BANTAM COMPUTER BOOKS

MANUAL OFICIAL DO AmigaDOS

Release 2.04

Inclui todas as versões até 2.04



Commodore-Amiga, Inc.

Berkeley

EBRAS Editora Brasileira

AMIGA™



Aprenda com a Commodore-Amiga, Inc. a dominar o AmigaDOS 2.04, o mais novo sistema operacional do Amiga publicado em português em três partes: o Manual do Usuário, o Manual de Desenvolvimento e o Manual de Referência Técnica.

O **Manual Oficial do AmigaDOS** descreve este poderoso sistema operacional mais profundamente do que qualquer outra documentação publicada e oferece uma referência completa aos comandos, funções e à estrutura interna do AmigaDOS. Você receberá informações detalhadas sobre as duas interfaces do usuário, *Workbench* e *Command Line Interface (CLI)* e sobre quais programas e comandos podem ser executados sob uma determinada interface. Com a versão 2 do AmigaDOS foi adicionado um Shell à CLI – o que faz com que a CLI seja ainda mais flexível e poderosa. Além disso, você aprenderá sobre:

- As diferenças entre a CLI e o Shell
- A seqüência de inicialização
- O diretório de *scripts*
- Chamadas à biblioteca/diretório
- *Scripts* de comandos
- Como abrir uma janela no Shell
- O novo Shell e seus inúmeros ícones
- Janelas de tela
- Códigos e mensagens de erros completos
- Glossário completo de termos do AmigaDOS

O Manual de Desenvolvimento, parte do livro, mostra como programar no Amiga e inclui uma cobertura completa do Linkeditor do AmigaDOS (ALink). Neste manual você aprenderá mais sobre:

- Como usar ALink
- A sintaxe dos comandos
- Saídas de MAP e XREF
- *Overlays*
- O guia de referência aos símbolos

Berkeley **ISBN 85-7251-029-X**

054 CITEC

Commodore Amiga, Inc.

LITEC
LIVRARIA EDITORA TÉCNICA LTDA
Rua dos Timbiras, 257 - CEP: 01208-010 - São Paulo
Tel.: 222.0477 - Fax: 220.2058
REF. 2133 PREÇO 10,30 URVs

Manual Oficial do AmigaDOS

TRADUÇÃO
CLAUDIO JOSÉ DA COSTA
TRADUTOR DE INFORMÁTICA



Copyright (c) 1991 by Commodore Capital Inc. Published by arrangement with
Bantam Books, a division of Bantam Doubleday Dell Publishing Group, Inc.

Translation Copyright (c) 1992, by Berkeley Brasil Editora.

Nenhuma parte desta publicação poderá ser reproduzida ou transmitida por qualquer modo ou meio, seja este eletrônico, fotográfico, mecânico, ou outros, sem autorização prévia e escrita da Berkeley Brasil Editora.

Editor de Aquisições: Ricardo Reinprecht
Editora: Gisella Narcisi
Marketing: Anna Maria Narcisi
Capa: Pit
Editoração Eletrônica e Produção: *RioTexto*

CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional dos Editores de Livros, RJ.

M251 Manual Oficial do AmigaDOS / Commodore Amiga ; tradução: Claudio José da Costa. – Rio de Janeiro : Berkeley Brasil Ed., 1992.

Tradução de: AmigaDOS
ISBN 85-7251-029-X

1. AmigaDOS (Sistema operacional de computador). 2. Sistemas operacionais de computador. I. Commodore Amiga.

CDD - 001.6425
CDU - 681.32

91-1020

Contato com autores, marketing e informações editoriais:

Berkeley Brasil Editora
Rua Almirante Tamandaré, 66 Grupo 313 - Flamengo
22210 - Rio de Janeiro - RJ
Tel/FAX (021) 225-3882

Distribuição e comercialização:

CITEC Ciência e Tecnologia Livraria Editora Ltda.
Rua São Januário, 817 - São Cristóvão
20921 - Rio de Janeiro - RJ
Tels: (021) 580-0034/580-5480/580-1918
FAX (021) 580-5996

Distribuição São Paulo:

DISAL S.A. Distribuidores Associados de Livros
Rua Vitória, 486/496 – Centro
01210 – São Paulo – SP
Tel.: (011) 221-1011 – FAX (011) 223-0306 – Telex (11)39698

Prefácio

Este livro, o Manual Oficial do AmigaDOS, é dividido em três partes:

O Manual do Usuário

O *Manual do Programador*

O *Manual de Referências Técnicas*

O *Manual do Usuário* contém informações do interesse de todos os usuários do Amiga. Há diversos comandos que o AmigaDOS reconhece além daqueles podem ser acessados através do Workbench. Ativando a Interface de Linhas de Comando (CLI), o usuário terá esses novos comandos à sua disposição.

O *Manual do Programador* ensina como utilizar os recursos do AmigaDOS em seus programas, em vez de acessá-los através da interface de linhas de comando. Esta seção também apresenta a documentação completa do Amiga Linker.

O *Manual de Referências Técnicas* descreve as estruturas internas de dados do AmigaDOS. Ele mostra como são armazenados os dados nos discos e revela o formato dos "arquivos objeto" utilizados pelo AmigaDOS. Programadores e usuários mais avançados certamente acharão muito úteis as informações encontradas nesta seção.

Em conjunto, estas três partes constituem o guia essencial para a compreensão do AmigaDOS.

Agradecimentos

Esta terceira edição do Manual do AmigaDOS foi escrita por Dan Baker, Randell Jesup, John Orr, John Toebes e Isabelle Vesey e editada por Jono Hardjowirogo e Dan Baker. O manual original foi escrito por Tim King.

Agradecimentos especiais a Bruce Barret, Patria Brown, Pamela Clare, Alan Coslett, Andy Finkel, Paul Floyd, Jessica King, Liz Laban, Rob Peck, Carolyn Scheppner e Keith Stobie. Sem suas generosas contribuições e sugestões, a elaboração deste manual não teria sido possível.

Sumário

PARTE I MANUAL DO USUÁRIO 1

Prefácio *iii*

Agradecimentos *vii*

Introdução *3*

Como Abrir uma Janela do Shell 3

Workbench e CLI: Relações e Diferenças 3

Capítulo 1 Introdução ao AmigaDOS *5*

Os Processos do AmigaDOS 5

Operação do Terminal 6

Utilizando o Sistema de Arquivo 8
Nomeando arquivos *8*
Usando Diretórios *9*
Selecionando o Diretório Corrente *10*
Selecionando o Dispositivo Corrente *11*
Anexando Comentários *12*
Decifrando os Nomes dos Dispositivos *13*
Usando Convenções de Diretórios
e Dispositivos Lógicos *15*

Utilizando os Comandos do AmigaDOS 18
Rodando Comandos em Segundo Plano *18*
Executando Arquivos de Comandos *19*
Direcionando a Entrada e Saída dos Comandos *19*
Interrompendo o AmigaDOS *20*

O Processo de Validação 20

Comandos Mais Usados 20
Utilizando o Shell *21*

Alguns Comandos do AmigaDOS 21
Aos Novos Usuários *22*
Para Começar *22*
Copiando um Disco *22*
Formatando um Disco *23*
Preparando um Disco de Boot *23*
Preparando um Disco CLI *24*
Renomeando um Disco *24*
Consultando o Diretório *24*
Usando o Comando LIST *25*
Usando o Comando PROTECT *26*
Obtendo Informações Sobre os Arquivos *26*
Mudando o Diretório Corrente *27*
Acertando a Data e a Hora *27*
Redirecionando a Saída de um Comando *27*
Listando o Conteúdo de um Arquivo de Texto *28*
Mudando o Nome de um Arquivo *28*
Apagando Arquivos *28*
Copiando Arquivos *29*
Criando um Novo Diretório *30*
Será Que o Meu Arquivo Está Nesse Disco? *30*
Automatizando a Seqüência de Boot *31*
Associando Discos *31*
Comentários Finais 33

Capítulo 2 Os Comandos do AmigaDOS *35*

Argumentos e Opções dos Comandos 35

Convenções dos Comandos 36

Formatos 36

Gabaritos 37

Especificações dos Comandos do AmigaDOS 39
ADDBUFFERS *39* ADDMONITOR *40* ALIAS *41* ASK *42*
ASSIGN *43* AUTOPOINT *46* AVAIL *47* BINDDRIVERS *47*
BINDMONITOR *48* BLANKER *49* BREAK *49*
CALCULATOR *50* CD *51* CHANGETASKPRI *52* CLOCK *52*
CMD *53* COLORS *54* COPY *55* CPU *57* DATE *58*
DELETE *59* DIR *61* DISKCHANGE *62* DISKCOPY *63*
DISKDOCTOR *64* DISPLAY *65* ECHO *66* ED *67*
EDIT *72* ELSE *75* ENDCLI *76* ENDIF *76* ENDSHELL *77*
ENDSKIP *77* EVAL *78* EXCHANGE *80* EXECUTE *80*

Resumo dos Comandos de Pontos 83
FAILAT *84* FAULT *85* FILENOTE *86* FIXFONTS *87*
FKEY *87* FONT *87* FORMAT *89* GET *89* GETENV *90*
GRAPHICDUMP *90* ICONEDIT *91* ICONTROL *91* ICONX *92*
IF *93* IHELP *94* INFO *95* INITPRINTER *96* INPUT *96*
INSTALL *97* IPREFS *98* JOIN *98* KEYSHOW *99*
LAB *99* LIST *99* LOADWB *102* LOCK *103*
MAKEDIR *103* MAKELINK *104* MEMACS *104*
MORE *110* MOUNT *111* NEWCLI *111* NEWSHELL *112*
NOCAPSLOCK *113* NOFASTMEM *113* OVERSCAN *114*
PALETTE *114* PATH *115* POINTER *116* PRINTER *117*
PRINTERGFX *117* PRINTFILES *118* PROMPT *118*
PROTECT *119* QUIT *121* RELABEL *121* REMRAD *122*
RENAME *122* RESIDENT *123* RUN *125* SAY *126*
SCREENMODE *127* SEARCH *127* SERIAL *129*
SET *129* SETCLOCK *130* SETDATE *131* SETENV *131*
SETFONT *132* SETMAP *133* SETPATCH *134* SKIP *134*
SORT *135* STACK *136* STATUS *137* TIME *137*
TYPE *138* UNALIAS *138* UNSET *138* UNSETENV *139*
VERSION *139* WAIT *140* WBCONFIG *140*
WBPATTERN *141* WHICH *142* WHY *143*

Comandos do AmigaDOS – Tabela de Referência Rápida 144

Capítulo 3 Mensagens de Erro do AmigaDOS *149*

Capítulo 4 Glossário *157*

PARTE II MANUAL DO PROGRAMADOR 163

Capítulo 5 Programando no Amiga *165*

Introdução 165

Desenvolvimento de Programas para o Amiga 165
Primeiros Passos *165*
Chamando as Bibliotecas Residentes *166*
Criando um Programa Executável *166*

Rodando um Programa Sob o CLI 167

Ambiente Inicial em Assembler *167*
Ambiente Inicial em C *167*
Falhas de Execução das Rotinas *168*
Terminando um Programa *168*

Rodando um Programa Sob o Workbench 169

Fundamentos da Programação de I/O 169
Usando Handlers de Arquivo *173*
I/O Bufferizado *174*
Análise de Sintaxe da Linha de Comandos *175*

Capítulo 6 Chamando o AmigaDOS *181*

Sintaxe 181
Valores dos Registradores *181*
Grafia dos Nomes *181*
Valores de Retorno Booleanos *181*
Valores *182*

Funções do AmigaDOS 182
AbortPkt *182* AddBuffers *182* AddDosEntry *183*
AddPart *184* AddSegment *185* AllocDosObject *186*
AssignAdd *186* AssignLate *187* AssignLock *188*
AssignPath *188* AttemptLockDosList *189* ChangeMode *189*
CheckSignal *190* Cli *191* Close *191* CompareDates *192*
CreateDir *192* CreateNewProc *193* CreateProc *194*
CurrentDir *195* DateStamp *195* DateToStr *196* Delay *197*
DeleteFile *198* DeleteVar *198* DeviceProc *199* DoPkt *200*
DupLock *201* DupLockFromFH *201* EndNotify *202*
ErrorReport *202* ExAll *203* Examine *206* ExamineFH *207*
Execute *207* Exit *209* ExNext *209* Fault *211* FGetC *211*
FGets *212* FilePart *213* FindArg *213* FindCliProc *214*
FindDosEntry *215* FindSegment *215* FindVar *216*
Flush *217* Format *217* FPutC *218* FPuts *218* FRead *219*
FreeArgs *220* FreeDeviceProc *220* FreeDosEntry *221*
FreeDosObject *221* FWrite *222* GetArgStr *222*
GetConsoleTask *223* GetCurrentDirName *223*
GetDeviceProc *224* GetFileSysTask *225* GetProgramDir *225*
GetProgramName *226* GetPrompt *226* GetVar *227*
Info *228* Inhibit *229* Input *229* InternalLoadSeg *230*
InternalUnLoadSeg *231* IoErr *232* IsFileSystem *232*
IsInteractive *233* LoadSeg *233* Lock *234* LockDosList *234*
LockRecord *235* LockRecords *236* MakeDosEntry *237*
MakeLink *237* MatchEnd *238* MatchFirst *239* MatchNext *240*
MatchPattern *241* MatchPatternNoCase *241* MaxCli *242*
NameFromFH *243* NameFromLock *243* NewLoadSeg *244*
NextDosEntry *245* Open *245* OpenFromLock *246*
Output *247* ParentDir *247* ParentOfFH *248* ParsePattern *248*

ParsePatternNoCase *249* PathPart *250* PrintFault *250*
PutStr *251* Read *251* ReadArgs *252* ReadItem *255*
ReadLink *255* Relabel *256* RemAssignList *257*
RemDosEntry *257* RemSegment *258* Rename *258*
ReplyPacket *259* RunCommand *259* SameDevice *260*
SameLock *261* Seek *261* SelectInput *262* SelectOutput *263*
SendPkt *263* SetArgStr *264* SetComment *264*
SetConsoleTask *265* SetCurrentDirName *265*
SetFileDate *266* SetFileSize *266* SetFileSysTask *267*
SetIoErr *268* SetMode *268* SetProgramDir *269*
SetProgramName *269* SetPrompt *270* SetProtection *270*
SetVar *271* SetVBuf *272* SplitName *273* StartNotify *274*
StrToDate *274* StrToLong *276* SystemTagList *276*
UnGetC *277* UnLoadSeg *278* UnLock *279*
UnLockDosList *279* UnLockRecord *279* UnLockRecords *280*
VFPrintf *281* VFWritef *281* VPrintf *282* WaitForChar *283*
WaitPkt *284* Write *284* WriteChars *285*

Funções do AmigaDOS – Tabela de Referência Rápida 286

Capítulo 7 O Linker *291*

Introdução 291

Usando o Linker 292
Sintaxe da Linha de Comando *293*
Arquivos WITH *294*
Erros e Outras Exceções *295*
Saídas MAP e XREF *296*

Overlays 296
A Diretiva OVERLAY *297*
Referências a Símbolos *299*
Advertências Sobre Alguns Pontos *300*

Mensagens e Códigos de Erro 300

Capítulo 8 Entrada e Saída dos Dispositivos do AmigaDOS *303*

Os Dispositivos do AmigaDOS 304

Comunicação com os Dispositivos do AmigaDOS 307

PARTE III MANUAL DE REFERÊNCIAS TÉCNICAS 309

Capítulo 9 O Sistema de Arquivo *311*

Estrutura dos Arquivos do AmigaDOS 311

Root Block 311
Root Block do FFS *313*

User Directory Blocks 316
Algoritmo de Hashing *316*
User Directory Blocks do FFS *318*

File Header Block 319
File Header Block do OFS e FFS *319*
Hard Links e Soft Links *320*

File List Block 320
File List Block do OFS e FFS *321*

Blocos de Dados 321
Bloco de Dados do OFS *322*
Bloco de Dados do FFS *322*

Capítulo 10 Estrutura dos Arquivos Binários do Amiga *323*

Introdução 323
Terminologia *323*

Estrutura dos Arquivos Objeto 325

Arquivos de Carregamento 335

Exemplos 338

Estrutura dos Arquivos de Bibliotecas do Amiga 341
Exemplo de um Arquivo de Biblioteca *341*
Estrutura da Nova Biblioteca *342*
Exemplo de hunk_lib *343*
Exemplo de hunk_index *345*

Capítulo 11 Estrutura de Dados do AmigaDOS *349*

Introdução 349

Estruturas de Dados dos Processos 350
Redirecionando os Requesters do Sistema *354*

Estrutura das Bibliotecas do DOS 354
A Subestrutura Info *356*

Alocação da Memória 359

Listas de Segmentos 359

File Handles 360

Locks 360

Os Packets do AmigaDOS 361

Tipos de Packets 363
Entrada/Saída Básica *364*
Informação/Manipulação de Arquivos/Diretórios *367*
Informação/Manipulação de Volumes *378*
Manutenção e Controle do Handler *379*
Ações Internas do Handler *381*
Packets Obsoletos *382*
Packets Somente de Console *382*
Sumário dos Números de Packets Definidos *382*

Utilizando Packets Diretamente 385

Capítulo 12 Informações Adicionais para o Programador Avançado *389*

Tabela de Overlay de Hunks – Visão Geral 390
Projetando uma Árvore de Overlay *390*
Descrevendo a Árvore *391*

Criando um Novo Dispositivo para Ser Executado Sob o AmigaDOS 394

Criando Novos Dispositivos de Disco 395

Usando o AmigaDOS sem o Workbench / Intuition 395

Índice Analítico 397

Parte I

MANUAL DO USUÁRIO

Introdução

Este manual descreve o AmigaDOS e seus comandos. O Interpretador de Linhas de Comando (em inglês, *Command Line Interpreter* ou CLI) lê os comandos do AmigaDOS digitados numa janela e os traduz em ações que deverão ser executadas pelo computador. Neste sentido, o CLI é similar às interfaces "tradicionais" da maioria dos computadores: o usuário digita os comandos e a interface exibe textos em resposta.

Como Abrir uma Janela do Shell

Para ativar o CLI, também conhecido como Shell, inicialize o Amiga e use o mouse para selecionar o ícone do disco do Workbench (se você estiver utilizando o Amiga com software pré-instalado no disco rígido, selecione com o mouse o ícone do sistema 2.0). Ao surgir a janela do Workbench, selecione o ícone do Shell (um box contendo o símbolo "1>"). Feito isto, a janela do Shell se abrirá.

Para utilizar o Shell, basta digitar em seguida os comandos do AmigaDOS que você desejar. A janela do Shell pode ter o seu tamanho e localização na tela modificados como qualquer outra. Para fechar a janela do Shell, digite "END-SHELL", ou clique o gadget no canto superior esquerdo da janela.

Workbench e CLI: Relações e Diferenças

Digite "DIR" para exibir uma lista dos arquivos (e diretórios) contidos no diretório do disco atual. Esta é a lista dos arquivos que compõem o seu Workbench. Você pode notar que existem mais arquivos no diretório do que ícones no Workbench. A razão disto é que o Workbench só exibe um arquivo "X" caso este possua um arquivo "X.info" associado. O Workbench utiliza o arquivo ".info" para manipular o ícone.

O programa de cópia de discos, por exemplo, possui dois arquivos: um arquivo "Diskcopy", contendo o programa propriamente dito, e outro chamado "Diskcopy.info", contendo as informações para uso do Workbench. No caso de arquivos de dados gráficos, como "mount.pic" por exemplo, o arquivo "mount.pic.info" contém informações sobre o ícone e o nome do programa (default) que deverá processá-lo (Graphi-

Craft). Neste caso, quando o usuário "abre" o arquivo de dados, o Workbench roda o programa e passa o nome do arquivo (mount.pic) para ele.

Os subdiretórios do AmigaDOS correspondem às "gavetas" (drawers) que aparecem na tela do Workbench. Dispositivos de acesso aleatório tais como discos (DF0:) são equivalentes aos ícones de discos que você já viu.

Repare, contudo, que nem todos os programas e comandos podem ser executados tanto a partir do Workbench como do Shell. Muitos dos comandos do AmigaDOS descritos no Capítulo 2 deste manual funcionam somente sob o ambiente do Shell.

Ao longo deste livro, os termos "CLI" e "Shell" serão usados para se referir à janela especial onde você pode digitar os comandos do AmigaDOS.

Capítulo 1

Introdução ao AmigaDOS

Este capítulo fornece uma visão geral do sistema operacional AmigaDOS, incluindo informações sobre a operação do terminal, a estrutura dos diretórios e a utilização dos comandos do sistema. No final deste capítulo, você encontrará um exemplo de sessão de trabalho com o AmigaDOS.

Os Processos do AmigaDOS

O AmigaDOS é um sistema operacional **multitarefa** projetado para o Amiga. Normalmente, o AmigaDOS é utilizado por apenas um usuário. A multitarefa é uma facilidade que permite executar vários trabalhos ao mesmo tempo, ou ainda suspender determinada tarefa enquanto se executa uma outra.

Cada **processo** do AmigaDOS representa um processo particular do sistema operacional – por exemplo, o sistema de arquivo (você o conhecerá em detalhes mais adiante). Somente um processo é executado de cada vez; os outros ficam esperando que alguma coisa aconteça, ou então são interrompidos e permanecem aguardando até poderem prosseguir. Cada processo tem uma **prioridade** associada, sendo que o processo com a mais alta prioridade que estiver livre para rodar é executado. Os processos de prioridade mais baixa só rodam quando os de maior prioridade estiverem, por algum motivo, em condição de espera – aguardando pela chegada de informações contidas num disco, por exemplo.

O sistema padrão do AmigaDOS emprega diversos processos, conhecidos como **processos do sistema**, que prescindem da interferência do usuário. Este é o caso do processo que manipula a linha serial. Outros exemplos são os processos que manipulam o console e a gravação e leitura de dados no disco (quando a configuração do hardware conta com mais de um drive, existe um processo separado para cada drive).

O AmigaDOS dispõe de um processo do qual o usuário pode se servir, chamado **Interface de Linhas de Comando** ou **Shell.** Os processos do Shell lêem **comandos** e os executam em seguida. Vários desses processos – que são numerados de 1 em diante – podem ser executados simultaneamente. Você pode criar novos processos Shell utilizando os comandos NEWSHELL ou RUN. Para remover um processo

Shell, basta usar o comando ENDSHELL (uma descrição detalhada destes comandos é apresentada no Capítulo 2).

Operação do Terminal

O usuário pode dirigir as informações digitadas no terminal para a Interface de Linhas de Comando (Shell), que diz ao AmigaDOS para carregar um programa, ou então enviar essas informações para um programa rodando sob o ambiente do Shell. Em ambos os casos, um processo de manipulação do **console** (ou **terminal**) se encarrega de controlar a entrada e a saída de dados. Este processo também executa a edição local das linhas e diversas outras funções.

Você pode digitar linhas de comando de até 512 caracteres. Para corrigir um erro, pressione a tecla BACKSPACE, isto apagará o último caractere digitado. Para apagar a linha inteira, pressione simultaneamente as teclas CTRL e X (a combinação de **teclas de controle** será referenciada daqui por diante, neste manual, como "CTRL-X"). Você também pode utilizar as teclas de setas para a esquerda e para a direita a fim de deslocar o cursor até a posição da linha onde haja a necessidade de efetuar alguma correção.

O Shell permite ainda que você pesquise a última ocorrência de um comando específico. Para tanto, digite a linha de comando desejada (ou o começo dela) e pressione simultaneamente as teclas SHIFT e cursor para cima (ou CTRL-R). Por exemplo: se você digitar DIR e em seguida SHIFT-cursor para cima, aparecerá a última linha de comando onde um DIR foi executado. As teclas SHIFT e cursor para baixo remetem ao final do buffer de comandos, deixando o cursor numa linha em branco.

O Shell mantém um buffer com uma cópia (ou **histórico**) dos últimos comandos digitados, possibilitando ao usuário chamar, editar e introduzir novamente linhas de comando já executadas. Isto é particularmente útil quando você necessita repetir algum comando ou introduzir uma seqüência de comandos semelhantes.

Esse buffer tem um tamanho de 2K e o número de linhas que ele comporta varia em função da extensão das linhas de comando. Quando o buffer fica cheio, as linhas mais antigas são perdidas. Para acessar as linhas contidas no buffer de comandos, basta utilizar as teclas de movimentação do cursor:

cursor para cima	Move em direção ao início do buffer de comandos (primeiras linhas).
cursor para baixo	Move em direção ao final do buffer de comandos (últimas linhas).

Quando o usuário começa a escrever uma linha de comando, o AmigaDOS espera o término da digitação antes de exibir qualquer outra mensagem no vídeo. Desse modo você pode continuar digitando sem que haja o risco de se misturarem a entrada e a saída de dados.

O AmigaDOS reconhece o término da digitação de uma linha quando o usuário aciona a tecla RETURN. O sistema também considera que uma linha está encerrada caso a mesma seja cancelada (para fazer isso, digite CTRL-X ou pressione BACKSPACE até que todos os caracteres dessa linha sejam apagados). Após receber a linha, o AmigaDOS continuará a exibir as mensagens que porventura tenham sido

retidas no início da digitação. Se você quiser fazer uma pausa para poder visualizar melhor o que está aparecendo na tela, simplesmente digite qualquer caractere (ou então pressione a barra de espaço) e a saída no vídeo será interrompida. Pressionando-se CTRL-X, BACKSPACE ou RETURN, a imagem no vídeo voltará a ser atualizada (note que se você teclar RETURN, o AmigaDOS tentará executar a linha de comando digitada depois que o programa corrente terminar).

O AmigaDOS reconhece CTRL-\ como indicador de fim de arquivo. Em determinadas circunstâncias, esta combinação pode ser utilizada para indicar o término de um arquivo de dados (para saber quando isso acontece, veja o tópico "Entendendo os Nomes dos Dispositivos", mais à frente neste capítulo).

Se você observar o surgimento de caracteres estranhos na tela ao digitar alguma coisa pelo teclado, é provável que as teclas CTRL-O tenham sido acionadas inadvertidamente. O AmigaDOS reconhece esta combinação como uma instrução para o dispositivo de console (CON:) exibir o conjunto de caracteres alternativo. Pressione CTRL-N para desfazer esta condição, e os caracteres subseqüentes deverão aparecer novamente com o formato normal (outra alternativa seria pressionar ESC-C para ressetar o console, isto limpa a tela e mostra o texto no modo normal).

A tabela abaixo resume os comandos de edição disponíveis no Shell.

Comandos de edição do Shell

cursor para a esquerda	Move o cursor um caractere para a esquerda.
cursor para a direita	Move o cursor um caractere para a direita.
Shift-cursor p/ esquerda	Move o cursor para o início da linha.
Shift-cursor p/ direita	Move o cursor para o fim da linha.
Backspace	Deleta o caractere à esquerda do cursor.
Del	Deleta o caractere destacado pelo cursor.
Ctrl-H	Deleta o último caractere (o mesmo que Backspace).
Ctrl-M	Processa a linha de comando (o mesmo que Return).
Ctrl-J	Acrescenta um line-feed.
Ctrl-W	Deleta a palavra à esquerda do cursor.
Ctrl-X	Deleta a linha atual.
Ctrl-K	Deleta todos os caracteres do cursor até o fim da linha.
Ctrl-Y	Repõe os caracteres apagados com Ctrl-K.
Ctrl-U	Deleta todos os caracteres do cursor até o início da linha.
Barra de espaço	Suspende a saída (interrompe o rolamento das mensagens para o alto da tela).
Backspace	Reinicia a saída (dá continuidade ao rolamento das mensagens para o alto da tela).
Ctrl-C	Envia um comando BREAK ao processo atual (suspende o processo).
Ctrl-D	Envia um comando BREAK ao script atual (suspende o script).
Ctrl-S	Suspende a saída.
Ctrl-Q	Reinicia a saída suspensa com Ctrl-S.
Ctrl-\	Fecha a janela do Shell.

Finalmente, saiba que para o AmigaDOS é indiferente a digitação dos comandos e **argumentos** com letras maiúsculas ou minúsculas. O AmigaDOS exibe o **nome do arquivo** com a mesma grafia utilizada quando o usuário o criou, mas é capaz de encontrar qualquer arquivo, não importando para isso se você especificar o nome do mesmo com letras maiúsculas ou minúsculas.

Utilizando o Sistema de Arquivo

Esta seção descreve o sistema de arquivo (em inglês, *filing system*) do AmigaDOS. Em particular, você verá como nomear, organizar e recuperar os seus arquivos.

Um arquivo é o menor elemento reconhecido pelo AmigaDOS. A maneira mais simples de se identificar um arquivo é através de seu nome (o que é demonstrado no tópico abaixo, "Nomeando Arquivos"). Em determinadas situações, no entanto, pode ser necessário identificar um arquivo com maior exatidão. Nesse caso, a identificação do arquivo deverá incluir também o nome do dispositivo ou volume, e/ou o nome do diretório (ou diretórios). Isto será discutido nas seções posteriores.

Nomeando arquivos

O AmigaDOS conserva as informações em disco num determinado número de **arquivos**, nomeados de maneira que você possa identificá-los e recuperá-los. Os nomes dos arquivos deverão ter um tamanho máximo de 30 caracteres. Nos nomes dos arquivos poderão ser usadas todas as letras que aparecem impressas no vídeo, com exceção da barra (/) e os dois pontos (:). Isto quer dizer que você pode incluir espaços, aspas (") e os sinais de mais (+) ou igual (=) nos nomes de seus arquivos. A única ressalva é que o CLI reconhece estes sinais como caracteres especiais, e você deverá escrever todo o nome do arquivo entre aspas quando utilizá-los. Para introduzir o caractere aspas no nome de um arquivo, deve-se digitar antes do mesmo um asterisco (*). Esta regra também vale caso você queira incluir o próprio caractere asterisco no nome do arquivo. Exemplificando, um arquivo chamado

```
A*B = C"
```

deverá ser digitado como se segue:

```
A**B = C*"
```

para que o CLI o aceite.

Nota: este uso do asterisco contrasta com o de muitos outros sistemas operacionais, onde este símbolo é empregado como curinga (uma chave especial que pode ser utilizada em substituição ao nome do arquivo). No Amiga, o padrão que serve de curinga é #?. Um asterisco sozinho no AmigaDOS representa o teclado e a janela atual. Por exemplo,

```
COPY arquivo to *
```

copia o arquivo especificado na tela.

Além disso, o uso de espaços no nome de um arquivo, diretório ou dispositivo torna obrigatório o uso de aspas em todas as referências a esse nome (por exemplo: copy "df0:Meu arquivo" to ram).

Evite usar espaços antes ou depois do nome do arquivo, pois isto pode provocar confusão.

Usando Diretórios

O sistema de arquivo também permite o uso de **diretórios** como uma maneira de agrupar os arquivos em unidades lógicas. Você pode, por exemplo, utilizar dois diretórios diferentes para guardar os fontes e a documentação de seus programas, ou ainda para manter separados os arquivos pertencentes a duas pessoas.

Cada arquivo num disco pertence a um diretório. Um disco vazio contém um diretório chamado de **diretório raiz**. Se você criar um arquivo num disco vazio, este arquivo pertencerá ao diretório raiz desse disco. Entretanto, um diretório pode conter outros diretórios (conhecidos como **subdiretórios**). Assim, cada diretório pode conter arquivos, ou mais diretórios, ou uma mistura de ambos. Note que todo arquivo é único dentro do diretório a que pertence Assim, um arquivo chamado "pedro" no diretório "joão" é completamente diferente de um arquivo "pedro" no diretório "maria".

Esta estrutura significa que dois usuários repartindo um mesmo disco não precisam se preocupar com a possibilidade de sobreposição acidental de seus arquivos, basta que cada usuário crie os seus arquivos num diretório individual.

ATENÇÃO: Quando você cria um arquivo com o mesmo nome de outro arquivo já existente no diretório corrente, o AmigaDOS apaga o arquivo existente. Nenhuma mensagem dando conta deste fato é exibida.

Você também pode se valer dessa estrutura de arquivo para organizar as informações nos discos, mantendo os diferentes tipos de arquivos em diretórios específicos. O exemplo a seguir ajuda a clarificar isto. Considere um disco contendo dois diretórios, chamados "joão" e "maria". O diretório "joão" contém dois arquivos, chamados "texto" e "carta". O diretório "maria" contém um arquivo chamado "dados" e dois diretórios chamados "carta" e "fatura". Cada um desses diretórios contém um arquivo chamado "18jun". A figura 1-A representa graficamente essa estrutura:

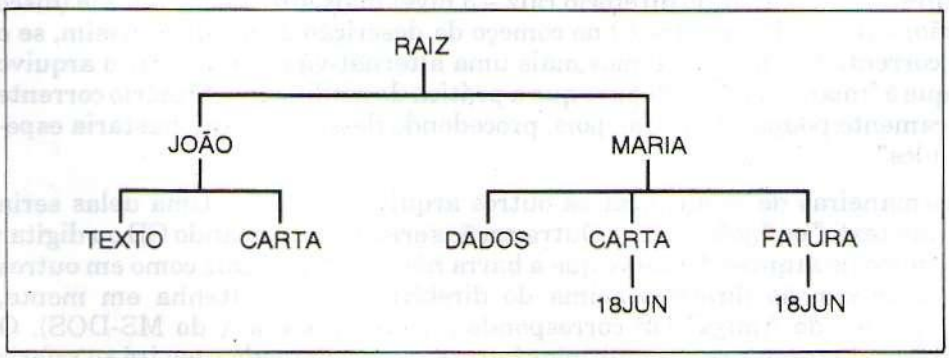


Figura 1-A Usando a estrutura do diretório.

Nota: O diretório "joão" tem um arquivo chamado "carta", enquanto que o diretório "maria" tem um subdiretório chamado "carta". Repare que não há confusão, uma vez que ambos estão em diretórios diferentes. Note ainda que não há limite de profundidade para se "aninhar" os diretórios.

Para identificar com precisão um arquivo, é necessário indicar o diretório onde esse arquivo se encontra, o diretório a que esse diretório pertence, e assim por diante. Para tanto basta que você digite os nomes dos diretórios no caminho (ou *path*) até o arquivo desejado. Para separar o nome de cada diretório do próximo diretório ou nome do arquivo, digite uma barra (/). Assim, a especificação completa dos arquivos de dados do disco mostrado na figura 1-A seria a seguinte:

```
joao/texto
joao/carta
maria/dados
maria/carta/18jun
maria/fatura/18jun
```

Selecionando o Diretório Corrente

Como pode se tornar bastante complicado digitar a descrição completa de um arquivo, o AmigaDOS trabalha com a noção de um **diretório corrente**. Salvo indicação em contrário, é nesse diretório que o sistema irá procurar os arquivos especificados. Para definir o diretório corrente, deve-se usar o comando CD (de *current directory*). Se você estabelecer "maria" como o diretório corrente, os seguintes nomes serão suficientes para especificar os arquivos de dados contidos nesse diretório:

```
dados
carta/18jun
fatura/18jun
```

Qualquer diretório pode ser selecionado como o diretório corrente, para especificar quaisquer arquivos deste diretório, bastará digitar o nome do arquivo. Para especificar arquivos dentro de subdiretórios, você deverá digitar os nomes dos diretórios até o arquivo desejado, partindo do novo diretório corrente.

Todos os arquivos do disco continuarão disponíveis, não importando o nível do diretório corrente que você selecionar. Para instruir o AmigaDOS a procurar em todos os diretórios a partir do diretório raiz – o nível mais alto de um volume (disco ou partição) – digite dois pontos (:) no começo da descrição do arquivo. Assim, se o diretório corrente for "maria", temos mais uma alternativa para acessar o arquivo "dados", que é ":maria/dados". Repare que a prática de modificar o diretório corrente visa basicamente poupar digitação pois, procedendo dessa maneira, bastaria especificar "dados".

Há várias maneiras de se alcançar os outros arquivos do disco. Uma delas seria digitar ":joão/texto" ou ":joão/carta". Outra opção seria usar o comando CD ou digitar / antes do nome do arquivo. Observe que a barra não se refere à raiz como em outros sistemas, mas sim ao diretório acima do diretório corrente (tenha em mente, portanto, que a / do AmigaDOS corresponde à / do Unix e à \ do MS-DOS). O AmigaDOS aceita o emprego de múltiplas barras, sendo que cada uma irá se referir ao nível de diretório imediatamente superior.

Assim, supondo que o diretório corrente seja "maria:carta", você poderia especificar o arquivo ":maria/fatura/18jun" como "/fatura/18jun". Para se referir aos arquivos em "joão", bastaria digitar:

```
CD :joao
```

ou

```
CD //joao
```

Então você poderia especificar qualquer arquivo em "joão" somente pelo nome. Naturalmente, nada o impede de lançar mão do recurso das barras para se referir diretamente a um arquivo específico. Por exemplo,

```
TYPE //joao/carta
```

listará o conteúdo do arquivo sem que haja a necessidade de selecionar "joão" como o diretório corrente.

Para ir direto ao nível raiz, digite sempre dois pontos (:) seguidos pelo nome do diretório. Para utilizar barras, é necessário que você conheça o número exato de níveis para trás a ser percorrido até chegar no arquivo desejado.

Selecionando o Dispositivo Corrente

Como se sabe, o Amiga é capaz de operar com mais de um drive (ou dispositivo de disco). Cada dispositivo de disco recebe um nome, no formato DFn (tal como DF1) onde "n" se refere ao número do dispositivo – por ora, o Amiga aceita nomes de dispositivos de DF0 até DF3. Cada disco também pode ser associado a um nome individual, conhecido como nome de volume, você verá isto em detalhes mais adiante.

Além disso, o dispositivo lógico SYS: é associado ao disco utilizado para dar o boot no sistema. Você pode empregar este nome no lugar do nome do dispositivo de disco correspondente (ou seja, é lícito usar SYS: no lugar de DF0:, por exemplo).

O diretório corrente, por sua vez, é também associado a um **drive corrente** (o drive onde se acha o diretório). Como já vimos, iniciar a descrição de um arquivo com dois pontos (:) serve para identificar o diretório raiz do drive corrente. Para designar o diretório raiz de um drive particular, deve-se digitar o nome do drive antes dos dois pontos. Assim, para especificar o arquivo "dados" no diretório "maria", pode-se usar "DF1:maria/dados" (assumindo que você tenha inserido o disco no drive DF1:). Logo, para designar um arquivo no drive DF0: chamado "pedro/projetos", basta digitar "DF0:pedro/projetos", não importando qual seja o diretório corrente.

Nota: Quando você indicar um drive ou qualquer outro dispositivo, sozinho ou dentro do nome de um diretório, não esqueça de digitar dois pontos no final (DF1:, por exemplo).

A figura 1-B ilustra a estrutura de uma descrição de arquivo. A figura 1-C mostra alguns exemplos de descrições de arquivos aceitas pelo AmigaDOS.

À esquerda do (:)	À direita do (:)	À direita da (/)
Nome do dispositivo ou Nome do volume	Nome do diretório ou Nome do arquivo	Nome do Subdiretório ou Nome do arquivo

Figura 1-B Estrutura de uma descrição de arquivo.

```
SYS:commands
DF0:joão
DF1:maria/carta
DF2:maria/carta/18jun
DOC:relatório/parte1/figuras
FONTS:silly-font
C:cls
```

Figura 1-C Exemplos de descrições de arquivos.

Para acessar um arquivo num determinado disco você pode digitar o nome do disco, conhecido como **nome do volume**, ao invés do nome do dispositivo. Se o arquivo se encontra no disco "MCC", por exemplo, é válido especificá-lo digitando o nome "MCC:pedro/projetos". É possível também usar o nome do volume para se referir a um disco independentemente do drive onde ele se encontra. O nome do volume é atribuído no momento em que o usuário formata o disco (para maiores detalhes, veja "FORMAT" no Capítulo 2). Para modificar o nome do volume deve ser usado o comando RELABEL.

Ao contrário do que acontece com o volume, o dispositivo não é realmente parte integrante do nome. O AmigaDOS pode, por exemplo, ler um arquivo criado em DF0: em outro drive, tal como DF1:, se você inserir o disco neste drive (assumindo, naturalmente, que os drives sejam compatíveis). Isto é: o arquivo "joão", criado num disco no drive DF0: e portanto conhecido como "DF0:joão", continua a ser lido normalmente se você inserir o disco no drive DF1:. Só que agora, o mesmo passará a ser conhecido como "DF1:joão".

Anexando Comentários

Embora o nome possa fornecer uma pista, geralmente precisamos consultar o próprio arquivo quando desejamos obter informações mais precisas sobre o seu conteúdo. O AmigaDOS, contudo, apresenta uma solução prática para contornar esse problema. O comando chamado FILENOTE permite que você anexe comentários a seus arquivos. Um comentário pode ter até 79 caracteres (como sempre, se forem utilizados espaços, o texto deverá ser escrito entre aspas). Qualquer tipo de

informação pode constar nesse comentário: a data de criação do arquivo, se determinado bug foi ou não consertado, o número da versão do programa, ou qualquer outro dado capaz de auxiliar na identificação do mesmo.

Cabe ao usuário associar os comentários aos arquivos que desejar, utilizando para isso o comando FILENOTE. Um arquivo ao ser criado não possui comentários, mesmo sendo cópia de um arquivo comentado (os comentários não são copiados para um novo arquivo, embora sejam preservados quando o próprio arquivo é regravado). Caso você queira escrever um programa capaz de copiar um arquivo e o comentário correspondente, será preciso dispender algum trabalho extra para copiar o comentário (maiores detalhes no Capítulo 6).

O comentário não é modificado quando o usuário renomeia o arquivo. O comando RENAME apenas troca o nome do arquivo, tanto o seu conteúdo como os comentários anexados permanecem inalterados. Para maiores detalhes, veja a descrição dos comandos LIST e FILENOTE no Capítulo 2.

Decifrando os Nomes dos Dispositivos

Os dispositivos são nomeados a fim de que o usuário possa se referir a eles por meio de nomes. Nomes de drives como DF0: são exemplos de **nomes de dispositivos**. Lembre-se de que você pode se referir aos nomes de arquivos e dispositivos usando letras maiúsculas ou minúsculas. No caso dos drives, o nome do dispositivo é seguido pelo nome de um arquivo, uma vez que o AmigaDOS suporta arquivos nestes dispositivos. Além disso, o nome do arquivo pode incluir diretórios, pois o AmigaDOS também suporta diretórios.

É possível ainda criar arquivos na memória, utilizando-se um dispositivo chamado RAM:. Este dispositivo implementa um sistema de arquivo na memória que suporta todos os comandos do sistema de arquivo normal.

Nota: Se você estiver utilizando o AmigaDOS versão 1.3 ou anterior, lembre-se de que RAM: requer que a biblioteca L:/ram-handler esteja no disco. A partir da versão 2.0, a RAM-handler fica residente em ROM.

O dispositivo RAM: possibilita, por exemplo, a criação de um diretório para a cópia de todos os comandos na memória. Para fazer isto, digite:

```
MAKEDIR ram:c
COPY sys:c TO ram:c
ASSIGN C: RAM:C
```

Para observar o resultado, digite DIR RAM:. Isto deverá incluir o conteúdo do diretório "c" (listado por DIR como c(dir)). A cópia dos comandos na memória faz com que o carregamento dos mesmos fique muito mais rápido, mas em contrapartida deixa pouca memória sobrando para se fazer qualquer outra coisa. Além disso, todos os arquivos contidos no dispositivo RAM: serão perdidos quando você ressetar o equipamento.

O AmigaDOS possui vários outros dispositivos que podem ser utilizados pelo usuário. Os parágrafos seguintes discutem esses dispositivos, tais como NIL:, SER:, PAR:, PRT:, CON: e RAW:.

NIL:, particularmente, é um dispositivo "fantasma" (isto é, simulado). O AmigaDOS simplesmente descarta a saída dirigida para este dispositivo. A leitura a partir de NIL: resulta numa indicação de final de arquivo. Por exemplo, você pode digitar:

```
EDIT abc TO nil:
```

para usar o editor a fim de dar uma espiada no conteúdo do arquivo chamado "abc" enquanto o AmigaDOS descarta a saída editada.

O dispositivo SER: tem por objetivo possibilitar o acesso a qualquer dispositivo conectado à linha serial (geralmente, uma impressora). Assim, a seguinte seqüência de comandos:

```
COPY xyz TO ser
```

faz com que o AmigaDOS envie o conteúdo do arquivo "xyz" para a linha serial. Note que o dispositivo serial só copia em múltiplos de 400 bytes de cada vez, copiar com SER:, desse modo, pode parecer granular.

O mesmo que foi dito acima se aplica ao dispositivo PAR:, só que em relação à porta serial.

O dispositivo PRT: (de PRinTer, impressora) se refere à impressora selecionada pelo usuário no programa Preferences. Através deste programa, você pode definir se a conexão de sua impressora será feita através da porta serial ou paralela. Assim, a seqüência de comandos:

```
COPY xyz TO PRT:
```

imprime o arquivo "xyz", não importando onde a impressora esteja conectada.

Todos os dados enviados à saída PRT: serão traduzidos pelo driver de impressora selecionado no programa Preferences. Este driver converte os códigos de escape padrão ANSI nos códigos específicos requeridos pela impressora. PRT: transforma, por exemplo, todos os códigos de avanço de linha numa seqüência de retorno de carro e avanço de linha. Algumas impressoras, contudo, não aceitam caracteres convertidos. Para contornar esse problema e enviar um arquivo com avanço de linha sem retorno de carro, certifique-se de usar PRT:RAW em vez de PRT:.

O AmigaDOS também suporta múltiplas janelas. Para criar uma nova janela, você pode especificar o dispositivo CON:. O formato de CON: é o seguinte:

```
CON:x/y/largura/altura/[título]
```

onde "x" e "y" correspondem às coordenadas horizontal e vertical; "largura" e "altura" são inteiros que descrevem a largura e a altura da nova janela, e "título" (opcional) é a a mensagem a ser exibida na barra de título da janela. É necessário digitar todas as barras (/) do comando, inclusive a última. O string do título pode conter até 30 caracteres, incluindo espaços. Caso o título contenha espaços, toda a descrição do comando deverá ser digitada entre aspas, como mostra o exemplo abaixo:

```
"CON:20/10/300/100/minha janela"
```

A partir da versão 2.0 do AmigaDOS, CON: dispõe de recursos adicionais para a criação de janelas. Consulte o Capítulo 8 para maiores informações a respeito.

Há outro dispositivo de janelas, porém de pouca utilidade para o usuário comum,

chamado RAW: (mais detalhes sobre isto no Capítulo 8). RAW pode ser utilizado para se criar um dispositivo de janelas bruto similar a CON:. Ao contrário de CON:, no entanto, RAW: não efetua a conversão de caracteres e não permite que o conteúdo de uma linha seja modificado. Noutras palavras, RAW: aceita a entrada de caracteres e os exibe exatamente da maneira como foram digitados. Isto significa que os caracteres serão enviados para um programa imediatamente, sem que você tenha a chance de corrigir qualquer engano com a tecla BACKSPACE. Normalmente RAW: é empregado em programas que requerem a entrada e saída de caracteres sem conversão.

ATENÇÃO: RAW: foi previsto para usuários experientes. Não utilize RAW: experimentalmente.

Existe ainda um nome especial, que é o * (asterisco). Você pode usá-lo para se referir à janela atual, tanto para a entrada como para a saída de dados. Assim como o comando COPY é capaz de efetuar a cópia de um arquivo para outro, usando * podemos copiar o conteúdo da janela atual para uma outra janela. Por exemplo,

```
COPY * TO CON:20/20/350/150/
```

Também é possível copiar da janela atual para a janela atual:

```
COPY * TO *
```

Ou ainda, de um arquivo para a janela atual:

```
COPY joao/carta TO *
```

O AmigaDOS terminará a cópia ao encontrar o fim do arquivo. Para interromper o processo, use o comando CTRL-\. Lembre-se de que * NÃO é um curinga.

Usando Convenções de Diretórios e Dispositivos Lógicos

Além dos dispositivos físicos mencionados anteriormente, o AmigaDOS suporta vários **dispositivos lógicos**. O AmigaDOS utiliza esses dispositivos para encontrar arquivos freqüentemente requisitados pelos programas do usuário. Desse modo os programas podem se referir a um nome de dispositivo padrão, independentemente de onde tais arquivos se encontrem armazenados. Você pode modificar as associações de todos esses dispositivos lógicos a fim de referenciar qualquer diretório.

Os dispositivos lógicos descritos nesta seção estão relacionados na figura 1-D.

Nome do dispositivo lógico: SYS:
Nome do diretório típico: Workbench:

"SYS" representa o diretório raiz do disco de sistema. Quando você inicializa o Amiga, o AmigaDOS associa SYS: ao nome do diretório do disco contido em DF0:. Por exemplo: se o disco no drive DF0: tiver o nome de volume Workbench, o AmigaDOS associará SYS: a Workbench:. Daí para a frente, quando qualquer programa se referir a SYS: será utilizado o diretório raiz do disco do Workbench.

Nome	Descrição	Diretório
SYS:	Diretório raiz do disco de sistema.	SYS:
C:	Diretório de Comandos.	SYS:C
L:	Diretório de Bibliotecas.	SYS:L
S:	Diretório de Scripts.	SYS:S
LIBS:	Diretório para chamadas Open Library.	SYS:LIBS
DEVS:	Diretório para chamadas Open Device.	SYS:DEVS
FONTS:	Fontes carregáveis para Open Fonts.	SYS:FONTS
	Espaço temporário de trabalho.	RAM:T

Figura 1-D Dispositivos lógicos.

Nome do dispositivo lógico: C:
Nome do diretório típico: Workbench:c

"C" representa o diretório de comandos (*commands directory*). Quando você digita um comando para o CLI (DIR <cr>, por exemplo), o AmigaDOS procura por esse comando primeiramente no diretório corrente. Caso não consiga encontrar o comando nesse diretório, o sistema procurará por "C:DIR". Assim, se você associar "C:" a outro diretório (por exemplo, "disco_boot:c"), o AmigaDOS lerá e executará "disco_boot:c/DIR".

Nome do dispositivo lógico: L
Nome do diretório típico: Workbench:l

"L" representa o diretório das bibliotecas (*library directory*). Neste diretório ficam os overlays dos comandos mais extensos e as partes não residentes do sistema operacional. As bibliotecas de run-time baseadas em disco (Aux-Handler, Port-Handler etc.), por exemplo, são guardadas aqui. O AmigaDOS requer este diretório para poder operar.

Nome do dispositivo lógico: S
Nome do diretório típico: Workbench:s

"S" representa o diretório de scripts. Este diretório contém os scripts de comandos procurados e executados pelo comando EXECUTE. O comando EXECUTE procura primeiro pelo script (ou batch) no diretório corrente. Se não conseguir encontrá-lo, EXECUTE passará a procurar no diretório ao qual S: estiver associado.

Nome do dispositivo lógico: LIBS:
Nome do diretório típico: Workbench:LIBS

O sistema procura aqui pela biblioteca chamada se ela não estiver carregada na memória.

Nome do dispositivo lógico: DEVS
Nome do diretório típico: Workbench:DEVS

As chamadas Open Device do sistema operacional procuram aqui pelo dispositivo se ele não estiver carregado na memória.

Nome do dispositivo lógico: FONTS
Nome do diretório típico: Workbench:FONTS

As chamadas Open Fonts procuram aqui pelas suas fontes carregáveis se elas não estiverem carregadas na memória.

Nota: Além dos diretórios acima, que podem ser reassociados pelo usuário, muitos programas abrem arquivos no diretório "T". Este diretório é utilizado para armazenar arquivos temporariamente. Programas como editores de textos guardam neste diretório seus arquivos de trabalho temporários e cópias back-up do último arquivo editado. Se você ficar sem espaço num disco, este é o primeiro lugar onde procurar por arquivos que não sejam mais necessários.

Quando o sistema é inicializado, o AmigaDOS associa C: ao diretório :C. Isto significa que se você der um boot usando um disco formatado com o comando:

```
FORMAT DRIVE DF0: NAME "Meu.Disco.Boot"
```

SYS: será associado a "Meu.Disco.Boot". O dispositivo lógico C: será associado ao diretório C neste mesmo disco (isto é, Meu.Disco.Boot:c). Do mesmo modo, as seguintes associações serão feitas:

```
C:          Meu.Disco.Boot:c
L:          Meu.Disco.Boot:l
S:          Meu.Disco.Boot:s
LIBS:       Meu.Disco.Boot:libs
DEVS:       Meu.Disco.Boot:devs
FONTS:      Meu.Disco.Boot:fonts
```

Caso um diretório não esteja presente, o dispositivo lógico correspondente será associado ao diretório raiz.

Se você tem um disco rígido não inicializável (aqui chamado de DH0:) e quiser usar nele os arquivos do sistema, basta dar os seguintes comandos:

```
ASSIGN SYS:       DH0:
ASSIGN C:         DH0:C
ASSIGN L:         DH0:L
ASSIGN S:         DH0:S
ASSIGN LIBS:      DH0:LIBS
ASSIGN DEVS:      DH0:DEVS
ASSIGN FONTS:     DH0:FONTS
```

Se o seu disco rígido for inicializável (isto é, capaz de dar o boot no sistema), não é preciso fazer estas associações: o sistema se encarregará disto para você.

Tenha sempre em mente que as associações são globais para todos os processos do Shell. Trocar uma associação numa janela faz com que a mesma associação seja trocada em todas as janelas.

Aqueles que preferirem utilizar a sua própria biblioteca de fontes especiais podem digitar:

```
ASSIGN FONTS: "Disco de fontes especiais:MinhasFontes"
```

Se você quiser que os comandos sejam carregados mais depressa (e dispõe de memória para gastar), digite:

```
makedir ram:c
copy sys:c ram:c all
assign c: ram:c
```

Como já vimos, isto copiará todos os comandos normais do AmigaDOS no ram disk e reassociará o diretório de comandos para que o sistema possa encontrá-los nessa área da memória. Outra maneira de conferir maior velocidade aos comandos do AmigaDOS é torná-los residentes (veja a descrição do comando RESIDENT no Capítulo 2).

Utilizando os Comandos do AmigaDOS

Um comando do AmigaDOS consiste no nome do comando e seus argumentos (se existirem). Para executar um comando do AmigaDOS, basta digitar o nome do comando e seus argumentos após o prompt do Shell.

Quando você digita o nome de um comando, esse comando é executado como parte da Interface de Linhas de Comando (Shell). Você pode seguir digitando os nomes de outros comandos, mas o AmigaDOS não os executará até que o comando corrente tenha terminado. Quando isto ocorrer, o prompt do Shell reaparecerá. Nesse caso, diz-se que o comando está rodando interativamente.

O prompt do Shell inicialmente é um "n>", onde "n" corresponde ao número do processo do Shell. Entretanto, isto pode ser modificado com o uso do comando PROMPT (para maiores detalhes, veja PROMPT no Capítulo 2).

ATENÇÃO: Se você executar um comando interativamente e ele falhar. o AmigaDOS tentará executar o próximo comando digitado. Dependendo das circunstâncias, pode ser perigoso enumerar muitos comandos em seguida. Por exemplo, se você introduzir a seqüência:

```
COPY a TO b
DELETE a
```

e o comando COPY for abortado (talvez porque o disco esteja cheio), DELETE será executado e você perderá o seu arquivo.

Rodando Comandos em Segundo Plano

É possível instruir o AmigaDOS a executar um ou mais comandos em segundo plano. Para fazer isto use o comando RUN, que cria um novo Shell como um processo

separado da mesma prioridade (ou *background process*, em inglês). Assim, o AmigaDOS executará ao mesmo tempo as tarefas enumeradas após o comando RUN e as tarefas que você especificar nas linhas de comando subseqüentes. Você pode, por exemplo, examinar o conteúdo do diretório ao mesmo tempo em que imprime um arquivo de texto. Para fazer isso, digite:

```
RUN TYPE arquivo_texto to PRT:
LIST
```

Desse modo, RUN criará um novo Shell e procederá à impressão do arquivo de texto, ao mesmo tempo em que você lista o diretório de seus arquivos na janela do Shell original.

RUN é capaz de executar múltiplos comandos do AmigaDOS, cada comando será executado na ordem determinada. A linha contendo os comandos após o RUN é chamada de linha de comandos. Para terminar uma linha de comandos, tecle RETURN. Para estender uma linha de comandos (ou seja, fazer com que a linha seguinte seja interpretada como uma continuação da primeira), digite um sinal de mais (+) antes de acionar RETURN em todas as linhas, exceto a última. Por exemplo,

```
RUN JOIN arquivo_texto1 arquivo_texto2 AS arquivo_texto +
SORT arquivo_texto TO texto_classificado +
TYPE texto_classificado to PRT:
```

Se você quiser executar um comando usando RUN e depois fechar a janela do Shell a partir da qual esse comando foi chamado, será preciso redirecionar a entrada e a saída de RUN. Para fazer isso, use o comando RUN<NIL:>NIL:

Executando Arquivos de Comandos

Ao invés de digitar seqüências de comandos diretamente pelo teclado, o usuário pode empregar o comando EXECUTE, que executa as linhas de comando contidas num arquivo. O Shell lê as seqüências de comandos desse arquivo até encontrar um erro ou o final do arquivo. Ao encontrar um erro, o AmigaDOS não executará os comandos subseqüentes na linha RUN ou no arquivo especificado por EXECUTE, a menos que tenha sido usado o comando FAILAT (veja o Capítulo 2). Note que o Shell só exibe o prompt após a execução de comandos que tenham rodado interativamente.

Direcionando a Entrada e Saída dos Comandos

O AmigaDOS permite que a entrada e a saída dos dados seja redirecionada mediante o uso dos símbolos > e < como comandos. Quando o usuário digita um comando, normalmente o AmigaDOS exibe a saída correspondente na tela. Para fazer com que a saída desse comando seja enviada para um arquivo, deve-se usar o comando >. Para fazer com que um programa aceite a entrada de dados a partir de um arquivo, ao invés do teclado, use o comando <. Os comandos < e > agem como guardas de trânsito, direcionando o fluxo da informação. Por exemplo: se você quiser direcionar a saída do comando DATE para um arquivo chamado "arquivo_texto", basta digitar a seguinte linha:

```
DATE > arquivo_texto
```

Caso deseje direcionar a saída para um arquivo que já existe, você deverá usar o

comando >>. A seqüência a seguir direciona a saída do comando TYPE para um arquivo já existente:

```
TYPE >>mais_texto texto_original
```

Desse modo, o texto armazenado no arquivo "texto_original" será acrescentado ao texto contido no arquivo "mais_texto". A partir da versão 2.0 do AmigaDOS, se o usuário redirecionar a saída com >> para um arquivo inexistente, o arquivo será criado automaticamente.

Interrompendo o AmigaDOS

O AmigaDOS permite a indicação de quatro níveis de atenção para interrupções, ativados através de CTRL-C, CTRL-D, CTRL-E e CTRL-F. No caso de EDIT, por exemplo, pressionar CTRL-C instrui o sistema a interromper o que estava fazendo e voltar para ler os próximos comandos fornecidos na linha do EDIT. Para mandar o CLI interromper uma seqüência de comandos iniciada por EXECUTE assim que o comando em curso terminar, pode-se digitar CTRL-D. Já CTRL-E e CTRL-F são usados apenas por certos comandos em ocasiões especiais, veja o Capítulo 2 para maiores informações a respeito.

Nota: A detecção e resposta a estas flags de interrupção são de responsabilidade do programador. O AmigaDOS é incapaz de eliminar um programa por si mesmo.

O Processo de Validação

Quando o usuário insere um disco para ser atualizado, o AmigaDOS cria um processo em baixa prioridade que confere validade a toda a estrutura do disco. Até que esse processo de restart termine o seu trabalho não será possível criar arquivos nesse disco. Pode-se, entretanto, ler arquivos existentes.

As versões mais antigas do AmigaDOS (1.3 e anteriores) executam processamentos adicionais quando um novo disco é inserido. Quando o processo de restart se completa o AmigaDOS checa se você acertou a data e hora do sistema. Para isto devem ser usados os comandos DATE, SETCLOCK ou TIME (veja o Capítulo 2). Se você não especificar uma nova data, o AmigaDOS ajustará a data do sistema para o dia e a hora do arquivo mais recente contido no disco inserido. Isto garante que as novas versões dos arquivos terão datas mais recentes, mesmo que o dia e hora reais não estejam corretos.

Nas versões 1.3 e anteriores, se o usuário pedir a data e a hora antes que a validação esteja completa, o AmigaDOS as exibirá sem ajuste. Neste caso você deve esperar que o processo do validação termine, ou então introduzir a data e a hora certas usando o comando DATE.

Comandos Mais Usados

Este manual descreve os vários comandos do AmigaDOS. O Interpretador de Linhas de Comando (Shell) lê os comandos do AmigaDOS digitados numa janela e os traduz

nas ações que deverão ser executadas pelo computador. Nesse sentido, o Shell se assemelha às interfaces tradicionais da maioria dos computadores: o usuário digita os comandos e a interface exibe textos em resposta.

Utilizando o Shell

Para usar a interface do Shell, basta selecionar a janela do Shell e digitar os comandos desejados, cuja descrição é apresentada neste manual. A janela (ou janelas) do Shell pode ter seu tamanho e localização modificados como qualquer outra. Para fechar a janela do Shell, digite "ENDSHELL".

Nem todos os comandos ou programas podem ser executados indistintamente a partir do Workbench ou do ambiente do Shell. Muitos dos comandos do AmigaDOS descritos no Capítulo 2 funcionam apenas sob o Shell.

Neste livro, os termos "CLI" e "Shell" são usados para se referir à janela especial onde o usuário digita os comandos do AmigaDOS. Embora esses termos sejam aqui empregados com o mesmo significado, existe uma pequena diferença entre eles. Por enquanto, tenha apenas em mente que tanto CLI como Shell se referem ao lugar onde você pode digitar os comandos do AmigaDOS.

Alguns Comandos do AmigaDOS

Todos os comandos disponíveis no Shell são explicados detalhadamente no Capítulo 2. Entretanto, por experiência sabemos que a maioria das pessoas costuma utilizar poucas vezes as opções mais avançadas. Assim, fornecemos aqui um sumário dos comandos mais utilizados e seus formatos mais comuns.

Os comandos resumidos abaixo (seguidos dos nomes que recebem no AmigaDOS) executam operações tais como:

- Copiar um disco (DISKCOPY).
- Formatar um novo disco (FORMAT).
- Habilitar um disco formatado a dar o boot no sistema (INSTALL).
- Criar um disco CLI.
- Renomear um disco (RELABEL).
- Consultar o diretório de um disco (DIR).
- Obter informações sobre os arquivos (LIST).
- Proteger um disco contra apagamento acidental (PROTECT).
- Obter informações sobre um disco ou partição (INFO).
- Mudar o diretório corrente (CD).
- Acertar a data e a hora (DATE).
- Redirecionar a saída de um comando (>).
- Exibir na tela o conteúdo de um arquivo de texto (TYPE).
- Renomear um arquivo (RENAME).
- Apagar um arquivo (DELETE).
- Criar um novo diretório (MAKEDIR).

- Copiar arquivos num sistema de dois drives (COPY).
- Copiar arquivos num sistema de um drive (COPY).
- Procurar arquivos num disco (DIR OPT A).
- Executar alguma tarefa automaticamente por ocasião do boot (usando a Startup-Sequence).
- Dizer ao AmigaDOS onde procurar por certas coisas (ASSIGN).
- Abrir uma nova janela do Shell (NEWSHELL).
- Fechar uma janela do Shell já existente (ENDSHELL).

Aos Novos Usuários

Sugerimos aos usuários ainda não iniciados no AmigaDOS que leiam e testem cada um dos próximos itens em seqüência. Cada comando apresentado deixará um disco de teste em tal estado que o comando seguinte deverá funcionar exatamente como descrito. Mais tarde, quando você já estiver familiarizado com o sistema, os subtítulos das seções abaixo servirão para reavivar a sua memória.

Para Começar

Antes de mais nada, providencie dois disquetes virgens de face dupla e tenha à mão o seu disco do Workbench. Proteja o disco mestre contra gravação e habilite para gravação os discos de teste. A maioria dos comandos apresentados a seguir assumem que o equipamento tem apenas um drive, mas a versão do comando para dois drives será fornecida quando necessário.

Os comandos que você deverá digitar aparecem nos tópicos a seguir endentados a partir da margem esquerda. Após digitar o comando, tecle RETURN para devolver o controle ao AmigaDOS. Os comandos são escritos em caixa alta apenas para distingüi-los do restante do texto. Lembre-se de que o AmigaDOS aceita a digitação de comandos tanto em letras maiúsculas como minúsculas.

As notações "df0:" e "drive 0" nos parágrafos a seguir se referem ao drive que vem instalado no Amiga. A notação "df1:" indica o primeiro drive externo de 3 1/2" (atente para o fato de que alguns sistemas utilizam "df2:" para designar o drive externo).

Ocasionalmente, você verá um ponto e vírgula (;) nas linhas de comando a serem digitadas. O que vier a seguir nestas linhas será tratado como comentário pelo AmigaDOS. O AmigaDOS ignora o comentário, logo não é preciso digitá-lo junto com o comando. Ele está ali apenas para o seu esclarecimento.

Uma forma limitada de ajuda sobre a operação dos comandos pode ser obtida digitando-se o nome do comando seguido por uma interrogação (?) e RETURN. Isto exibirá um "gabarito" contendo a seqüência de parâmetros esperados pelo comando e as palavras-chave por ele reconhecidas.

Copiando um Disco

Você pode empregar a seqüência a seguir para fazer um back-up de seu disco mestre do sistema ou de qualquer outro disco.

Nos sistemas de um drive:

```
DISKCOPY FROM df0: TO df0:
```

Nos sistemas de dois drives:

```
DISKCOPY FROM df0: TO df1:
```

Siga as instruções que aparecem na tela. Em sistemas de um drive, você será convidado a inserir o disco mestre (FROM). Em seguida, o AmigaDOS solicitará que você retire este disco e introduza o disco onde será gravada a cópia (TO). A troca de discos se repetirá até que todo o disco mestre tenha sido duplicado. Num sistema de dois drives, você será instruído a introduzir o disco mestre no drive df0: (o drive instalado) e o disco que irá receber a cópia no drive df1: (o primeiro drive externo).

Remova o seu disco mestre e guarde-o num lugar seguro. Deixe a cópia habilitada para gravação, de maneira que nesse disco possam ser armazenadas outras informações. Introduza a cópia no drive instalado e inicialize com ela o sistema (veja o tópico "Preparando um Disco para o Boot", mais abaixo).

Após a inicialização, entre no modo CLI novamente.

Formatando um Disco

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0, e você deverá ter um disco virgem à disposição.

Algumas vezes, ao invés de simplesmente copiar o conteúdo de um disco, será necessário preparar um disco de dados para o seu sistema. Mais tarde você poderá copiar os arquivos que quiser neste disco. Formate o seu segundo disco virgem com o seguinte comando:

```
FORMAT DRIVE df0: NAME "qualquer_nome"
```

E siga as instruções apresentadas. Tanto o drive instalado como o drive externo podem ser usados para formatar um disco.

Quando a formatação estiver completa, espere a luz que indica a atividade do drive se apagar e retire o disco recém-formatado. Introduza novamente o disco do Workbench. Você já pode armazenar arquivos no disco formatado. Contudo, esse disco não pode ser usado para inicializar o sistema.

Preparando um Disco de Boot

Para testar este comando você deverá inserir sua cópia do Workbench no drive 0 e ter o disquete recém-formatado à mão.

Um disco "bootável" é aquele que pode ser usado para inicializar o Amiga na seqüência do processo Kickstart. Para habilitar um disco formatado a dar o boot, digite o comando:

```
INSTALL ?
```

Nota: Ao usar esse comando num sistema com apenas um drive, NÃO ESQUEÇA de digitar o ponto de interrogação! Caso contrário, o AmigaDOS tentará executar o INSTALL no disco inserido no drive 0.

O AmigaDOS responderá:

```
DRIVE/A, NO BOOT/S, CHECK/S
```

Remova o disco do Workbench, introduza o disco formatado e então digite:

```
df0:
```

e pressione RETURN. O AmigaDOS gravará os setores do boot no disco em df0:. Agora, após a luz do drive se apagar, você já pode executar um reset geral (CTRL-Amiga-Amiga). Quando o sistema se reinicializar você irá direto para o CLI, ao invés do Worbench.

O disco formatado contém agora um CLI e nada mais. Isto significa que embora você possa ver o interpretador, ele ainda não é capaz de executar nenhum dos comandos apresentados nesta seção. O CLI necessita de vários outros arquivos para funcionar plenamente. Esses arquivos estão contidos no diretório C do disco mestre.

Preparando um Disco CLI

Há outra maneira de preparar um disco "bootável" mais útil, uma vez que este deixa o diretório de comandos do CLI intacto. Passo a passo, eis o procedimento para transformar um disquete contendo uma cópia do Worbench num disco CLI:

1. Tire uma cópia do disco do Worbench.
2. Abra o Shell como descrito anteriormente.
3. Selecione a janela do Shell e digite o seguinte comando:

```
RENAME FROM s/startup-sequence TO s/NO-startup-sequence
```

Espere a luz do drive se apagar e execute um reset geral, a transformação do disco de Worbench num disco CLI estará terminada (se você quiser restaurar o Workbench, proceda outra vez ao RENAME, com a seqüência de nomes ao contrário). Repare que caso o AmigaDOS não consiga encontrar um arquivo com o nome "startup-sequence" no diretório "s", ele entrará no modo de comando e ficará aguardando que você digite um comando.

Renomeando um Disco

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Se você não estiver satisfeito com o nome do disco que acabou de formatar ou copiar, é possível modificar o nome do volume usando o comando RELABEL:

```
RELABEL df0: novo_nome
```

Este comando muda o nome do volume contido no drive df0: para "novo_nome".

Consultando o Diretório

Para testar este comando sua cópia do Workbench deverá estar no drive 0.

Você pode visualizar o conteúdo de um disco empregando o comando:

```
DIR ou DIR df0:
```

Isto listará o conteúdo do diretório corrente. Por exemplo, o comando:

```
DIR df0: ou DIR c
```

lista o conteúdo do diretório c no drive df0:. Os diretórios correspondem aos drawers (as "gavetas" que aparecem quando a tela do Workbench é visível).

Você também pode consultar o diretório de uma outra unidade de disco (se o seu sistema tiver mais de um drive) simplesmente especificando o nome correspondente. Por exemplo, o comando:

```
DIR df1:
```

lista o conteúdo de um disco inserido no drive df1:

É possível também consultar o diretório de um disco que ainda não esteja no drive, especificando-se o nome do volume desse disco. Por exemplo, o conteúdo daquele disco formatado que acabamos de renomear pode ser visualizado com o seguinte comando:

```
DIR novo_nome:
```

O AmigaDOS pedirá que você introduza o disco "novo_nome" no drive, de maneira que DIR possa ler seu conteúdo e mostrar o diretório na tela. Não faça isso por enquanto porque não há neste disco arquivos que DIR possa ler. Nós acrescentaremos alguns arquivos mais tarde.

Usando o Comando LIST

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Como vimos, o comando DIR mostra os nomes dos arquivos contidos num diretório. O comando LIST apresenta algumas informações adicionais sobre esses arquivos. Digite:

```
LIST
```

O AmigaDOS mostrará uma série de informações sobre todos os arquivos contidos no diretório, tais como o tamanho dos mesmos, os flags de proteção de cada arquivo, se o nome listado é de um arquivo ou de um diretório, além da data e hora em que os arquivos foram criados.

Especificando-se o nome de um diretório após LIST, serão exibidas as informações sobre os arquivos contidos nesse diretório. Por exemplo,

```
LIST c
```

Na segunda coluna após o nome do arquivo são mostrados os flags de proteção do mesmo. O acrônimo "rwed" se refere aos flags de proteção contra leitura, gravação, execução e apagamento (*read, write, execute* e *delete*). Quando o flag correspondente é setado, utilizando-se o comando PROTECT, o arquivo é habilitado para ser lido, sobreposto, executado ou apagado, respectivamente.

Usando o Comando PROTECT

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Este comando protege (ou desprotege) um arquivo de ser apagado acidentalmente. Experimente digitar:

```
DATE > meu_arquivo
PROTECT meu_arquivo
LIST meu_arquivo
```

Você notará que todos os flags de proteção estão definidas como "--------". Se agora você digitar:

```
DELETE meu_arquivo
```

O AmigaDOS responderá:

```
"Not Deleted - file is protected from deletion"
(Não deletado - arquivo está protegido contra apagamento)
```

Para reabilitar o apagamento desse arquivo, digite:

```
PROTECT meu_arquivo d
```

Maiores informações sobre os flags de proteção são apresentadas na descrição do comando PROTECT (Capítulo 2).

Obtendo Informações Sobre os Arquivos

Sua cópia do Workbench ou CLI deverá estar no drive 0. Digite o comando:

```
INFO
```

Isto mostrará a quantidade de espaço ocupado e de espaço livre nos seus discos, se eles são somente de leitura ou de gravação, bem como o nome dos volumes. Você pode abrir mais espaço num disco apagando arquivos, ou mudar o nome do volume com o comando RELABEL.

Se você quiser obter informações sobre um disco que ainda não esteja inserido no drive (em sistemas com apenas um drive), dê o comando desta maneira:

```
INFO ?
```

O AmigaDOS responderá:

```
DEVICE:
```

O AmigaDOS carregou o comando INFO do disco do CLI e está apresentando o gabarito desse comando. A resposta "DEVICE:" indica que você pode digitar o nome de qualquer dispositivo sobre o qual deseje obter informações. Por enquanto você não precisa fazer mais nada além de teclar RETURN para executar o comando. Remova o disco do CLI e introduza o disco no qual você deseja que INFO atue. Espere a luz do drive acender e apagar e pressione RETURN. O AmigaDOS dará as informações sobre o disco que você acabou de inserir no drive. Este método funciona tanto para DIR como para INFO.

Mudando o Diretório Corrente

Até agora nós permanecemos na "raiz", o nível hierárquico mais elevado do diretório do disco. Você pode encontrar mais informações sobre a estrutura de árvore do diretório no tópico "Usando Diretórios", visto mais acima. Para descobrir em que nível você está posicionado na árvore do diretório, use o comando:

```
CD
```

Para mudar o diretório corrente, é preciso dizer ao sistema qual será o novo diretório corrente. Por exemplo: ao executar o comando DIR com o disco CLI no drive df0: você pode observar a mensagem "c(dir)". Para fazer deste o diretório corrente, basta digitar:

```
CD df0:c
```

A partir de agora, quando você der o comando DIR, ele exibirá o conteúdo deste nível do sistema de arquivo. O comando CD, sozinho, mostra o nome do diretório corrente. Para voltar ao diretório raiz (o nível superior), digite:

```
CD :
```

no volume corrente (caso você se refira a seus discos pelo nome de volume), ou:

```
CD df0:
```

no drive instalado.

Acertando a Data e a Hora

Você pode acertar o relógio do Amiga usando o comando DATE:

```
DATE 12:00:00 12-Oct-91
```

O relógio do sistema contará o tempo a partir desta data e hora. Se o seu equipamento dispõe de relógio de tempo real alimentado a bateria, proceda ao ajuste como mencionado anteriormente e em seguida acrescente o comando:

```
SETCLOCK SAVE
```

Este comando copia a hora do AmigaDOS no relógio de tempo real.

Redirecionando a Saída de um Comando

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Normalmente, a saída de todos os comandos é dirigida para a tela. Você pode alterar essa condição usando o comando ">". O parêntese angular para a direita significa: envie a saída para o arquivo que se segue. Eis um exemplo:

```
DATE > arq_data
```

Este comando copia (ou regrava) a saída do comando DATE no arquivo "arq_data". Execute o comando para que possamos usar este arquivo no próximo exemplo.

Listando o Conteúdo de um Arquivo de Texto

Para visualizar o conteúdo de um arquivo de texto, use o comando TYPE:

```
TYPE arq_data
```

Isto listará o conteúdo do arquivo "arq_data". Desejando fazer uma pausa para poder ler o que está escrito na tela, pressione a barra de espaço. Teclando-se BACKSPACE a impressão continuará do ponto onde parou. Se você quiser encerrar o comando TYPE, digite CTRL-C.

Mudando o Nome de um Arquivo

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

O comando RENAME permite mudar o nome de um arquivo:

```
RENAME FROM arq_data TO outro_nome
```

ou

```
RENAME arq_data outro_nome
```

Isto trocará o nome do arquivo "arq_data" para "outro_nome". Agora use o comando TYPE para se certificar de que o novo nome se aplica ao mesmo conteúdo do arquivo:

```
TYPE outro_nome
```

Observe que a forma alternativa do comando dispensa a digitação das palavras FROM e TO. A maioria dos comandos do AmigaDOS têm uma forma abreviada, mais curta que as apresentadas neste tutorial. A forma mais longa do comando é empregada aqui com o objetivo de lhe apresentar o que o comando faz. Não deixe de examinar as páginas que contêm a descrição dos comandos a fim de se familiarizar com as formas alternativas disponíveis.

Apagando Arquivos

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Durante a elaboração de um programa ou de um texto, é comum o usuário desenvolver várias versões de seu trabalho e mais tarde decidir apagar os arquivos contendo as versões de que não necessita mais. O comando DELETE permite que você elimine esses arquivos, liberando espaço no disco para o AmigaDOS reutilizar.

Nota: Depois de mandar que o AmigaDOS DELETE um arquivo, não será possível recuperá-lo. Certifique-se de que você realmente deseja apagar tal arquivo.

Como exemplo, esta seqüência de comandos cria um arquivo através do redirecionamento do comando DIR, lista o conteúdo do arquivo na tela para demonstrar que o mesmo existe, e apaga-o em seguida:

```
DIR > arq_dir
TYPE arq_dir
DELETE arq_dir
TYPE arq_dir
```

Ao último comando da seqüência acima o AmigaDOS responderá:

```
Can't Open arq_dir
```

indicando que o arquivo não foi encontrado, pois você o apagou.

Copiando Arquivos

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Num sistema com dois drives, fazer cópias de arquivos é bastante fácil:

```
COPY FROM df0:fonte df1:destino
```

Em sistemas com um drive o procedimento para a cópia de arquivos é um pouco mais complicado. Antes de mais nada, é preciso transferir certos arquivos contidos no disco de sistema para uma área da memória chamada de dispositivo RAM: ou ramdisk. Em seguida, devemos selecionar o ramdisk como diretório corrente, finalmente, a cópia dos arquivos deverá ser executada do ramdisk para o disco destino. A seqüência a seguir ilustra este procedimento.

Certifique-se de que o disco do Workbench ou CLI está no drive interno e digite:

```
COPY df0:c/cd RAM:
COPY df0:c/copy RAM:
CD RAM:
```

Insira no drive o disco contendo o arquivo que você quer copiar (neste exemplo, vamos copiar o comando EXECUTE do disco do Workbench ou CLI, que já está no drive). Em seguida digite o comando:

```
COPY df0:c/execute ram:execute
```

ou

```
COPY df0:c/execute ram:
```

Remova o disco fonte e insira o disco destino (aquele onde você deseja gravar a cópia do arquivo). O disco destino pode ser qualquer disquete do Amiga previamente formatado. Então digite:

```
COPY ram:execute df0:execute
```

Feito isto, o comando EXECUTE estará copiado do disco fonte para o disco destino.

Retire o disco destino do drive e introduza novamente o disco do CLI ou Workbench. Dê o comando:

```
CD df0:
```

e você estará de volta ao lugar de onde começou. O único comando que você ainda pode precisar dar agora é:

```
DELETE RAM:cd RAM:copy RAM:execute
```

a fim de liberar a memória ramdisk do sistema para outros usos.

Criando um Novo Diretório

Para criar um novo diretório (chamado, por exemplo, "novodir") a partir do diretório corrente, use o comando MAKEDIR:

```
MAKEDIR novodir
```

A partir de agora, quando você der o comando DIR, um novo registro será visível:

```
novodir (dir)
```

Você também pode utilizar o comando RENAME para mover um arquivo de um diretório para outro, no mesmo disco:

```
MAKEDIR novodir
RENAME FROM outro_nome TO novodir/outro_nome
```

Isto moverá o arquivo "outro_nome" do diretório corrente para o diretório "novodir" recém-criado. Para conferir se o arquivo realmente foi movido, dê o comando:

```
DIR
```

E depois digite:

```
DIR novodir
```

O AmigaDOS consultará o diretório "novodir" e responderá que o arquivo "outro_nome" se encontra ali.

Será Que o Meu Arquivo Está Nesse Disco?

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

Algumas vezes você pode querer visualizar o conteúdo de um disco sem precisar consultar um diretório de cada vez. Nesse caso, basta empregar o comando DIR com a seguinte opção:

```
DIR OPT A
```

Isto listará todos os diretórios e subdiretórios do disco. Lembre-se de que a barra de espaço e a tecla BACKSPACE podem ser utilizadas para, respectivamente, produzir uma pausa e dar continuidade à listagem.

Se preferir olhar com mais calma o conteúdo do disco, você pode redirecionar a saída do diretório para um arquivo:

```
DIR > arq_dir ALL
```

Note que o comando de redirecionamento e o nome do arquivo devem vir antes da opção do comando DIR.

Se você quiser, pode agora digitar "TYPE arq_dir" e pressionar a barra de espaço para fazer uma pausa na listagem. A tecla RETURN também pode ser usada para dar continuidade à impressão da lista.

Como alternativa, é possível utilizar o editor de textos ED (veja o Capítulo 2) para verificar o conteúdo de um arquivo. Digite o seguinte:

```
ED arq_dir
```

Use as teclas de direção para movimentar o cursor através do arquivo.

Use a combinação de teclas ESC e T <RETURN> para ir para o início do arquivo (vamos nos referir a esta combinação como ESC-T, significando ESC e depois T).

Use a combinação de teclas ESC-B <RETURN> para ir para o fim do arquivo.

Use a combinação de teclas ESC-M e depois um número <RETURN> para ir para a linha correspondente do arquivo.

Use a combinação de teclas ESC-Q <RETURN> para sair sem modificar o arquivo ou;

Use ESC-X <RETURN> para gravar as modificações realizadas no arquivo.

Automatizando a Seqüência de Boot

Existe um arquivo no diretório "s" do disco do CLI ou Workbench chamado "startup-sequence". Trata-se de um arquivo script contendo uma seqüência de comandos que o AmigaDOS executa toda vez que o usuário reinicializa o sistema (em termos, este arquivo é o equivalente ao AUTOEXEC.BAT do MS-DOS). Da startup-sequence do disco do Workbench fazem parte ainda os comandos LOADWB (de *Load the Workbench*), que carrega o programa, e ENDCLI, que basicamente faz o programa Workbench assumir o controle. Você pode criar a sua própria startup-sequence, usando os editores ED ou MEMACS para elaborar uma versão personalizada de uma seqüência de comandos EXECUTE. A descrição e o tutorial sobre o comando EXECUTE apresentados no Capítulo 2 fornecem maiores detalhes sobre os comandos que podem ser incluídos neste arquivo. Note que a startup-sequence também pode ser empregada para fazer com que um programa se auto-execute logo após o seu carregamento.

ATENÇÃO: Tome o cuidado de modificar somente a cópia do seu disco mestre caso você decida criar uma nova startup-sequence. Jamais modifique o disco mestre. Além disso, os usuários do AmigaDOS versão 2.0 ou posterior deverão acrescentar os novos comandos ao arquivo s:user-startup, ao invés do arquivo s:startup-sequence.

Nota: A startup-sequence do AmigaDOS 2.0 procura por um arquivo chamado "s:user-startup" e executa o mesmo se o encontrar. Caso você seja usuário desta versão, efetue as alterações desejadas no arquivo s:user-startup, ao invés de modificar o arquivo s:startup-sequence.

Associando Discos

Para testar este comando sua cópia do Workbench ou CLI deverá estar no drive 0.

De vez em quando você desejará mudar de disco e então continuar com o seu trabalho. Suponha, por exemplo, que você tenha inicializado o sistema com o disco

do Workbench e mais tarde descubra que precisa usar o disco do CLI. Se o disco do CLI tiver um diretório contendo os comandos executáveis que você deseja empregar (como um diretório "c", por exemplo), essa mudança poderá ser efetuada utilizando-se o comando ASSIGN.

Se ASSIGN não for usado, será preciso trocar os discos para poder executar os comandos. Vejamos um exemplo. A intenção é mudar de disco e passar a usar "meu_disco:" como disco principal. Antes de começar, você deve criar um disco chamado "meu_disco". Para tanto, tire uma cópia do Workbench (veja as instruções referentes à cópia de discos no início desta seção) e, usando o comando RELABEL, modifique o nome da cópia para "meu_disco".

Feito isto, digite:

```
CD meu_disco:
```

O AmigaDOS responderá "Insert meu_disco into any drive". Insira o disco em qualquer drive e dê o comando:

```
DIR
```

O AmigaDOS exibirá a mensagem "Insert Workbench [*ou o nome de outro disco que tenha sido usado para dar o boot*] in any drive". Desde a inicialização, o sistema sabe que o comando DIR está no diretório "c" do disco de boot. Somente depois que conseguir encontrar e ler o comando DIR no diretório "c" o AmigaDOS poderá examinar o conteúdo do disco "meu_disco" (quando então mostrará a mensagem "Insert meu_disco in any drive"). Qualquer outro comando do AmigaDOS que você der agora resultará, do mesmo modo, na necessidade de trocar de disco.

Para evitar esse inconveniente, basta usar o comando ASSIGN da maneira que se segue:

```
ASSIGN c: meu_disco:c
```

O AmigaDOS exibirá então a mensagem "Insert meu_disco into any drive". Daí por diante, todos os comandos do AmigaDOS serão procurados no diretório de comandos (o diretório "c") de meu_disco, e o sistema não mais solicitará que o usuário insira o disco original para executar comandos simples.

Depois de ter feito isto, pode ser que você queira digitar:

```
CD meu_disco:
```

Há outras coisas que o AmigaDOS é capaz de associar. É só dar o comando

```
ASSIGN LIST
```

para descobrir que outras coisas são essas. Quando você roda um programa que faz uso de um dispositivo serial (modem, impressora), o AmigaDOS procura no diretório corrente associado a DEVS: para localizar o dispositivo. Se todos os diretórios do sistema estiverem no mesmo disco onde se encontra o programa, pode-se evitar que o AmigaDOS peça para reinserir o disco original incluindo-se no disco do programa um arquivo script que reassocie todos os dispositivos a esse disco. O conteúdo de tal arquivo script para um disco chamado "meu_disco" seria o seguinte:

```
ASSIGN SYS: meu_disco:
```

```
ASSIGN S: meu_disco:s
ASSIGN DEVS: meu_disco:devs
ASSIGN L: meu_disco:l
ASSIGN FONTS: meu_disco:fonts
ASSIGN LIBS: meu_disco:libs
```

Para criar o arquivo script, use o comando:

```
COPY FROM * TO reassign
```

E em seguida digite as linhas de comandos acima. Após digitar a última linha, tecle CTRL-\ para indicar o final do arquivo. O "*" representa o teclado e a janela atual do CLI, o que faz deste método de criação de arquivos uma alternativa prática caso você não queira utilizar os editores de texto ED ou EDIT com essa finalidade.

Criado o arquivo script, você poderá executá-lo digitando:

```
EXECUTE reassign
```

Agora, todos os ASSIGN utilizados pelo sistema estão ajustados para "meu_disco" ao invés do disco de boot original.

Comentários Finais

O Capítulo 2 contém uma seção de referência que mostra os gabaritos de todos os comandos do AmigaDOS. Consulte a descrição de cada comando a fim de obter maiores informações sobre os mesmos. Quando estiver familiarizado com os comandos e as formas nas quais eles podem ser usados, a lista de referência rápida encontrada no final do capítulo será útil para recordá-lo dos comandos disponíveis.

Capítulo 2

Os Comandos do AmigaDOS

Este capítulo fornece as especificações completas de todos os comandos do AmigaDOS 2.0. O AmigaDOS foi inteiramente aperfeiçoado e diversos novos comandos foram acrescentados. Muitos comandos agora são internos (incorporados ao Shell) para maior rapidez, conveniência e redução do consumo de memória. Este capítulo inclui:

- As convenções dos comandos: uma explicação dos símbolos e abreviaturas utilizados nas descrições dos comandos.
- As especificações de cada comando, inclusive para os programas do Workbench e Preferences.
- Uma tabela com as mensagens de erro.

Uma lista de referência rápida dos comandos do AmigaDOS é fornecida no final deste capítulo.

Argumentos e Opções dos Comandos

Para chamar os comandos do AmigaDOS, nem sempre basta digitar o nome do comando desejado após o prompt do Shell. Diversos comandos requerem o emprego de argumentos ou opções de suporte a fim de fornecer ao Amiga informações adicionais a respeito do que você deseja que ele faça. Por exemplo, ao digitar:

```
1> DIR Utilities
```

você estará dizendo ao Amiga para gerar uma lista dos arquivos e subdiretórios contidos no diretório Utilities. Nesta linha de comando, Utilities é um argumento do comando. Entretanto, se você digitar:

```
1> DIR Utilities FILES
```

somente a lista dos arquivos contidos no diretório Utilities será mostrada. Aqui, FILES é uma opção do comando.

Convenções dos Comandos

Na seção "Especificações dos Comandos", cada comando do AmigaDOS é descrito de acordo com um esquema padronizado.

Exemplos:
Quando exemplos forem fornecidos, o comando e qualquer saída no vídeo aparecerão endentados a partir da margem esquerda do texto principal. Um prompt genérico "1>" indicará o que deve ser digitado após o prompt do Shell. Para maior clareza, os nomes dos comandos e seus argumentos serão escritos em caixa alta. Na digitação dos comandos, contudo, é indiferente o uso de letras maiúsculas ou minúsculas. Para executar a linha de comando, pressione a tecla Return ao encerrá-la.

Formato:
Todos os argumentos e opções aceitos pelo comando.

Path:
O diretório onde normalmente é encontrado o comando. Na maioria dos casos será o diretório C:. Como exceções temos os comandos Internal, que são copiados na memória, e os programas do Workbench.

Propósito:
Uma breve explicação sobre a função do comando.

Especificação:
Uma descrição do comando e de todos os seus argumentos.

Gabarito:
Um lembrete interno sobre o formato do comando. O gabarito é embutido no código do programa; se você digitar o nome de um comando seguido por um ponto de interrogação (?), o gabarito desse comando será apresentado na tela.

Lembre-se de que os comandos e os argumentos devem ser separados por espaços (não precisa ser apenas um espaço, múltiplos espaços são aceitos). Nenhuma outra pontuação deve ser utilizada, salvo quando requisitada pela sintaxe do comando específico.

Formatos

A descrição dos comandos do AmigaDOS inclui uma listagem do formato e um gabarito do comando. Nas listas de formatos, os argumentos são cercados por diferentes tipos de parênteses, os quais servem para indicar o tipo do argumento (os parênteses não são parte integrante do comando, e portanto não devem ser digitados).

< > Parênteses angulares cercam os argumentos cuja indicação é obrigatória. Por exemplo, <nome do arquivo> significa que você deverá digitar nesta posição o nome do arquivo apropriado. A menos que esteja cercado por colchetes (veja abaixo), o argumento é indispensável e o comando não funcionará caso o mesmo não seja especificado.

[ ] Colchetes cercam os argumentos e as palavras-chave opcionais, ou seja, aqueles que são aceitos pelo comando, mas não são indispensáveis para o seu funcionamento.

{ } Chaves cercam itens que podem ser fornecidos uma vez ou repetidos qualquer número de vezes. Por exemplo, {<args>} significa que vários itens podem ser enumerados neste argumento.

| Uma barra vertical é usada para separar opções dentre as quais você pode escolher apenas uma. Por exemplo, [OPT R|S|RS] significa que você pode escolher ou a opção R, ou a opção S, ou as duas opções (RS).

O formato do comando COPY, por exemplo, é o seguinte:

```
COPY [FROM] {<nome|padrão>} [TO] <nome|padrão> [ALL] [QUIET]
 [BUF|BUFFER = <n>] [CLONE] [DATE] [NOPRO] [COM]
```

A palavra-chave [FROM] é opcional. Se não for especificada, o comando reconhecerá o nome do arquivo (ou padrão) a ser copiado através de sua posição na linha de comando.

O argumento {<nome|padrão>} precisa ser indicado. Você deverá substituí-lo pelo nome ou padrão apropriado. As chaves indicam que pode ser fornecido mais de um argumento.

A palavra [TO] é opcional. Se não for especificada, o comando reconhecerá o nome do arquivo (ou padrão) no qual será feita a cópia através de sua posição na linha de comando.

Os argumentos [ALL], [QUIET], [CLONE], [DATE], [NOPRO] e [COM] são opcionais.

O argumento [BUF|BUFFER = <n>] é opcional. Caso deseje indicá-lo você poderá usar BUF ou BUFFER, seguido de um argumento numérico. Por exemplo, tanto BUF=5 como BUFFER=5 serão aceitos pelo comando.

Gabaritos

O gabarito é uma descrição condensada do comando, mais curta que a apresentada nas listas de formatos e que vem embutida no próprio sistema. Digitando-se um ponto de interrogação (?) após o nome do comando, o gabarito aparecerá na tela para recordá-lo da sintaxe apropriada.

Nas listas dos gabaritos, os argumentos e opções do comando são separados por vírgulas e seguidos por um código especial que indica o tipo do argumento. O código não deve ser digitado como parte integrante do comando; ele serve apenas para lembrá-lo do tipo de argumento que o comando aceita. Estes códigos são os seguintes.

/A Sempre (*always*) necessário. É obrigatória a indicação do argumento.

/K Palavra-chave (*keyword*) necessária. A palavra-chave do argumento deverá ser fornecida junto com o argumento (normalmente a palavra-chave é opcional).

/S — Chave (*switch*) do argumento. Para ligar a chave, deve-se incluir a palavra-chave indicada na linha de comando.

/N — Número. O argumento em questão é numérico.

/M — Múltiplos argumentos são aceitos. É a convenção do gabarito equivalente aos colchetes. Não há limite para o número de argumentos que podem ser fornecidos, qualquer número será aceito (o /M substitui o emprego de múltiplas vírgulas para indicar o número de elementos com os quais o comando pode operar, utilizado nas primeiras versões do AmigaDOS).

/F — Argumento final. É necessário que o último argumento da linha de comando seja um string. O restante da linha será tratado como o string desejado. Não é preciso digitar o string entre aspas, mesmo que ele contenha espaços.

= — Um sinal de igual indica que duas diferentes formas da palavra-chave são equivalentes. Qualquer uma das duas será aceita. O sinal não deve ser digitado como parte integrante da linha de comando.

Por exemplo: se você digitar COPY? o gabarito do comando COPY será apresentado, tal como se segue.

```
FROM/A/M,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K/N,CLONE/S,DATES/S,
NOPRO/S,COM/S
```

FROM/A/M indica que é obrigatório especificar o primeiro argumento e que múltiplos argumentos serão aceitos. TO/A indica que o segundo argumento também precisa ser especificado. ALL/S, QUIET/S, CLONE/S, DATES/S, NOPRO/S e COM/S são chaves de argumentos: se a palavra estiver presente na linha, a chave correspondente será ligada. BUF=BUFFER/K/N indica que o argumento é numérico (/N). Este argumento é opcional, mas caso seja fornecido, devem ser incluídos tanto a palavra-chave (/K) como o argumento. Você pode optar entre "BUF" ou "BUFFER" como palavra-chave.

Especificações dos Comandos do AmigaDOS

ADDBUFFERS

Formato: ADDBUFFERS <drive>[<n>]
Gabarito: DRIVE/A,BUFFERS/N
Propósito: Adicionar buffers ao sistema de arquivo.
Path: C:ADDBUFFERS
Especificação: ADDBUFFERS acrescenta <n> buffers à lista de buffers disponíveis para <drive>. A alocação de buffers adicionais agiliza as operações que envolvem acesso ao disco de maneira significativa. Entretanto, cada buffer adicional reduz a memória livre em aproximadamente 500 bytes. A alocação default é de 5 buffers para drives de disco flexível e geralmente 30 para discos rígidos.

O número de buffers que podem ser acrescentados depende da quantidade de memória extra disponível. Não há um limite superior fixo, embora na prática o acréscimo de muitos buffers possa prejudicar a performance geral do sistema, por roubar memória de acesso aleatório (RAM) das outras funções.

> Buffer é uma área da memória reservada para o armazenamento temporário de dados.

Se um número negativo for especificado, a quantidade de buffers indicada será subtraída da alocação atual. A quantidade mínima de buffers permitida é 1, embora não seja aconselhável utilizar apenas um buffer.

De maneira geral, 30 buffers são recomendados para um drive de disco flexível num sistema com 512K de memória. O número apropriado de buffers para um disco rígido depende do tipo e capacidade do drive. No caso do Commodore 2091, deve ser utilizado o valor default indicado no HDToolbox (este valor pode ser exibido selecionando-se o gadget Advanced Options na tela de particionamento). Como regra geral, você pode alocar de 30 a 50 buffers para cada megabyte de RAM do seu equipamento.

Se apenas o argumento <drive> for especificado, ADDBUFFERS exibirá o número de buffers que se encontram alocados para esse drive.

Exemplos

```
1> ADDBUFFERS DF1: 25
```

```
DF1: has 30 buffers
```

adiciona 25 buffers ao drive DF1:.

```
1> ADDBUFFERS DF0:
DF0: has 20 buffers
```

exibe o número de buffers alocados para o drive DF0:.

ADDMONITOR

Formato: ADDMONITOR NUM=%d NAME=%s

Gabarito: NUM/N/A,NAME/A,HBSTRT/K,HBSTOP/K,HSSTRT/K,HSSTOP/K,VBSTRT/K,VBSTOP/K,VSSTRT/K,VSSTOP/K,MINROW/K,MINCOL/K,TOTROWS/K,TOTCOLS/K,BEAMCONO/K

Propósito: Informar ao Amiga que um monitor não RGB foi ligado ao sistema.

Path: SYS:System/AddMonitor

Especificação: ADDMONITOR deve ser executado quando o usuário ligar ao sistema um monitor A2024 ou Multiscan, ou ainda qualquer monitor cujo padrão de vídeo seja diferente daquele adotado no país (PAL onde se usa NTSC, ou vice-versa).

AddMonitor requer dois argumentos. Os valores aceitos por NUM e NAME são listados abaixo:

	NUM=	NAME=
Para um monitor NTSC.	1	NTSC
Para um monitor PAL.	2	PAL
Para um Multiscan.	3	Multiscan
Para um A2024	4	A2024

As opções adicionais controlam recursos especiais de hardware do Amiga. Certos periféricos voltados para aplicações gráficas requerem o uso dessas opções para poderem funcionar corretamente. Neste caso, as opções deverão ser explicadas na documentação que acompanha o hardware.

Após empregar o comando AddMonitor, você deverá utilizar o editor ScreenMode a fim de selecionar o novo modo de vídeo. Para que o sistema reconheça o monitor por ocasião do boot, arraste o ícone apropriado do drawer MonitorStore para o drawer Monitors.

Exemplos: Se você deseja ligar um monitor Multiscan, digite:

```
1> ADDMONITOR NUM=3 NAME=Multiscan
```

Se você deseja ligar um monitor A2024, digite:

```
1> ADDMONITOR NUM=4 NAME=A2024
```

Em seguida, use o editor ScreenMode do programa Preferences a fim de selecionar o modo de vídeo apropriado para o seu monitor.

ALIAS

Formato: ALIAS [<nome>] [<string>]
Gabarito: NAME,STRING/F
Propósito: Atribuir ou exibir nomes alternativos de comandos.
Path: Internal
Especificação: ALIAS permite a criação de nomes alternativos (ou *aliases*) para os comandos do AmigaDOS. Empregar um alias equivale a substituir uma sentença inteira por uma única palavra. Com ALIAS é possível abreviar os comandos que você utiliza com maior freqüência, ou ainda substituir o nome padrão de um comando por outro de sua escolha.

Ao encontrar o parâmetro <nome>, o AmigaDOS o substituirá pelo <string> definido, integrará o resultado com o restante da linha de comando e tentará interpretar e executar a linha resultante como um comando do AmigaDOS. Desse modo, <nome> é o nome alternativo, ou alias, e <string> é o comando cujo nome deverá ser substituído pelo alias.

O comando ALIAS <nome> exibe o <string>, ou seja, o nome padrão do comando que foi substituído pelo alias. ALIAS sozinho lista todos os aliases em uso.

Note que os aliases são locais ao Shell em que foram definidos. Um novo Shell criado por meio do comando NEWSHELL compartilhará os mesmos aliases com o Shell matriz. Contudo, um Shell criado com o item Execute Command do menu do Workbench não será capaz de reconhecer os aliases criados no Shell original. Para criar um alias global, reconhecido por todos os Shells, deve-se incluir o alias no arquivo S:Shell-startup.

Um alias deverá obrigatoriamente estar no início da linha de comando. Após o alias você poderá continuar especificando normalmente os argumentos da linha de comando. Contudo, não é permitido usar aliases para uma série de argumentos de comando. Se quiser criar um script usando a opção LFORMAT do comando LIST, por exemplo, você não poderá definir um alias para representar o argumento LFORMAT.

É possível suprir o nome de um arquivo ou outra instrução num alias introduzindo-se colchetes ([ ]) no <string>. Qualquer argumento digitado depois do alias será inserido no lugar dos colchetes. Para remover um alias, use o comando UNALIAS.

Exemplos:

```
1> ALIAS d1 DIR DF1:
```

faz com que, digitando-se d1, seja listado o conteúdo do diretório do disco inserido em DF1:, tal como se você tivesse digitado DIR DF1:.

```
1> ALIAS hex TYPE [ ] HEX NUMBER
```

cria um alias chamado "hex", que exibe o conteúdo do arquivo especificado em formato hexadecimal. Os colchetes indicam onde será inserido o nome do arquivo. Se você digitar:

```
1> hex MeuArquivo
```

o conteúdo de MeuArquivo será exibido em formato hexadecimal e com linhas numeradas.

ASK

Formato: ASK <prompt>
Gabarito: PROMPT/A
Propósito: Obter uma resposta do usuário durante a execução de um arquivo script.
Path: Internal
Especificação: ASK é usado em scripts para exibir a mensagem <prompt> na janela atual e aguardar pela entrada de uma resposta do usuário através do teclado. As respostas aceitas são Y (sim), N (não) e RETURN (não). Se "Y" for pressionado, ASK atribuirá o valor 5 ao flag de condição (WARN). Caso "N" seja pressionado, o valor desse flag será zero. Pode-se então usar uma declaração IF para testar a resposta.

As mensagens utilizadas como <prompt> que contiverem espaços deverão ser digitadas entre aspas.

Exemplo: Suponha um script contendo os seguintes comandos:

```
ASK Continua?
IF WARN
  ECHO Sim
ELSE
  ECHO Não
ENDIF
```

Quando o comando ASK for encontrado, a mensagem "Continua?" será apresentada na tela. Se o usuário pressionar Y, a palavra "Sim" será mostrada na tela. Teclando-se N, aparecerá "Não".

Veja também: IF, ELSE, ENDIF

ASSIGN

Formato: ASSIGN [<nome>]:{dir}] [LIST] [EXISTS] [DISMOUNT] [DEFER] [PATH] [ADD] [REMOVE] [VOLS] [DIRS] [DEVICES]

Gabarito: NAME,TARGET/M,LIST/S,EXISTS/S,DISMOUNT/S,DEFER/S,PATH/S,ADD/S,REMOVE/S,VOLS/S,DIRS/S,DEVICE/S

Propósito: Controlar a atribuição de nomes de dispositivos lógicos aos diretórios do sistema de arquivo.

Path: C:ASSIGN

Especificação: ASSIGN permite que nomes de dispositivos lógicos mais curtos e convenientes sejam utilizados para referenciar os diretórios do sistema, no lugar dos nomes usuais ou paths completos dos mesmos. Este comando associa nomes alternativos aos diretórios tal como ALIAS atribui nomes alternativos aos comandos. ASSIGN possibilita ao usuário remover e criar associações, além de listar as associações atuais, no todo ou em parte.

Fornecendo-se os argumentos <nome> e {dir}, ASSIGN associará o nome ao diretório especificado. O AmigaDOS passará então a acessar este diretório quando o nome do dispositivo lógico for referenciado. Caso o <nome> indicado já tenha sido associado a um diretório, o novo diretório substituirá o antigo (não se esqueça de digitar os dois pontos (:) depois do argumento <nome>).

Fornecendo-se somente o argumento <nome>, qualquer associação existente de um diretório com aquele dispositivo lógico será cancelada.

É possível associar vários nomes de dispositivos lógicos ao mesmo diretório, utilizando-se múltiplos comandos ASSIGN. Para associar um nome de dispositivo lógico a vários diretórios, deve-se especificar cada diretório depois do argumento <nome> ou então usar a opção ADD. Quando a opção ADD é especificada, qualquer diretório associado a <nome> já existente não será cancelado. Em vez disso, o novo diretório será adicionado à lista de associações, e o sistema procurará por ambos os diretórios quando <nome> for encontrado. Se o diretório original não estiver disponível, o diretório acrescentado satisfará ASSIGN.

Para deletar um nome da lista de associações, use a opção REMOVE.

O comando ASSIGN, sem nenhum argumento ou seguido da opção LIST, exibirá uma lista de todas as associações atuais. Se as chaves VOLS, DIRS ou DEVICES estiverem presentes, ASSIGN se limitará a listar os volumes, diretórios ou dispositivos, respectivamente.

Fornecendo-se a palavra-chave EXIST junto com um nome de dispositivo lógico, o AmigaDOS procurará tal nome na lista de ASSIGN e exibirá o volume e diretório associados a esse dispositivo. Se o nome do dispositivo não for encontrado, o flag de condição receberá o valor 5 (WARN). Esta opção costuma ser usada em scripts.

Normalmente, quando o argumento {dir} é fornecido o AmigaDOS tenta localizar imediatamente tal diretório. Isto implica em dizer que se os comandos ASSIGN estiverem contidos na S:startup-sequence, será obrigatória a presença dos diretórios associados em um disco montado por ocasião do boot (caso algum diretório não seja encontrado durante o boot, será exibida uma mensagem solicitando que o usuário insira o disco contendo esse diretório). Contudo, existem duas novas opções, DEFER e PATH, que fazem com que o sistema espere até que o diretório seja efetivamente requisitado antes de procurar pelo mesmo.

A opção DEFER cria uma associação de "vínculo posterior". Esta associação só terá efeito quando o objeto associado for referenciado pela primeira vez, ao invés de começar a valer a partir do instante em que a associação é feita. Isto elimina a necessidade de inserir os discos contendo os diretórios associados durante a etapa de inicialização do sistema. Quando a opção DEFER é usada, o disco contendo o diretório associado não é considerado indispensável até que o objeto seja efetivamente chamado.

Se, por exemplo, você associar FONTS: ao diretório DF0:Fonts usando a opção DEFER, o sistema associará FONTS: ao disco que estiver inserido no drive DF0: na hora em que FONTS: for chamado pela primeira vez. Assim, caso um disco do Workbench esteja no drive DF0: na hora em que o diretório FONTS: for requisitado, o sistema associará FONTS: ao disco do Workbench. Entretanto, se mais tarde você remover o disco do Workbench e inserir um outro disco contendo o diretório Fonts, o sistema irá requisitar especificamente o disco do Workbench quando FONTS: for chamado.

A opção PATH estabelece uma associação "não vinculada". Esta associação é semelhante à anterior, porém o ASSIGN será reavaliado cada vez que se verificar uma referência ao nome associado. Este arranjo evita que o sistema precise ficar esperando por um determinado volume para poder usar um diretório particular (como na situação descrita no exemplo acima). Se você associar FONTS: a DF0:Fonts com a opção PATH, qualquer disco em DF0: será examinado quando FONTS: for requisitado. Contanto que esse disco contenha um diretório Fonts, ASSIGN o aceitará. Não é permitido associar múltiplos diretórios usando a opção PATH.

A opção PATH é particularmente útil para usuários de sistemas de disco flexível, uma vez que elimina a necessidade de reinserir o disco original do Workbench usado para dar o boot no sistema. Contanto que o drive associado com a opção PATH contenha um disco com o nome do diretório associado, o sistema utilizará esse disco.

ATENÇÃO: A opção DISMOUNT (chamada REMOVE na versão 1.3) desconecta um volume ou dispositivo da lista de dispositivos montados. Esta opção não desembaraça a memória, somente remove o nome do dispositivo da lista. É impossível cancelar um DISMOUNT sem reinicializar o equipamento. DISMOUNT foi previsto, a princípio, para ser utilizado durante o desenvolvimento de programas. O emprego descuidado desta opção pode causar um erro do tipo *software failure*.

Exemplos:

```
1> ASSIGN FONTS: MinhasFontes:DirFontes
```

associa o diretório FONTS:, do sistema, ao diretório "DirFontes" do disco "MinhasFontes:".

```
1> ASSIGN LIST
```

Volumes:

```
Ram Disk [Mounted]
Workbench 2.0 [Mounted]
MinhasFontes [Mounted]
```

Diretórios:

```
CLIPS       Ram Disk:Clipboards
ENV         Ram Disk:Env
T           Ram Disk:T
ENVARC      Workbench2.0:Prefs/Env-Archive
SYS         Workbench2.0
C           Workbench2.0:C
S           Workbench 2.0:S
L           Workbench 2.0:L
FONTS       MinhasFontes:DirFontes
DEVS        Workbench 2.0:Devs
LIBS        Workbench 2.0:Libs
```

Dispositivos:

```
PIPE AUX SPEAK RAM CON
RAW PAR SER PRT DF0 DF1
```

exibe uma lista de todas as associações atuais.

```
1> ASSIGN FONTS: EXISTS
FONTS MinhasFontes:DirFontes
```

pergunta sobre a associação de FONTS:. O AmigaDOS responde informando que FONTS está associado ao diretório DirFontes do volume MinhasFontes.

```
1> ASSIGN LIBS: SYS: Libs BigAssem:Libs PDAssem:Libs
```

esta é uma associação de múltiplos diretórios que cria um path contendo três diretórios Libs. Esses diretórios serão examinados em seqüência toda vez que Libs: for chamado.

```
1> ASSIGN DEVS: DISMOUNT
```

remove as associações de DEVS: do sistema.

```
1> ASSIGN DiscTrab: DF0: DEFER
1> ASSIGN DiscTrab: EXISTS
DiscTrab <DF0:>
```

estabelece uma associação de vínculo posterior com o dispositivo lógico DiscTrab:. O disco não precisa ser inserido no drive DF0: até a primeira vez em que se verificar

uma referência ao nome DiscTrab:. Observe que ASSIGN exibe DF0: entre parênteses angulares a fim de assinalar o emprego da opção DEFER. Após a primeira referência a DiscTrab:, o nome de volume do disco contido naquele instante em DF0: substituirá <DF0:>.

```
1> ASSIGN C: DF0:C PATH
1> ASSIGN C: EXISTS
C           [DF0:C]
```

referencia o diretório C (o diretório de comandos) de qualquer disco contido em DF0: no instante em que um comando é procurado. Observe que ASSIGN exibe DF0:C entre colchetes a fim de indicar uma associação não vinculada.

```
1> ASSIGN LIBS: Zcad:Libs ADD
```

acrescenta ZCadLibs à lista de diretórios a que LIBS: foi associado.

```
1> ASSIGN LIBS: ZCad:Libs REMOVE
```

remove ZCadLibs da lista de diretórios a que LIBS: foi associado.

AUTOPOINT

Formato:	AUTOPOINT [CX_PRIORITY=<n>]
Gabarito:	CX_PRIORITY/K/N
Propósito:	Ativar automaticamente a janela sobre a qual o apontador for posicionado.
Path:	Extras2.0:Tools/Commodities/AutoPoint
Especificação:	Após AUTOPOINT ser executado, a janela que estiver sob o apontador (o cursor em forma de seta) será ativada automaticamente, sem que o usuário precise clicar o botão do mouse.

O argumento CX_PRIORITY determina a prioridade de AutoPoint em relação aos demais programas Commodity Exchange, tal como se o usuário definisse um ToolType com o valor CX_PRIORITY=<n> na janela Information do ícone do programa. Todos os programas Commodity Exchange possuem um nível de prioridade default igual a 0. Especificando-se um valor <n> superior a 0, AutoPoint tomará prioridade sobre qualquer outro programa Commodity Exchange.

Para abandonar o AutoPoint quando este é executado a partir de um Shell, digite Ctrl-E ou use o comando BREAK.

Exemplo:

```
1> AUTOPOINT
```

inicia o programa AutoPoint.

AVAIL

Formato: AVAIL [CHIP | FAST | TOTAL] [FLUSH]
Gabarito: CHIP/S,FAST/S,TOTAL/S,FLUSH/S
Propósito: Reportar a quantidade de memória (Chip e Fast RAM) disponível.
Path: C:AVAIL
Especificação: AVAIL fornece um resumo sobre a utilização de memória pelo sistema (Chip e Fast RAM). Para cada tipo de memória, AVAIL reportará a quantidade máxima que pode ser manipulada pelo sistema, a quantidade de que o equipamento efetivamente dispõe, quanto se encontra em uso, e o maior bloco contíguo de memória ainda não alocada.

Utilizando-se as opções CHIP, FAST e/ou TOTAL, AVAIL listará o número de bytes livres somente para a Chip ou Fast RAM e/ou a memória total disponível, ao invés de exibir um resumo completo. O valor obtido pode ser utilizado para comparações em scripts. A opção FLUSH faz com que todas as bibliotecas e módulos de dispositivos não utilizados sejam apagados e eliminados da memória.

Exemplos:

```
1> AVAIL
Type Available    In-Use      Maximum     Largest
chip 233592       282272      515864      76792
fast 341384       182896      524280      197360
tot 574976        465168      1040144     274152

1> AVAIL CHIP
   233592
```

O Amiga utiliza dois tipos diferentes de memória RAM. A Chip RAM é usada para o armazenamento de dados relativos ao processamento de gráficos e sons. A Fast RAM é a memória de uso geral do sistema, onde são armazenados os programas.

BINDDRIVERS

Formato: BINDDRIVERS
Gabarito: (nenhum)
Propósito: Vincular drivers de dispositivos ao hardware.
Path: C:BINDDRIVERS

Especificação: BINDDRIVERS é utilizado para carregar e executar drivers de dispositivos para periféricos configurados pela biblioteca de expansão. Estes drivers de dispositivos deverão estar no diretório SYS:Expansion a fim de que BINDDRIVERS possa encontrá-los.

Normalmente, BINDDRIVERS é incluído no arquivo Startup-sequence. Caso os drivers para um hardware de expansão estejam contidos no diretório Expansion, é preciso que um comando BINDDRIVERS conste na Startup-sequence, ou o hardware não será configurado quando o sistema for inicializado.

BINDMONITOR

Formato: BINDMONITOR
Gabarito: MONITORID/A,MONITORNAME/A
Propósito: Associar nomes aos diferentes modos de vídeo
Path: SYS:System/BindMonitor
Especificação: BINDMONITOR associa nomes aos diferentes modos de vídeo suportados pela biblioteca gráfica. Os argumentos aceitos correspondem aos ToolTypes do ícone Mode_Names do Workbench.

Argumentos aceitos por BINDMONITOR

0x00000 Lores	0x00004 Lores-Interlaced
0x08000 Hires	0x08004 Hires-Interlaced
0x08020 SuperHires	0x08024 SuperHires Interlaced
0x11000 NTSC:Lores	0x11004 NTSC:Lores-Interlaced
0x19000 NTSC:Hires	0x19004 NTSC:Hires-Interlaced
0x19020 NTSC:SuperHires	0x19024 NTSC:SuperHires-Interlaced
0x21000 PAL:Lores	0x21004 PAL:Lores-Interlaced
0x29000 PAL:Hires	0x29004 PAL:Hires-Interlaced
0x29020 PAL:SuperHires	0x29024 PAL:SuperHires-Interlaced
0x31004 VGA-ExtraLores	0x31005 VGA-ExtraLores-Interlaced
0x39004 VGA-Lores	0x39005 VGA-Lores-Interlaced
0x39024 Productivity	0x39025 Productivity Interlaced
0x41000 A2024_10Hz	0x49000 A2024_15Hz

A ROM do Amiga reconhece 0x08000 como o display de 640 X 200 pontos. O comando BindMonitor, por sua vez, atribui a 0x08000 o nome de "Hires". Os nomes associados aos modos de vídeo aparecem no gadget Choose Display Mode do editor ScreenMode.

Exemplo:

```
1> BINDMONITOR 0x08000MeuDisplay
```

em vez de Hires, o nome "MeuDisplay" aparecerá no gadget Choose Display Mode do editor ScreenMode.

BLANKER

Formato: BLANKER [SECONDS=<n>] [CX_POPKEY=<tecla(s)>] [CX_POPUP=<yes | no>] [CX_PRIORITY=<n>]

Gabarito: SECONDS/K/N,CX_POPKEY/K,CX_POPUP/K,CX_PRIORITY/K/N

Propósito: Apagar a tela do monitor se nenhum dado for recebido dentro do intervalo especificado de tempo.

Path: Extras2.0Tools/Commodities/Blanker

Especificação: BLANKER é um programa Commodity Exchange que apaga a tela caso nenhuma entrada de dados seja recebida, através do mouse ou teclado, após um período especificado de tempo. O argumento SECONDS=<n> estabelece o intervalo de tempo a ser transcorrido, em segundos. A faixa aceita é de 1 a 9999. O default é de 60 segundos.

CX_POPKEY=<tecla(s)> permite especificar a hot key (a tecla de atalho para trazer de volta a janela do programa). Caso você queira enumerar mais de uma tecla, digite todo o argumento entre aspas (por exemplo: "CX_POPKEY=Shift F1").

CX_POPUP=no evita a abertura da janela do Blanker (por default, a janela do programa se abre quando o comando é executado).

CX_PRIORITY=<n> determina a prioridade do Blanker em relação aos demais programas Commodity Exchange. Todos os programas Commodity Exchange têm uma prioridade default igual a 0.

Para eliminar o Blanker quando este é executado através de um Shell, pressione Ctrl-E.

Exemplos:

```
1> BLANKER SECONDS=45
```

a janela do programa Blanker se abrirá e o número 45 será exibido em seu gadget de texto. Se nenhuma entrada de dados através do mouse ou teclado for recebida durante um intervalo de 45 segundos, a tela será apagada.

```
1> BLANKER CX_POPUP=no
```

o programa Blanker será iniciado sem que sua janela se abra. Caso nenhuma entrada de dados seja recebida durante 60 segundos (o intervalo de tempo default), a tela será apagada.

BREAK

Formato: BREAK <processo> [ALL | C | D | E | F]

Gabarito: PROCESS/A/N,ALL/S,C/S,D/S,E/S,F/S

Propósito: Setar os flags de atenção no processo especificado.
Path: C:BREAK
Especificação: BREAK seta os flags de atenção especificados no <processo> indicado. A opção C seta o flag Ctrl-C, a opção D seta o flag Ctrl-D, e assim por diante. ALL seta todos os flags, de Ctrl-C a Ctrl-F. Por default, o AmigaDOS seta apenas o flag Ctrl-C.

A ação de BREAK equivale ao usuário selecionar o processo relevante clicando-o em sua janela, e pressionar a combinação de teclas Ctrl apropriada.

Ctrl-C é usado, por default, a fim de enviar um sinal BREAK para interromper um processo. Um processo abortado desta maneira exibirá a mensagem ***BREAK na janela do Shell. Ctrl-D é usado para parar execução de um arquivo script. Ctrl-E é usado para sair dos programas Commodity Exchange. Ctrl-F não tem uso corrente.

Use o comando STATUS para exibir os números dos processos correntes.

Exemplos:

```
1> BREAK 7
```

seta o flag de atenção Ctrl-C do processo 7. Isto equivale a ativar o processo 7 e pressionar as teclas Ctrl-C.

```
1> BREAK 5 D
```

seta o flag de atenção Ctrl-D do processo 5.

Veja também: STATUS

CALCULATOR

Formato: CALCULATOR
Gabarito: (nenhum)
Propósito: Ativar uma calculadora na tela.
Path: SYS:Utilities/Calculator
Especificação: CALCULATOR ativa o programa Calculator. Pode-se efetuar uma operação de *cut-and-paste* dos dados da tela do programa para qualquer janela do console, como a do Shell ou a do editor ED.

Para abandonar o programa, selecione o gadget que fecha a janela.

Exemplo:

```
1> CALCULATOR
```

CD

Formato: CD [<dir | padrão>]
Gabarito: DIR
Propósito: Selecionar, modificar ou exibir o diretório corrente.
Path: Internal
Especificação: CD, sem nenhum argumento, exibe o nome do diretório corrente. Seguido por um nome válido de diretório, CD fará deste o diretório corrente.

CD não procura no disco o diretório especificado, assumindo que o mesmo se encontra no mesmo nível do diretório corrente. Caso não esteja, o usuário deverá fornecer o path completo até tal diretório. Se CD não conseguir encontrar o diretório especificado no diretório corrente ou no path indicado, a mensagem de erro "Can't find <directory>" será exibida.

Para subir um nível na hierarquia dos arquivos, em direção ao diretório matriz do diretório corrente, deve-se digitar CD seguido por um espaço e uma barra (caso o diretório corrente seja um diretório raiz, CD / não terá efeito). Pode-se acessar outro diretório na matriz digitando-se o nome do mesmo após a barra. Múltiplas barras são aceitas, sendo que cada barra o conduzirá um nível acima na estrutura de diretórios. Ao usar múltiplas barras, certifique-se de não deixar espaços entre elas.

Para alcançar diretamente o diretório raiz do dispositivo corrente, use CD seguido por um espaço e dois pontos (CD :).

CD também admite a comparação de padrões. Se mais de um diretório coincidir com o padrão fornecido, uma mensagem de erro será exibida.

Exemplos:

```
1> CD DF1:Work
```

seleciona o diretório Work, do disco contido no drive DF1:, como o diretório corrente.

```
1> CD SYS:Com/Basic
```

faz do subdiretório Basic, no diretório Com, o diretório corrente.

```
1> CD //
```

sobe dois níveis na estrutura de diretórios e faz de SYS: o diretório corrente.

```
1> CD SYS:Li#?
```

usa a comparação de padrões para designar o nome do diretório corrente (no caso, Libs).

CHANGETASKPRI

Formato: CHANGETASKPRI <prioridade> [<processo>]
Gabarito: PRI=PRIORITY/A/N,PROCESS/K/N
Propósito: Mudar a prioridade de um processo em execução.
Path: C:CHANGETASKPRI
Especificação: Sendo multitarefa, o Amiga utiliza números de prioridade a fim de determinar a ordem na qual as tarefas correntes deverão ser executadas. Normalmente, a maioria das tarefas tem uma prioridade igual a 0 e a unidade central de processamento (CPU) divide quantidades iguais de seu tempo e ciclos de instruções entre elas. O comando CHANGETASKPRI permite que o usuário altere a prioridade de um processo Shell (o processo atual será assumido caso nenhum outro seja especificado). Todas as tarefas iniciadas no <processo> herdarão a prioridade do mesmo.

A faixa de valores aceitos por <prioridade> compreende os inteiros entre -128 e 127, sendo que os valores mais altos determinam a mais alta prioridade (ou seja, maior proporção de tempo da CPU será alocado para o processo). Contudo, não é aconselhável introduzir valores acima de +10, sob pena de prejudicar o funcionamento de importantes tarefas do sistema. Um nível de prioridade muito baixo (inferior a zero), por outro lado, pode resultar num processo que demorará mais tempo que o razoável para ser executado.

Exemplo:

```
1> CHANGETASKPRI 4 Process 2
```

O processo 2 tem sua prioridade alterada para 4. Quaisquer tarefas iniciadas nesse Shell terão igualmente uma prioridade igual a 4. Logo, as mesmas terão prioridade sobre quaisquer outras tarefas do usário criadas sem o uso de CHANGETASKPRI, uma vez que essas tarefas possuem uma prioridade igual a 0.

Veja também: STATUS

CLOCK

Formato: CLOCK [DIGITAL] [[LEFT] <n>] [[TOP <n>] [[WIDTH <n>] [[HEIGHT <n>] [24HOUR] [SECONDS] [DATE]
Gabarito: DIGITAL/S,LEFT/N,TOP/N,WIDTH/N,HEIGHT/N,24HOUR/S,SECONDS/S,DATE/S
Propósito: Ativar um relógio na tela.
Path: SYS:Utilities/Clock

Especificação: A opção DIGITAL ativa um relógio do tipo digital. Já as opções LEFT, TOP, WIDTH e HEIGHT permitem que o usuário determine o tamanho e a posição do relógio na tela. Estas palavras-chave são opcionais; o relógio compreenderá os argumentos numéricos através de sua posição na linha de comando, tal como descrito abaixo.

Primeiro número: o relógio será posicionado <n> pixels a partir da extremidade esquerda da tela.
Segundo número: o relógio será posicionado <n> pixels abaixo do topo da tela.
Terceiro número: o relógio terá <n> pixels de largura.
Quarto número: o relógio terá <n> pixels de altura.

Observe que caso você queira especificar apenas a largura e altura do relógio, por exemplo, será indispensável o uso das palavras-chave WIDTH e HEIGHT. Se forem digitados apenas os dois números, o relógio os interpretará como as posições a partir da esquerda e do topo (LEFT e TOP).

Nota: WIDTH e HEIGHT não poderão ser usados quando for designada a opção DIGITAL. Não é permitido ao usuário alterar o tamanho do relógio digital, somente determinar a sua posição na tela.

A opção 24HOUR ativa o relógio no modo 24 horas. Caso não seja indicada, o relógio será ativado no modo 12 horas (onde a notação 1:00 PM corresponde a 13:00 horas, por exemplo). As opções SECONDS e DATE, quando especificadas, fazem com que o relógio mostre, respectivamente, os segundos e a data.

Exemplos: Para ativar um relógio posicionado a 75 pixels da extremidade esquerda e 75 pixels abaixo do topo da tela, com 300 pixels de largura e 100 pixels de altura, digite:

```
1> CLOCK 75 75 300 100
```

Para usar as opções SECONDS, DATE e 24HOUR, digite:

```
1> CLOCK SECONDS DATE 24HOUR
```

Para ativar um relógio digital posicionado a 320 pixels da extremidade esquerda e na barra de título da tela (0 pixels abaixo do topo), digite:

```
1> CLOCK DIGITAL 320 0
```

CMD

Formato: CMD <nome do dispositivo> <nome do arquivo> [OPT s|m|n]
Gabarito: DEVICENAME/A,FILENAME/A,OPT/K
Propósito: Redirecionar a saída da impressora para um arquivo.
Path: Extras2.0:Tools/CMD
Especificação: O <nome do dispositivo> pode ser *serial*, *paralell* ou

printer, e equivale ao mesmo dispositivo especificado no editor Printer. O <nome do arquivo> corresponde ao nome do arquivo para o qual deverá ser enviada a saída redirecionada.

As opções de CMD são as seguintes.

s Saltar qualquer dado inicial curto (em geral para um reajuste ao se redirecionar um dump de tela).

m Interceptar múltiplos arquivos até um que um comando BREAK seja recebido ou que Ctrl-C seja pressionado.

n Notificar o usuário sobre progresso da operação por meio da exibição de mensagens na tela.

Exemplo:

```
1> CMD parallel ram:arqcmd
```

qualquer saída dirigida para a porta paralela será redirecionada para o arquivo em RAM: chamado arqcmd.

COLORS

Formato: COLORS [<planos de profundidade> <nível de resolução>]

Gabarito: BITPLANES,SCREENTYPE

Propósito: Modificar as cores da tela da frente.

Path: Extras2.0:Tools/Colors

Especificação: COLORS possibilita ao usuário modificar as cores da tela da frente. Enumerando-se os valores para as opções <planos de profundidade> e <nível de resolução>, será exibida uma tela de teste personalizada. Os valores aceitos para as opções de COLORS são os seguintes.

<planos de profundidade — Especifica a profundidade da tela de teste:

1	2 cores
2	4 cores
3	8 cores
4	16 cores
5	32 cores

<nível de resolução> — Especifica a resolução da tela de teste:

0	320 X 200 pixels
1	320 X 400 pixels
2	640 X 200 pixels
3	640 X 400 pixels

O valor de <planos de profundidade> se restringirá a 4 ou menos quando o valor do <nível de resolução> for igual a 2 ou 3.

Exemplo:

```
1> COLORS 3 2
```

abre uma tela de teste personalizada exibindo uma janela para a programação de cores. A tela terá oito cores e uma resolução de 640 X 200 pixels (Hires).

COPY

Formato: COPY [FROM] {<nome | padrão>} [TO] <nome | padrão> [ALL] [QUIET] [BUF | BUFFER=<n>] [CLONE] [DATES] [NOPRO] [COM] [NOREQ]

Gabarito: FROM/M,TO/A,ALL/S,QUIET/S,BUF=BUFFER/K/N,CLONE/S,DATE/S, NOPRO/S,COM/S,NOREQ/S

Propósito: Copiar arquivos ou diretórios.

Path: C:COPY

Especificação: COPY copia o arquivo ou diretório especificado com o argumento FROM para o arquivo ou diretório especificado com o argumento TO. É possível copiar vários itens de uma só vez, fornecendo-se mais de um argumento FROM, nesse caso, os nomes dos arquivos/diretórios deverão vir separados por espaços. Pode-se usar a comparação de padrões a fim de copiar ou excluir itens cujos nomes compartilhem de um conjunto comum de caracteres ou símbolos.

Durante o procedimento de cópia, o arquivo FROM sobreporá o arquivo TO se este já existir. Caso o diretório destino indicado não exista, COPY criará automaticamente um diretório com o mesmo nome. Pode-se também usar um par de aspas ("") para designar o diretório corrente ao especificar o destino (certifique-se de não digitar nenhum espaço entre as duas aspas).

Quando o argumento de FROM for um diretório, somente os arquivos desse diretório serão copiados (ou seja, os subdiretórios serão excluídos). Use a opção ALL para copiar o diretório completo, incluindo seus arquivos, subdiretórios e os arquivos dos subdiretórios. Caso você deseje que a cópia do diretório tenha o mesmo nome que o original, basta indicar o nome do diretório como argumento de TO.

COPY exibe na tela o nome de cada arquivo quando o mesmo é copiado. A opção QUIET inibe a saída no vídeo.

A opção BUF= é usada para selecionar o número de buffers de 512 bytes que serão usados durante o procedimento de cópia (o default é de 200 buffers, o que equivale a aproximadamente 100K de RAM). Pode ser aconselhável limitar o número de buffers ao se efetuar uma cópia para o dispositivo RAM:. Se você fizer BUF=0, será usado um buffer do mesmo tamanho do arquivo a ser copiado.

Normalmente, COPY transfere para o arquivo TO a data e a hora em que a cópia foi realizada. Quaisquer comentários anexados ao arquivo original FROM serão ignorados. Os flags de proteção do arquivo FROM, no entanto, serão copiadas para

o arquivo TO. As seguintes opções permitem ao usuário alterar esses defaults.

DATES: Copia a data de criação do arquivo FROM para o arquivo TO.

COM: Copia qualquer comentário anexado ao arquivo FROM para o arquivo TO.

CLONE: Copia a data, comentários e flags de proteção do arquivo FROM para o arquivo TO.

NOPRO Os flags de proteção do arquivo FROM não serão copiadas para o arquivo TO. O arquivo TO receberá os flags de proteção padrão.

Normalmente, COPY exibe uma mensagem solicitando a intervenção do usuário caso, por alguma razão, não seja possível prosseguir com a cópia. A opção NOREQ faz com que esse procedimento seja suprimido. Isto pode ser usado em scripts, a fim de prevenir que um colapso de COPY interrompa a execução do script. Por exemplo: quando um script solicita que determinado arquivo seja copiado e o sistema não consegue encontrar esse arquivo, normalmente o script pára e mostra uma mensagem dando conta desse fato, aguardando que o usuário indique o que deve ser feito. Com a opção NOREQ, o comando COPY seria abortado e o script continuaria normalmente.

Exemplos:

```
1> COPY arquivo1 TO :Work/arquivo2
```

copia o arquivo1 no diretório corrente para o arquivo2 no diretório Work.

```
1> COPY ~ (#?.info) TO DF1:Backup
```

copia todos os arquivos do diretório corrente que não terminam em .info para o diretório Backup, no disco inserido no drive DF1:. Este é um uso conveniente da combinação de padrões para poupar espaço de armazenamento quando ícones não são necessários.

```
1> COPY Work:Teste TO ""
```

copia os arquivos do diretório Teste, no disco Work, para o diretório corrente.

```
1> COPY Work:Teste TO DF0: Teste ALL
```

copia todos os arquivos e quaisquer subdiretórios do diretório Teste, no disco Work, para o diretório Teste em DF0:. Se em DF0: ainda não existir um diretório chamado Teste, o AmigaDOS o criará.

```
1> COPY DF0: TO DF1: ALL QUIET
```

copia todos os arquivos e diretórios do disco inserido em DF0: para o disco em DF1:, sem mostrar na tela os nomes dos arquivos/diretórios copiados (este procedimento para a cópia do conteúdo de um disco é bastante lento, comparado com o comando DiskCopy).

CPU

Formato: CPU [CACHE] [BURST] [NOBURST] [DATACACHE] [DATABURST] [NODATACACHE] [NODATABURST] [INSTCACHE] [INSTBURST] [NOINSTCACHE] [NOINSTBURST] [FASTROM] [NOFASTROM] [NOMUTTEST] [CHECK 68010 | 68020 | 68030 | 68881 | 68882 | 68851 | MMU | FPU]

Gabarito: CACHE/S,BURST/S,NOCACHE/S,NOBURST/S,DATACACHE/S,DATABURST/S,NODATACACHE/S,NODATABURST/S,INSTCACHE/S,INSTBURST/S,NOINSTCACHE/S,NOINSTBURST/S,FASTROM/S,NOFASTROM/S,NOMUTEST/S,CHECK/K

Propósito: Ativar ou desativar os caches da CPU, testar se um determinado processador está instalado, carregar a imagem da ROM na memória rápida de 32 bits, ou ativar um handler de acesso ilegal à memória, que enviará informações via porta serial a 9600 bauds se uma tarefa acessar a página zero (os 256 bytes mais baixos) ou a memória acima de 16M.

Path: C:CPU

Especificação: O comando CPU permite ajustar várias opções do microprocessador instalado no Amiga. Este comando também exibe o processador em uso e as opções que se encontram ativadas.

Nota: Muitas dessas opções funcionam apenas com determinados membros da família de processadores 680XO. O microprocessador 68020, por exemplo, possui um tipo especial de memória conhecido como cache de instruções. Este cache faz com que as instruções sejam executadas mais depressa. Já o processador 68030 tem dois tipos de memória cache: de instruções e de dados. Caso o equipamento disponha de Static Column Dynamic RAM (SCRAM) instalada, o usuário poderá também designar um modo especial de acesso para os caches de dados e de instruções, conhecido como *burst mode*. Este procedimento é capaz de incrementar ainda mais a velocidade de acesso em determinadas circunstâncias. As opções do comando CPU, descritas abaixo, especificam o tipo de memória e de microprocessador que deve ser utilizados. Se opções mutuamente excludentes forem especificadas, a opção mais segura terá efeito.

CACHE: Ativa os caches de dados e instruções (somente microprocessador 68030).
NOCACHE: Desativa os caches de dados e instruções.
BURST: Ativa o modo burst para dados e instruções (somente microprocessador 68030 com SCRAM).
NOBURST: Desativa o modo burst para dados e instruções.
DATACACHE: Ativa o cache de dados (somente microprocessador 68030).
NODATACACHE: Desativa o cache de dados.
DATABURST: Ativa o modo burst para dados (somente microprocessador 68030 com SCRAM).

NODATABURST: Desativa o modo burst para dados.
INSTCACHE: Ativa o cache de instruções.
INSTBURST: Ativa o modo burst para instruções (caso haja memória SCRAM instalada).
NOINSTCACHE: Desativa o cache de instruções.
NOINSTBURST: Desativa o modo burst para instruções.
FASTROM: Copia os dados da ROM na memória RAM de 32 bits, o que torna o acesso aos dados significativamente mais rápido. O comando CPU protege esta área da RAM contra gravação, impedindo que os dados possam ser alterados.
NOFASTROM: Desativa a FASTROM.
NOMMUTTEST: Permite a troca da MMU (Memory Management Unit) sem verificar se a mesma se encontra em uso.

A opção CHECK, em conjunto com uma palavra-chave (68010, 68020, 68030, 68881, 68882 ou 68851), permite verificar se o microprocessador correspondente está instalado.

Exemplos:

```
1> CPU
System:  68030  68881  (INST: NoCache Burst)  (DATA: Cache
NoBurst)

1> CPU Burst Cache Check MMU
System:  68030  68881  (INST:  Cache Burst)  (DATA: Cache Burst)
1> CPU NoBurst DataCache NoInstCache
System:  68030  68881  (INST: NoCache NoBurst)  (DATA: Cache
NoBurst)
1> CPU Burst Cache FastROM
System:  68030  68881  FastROM  (INST:  Cache Burst)  (DATA:
Cache Burst)

1> CPU NoFastROM NoDataCache
System:  68030  68881  (INST:  Cache Burst)  (DATA: NoCache Burst)
```

DATE

Formato: DATE [<dia>] [<data>] [<hora>] [TO|VER <nome do arquivo>]
Gabarito: DAY,DATE,TIME,TO=VER/K
Propósito: Exibir ou acertar a data e/ou hora do sistema.
Path: C:DATE
Especificação: DATE, sem argumento, exibe a data e hora atuais do sistema, incluindo o dia da semana. Para mostrar o tempo, o sistema utiliza um relógio de 24 horas.

O comando DATE <data> acerta a data. O formato de <data> é DD-MMM-AA (dia-mês-ano). O zero antes da data é opcional, mas os hífens entre os argumentos

são obrigatórios. As três primeiras letras do mês (em inglês) também precisam ser enumeradas, bem como os dois últimos dígitos do ano.

Caso a data já tenha sido ajustada, você poderá acertá-la indicando o nome do dia, isto acertará a data desse dia da semana em diante. Pode-se também utilizar as palavras *tomorrow* (amanhã) ou *yesterday* (ontem) como o argumento de <dia>.

O comando DATE <hora> acerta a hora. O formato para a <hora> é HH:MM:SS (horas:minutos:segundos), sendo que os segundos são opcionais.

Se o seu Amiga não estiver equipado com o relógio de tempo real alimentado a bateria e você não fornecer a data, o sistema assumirá automaticamente, por ocasião da inicialização, a data de criação do arquivo mais recente contido no disco de boot.

Especificando-se as opções TO ou VER, seguidas pelo nome de um arquivo, a saída do comando DATE será enviada para o arquivo indicado, sobrepondo o seu conteúdo (se existir).

Nota: Os ajustes feitos através do comando DATE acertam apenas o relógio do software, e serão perdidos assim que o usuário desligar o equipamento. Para acertar o relógio do hardware (alimentado a bateria) a partir do Shell, é preciso acertar a data e então usar o comando SETCLOCK SAVE.

Exemplos:

```
1> DATE
```

exibe a data e hora atuais.

```
1> DATE 6-Sept-82
```

acerta a data em 6 de janeiro de 1982, sem modificar a hora (a data mais antiga que o sistema aceita é 1º de janeiro de 1978).

```
1> DATE tomorrow
```

acerta a data para um dia à frente.

```
1> DATE to ArqData
```

envia a data atual para um arquivo chamado ArqData.

```
1> DATE 23:00
```

acerta o relógio para 11:00 h da noite.

DELETE

Formato:	DELETE {<nome | padrão>} [ALL] [Q | QUIET] [FORCE]
Gabarito:	FILE/M/A,ALL/S,QUIET/S,FORCE/S
Propósito:	Deletar arquivos ou diretórios.
Path:	C:DELETE
Especificação:	O comando DELETE tenta deletar (isto é, apagar) o

arquivo especificado. Enumerando-se mais de um arquivo, o AmigaDOS prosseguirá até apagar o último arquivo da lista.

É lícito usar a comparação de padrões para deletar arquivos. O padrão pode especificar tanto nomes de arquivos como níveis de diretório. Todos os arquivos que coincidirem com os padrões serão deletados. Para abortar um DELETE de múltiplos arquivos, pressione Ctrl-C.

O AmigaDOS não solicita a confirmação do usuário para proceder a um DELETE. Um erro num DELETE executado por meio de comparação de padrões pode ter severas conseqüências, pois os arquivos apagados não poderão ser recuperados. Certifique-se de entender o mecanismo da comparação de padrões antes de utilizar este recurso, e mantenha sempre cópias backup de seus arquivos mais importantes.

ATENÇÃO: Se você tentar deletar um diretório que contém arquivos, será exibida uma mensagem avisando que essa operação não pode ser completada porque o diretório não está vazio. Para evitar que isto aconteça, basta utilizar a opção ALL. DELETE ALL apagará o diretório especificado, seus subdiretórios e todos os arquivos.

Normalmente, os nomes dos arquivos aparecem na tela à medida em que vão sendo deletados. Para suprimir a saída na tela, use a opção QUIET.

Se o flag de proteção contra apagamento do arquivo estiver ressetada (isto é, quando a flag d for igual a zero), esse arquivo só poderá ser apagado utilizando-se a opção FORCE.

Exemplos:

```
1> DELETE Xyz
```

apaga o arquivo Xyz do diretório corrente.

```
1> DELETE Work/Prog1 Work/Prog2 Work
```

apaga os arquivos Prog1 e Prog2 do diretório Work, e em seguida deleta o próprio diretório Work (desde que não haja mais nenhum arquivo nesse diretório).

```
1> DELETE T#?/#?(1|2)
```

apaga todos os arquivos terminando com 1 e 2 dos diretórios que começam com a letra T.

```
1> DELETE DF1:#? ALL FORCE
```

deleta todos os arquivos do disco contido em DF1:, inclusive aqueles protegidos contra apagamento.

DIR

Formato: DIR [<dir | padrão>] [OPT A | I | AI | D | F] [ALL] [DIRS] [FILES] [INTER]

Gabarito: DIR,OPT/K,ALL/S,DIRS/S,FILES/S,INTER/S

Propósito: Exibir uma lista ordenada dos arquivos contidos em um diretório.

Path: C:DIR

Especificação: DIR exibe os nomes dos arquivos e subdiretórios contidos no diretório indicado, ou no diretório corrente caso nenhum outro seja especificado. Os diretórios são exibidos em primeiro lugar, seguidos por uma lista alfabética dos arquivos em duas colunas. Pressionando-se Ctrl-C, a listagem dos diretórios será abortada.

As opções disponíveis para este comando são as seguintes.

ALL	Exibe todos os subdiretórios e seus arquivos.
DIRS	Exibe apenas os diretórios.
FILES	Exibe apenas os arquivos.
INTER	Entra no modo de listagem interativa.

Nota: As palavras-chave ALL, DIRS, FILES e INTER substituem as opções OPT A, D, F e I, respectivamente. As antigas palavras-chave foram mantidas a fim de preservar a compatibilidade com as primeiras versões do AmigaDOS. Não use OPT em conjunto com as palavras-chave inteiras (ALLS, DIRS, FILES e INTER).

No modo de listagem interativa é feita uma pausa depois de cada nome e exibido um ponto de interrogação (?), após o qual o usuário poderá introduzir os seguintes comandos:

Return:	Exibe o próximo nome da lista.
E:	Entra num diretório. Os nomes dos arquivos neste diretório serão exibidos.
B:	Retrocede um nível de diretório.
DEL ou DELETE:	Deleta um arquivo ou diretório vazio. Note que DEL não significa a tecla DEL, você deve digitar as letras D, E e L.
T:	Lista na tela o conteúdo do arquivo.
C ou COMMAND:	Permite a introdução de outros comandos do AmigaDOS.
Q:	Abandona o modo de listagem interativa.
?:	Exibe uma lista dos comandos disponíveis no modo interativo (help).

A opção COMMAND permite que a maioria dos comandos do AmigaDOS sejam executados durante a listagem interativa dos diretórios. Quando você quiser execu-

tar um comando, digite C (ou COM) após o prompt ?. Neste ponto, DIR perguntará pelo comando. Digite então o comando desejado e pressione Return. O comando será executado e em seguida DIR continuará.

Também é permitido digitar a opção C e o comando desejado numa só linha, introduzindo-se o comando entre aspas após o C. Por exemplo: C "type prefs.info hex" equivaleria a pressionar Q para abandonar o modo de listagem interativa, voltar para o prompt regular do Shell e digitar:

```
1> TYPE Prefs.info HEX
```

O conteúdo do arquivo Prefs.info seria então listado na tela em formato hexadecimal.

ATENÇÃO: Não é seguro formatar um disco a partir do modo de listagem interativa pois a formatação tem lugar imediatamente, sem exibir mensagens de confirmação. Além disso, iniciar um outro DIR interativo do modo de listagem interativo resultará numa saída confusa.

Exemplos:

```
1> DIR Workbench2.0:
```

exibe uma lista dos diretórios e arquivos contidos no disco do Workbench2.0.

```
1> DIR MeuDisco:#?.memo
```

exibe todos os diretórios e arquivos em MeuDisco que terminam em .memo.

```
1> DIR Extras2.0: ALL
```

exibe o conteúdo completo do disco Extras2.0 – todos os diretórios, subdiretórios e arquivos.

```
1> DIR Workbench2.0: DIRS
```

exibe apenas os diretórios do disco Workbench2.0.

```
1> DIR Workbench2.0 INTER
```

procede a uma listagem interativa do conteúdo do disco Workbench2.0.

DISKCHANGE

Formato: DISKCHANGE <dispositivo>
Gabarito: DEVICE/A
Propósito: Informar ao Amiga a troca de disco em um drive.
Path: C:DISKCHANGE
Especificação: DISKCHANGE destina-se apenas a usuários de drives de discos flexíveis de 5 1/4", ou drives de mídia removível sem hardware para troca automática de discos. Ao mudar o disco ou cartucho de um desses

drives, deve-se usar o comando DISKCHANGE para notificar a troca ao sistema.

DISKCHANGE também pode ser utilizado caso você queira editar a imagem de um ícone de disco, e visualizar o novo ícone na tela do Workbench imediatamente. Esta é a única maneira de exibir um ícone de disco rígido modificado sem precisar reinicializar o equipamento.

Exemplo: Quando surgir uma mensagem convidando-o a inserir um novo disco em seu drive de 5 1/4", conhecido como DF2:, insira o disco e em seguida digite:

```
1> DISKCHANGE DF2:
```

O AmigaDOS reconhecerá então o novo disco, e você poderá prosseguir com o seu trabalho.

DISKCOPY

Formato: DISKCOPY [FROM] <disco> TO <disco> [NOVERIFY] [MULTI] [NAME<nome>]

Gabarito: DISK/A,TO/A,DISK/A,NOVERIFY/S,MULTI/S,NAME/

Propósito: Copiar o conteúdo de um disco para outro.

Path: SYS:System/DiskCopy

Especificação: O comando DISKCOPY copia todo o conteúdo de um volume em outro disco. A palavra-chave FROM não precisa ser digitada, mas a palavra-chave TO é indispensável para que DISKCOPY funcione.

O argumento <disco> pode ser tanto o nome de um volume como o nome de um drive, tal como Workbench2.0 ou DF0:. Um ícone de disquete modificado pode ser exibido removendo-se o disco do drive e inserindo-se o mesmo novamente.

Normalmente, durante a cópia de um disco, o Amiga copia e verifica cada cilindro de dados. A opção NOVERIFY suprime o processo de verificação, agilizando a cópia.

A opção MULTI carrega os dados do disco fonte na memória, permitindo efetuar múltiplas cópias sem precisar repetir a leitura do disco fonte.

Por default, o disco destino terá o mesmo nome que o disco fonte. Especificando a opção NOME, você poderá dar um nome diferente ao disco destino.

Exemplos:

```
1> DISKCOPY DF0: to DF2:
```

copia o conteúdo do disco inserido no drive DF0: para o disco que está no drive DF2:, sobrepondo os dados eventualmente já contidos nesse disco.

```
1> DISKCOPY DF0: to DF2: NOVERIFY NAME NovoDisco
```

copia o conteúdo do disco inserido no drive DF0: para o disco que está no drive DF2:, e dá ao volume em DF2: o nome NovoDisco. Os dados do disco não serão verificados durante a cópia.

DISKDOCTOR

Formato: DISKDOCTOR <drive>
Gabarito: DRIVE/A
Propósito: Tentar restaurar o conteúdo de um disco danificado.
Path: C:DISKDOCTOR
Especificação: DISKDOCTOR tenta restaurar o conteúdo de um disco cujos dados tenham sido danificados, de maneira que você possa recuperar os arquivos e copiá-los num disco em bom estado. Ao detectar um disco danificado, o AmigaDOS exibe uma mensagem dando conta de que o disco não foi validado, ou que este apresenta um erro de leitura/gravação. Com o DISKDOCTOR é possível tentar consertar a estrutura dos arquivos desse disco.

O DISKDOCTOR pode ser usado tanto no sistema padrão de gravação de arquivos, o Old Filing System, como no sistema mais recente, o Fast Filing System (FFS). Antes de usar o DISKDOCTOR com o Fast Filing System, entretanto, você deve verificar se o valor da palavra-chave DosType, no arquivo MountList (localizado no diretório DEVS:) é 0x444F5301 (ou seja, o equivalente em ASCII a DOS1). Não tente usar o DISKDOCTOR numa partição ou disco Fast Filing System se a palavra-chave DosType não tiver o valor correto. Atente para o fato de que nem todas as partições Fast Filing System possuem uma entrada na MountList. No caso dessas partições auto-montáveis, não é necessário ajustar a MountList antes de usar o DISKDOCTOR.

As versões 1.3.5 e anteriores do DISKDOCTOR não trabalham com disquetes FFS.

ATENÇÃO: Antes de executar o DISKDOCTOR é conveniente tirar uma cópia do disco danificado pois, durante a sua operação, DISKDOCTOR tentará gravar os dados nesse mesmo disco. Em detèrminadas circunstâncias, isto poderá impedir o uso posterior de outros programas utilitários para o reparo desse disco. Após executar o DISKDOCTOR, você deverá copiar os arquivos restaurados em um outro disco, e reformatar o disco danificado.

DISKDOCTOR checa se há memória suficiente antes de começar a operar e modifica o boot block o tipo DOS.

Pode ser necessário executar o DISKDOCTOR várias vezes até que o disco fique novamente em condições de uso. Caso o DISKDOCTOR não consiga ler o root block, o disco defeituoso será renomeado Lazarus (referência ao personagem bíblico Lázaro, que foi ressuscitado).

Exemplo: Ao receber uma mensagem do tipo "Volume Workbench is not validated" ou "Error validating disk/Disk is unreadable", dando conta de que determinado disco não foi validado ou não pode ser lido, você pode usar o DISKDOCTOR para tentar

recuperar os arquivos desse disco. Por exemplo, supondo que o disco danificado esteja no drive DF1:, digite:

```
1> DISKDOCTOR DF1:
```

O AmigaDOS o convidará a inserir o disco a ser corrigido e pressionar Return. O DISKDOCTOR lerá então cada cilindro do disco, a cada erro encontrado será exibida a mensagem "Hard error Track <xx>, Surface <xx>". À medida em que os arquivos e diretórios forem repostos, seus nomes irão surgindo na tela. Ao terminar, o DISKDOCTOR exibirá a seguinte mensagem:

```
Now copy files required to a new disk and reformat this disk.
```

Copie os arquivos num outro disco e reformate o disco danificado. A ocorrência de um hard error pode significar que o disco sofreu algum tipo de dano físico. Se depois de reformatado esse disco continuar apresentando problemas, é aconselhável descartá-lo.

DISPLAY

Formato: DISPLAY {<nome do arquivo> | FROM <lista de arquivos>} [OPT mlbpenv] [t=<n>]

Gabarito: FILENAME/A/M,FROM/K,OPT/K,T/N

Propósito: Exibir gráficos gravados com o formato IFF ILBM.

Path: SYS:Utilities/Display

Especificação: DISPLAY exibe na tela os gráficos gravados com o formato IFF ILBM (Interchangeable File Format/Interleaved Bitmap). Enumerando-se uma série de arquivos na linha de comando, os gráficos serão exibidos na ordem determinada. Opcionalmente, você também pode criar um script contendo uma lista de todos os arquivos IFF que gostaria de exibir, e então usar o argumento FROM <lista de arquivos>.

As opções de DISPLAY são descritas abaixo (lembre-se de que a palavra-chave OPT deverá antecedê-las).

m Ao clicar o botão de seleção do mouse, será exibido o próximo arquivo da lista, clicando-se o botão do menu, o arquivo anterior será mostrado.

l Ao invés de encerrar após a exibição da última figura, Display volta para o primeiro arquivo e começa de novo.

b As figuras permanecerão em sua própria tela desativada, atrás da tela do Workbench. Esta opção é útil para imprimir as figuras enquanto você se ocupa com alguma outra tarefa.

p Imprime cada arquivo que é exibido. Alternativamente, você também pode pressionar Ctrl-P enquanto o arquivo está na tela.

e Esta opção diz a Display para tratar uma figura de seis níveis de profundidade (bitplanes) como uma imagem Extra Halfbrite. Esta opção se destina aos usuários de antigos editores gráficos HAM (Hold and Modify) que não salvam

uma trilha CAMG, normalmente, na ausência da CAMG, Display trata a imagem como uma figura HAM.

n Determina que as bordas não fiquem transparentes quando a imagem for sincronizada com uma fonte externa de vídeo (genlock).

v As figuras serão exibidas em modo full-video. Isto significa que a figura se estenderá até a máxima posição possível na extremidade direita da tela, indo um pouco além dos ajustes de overscan (largura máxima do display) efetuados no programa Preferences. Entretanto, quando for usada esta opção, não será possível mover a figura para os lados e Display não poderá centralizar a figura.

Uma trilha CAMG é a parte de um arquivo IFF que descreve em que modo de apresentação (viewmode) a figura deverá ser exibida.

O argumento t=<n> especifica o número de segundos em que o arquivo IFF será exibido. Isto permite efetuar um avanço automático durante a exibição dos arquivos.

Exemplos:

```
1> DISPLAY arq1 arq2 arq3
```

exibe os arquivos na ordem determinada. Para avançar de um arquivo para o outro, pressione Ctrl-C.

```
1> DISPLAY from Scriptlist
```

exibe os arquivos IFF enumerados no arquivo Scriptlist. A combinação de teclas Ctrl-C avança para o próximo arquivo.

```
1> DISPLAY from Scriptlist OPT mp
```

exibe os arquivos enumerados em Scriptlist. Clicando o botão de seleção você avançará para o próximo arquivo da lista. Ao clicar o botão do menu, será apresentado o arquivo anterior. Cada um dos arquivos será impresso ao ser exibido.

```
1> DISPLAY from ScriptList OPT t=5
```

exibe cada arquivo de ScriptList durante 5 segundos.

ECHO

Formato: ECHO [<string>] [NOLINE] [FIRST <n>] [LEN <n>] [TO <dispositivo/arquivo>]

Gabarito: /M,NOLINE/S,FIRST/K/N,LEN/K/N,TO/K

Propósito: Exibir um string.

Path: Internal

Especificação: ECHO imprime o string especificado na janela de saída ou dispositivo atual. O dispositivo default é a tela, mas pode ser qualquer outro dispositivo ou arquivo. Caso o string contenha espaços, certifique-se

de digitá-lo entre aspas. O comando ECHO é geralmente empregado em scripts.

Especificando-se a opção NOLINE, ECHO não moverá automaticamente o cursor para a próxima linha, após imprimir o string.

As opções FIRST e LEN possibilitam a impressão de um substring. FIRST <n> especifica a posição do caractere inicial e LEN <n> o número de caracteres do substring a serem impressos, a contar do primeiro caractere. Omitindo-se a opção FIRST e indicando-se apenas LEN, o substring impressa consistirá nos <n> caracteres mais à direita do string principal. Por exemplo: se o string tiver 20 caracteres e você especificar LEN=4, o 17º, 18º, 19º e 20º caracteres do string serão impressos.

Exemplos:

```
1> ECHO "bom dia para todos!"
bom dia para todos!
1>ECHO "bom dia para todos!" NOLINE FIRST 0 LEN 7
bom dia1>
```

ED

Formato: ED [FROM] <nome do arquivo> [SIZE <n>] [WITH <arquivo>] [WINDOW <especificação da janela>] [TABS <n>] [WIDTH <n>] [HEIGHT <n>] [COLS <n>] [ROWS <n>]

Gabarito: FROM/A,SIZE/N,WITH/K,WINDOW/K,TABS/N,WIDTH=COLS/N,HEIGHT=ROWS/N

Propósito: Editar arquivos de texto (editor full-screen).

Path: C:ED

Especificação: ED é um editor de textos full-screen apropriado para a preparação de código fonte ou arquivos de texto. O ED pode ser utilizado tanto para criar um novo arquivo como para alterar arquivos já existentes. O texto exibido na tela pode ser rolado para cima ou para baixo, utilizando-se as teclas do cursor. Para adicionar mais texto, basta digitá-lo na posição desejada. As teclas de deleção permitem remover o texto já existente. Alternativamente, você também pode posicionar o cursor e selecionar o texto com o mouse.

O argumento FROM especifica o arquivo a ser editado. Caso o arquivo já exista, ele será automaticamente carregado no ED e suas primeiras linhas serão exibidas na tela, prontas para serem editadas. Se o arquivo não existir, o ED o criará e apresentará uma tela vazia, pronta para a introdução do texto.

O argumento SIZE permite ao usuário ajustar o tamanho do buffer de texto utilizado pelo ED. O tamanho inicial desse buffer é baseado no tamanho do arquivo a ser editado (o mínimo é de 40.000 bytes).

O argumento opcional WITH especifica o nome de um arquivo de comandos ED, o qual pode ser usado para definir as atribuições das teclas de função, ou mesmo para executar automaticamente operações de edição num arquivo de texto já existente. O arquivo de comandos pode conter qualquer seqüência de comandos ED no modo estendido, sendo que cada comando deverá estar numa linha separada (uma lista completa dos comandos ED será fornecida mais adiante).

Os argumentos WINDOW, WIDTH e HEIGHT servem para definir o tipo do terminal, caso você não esteja utilizando o console do Amiga, ou simplesmente para ajustar o tamanho da janela do ED. WINDOW descreve o tipo do console (por exemplo, RAW:0/0/640/256/<título>, AUX: ou mesmo *). WIDTH e HEIGHT determinam o número de caracteres a serem exibidos horizontal e verticalmente.

O argumento TABS especifica o intervalo da tabulação automática. Esse intervalo corresponde ao número de espaços para a direita, a partir da posição atual, que o cursor deverá se mover quando a tecla Tab for pressionada. O valor default é 3.

Exemplos:

```
1> ED MeuTexto
```

inicia ED e carrega o arquivo de texto MeuTexto para ser editado.

```
1> ED MeuTexto SIZE 50000 TABS 8
```

inicia o ED e carrega o arquivo MeuTexto para ser editado. O tamanho do buffer de texto usado por ED será de 50.000 bytes e o intervalo da tabulação automática será de 8 caracteres.

Movimentação do cursor

O cursor pode ser posicionado em qualquer ponto do texto, bastando movimentar o apontador para o local desejado e pressionar o botão de seleção do mouse. Se preferir o teclado, você pode utilizar as teclas do cursor, a tecla Tab e diversas combinações de teclas Ctrl.

Para mover o cursor uma posição em qualquer direção, pressione a tecla do cursor correspondente. Se o cursor estiver na extremidade direita da tela, o ED rolará o texto para a esquerda, de modo que você possa visualizar o restante da linha. O ED rola o texto para baixo uma linha de cada vez, e dez caracteres horizontalmente de cada vez. Não é permitido mover o cursor além do início ou do final do arquivo, nem além das extremidades direita ou esquerda da linha. Se você tentar, o ED exibirá uma mensagem "Top of file" ou "Bottom of file".

Algumas maneiras alternativas de movimentar o cursor são descritas a seguir.

Shift-cursor para cima	Início do arquivo.
Shift-cursor para baixo	Final do arquivo.
Shift-cursor p/esquerda	Extremidade esquerda da janela do ED (indepentemente do ajuste de margens).
Shift-cursor p/direita	Final da linha atual.

Comandos Imediatos do ED – Tabela de Referência Rápida

A lista abaixo resume todos os comandos do ED no modo imediato. Para ativar qualquer um desses comandos, pressione ao mesmo tempo a tecla Ctrl e a tecla da letra indicada.

Ctrl-A	Insere uma linha.
CTRL-B	Apaga uma linha.
Ctrl-D	Rola o texto 12 linhas para baixo.
Ctrl-E	Move para o topo ou para a parte inferior da tela.
Ctrl-F	Alterna letras maiúsculas e letras minúsculas.
Ctrl-G	Repete o último comando estendido (veja lista a seguir).
Ctrl-H	Apaga o caractere à esquerda do cursor.
Ctrl-I	Move o cursor para a próxima posição de tabulação.
Ctrl-M	O mesmo que RETURN.
Ctrl-O	Apaga a palavra (ou espaços).
Ctrl-R	Move o cursor para o final da palavra anterior.
Ctrl-T	Move o cursor para o início da próxima palavra.
Ctrl-U	Rola o texto 12 linhas para cima.
Ctrl-V	Verifica tela.
Ctrl-Y	Apaga do cursor até o fim da linha.
Ctrl-[	Escape (entra no modo de comando estendido).
Ctrl-]	Move o cursor para o início ou para o fim da linha.

Comandos Estendidos do ED – Tabela de Referência Rápida

A lista a seguir descreve todos os comandos estendidos do ED. Para ativar um comando no modo estendido, pressione em primeiro lugar a tecla ESC. Uma linha especial de comando aparecerá na parte inferior da tela, indicando que o ED está pronto para aceitar as suas instruções. Digite um dos comandos apresentados na lista abaixo e pressione RETURN. O ED executará esse comando.

Por exemplo: para gravar um arquivo que você acabou de criar, pressione a tecla ESC, digite SA e tecle RETURN. O arquivo será gravado com o nome especificado no início da execução do ED. Para abandonar o ED sem gravar as modificações feitas no texto, tecle ESC, depois Q e então RETURN.

Na lista abaixo /s/ significa um string, /s/t/ representa dois strings a serem trocados, e n indica um número.

A /s/	Insere uma linha com o texto s após a linha atual.
B	Move para o final do texto.

BE	Marca o final de um bloco na posição do cursor.
BF /s/	Procura por uma ocorrência do string s antes do cursor.
BS	Marca o início de um bloco na posição do cursor.
CE	Move o cursor para o final da linha.
CL	Move o cursor uma posição para a esquerda.
CR	Move o cursor uma posição para a direita.
CS	Move o cursor para o início da linha.
D	Apaga a linha atual.
DB	Apaga um bloco. Use BS e BE para marcar o começo e o fim do bloco.
DC	Apaga o caractere na posição do cursor.
DF R	Exibe a atribuição atual da tecla de função R.
DL	Apaga o caractere à esquerda do cursor.
DW	Apaga os caracteres até o final da palavra atual.
E /s/t	Troca todas as ocorrências do string s pelo string t.
EL	Apaga os caracteres até o final da linha atual.
EM	Ativa novos menus.
EP	Move para o final da página.
EQ s/t/	Troca as ocorrências do string s pelo string t, solicitando antes a confirmação do usuário.
EX	Estende a margem direita.
F /s/	Localiza o string s.
FC	Alterna letras maiúsculas e letras minúsculas.
I /s/	Insere uma linha com o string s antes da linha atual.
IB	Insere o conteúdo do bloco marcado. Use BS e BE para marcar o começo e o fim do bloco.
IF /s/	Insere o arquivo s na posição atual.
J	Une a linha atual com a próxima linha.
LC	Faz distinção entre maiúsculas e minúsculas durante a localização de strings.
M n	Move para a linha n.
N	Move para o início da próxima linha.
NW	Cria um novo arquivo substituindo o existente.
OP	Abre um arquivo.

P	Move para o início da linha anterior.
PD	Próxima página.
PU	Página anterior.
Q	Abandona o ED sem gravar o texto.
RF	Carrega e executa um arquivo de comandos estendidos.
RK	Restaura as atribuições default das teclas de função e de controle.
RP	Repete o comando até a ocorrência de um erro.
S	Divide a linha na posição do cursor.
SA /s/	Grava o texto no arquivo s.
SB	Exibe na tela o conteúdo do último bloco marcado.
SF n/s	Atribui à tecla de função n o string s.
SH	Exibe informações.
SI nm/s/t	Seta o item n do menu para o tipo m com o texto s e t.
SL n	Ajusta a margem esquerda na coluna n.
SM	Imprime na linha de status o string especificado.
SR n	Ajusta a margem direita na coluna n.
ST n	Ajusta a distância da tabulação em n espaços.
T	Move para o início do texto.
TB	Move para a próxima posição de tabulação.
U	Desfaz as alterações na linha atual.
UC	Desfaz a distinção entre maiúsculas e minúsculas durante a localização de strings.
WB /s/	Grava o bloco marcado no arquivo /s/. Use BS e BE para marcar o começo e o fim do bloco.
WN	Move para o início da próxima palavra.
WP	Move para o espaço depois da palavra anterior.
X	Encerra a edição e grava o texto no arquivo especificado ao dar início ao ED.
XQ	Abandona o ED a menos que tenha sido feita alguma alteração no arquivo.

EDIT

Formato: EDIT [FROM] <nome do arquivo> [[TO] <nome do arquivo>] [WITH <nome do arquivo>] [VER <nome do arquivo> [[OPT P <linhas> | W <caracteres> | [PREVIOUS <linhas> | WIDTH <caracteres>]]

Gabarito: FROM/A,TO,WITH/K,VER/K,OPT/K,WIDTH/N,PREVIOUS/N

Propósito: Editar arquivos de texto processando o arquivo fonte seqüencialmente (editor de linhas).

Path: C:EDIT

Especificação: EDIT é um editor de textos orientado por linhas, que pode ser utilizado para o processamento seqüencial de arquivos sob o controle de comandos de edição. A edição é realizada movimentando-se linha por linha através do conteúdo de um arquivo de entrada (ou arquivo fonte), procedendo-se às modificações necessárias e passando-se o resultado para um arquivo de saída (ou arquivo destino).

O argumento FROM representa o arquivo fonte a ser editado, e precisa ser um arquivo já existente.

O argumento TO corresponde ao nome do arquivo destino. Este é o arquivo para o qual EDIT enviará a saída contendo as modificações realizadas durante a edição. Caso o argumento TO seja omitido, EDIT usará um arquivo temporário, e quando a edição estiver completa, gravará o conteúdo deste arquivo por cima do arquivo original.

O argumento WITH permite a utilização de um arquivo contendo os comandos de edição que EDIT deverá executar. Normalmente EDIT funciona de modo interativo, lendo os comandos que o usuário digita no teclado. Opcionalmente, você pode colocar comandos EDIT num arquivo a fim de que os mesmos sejam executados automaticamente. O argumento WITH especifica o nome desse arquivo de comandos.

O argumento VER determina o nome do arquivo para o qual EDIT enviará as mensagens de erro e verificações das linhas. Omitindo-se o argumento VER, EDIT exibirá as mensagens na tela.

EDIT tem a capacidade de retroceder um determinado número de linhas no conteúdo do arquivo fonte. Isto é possível porque EDIT não grava as linhas imediatamente no arquivo destino, retendo-as antes num buffer de saída. O argumento PREVIOUS enumera quantas linhas EDIT deverá retroceder no arquivo fonte.

O argumento WIDTH especifica o tamanho máximo da linha que EDIT pode manipular. Juntos, os argumentos PREVIOUS e WIDTH determinam o tamanho do buffer que EDIT precisará usar (tamanho do buffer = PREVIOUS * WIDTH). Os valores default são 40 para PREVIOUS e 120 para WIDTH.

Exemplos:

```
1> EDIT FROM Programa1 TO Programa1Novo WITH MeusComandos
```

inicia EDIT tendo Programa1 como o arquivo fonte e Programa1Novo como o arquivo destino. Os comandos EDIT encontrados no arquivo MeusComandos serão executados automaticamente, tal como se você os digitasse pelo teclado.

```
1> EDIT FROM Programa1 PREVIOUS 50 WIDTH 240 VER ArqVer
```

inicia EDIT tendo Programa1 como o arquivo fonte. Quaisquer modificações realizadas com EDIT serão gravadas nesse mesmo arquivo, já que o argumento TO foi omitido. O buffer de EDIT é ajustado de forma que o usuário poderá retroceder até 50 linhas no conteúdo do arquivo fonte. O tamanho da linha que EDIT pode manipular é de 240 caracteres. Todas as mensagens de erros e verificação de EDIT serão enviadas para o arquivo ArqVer.

Comandos do Edit – Tabela de Referência Rápida

Segue-se uma lista completa dos comandos do EDIT. As seguintes abreviações são usadas.

t Um string com delimitadores (por exemplo: /um string/).

n Número da linha no arquivo fonte. Um ponto (.) se refere à linha atual, um asterisco (*) corresponde à última linha do arquivo fonte.

sw Chave de comando (+ liga e desliga).

qs Um string ou string qualificado com delimitadores. Existem cinco tipos de delimitadores: B (início), E (fim), L (último), P (precisamente) e U (letras maiúsculas). Exemplos: o string qualificado "B/um string/" coincidirá com "um string" quando este ocorrer no início de uma linha. O string qualificado "U/x/" coincidirá tanto com um "X" maiúsculo como com um "x" minúsculo.

Comandos de Posicionamento

Mn	Move para a linha n.
M+	Move para a linha de número mais alto no buffer.
M	Move para a linha de número mais baixo no buffer.
N	Próxima linha.
P	Linha anterior.

Comandos de Busca

Fqs	Localiza o string qs.
BF qs	Localiza o string qs pesquisando para trás no arquivo fonte.
DF qs	Localiza o string qs e apaga as linhas intermediárias.

Comandos de Posicionamento de Caracteres

<	Move O Apontador de caracteres para a esquerda.
>	Move o apontador de caracteres para a direita.
#	Apaga o caractere na posição do apontador.
$	Caractere em caixa baixa na posição do apontador.
%	Caractere em caixa alta na posição do apontador.

j	Transforma em espaço o caractere na posição do apontador.
PA qs	Posiciona o apontador de caracteres depois de qs.
PB qs	Posiciona o apontador de caracteres antes de qs.
PR	Posiciona o apontador de caracteres no início da linha.

Comandos de Linha Atual

A qs t	Insere o string t após qs.
AP qs t	Insere o string t após qs, posiciona o apontador de caracteres após t.
B qs t	Insere o string t antes de qs.
BP qs t	Insere o string t antes de qs, posiciona o apontador de caracteres após qs.
CL t	Concatena a linha atual, o string t e a linha seguinte.
D n	Apaga a linha n. Especificando-se um segundo número de linha, apaga de n1 até n2 inclusive. Omitindo-se o número, apaga a linha atual.
DF qs	Apaga da linha atual até a linha contendo qs.
DFA qs	Apaga do fim de qs até o final da linha.
DFB qs	Apaga do início de qs até o final da linha.
DTA qs	Apaga do início da linha até depois de qs.
DTB qs	Apaga do início da linha até o começo de qs.
E qs t	Troca o string qs pelo string t.
EP qs t	Troca o string qs pelo string t e posiciona o apontador de caracteres após t.
I n	Insere o texto digitado pelo teclado antes da linha n (digite Z para encerrar a entrada de dados pelo teclado).
I t	Insere o texto contido no arquivo t antes da linha atual.
R n	Substitui a linha n pelo texto digitado através do teclado (digite Z para encerrar a entrada de dados pelo teclado). Especificando-se um segundo número de linha, substitui as linhas de n1 até n2 pelo texto digitado.
R n t	Substitui a linha n pelo texto contido no arquivo t. Especificando-se um segundo número de linha, substitui as linhas de n1 até n2 pelo arquivo.
SA qs	Divide a linha após qs.
SB qs	Divide a linha antes de qs.

Comandos Globais

GA qs t	Para cada qs encontrada enquanto o arquivo é processado, posiciona t após qs.
GB qs t	Para cada qs encontrada enquanto o arquivo é processado, posiciona t antes de qs.
GE qs t	Para cada qs encontrada enquanto o arquivo é processado, substitui qs por t.
CG n	Cancela o comando global n (cancela todos os comandos globais se n for omitido).
DG n	Desabilita o comando global n (desabilita todos os comandos globais se n for omitido).

EG n	Habilita o comando global n (habilita todos os comandos globais se n for omitido).
SHG	Exibe informações sobre os comandos globais utilizados.

Comandos de Verificação do Texto

?	Verifica a linha atual.
!	Verifica a linha com indicadores de caracteres.
T	Exibe todo o conteúdo do arquivo.
T n	Exibe n linhas do arquivo.
TL n	Exibe n linhas do arquivo, numerando-as.
TN	Exibe o conteúdo do arquivo até a mudança do buffer.
TP	Move para o começo do buffer e exibe as linhas do arquivo até chegar ao fim do buffer.
V	Liga/desliga o modo de verificação.

Comandos de Arquivo

FROM t	Utiliza como fonte o arquivo t.
FROM	Reverte para o arquivo fonte original.
TO t	Direciona a gravação das linhas para o arquivo t.
TO	Reverte para o arquivo destino original.
CF t	Fecha o arquivo t.

Outros Comandos

'	Repete o comando A, B ou E previamente utilizado.
= n	Estabelece n como a linha atual.
C t	Carrega e executa os comandos do arquivo t.
H n	Provoca uma pausa na linha n. Se no lugar de n for digitado um asterisco (*), produz uma pausa e em seguida volta à condição normal.
REWIND	Retrocede o arquivo fonte.
Q	Abandona o nível de comando, encerra a operação do EDIT se estiver no nível 1.
SHD	Exibe dados.
STOP	Abandona o EDIT sem gravar as modificações.
TR sw	Liga/desliga a remoção de espaços no final da palavra.
W	Encerra o EDIT e grava as modificações.
Z t	Modifica o indicador do término da entrada de dados pelo teclado (por default, a letra Z) para o string t.

ELSE

Formato:	ELSE
Gabarito:	(nenhum)
Propósito:	Especificar uma alternativa para uma declaração IF num arquivo script.

Path: Internal
Especificação: ELSE é usado num bloco IF de um script, com a finalidade de especificar uma ação alternativa caso a condição expressa por IF não seja verdadeira. Se a condição IF não for verdadeira, a execução do script saltará da linha IF para a linha após o ELSE. Todos os comandos intermediários serão ignorados. Se a condição IF for verdadeira, os comandos imediatamente seguintes à declaração IF serão executados até o ELSE, e então a execução saltará para a declaração ENDIF que conclui o bloco IF.

Exemplo: Imagine um script chamado Display, contendo o seguinte bloco:

```
IF exists <name>
  TYPE <name> OPT n
ELSE
  ECHO "<name> não está neste diretório"
ENDIF
```

E suponha que o seguinte comando foi usado para executar o script:

```
1> EXECUTE Display Work/Prg2
```

Se o arquivo Work/Prg2 puder ser encontrado no diretório corrente o comando TYPE <name> OPT n será executado, e o conteúdo do arquivo Work/Prg2 será exibido na tela com linhas numeradas.

Se o arquivo Work/Prg2 não puder ser encontrado no diretório corrente, o script saltará para o comando ECHO "<name> não está neste diretório" e a mensagem "Work/Prg2 não está neste diretório" será exibida na janela do Shell.

Veja também: IF, ENDIF, EXECUTE

ENDCLI

Formato: ENDCLI
Gabarito: (nenhum)
Propósito: Encerrar um processo Shell.
Path: Internal
Especificação: ENDCLI encerra a execução de um processo do Shell.

Veja também: ENDSHELL

ENDIF

Formato: ENDIF
Gabarito: (nenhum)

Propósito: Terminar um bloco IF num arquivo script
Path: Internal
Especificação ENDIF é usado em scripts no final de um bloco IF. Se a condição expressa por IF for falsa, ou se os comandos relacionados à condição verdadeira forem executados até encontrar um ELSE, a execução do script saltará para o próximo comando ENDIF. Toda declaração IF deverá terminar com um ENDIF.

O ENDIF se aplica ao comando IF ou ELSE mais recente.

Veja também: ELSE, IF

ENDSHELL

Formato: ENDSHELL
Gabarito: (nenhum)
Propósito: Encerrar um processo Shell.
Path: Internal
Especificação: ENDCLI encerra a execução de um processo do Shell.

ENDCLI também fecha uma janela do Shell.

ENDSHELL deve ser usado apenas com o Workbench carregado ou no caso de um outro Shell se encontrar em execução. Abandonando o Workbench e fechando o único Shell, você ficará impossibilitado de se comunicar com o Amiga. A única alternativa, então, será reinicializar o equipamento.

A janela do Shell pode não fechar se algum processo ativado através do Shell ainda estiver em andamento. Nesse caso, apesar de a janela continuar aberta, o Shell não aceitará a entrada de novos dados. É necessário terminar os processos pendentes para que se possa fechar a janela. Se você tiver ativado um editor através do Shell, por exemplo, a janela do Shell não fechará até que você abandone esse editor.

Em alguns casos é possível ativar um processo através do CLI e então fechar o Shell, bastando que a execução do processo tenha sido iniciada com RUN>:<comando>.

ENDSKIP

Formato: ENDSKIP
Gabarito: (nenhum)
Propósito: Terminar um bloco SKIP num arquivo script.
Path: Internal

Especificação: ENDSKIP é utilizado em scripts com o propósito de terminar a execução de um bloco SKIP. Um bloco SKIP permite pular comandos intermediários caso uma certa condição seja satisfeita. Quando um ENDSKIP é encontrado, a execução do script é retomada na linha após o ENDSKIP e o flag de condição recebe o valor 5 (WARN).

Veja também: SKIP

EVAL

Formato: EVAL <valor1> [<operação>] [<valor2>] [TO <arquivo>] [LFORMAT=<string>]

Gabarito: VALUE1/A,OP,VALUE2/M,TO/K,LFORMAT/K

Propósito: Resolver expressões simples.

Path: C:EVAL

Especificação: EVAL é utilizado para calcular e imprimir o resultado de uma expressão com números inteiros. A parte fracionária dos valores de entrada, bem como dos resultados, é truncada (eliminada).

Tanto o <valor1> como o <valor2> podem ser decimais, hexadecimais ou octais. Por default, EVAL utiliza a notação decimal. A base hexadecimal é indicada por um 0x ou #X antes do número, enquanto que um 0 ou um # antecedendo o número indicam a base octal. Os caracteres do alfabeto devem ser precedidos por um apóstrofo (').

O formato default de saída é decimal, mas a palavra-chave LFORMAT permite selecionar qualquer outro formato para a saída. LFORMAT especifica o string de formatação usado para imprimir a resposta. Você pode usar as opções %X (hexadecimal), %O (octal), %N (decimal) ou %C (caractere). As opções %X e %O exigem que seja indicado o número de dígitos da resposta (%X8, por exemplo, determina oito dígitos de saída hexadecimal). Além disso, ao utilizar LFORMAT você pode determinar que uma nova linha seja impressa, incluindo um *N no string de formatação.

As operações que podem ser efetuadas e seus símbolos correspondentes estão resumidos na tabela abaixo:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Módulo	mod
AND	&
OR	|
NOT	~
Shift p/esquerda	<<
Shift p/direita	>>

Negação	-
OR exclusivo	xor
equivalência a nível de bits	eqv

EVAL pode ser usado em scripts para atuar como um contador de loops. Nesse caso a opção TO, que envia a saída de EVAL para um arquivo, pode ser útil.

Também podem ser usados parênteses nas expressões.

Exemplos:

```
1> EVAL 4 * -5
-20
1> EVAL 0x4f / 010 LFORMAT="A resposta é %X4*N"
A resposta é 0009
1>
```

Esta expressão divide o número hexadecimal 4f (79 decimal) pelo octal 10 (8). O resultado é 0009, que corresponde à parte inteira da resposta (9.875). O prompt apareceria imediatamente após o 0009 se *N não tivesse sido incluído no string LFORMAT.

Imagine agora o seguinte script, chamado Loop:

```
.Key loop/a
;demo de um loop usando eval e skip
.Bra {
.Ket }
ECHO >ENV:Loop {loop}
LAB start
ECHO "Loop #" noline
TYPE ENV:Loop
EVAL <ENV:Loop >NIL: to=T:Qwe{$$} value2=1 op=- ?
TYPE >ENV:Loop T:Que{$$}
IF val $loop GT 0
SKIP start back
ENDIF
ECHO "Terminado"
```

Se você estivesse utilizando este script e digitasse:

```
1> EXECUTE Loop 5
Loop #5
Loop #4
Loop #3
Loop #2
Loop #1
Terminado
```

O primeiro comando ECHO envia o número fornecido como argumento do loop (introduzido como argumento do comando EXECUTE) para o arquivo ENV:Loop.

O segundo comando ECHO, acoplado com o comando TYPE, exibe o string "Loop #" seguido pelo número fornecido como o argumento do loop. Neste caso, é exibido Loop #5.

O comando EVAL pega o número no arquivo ENV:Loop e o utiliza como o <valor1>. O <valor2> é 1, e a operação é uma subtração. A saída do comando EVAL é enviada para o arquivo T:Qwe($$). Neste caso, o valor é 4.

O próximo comando NEXT envia o valor contido no arquivo T:Qwe($$) para o arquivo ENV:Loop. Neste caso, ele altera o valor em ENV:Loop de 5 para 4.

A declaração IF determina que enquanto o valor de Loop for maior do que 0, a execução do script deve recomeçar. Isto resulta na próxima linha sendo Loop #4.

O script continuará até que Loop seja igual a zero.

EXCHANGE

Formato: EXCHANGE [CX_POPKEY=<tecla>] [CX_POPUP=no] [CX_PRIORITY=<n>]

Gabarito: CX_POPKEY/K,CX_POPUP/K,CX_PRIORITY/K/N

Propósito: Monitorar e controlar os programas Commodity Exchange.

Path: SYS:Utilities/Exchange

Especificação: EXCHANGE é um programa Commodities Exchange que monitora e controla todos os outros programas Commodities Exchange. A opção CX_POPKEY=<teclas(s)> permite especificar a tecla-chave do programa. Caso você queira enumerar mais de uma tecla, certifique-se de incluir todo o argumento entre aspas (por exemplo: "CX_POPKEY=Shift F1"). CX_POPUP=no evita a abertura da janela do programa Exchange.

CX_PRIORITY=<n> determina a prioridade de Exchange sobre os demais programas Commodities Exchange. Todos os programas Commodities Exchange têm uma prioridade default igual a zero.

Para eliminar Exchange, pressione Ctrl-E.

Exemplo:

```
1> EXCHANGE "CX_POPKEY=Shift F1"
```

a janela do programa Exchange aparecerá na tela e sua execução se iniciará. Se você esconder a janela e depois quiser trazê-la de volta, a tecla-chave é a combinação de teclas Shift-F1.

EXECUTE

Formato: EXECUTE <script> [{<argumentos>}]

Gabarito: FILE/A

Propósito: Executar um script com a substituição opcional de argumentos.
Path: C:EXECUTE
Especificação: EXECUTE é usado para rodar os scripts de comandos do AmigaDOS. As linhas do script serão executadas tal como se o usuário as tivesse digitado após o prompt do Shell. Se o flag "s" de proteção de um arquivo estiver setado e o arquivo estiver no path atual, basta digitar o nome do arquivo – neste caso o comando EXECUTE não é necessário.

Para determinar a substituição de parâmetros devem ser utilizadas palavras-chave especiais nos scripts. Nesse caso o usuário poderá passar para o script as variáveis que tiver indicado na linha do comando EXECUTE. Antes de executar o script, o AmigaDOS compara os nomes dos parâmetros no script com os argumentos enumerados na linha de comando. Caso haja alguma coincidência, o AmigaDOS substituirá o nome do parâmetro no script pelos valores que você especificou na linha de comando. É possível também indicar valores default para o AmigaDOS utilizar caso nenhuma variável seja especificada na linha de comando. Se você não indicar uma variável e não houver um default especificado no script, então o valor do parâmetro ficará vazio (nenhuma substituição será efetuada).

As palavras-chave utilizadas para a substituição de parâmetros são relacionadas abaixo. Cada palavra-chave deverá ser precedida por um ponto (.).

A palavra-chave .KEY (ou .K) especifica tanto os nomes como as posições das palavras-chave um script, informando a EXECUTE quantos parâmetros devem ser esperados e como interpretá-los. Em outras palavras, .KEY serve como um gabarito para os valores dos parâmetros especificados pelo usuário. Apenas uma declaração .KEY é permitida por script. Quando presente, esta deverá ser a primeira linha do arquivo.

Os argumentos da linha .KEY podem ser indicados em conjunto com as diretivas /A e /K, que funcionam tal como os gabaritos do AmigaDOS. Os argumentos seguidos por /A deverão obrigatoriamente (*always*) ser especificados, enquanto que os argumentos seguidos por /K exigem o uso do nome desse argumento como uma palavra-chave (*keyword*). Por exemplo: se um script começa com .KEY filename/A, você deverá especificar um nome de arquivo logo após o nome do script, na linha de comando EXECUTE. O nome do arquivo será substituído nas linhas subseqüentes do script. Assim, se a primeira linha de um script for:

```
.KEY filename/A, TOname/K
```

Deve-se indicar uma variável de nome de arquivo. A variável TOname é opcional, mas se você quiser especificá-la, o uso da palavra-chave TOname é obrigatório. Suponha, por exemplo, que a primeira linha de um script seja:

```
1> EXECUTE Script ArquivoTexto TOname NovoArquivo
```

Antes de executá-lo, o AmigaDOS examinará o script à procura dos itens que estiverem entre os caracteres < e > (chamados respectivamente de BRA e KET, devido a *bracket*, "colchete" ou "parêntese"). Estes itens podem ser tanto uma

palavra-chave, como uma palavra-chave seguida de um valor default. Ao encontrar qualquer parâmetro entre parênteses angulares, EXECUTE tentará substituí-los por um valor. Assim, caso deseje usar no seu arquivo script um string contendo parênteses angulares, você deverá antes redefinir os caracteres "bracket" com os comandos .BRA e .KET (.BRA <car1> troca o parêntese à esquerda por <car1> e .KET <car2> troca o parêntese à direita por <car2>).

Por exemplo:

```
.KEY filename
ECHO "Esta linha NÃO imprime parênteses <angulares>."
.BRA {
.KET {
ECHO "Esta linha imprime parênteses <angulares>."
ECHO "O nome do arquivo especificado é {filename}.
```

O resultado seria o seguinte:

```
1> EXECUTE script ArquivoTeste
Esta linha NÃO imprime parênteses.
Esta linha imprime parênteses <angulares>.
O nome do arquivo especificado é ArquivoTeste.
```

A primeira declaração ECHO faz com que o AmigaDOS procure por uma variável para substituir pelo parâmetro <angulares>. Como nenhum argumento foi fornecido na linha de comando EXECUTE, é feita uma substituição por um string nulo. Os comandos .BRA e .KET dizem então ao script para empregar chaves a fim de cercar os parâmetros. Assim, quando a segunda declaração ECHO é executada, os parênteses angulares são impressos. A terceira declaração ECHO ilustra que as chaves agora têm a mesma função que possuíam os parênteses angulares.

Ao envolver uma palavra-chave entre os caracteres BRA e KET, pode-se também especificar um string default, para ser utilizada caso alguma variável não seja declarada na linha de comando. Há duas maneiras de se definir um default. A primeira requer que você especifique o default toda vez que referenciar um parâmetro. Neste caso, os dois strings devem ser separados com o cifrão ($).

Por exemplo, na seguinte declaração:

```
ECHO "<palavra1$palavra2> é o default para palavra1."
```

palavra2 é o valor default especificado para palavra1, e será impresso caso nenhuma outra variável seja indicada para palavra1. O inconveniente é que todas as vezes que você desejar indicar este default no seu script, será preciso usar <palavra1$palavra2>.

O comando .DOLLAR <car> (ou .DOL <car>) permite mudar o caractere delimitador do default de $ para <car>. Por exemplo:

```
.DOL #
ECHO "<palavra1#palavra2> é o default para palavra1."
```

A segunda maneira de definir um default é utilizar o comando .DEF. Isto permite indicar um default para cada palavra-chave específica. Por exemplo:

```
.DEF palavra1 "palavra2"
```

define palavra2 como o default para o parâmetro palavra1 ao longo do script. A seguinte declaração:

```
ECHO "<palavra1> é o default para palavra1."
```

resulta na mesma saída que a declaração ECHO anterior:

```
palavra2 é o default para palavra1.
```

Você pode embutir comentários num script bastando incluí-los depois de um ponto e vírgula (;) ou digitando um ponto (.) seguido por um espaço e então o comentário.

Resumo dos Comandos de Pontos

.KEY	Gabarito usado para especificar o formato dos argumentos, pode ser abreviado como .K.
.DOT <car>	Troca o caractere ponto (.) por <car>.
.BRA <car>	Troca o caractere do parêntese à esquerda (<) por <car>.
.KET	Troca o caractere do parêntese à direita (>) por <car>.
.DOLLAR <car>	Troca o caractere delimitador do default ($) por <car>, pode ser abreviado como .DOL.
.DEF <valor da palavra-chave>	Atribui um valor default para o parâmetro.
.<espaço>	Linha de comentários.
.\	Linha de comentários em branco.

Quando você determina que o AmigaDOS EXECUTE uma linha de comando, a primeira linha do script é verificada. Se essa linha iniciar com um comando de pontos, o AmigaDOS inspeciona o script à procura de parâmetros para substituição e monta um arquivo temporário no diretório T:. Caso o arquivo não comece com um comando de pontos, o AmigaDOS assume que nenhuma substituição de parâmetros é necessária e começa a executar o arquivo imediatamente, sem copiá-lo para T:. Portanto, se a tarefa que você deseja executar não exige a substituição de parâmetros, evite usar comandos de pontos, pois eles requerem acessos extras ao disco e aumentam o tempo de execução do programa.

O AmigaDOS oferece vários comandos que podem ser úteis em scripts, como IF, ELSE, SKIP, LAB e QUIT. Todos esses comandos – bem como o comando EXECUTE – podem ser aninhados dentro do script (noutras palavras, o próprio script pode conter comandos EXECUTE para executar outros arquivos script).

Para interromper a execução de um script, pressione Ctrl-D. Lembre-se de que Ctrl-D interrompe somente a execução do arquivo atual. Caso você tenha aninhado vários arquivos script, o conjunto de comandos EXECUTE só será interrompido pressionando-se Ctrl-C.

Para referenciar o número do processo corrente do Shell, use os caracteres <$$>. Este procedimento é útil na criação de arquivos temporários únicos, atribuições lógicas e nomes para PIPE.

Exemplos: Imagine um script chamado List, contendo os seguintes comandos:

```
.K filename
```

```
RUN COPY <filename> TO PRT: +
ECHO "Impressão de <filename> terminada"
```

O seguinte comando:

```
1> EXECUTE List Teste/Prg
```

agirá como se você tivesse digitado os seguintes comandos pelo teclado:

```
1> RUN COPY Teste/Prg TO PRT: +
1> ECHO "Impressão de Teste/Prg terminada"
```

Agora imagine um outro script chamado Display, usando mais alguns dos recursos descritos anteriormente:

```
.Key name/A
IF EXISTS <name>
TYPE <name> NUMBER ;se o arquivo estiver no diretório
especificado,
;exibe o conteúdo com linhas numeradas
ELSE
ECHO "<name> não está neste diretório"
ENDIF
```

O comando:

```
1>RUN EXECUTE Display Work/Prg2
```

exibirá o conteúdo do arquivo Work/Prg2 com linhas numeradas na tela, caso o mesmo esteja no diretório corrente. Se o arquivo não for encontrado, será exibida uma mensagem de erro. Além disso, devido ao parâmetro /A, a omissão do nome do arquivo na linha de comando, logo após o nome Display, resultará num erro.

Veja também: ECHO, FAILAT, IF, LAB, QUIT, RUN, SKIP

FAILAT

Formato: FAILAT [<n>]

Gabarito: RCLIM/N

Propósito: Determinar que uma seqüência de comandos não seja executada se o programa devolver um código de retorno igual ou maior que o valor especificado.

Path: Internal

Especificação: O AmigaDOS indica uma falha na operação de um comando através da geração de um código de retorno. Um código de retorno diferente de zero indica que houve algum tipo de erro na execução do comando. O valor do código de retorno, normalmente 5, 10 ou 20, indica a seriedade desse erro. Um código de retorno maior ou igual a certo limite (conhecido como o *limite*

de erro) encerrará uma seqüência de comandos não interativos (isto é, comandos especificados após um RUN ou utilizados em um script).

O comando FAILAT possibilita ao usuário alterar o limite de erro RCLIM (Return Code Limit), cujo valor inicial é 10. Aumentando o valor desse limite, você estará indicando que certos tipos de erros não devem ser considerados fatais, e portanto a execução dos comandos subseqüentes pode prosseguir normalmente. O argumento deverá ser obrigatoriamente um número positivo. O limite de erro reassumirá o seu valor inicial de 10 na saída da seqüência de comandos.

Omitindo-se o argumento, o limite de erro atual será exibido.

Exemplo: Imagine um script contendo as seguintes linhas:

```
COPY DF0:MeuArquivo to RAM:
ECHO "MeuArquivo sendo copiado."
```

Se MeuArquivo não puder ser encontrado, o script será abortado e a seguinte mensagem surgirá na janela do Shell:

```
COPY: object not found
COPY failed returncode 20:
```

Contudo, alterando-se o limite do código de retorno para um valor superior a 20, o script continuaria mesmo que o comando COPY falhasse. Por exemplo, se o script fosse modificado da seguinte maneira:

```
FAILAT 21
COPY DF0:MeuArquivo to RAM:
ECHO "MeuArquivo sendo copiado."
```

mesmo que MeuArquivo não seja encontrado, o script continuará sendo executado. A seguinte mensagem aparecerá na janela do Shell:

```
COPY: object not found
MeuArquivo sendo copiado.
```

Veja também: ECHO, EXECUTE

FAULT

Formato: FAULT <código(s) de erro>
Gabarito: /N/M
Propósito: Imprimir as mensagens correspondentes aos códigos de erro especificados.
Path: Internal
Especificação: FAULT imprime as mensagens correspondentes aos códigos de erro que o usuário especificar. Podem ser exibidas até 10 mensagens de erros de cada vez. Para introduzir mais de um código de erro como argumento

de FAULT, digite os números dos códigos desejados separados por vírgulas ou espaços.

Exemplo: Suponha que você recebeu a mensagem "Error when opening DF1:ArqTest 205" e deseja obter mais informações. Digite:

```
1> FAULT 205
FAULT 205: object not found
```

Isto revelará que o erro ocorreu porque o arquivo ArqTest não foi encontrado no drive DF1:.

FILENOTE

Formato: FILENOTE [FILE] <arquivo|padrão [[COMMENT] <comentário>] [ALL] [QUIET]

Gabarito: FILE/A,COMMENT/A,ALL/S,QUIET/S

Propósito: Anexar um comentário a um arquivo.

Path: C:FILENOTE

Especificação: FILENOTE anexa um comentário opcional de até 79 caracteres ao arquivo indicado, ou a todos os arquivos cujos nomes coincidam com o padrão especificado.

Caso contenha espaços, o <comentário> deverá ser escrito entre aspas. Para incluir aspas dentro do comentário, digite um asterisco (*) antes dos caracteres das aspas. Nesse caso o próprio comentário deverá vir escrito entre aspas, mesmo que não contenha espaços.

Se o argumento <comentário> for omitido, o comentário existente será deletado do arquivo especificado.

A função desempenhada por FILENOTE equivale à introdução de um comentário no gadget Comment da janela Information de um ícone. Quaisquer modificações feitas com FILENOTE serão refletidas na janela Information, e vice-versa.

Quando um arquivo já existente é escolhido como o arquivo destino de uma cópia (ou seja, quando este é especificado como o argumento TO de um comando COPY), seu conteúdo é sobreposto mas o comentário correspondente é preservado. Os comentários associados ao arquivo FROM, contudo, não serão copiados a menos que a opção CLONE ou COM do comando COPY tenha sido designada. Indicando-se a opção ALL, FILENOTE adicionará o <comentário> a todos os arquivos do diretório especificado. A opção QUIET faz com que a saída para o vídeo seja suprimida.

Exemplos:

```
1> FILENOTE Sonata "allegro ma non troppo"
```

anexa ao arquivo Sonata o comentário allegro ma non troppo.

```
1> FILENOTE Toccata "*"presto*""
```

neste exemplo, o comentário é "presto" (observe as aspas).

FIXFONTS

Formato: FIXFONTS
Gabarito: (nenhum)
Propósito: Atualizar os arquivos .font no diretório FONTS:
Path: SYS:System/FixFonts
Especificação: O comando FIXFONTS, que não possui argumentos, executa o programa FixFonts. A luz do drive permanecerá acesa enquanto o diretório FONTS: é atualizado. Quando a atualização estiver completa, a luz será apagada e o prompt do Shell reaparecerá.

Exemplo:

```
1> FIXFONTS
```

FKEY

Formato: FKEY [F1-F10=<string>] [SF1-SF10=<string>] [CX_PRIORITY=n] [CX_POPUP=yes | no] [CX_POPKEY=<tecla>]
Gabarito: KEY,CX_PRIORITY/K/N,CX_POPUP/K,CX_POPKEY/
Propósito: Atribuir strings de texto às teclas de função.
Path: Extras2.0:Tools/Commodities/Fkey
Especificação: FKEY é um programa Commodities Exchange que permite associar a cada uma das teclas de função (e também à combinação Shift-teclas de função) um string de texto definido pelo usuário. As atribuições das teclas de função podem ser visualizadas através do item Execute Command do menu do Workbench, ou numa janela do Shell.

Exemplo:

```
1> RUN FKEY F4=INFO\n CX_POPUP=no
```

associa o comando INFO à tecla F4. O programa FKey será iniciado sem que sua janela se abra, como determina a opção CX_POPUP=no. Se o usuário pressionar a tecla F4 enquanto trabalha numa janela do Shell, o efeito será o mesmo que digitar INFO e pressionar RETURN.

FONT

Formato: FONT [FROM <nome do arquivo>] [EDIT] [USE] [SAVE] [WORKBENCH] [SCREEN] [SYSTEM]

Gabarito:	FROM,EDIT/S,USE/S,SAVE/S,WORKBENCH/S,SCREEN/S,SYSTEM/S
Propósito:	Especificar as fontes utilizadas pelo sistema.
Path:	SYS:Prefs/Font
Especificação:	O comando FONT, sem nenhum argumento ou seguido do argumento EDIT, ativa o editor Font.

O argumento FROM deverá ser usado em conjunto com pelo menos uma das três chaves: WORKBENCH, SCREEN ou SYSTEM (mais de uma chave pode ser enumerada). O arquivo FROM deverá ser um arquivo previamente gravado com o item Save As do menu Project do editor Font. As chaves permitem que uma fonte particular seja utilizada em qualquer área da tela. Mesmo que tenha sido gravada como configuração específica de um determinado tipo de texto, a fonte contida no arquivo FROM poderá ser usada numa área diferente da tela, bastando que o usuário especifique a chave apropriada. Por exemplo: se o arquivo FROM foi criado quando você gravou uma fonte utilizada como texto da tela (Screen), a mesma fonte poderá ser usada como texto do ícone do Workbench bastando especificar a chave WORKBENCH após o nome do arquivo.

A opção USE fará com que a fonte seja carregada e utilizada na área indicada, tal como se você tivesse ativado o editor Font, selecionado o botão de rádio apropriado, escolhido a fonte e ativado o gadget Use. Para salvar essa fonte, basta especificar a opção SAVE.

Se você não especificar USE ou SAVE, a opção EDIT será assumida automaticamente, e o editor Font será ativado. Caso um arquivo FROM e uma chave WORKBENCH, SCREEN ou SYSTEM sejam indicados, a janela do editor Font se abrirá e a fonte contida no arquivo FROM será exibida próximo ao botão de rádio selecionado. Se nenhuma chave for especificada em conjunto com o arquivo FROM, o editor exibirá a última configuração utilizada.

Exemplos:

```
1> FONTS Prefs/Presets/Font.screen WORKBENCH
```

abre a janela do editor Font. A fonte previamente gravada no arquivo Font.screen será exibida no gadget de texto do ícone do Workbench. Para fechar a janela do editor, selecione o gadget Save, Use ou Cancel.

```
1> FONT Prefs/Presets/Font.screen WORKBENCH USE
```

faz com que a fonte do arquivo Font.screen seja utilizada no texto do ícone do Workbench. A janela do editor Font não se abrirá e a seleção da fonte será perdida quando o sistema for reinicializado.

```
1> FONT Prefs.Presets.Font.screen SCREEN WORKBENCH USE
```

faz com que a fonte do arquivo Font.screen seja utilizada tanto no texto da tela (Screen) como no texto do ícone do Workbench.

FORMAT

Formato: FORMAT DRIVE <drive> NAME <nome> [NOICONS] [QUICK] [FFS] [NOFFS]

Gabarito: DRIVE/A/K,NAME/A/K,NOICON/S,QUICK/S,FFS/S,NOFFS/S

Propósito: Formatar um disco para uso com o Amiga.

Path: SYS:System/Format

Especificação: Para formatar um disco o usuário deverá especificar as palavras-chave DRIVE e NAME em conjunto. O <nome> poderá ter um tamanho de 1 a 31 caracteres, e caso contenha espaços, deverá ser escrito entre aspas.

A opção NOICONS evita que um ícone Trashcan seja adicionado ao disco recém-formatado.

A opção QUICK determina que FORMAT formate e crie apenas o root block (e trilha), o boot block (e trilha), e crie os bitmap blocks (o mapa de setores livres, correspondente ao FAT do MS-DOS). Esta opção é útil para reformatar disquetes que já tenham sido formatados anteriormente.

Normalmente, os disquetes são formatados com o sistema de arquivos padrão (o Old Filing System). No caso dos discos rígidos, FORMAT usa as informações especificadas pelo programa HDToolbox ou pelo arquivo MountList para determinar o tipo do DOS (Old Filing System ou Fast Filing System) a ser utilizado. A opção FFS formata o disco no padrão Fast Filing System, cancelando as definições das palavras-chave da MountList e do tipo de DOS default.

Exemplos:

```
1> FORMAT DRIVE DF0: NAME DiscoVazio
```

formata o disco inserido no drive DF0:, apagando quaisquer dados existentes, e dá a ele o nome DiscoVazio. Para reformatar ou apagar um disco que já contenha dados, use a opção QUICK.

```
1> FORMAT DRIVE DF2: NAME NovoDisco QUICK
```

GET

Formato: GET <nome>

Gabarito: NAME/A

Propósito: Obter o valor de uma variável local.

Path: Internal

Especificação: GET é usado para obter e exibir o valor de uma variável de ambiente local. O valor será exibido na janela atual. As variáveis locais são reconhecidas somente no ambiente do Shell em que foram criadas,

ou nos Shells criados por meio da execução de um comando NEWSHELL. Se o usuário criar um novo Shell "abrindo" o ícone do Shell ou utilizando o item Execute Command do menu do Workbench, as variáveis de ambiente locais previamente criadas não serão reconhecidas.

Exemplo:

```
1> GET editor
Extras2.0:Tools/MEmacs
```

Veja também: SET

GETENV

Formato: GETENV <nome>
Gabarito: NAME/A
Propósito: Obter o valor de uma variável global.
Path: Internal
Especificação: GETENV é usado para obter e exibir o valor de uma variável de ambiente global. O valor será exibido na janela atual. As variáveis globais são armazenadas no diretório ENV: e reconhecidas por todos os Shells.

Exemplo:

```
1> GETENV editor
Extras2.0:Tools/MEmacs
```

Veja também: SETENV

GRAPHICDUMP

Formato: GRAPHICDUMP
[TINY|SMALL|MEDIUM|LARGE|<xdots>:<ydots>]
Gabarito: TINY/S,SMALL/S,MEDIUM/S,LARGE/S,<xdots>:<ydots>/S
Propósito: Imprimir uma cópia da tela.
Path: Extras2.0:Tools/GraphicDump
Especificação: GRAPHICDUMP envia um dump da tela da frente para a impressora, cerca de 10 segundos após o usuário introduzir o comando. As opções de tamanho, que correspondem aos ToolTypes aceitos pelo programa, determinam as dimensões da cópia:

TINY	1/4 da largura permitida pela impressora.
SMALL	Metade da largura permitida pela impressora.
MEDIUM	3/4 da largura permitida pela impressora.
LARGE	A largura total permitida pela impressora.

A altura da cópia é calculada de maneira que a perspectiva da tela seja preservada.

Para designar dimensões específicas, basta substituir <xdots> e <ydots>, respectivamente, pelos valores da largura e altura absoluta em pontos.

Exemplos:

```
1> GRAPHICDUMP SMALL
```

produz uma cópia impressa da tela da frente, que terá aproximadamente a metade da largura total permitida pela impressora.

```
1> GRAPHICDUMP 600:300
```

produz uma cópia impressa de 600 pontos de largura por 300 pontos de altura.

ICONEDIT

Formato: ICONEDIT
Gabarito: (nenhum)
Propósito: Editar o formato e o tipo dos ícones.
Path: Extras2.0:Tools/IconEdit
Especificação: O comando ICONEDIT, que não possui argumentos, abre a janela do programa IconEdit.

Exemplo:

```
1> ICONEDIT
```

ICONTROL

Formato: ICONTROL [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]
Gabarito: FROM,EDIT/S,USE/S,SAVE/S
Propósito: Especificar os parâmetros utilizados pelo Workbench.
Path: SYS:Prefs/IControl
Especificação: O comando ICONTROL, sem argumentos ou seguido do argumento EDIT, ativa o editor Icontrol. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor IControl. Caso você tenha gravado uma configuração especial do editor IControl no diretório

Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.
Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplos:

```
1> ICONTROL Prefs/Presets/IControl.pre USE
```

utiliza a configuração gravada no arquivo IControl.pre. A janela do editor não será aberta.

ICONX

Formato: ICONX
Gabarito: (nenhum)
Propósito: Permitir a execução de um arquivo script a partir de um ícone.
Path: C:ICONX
Especificação: O comando ICONX permite que o usário determine a execução de um arquivo script do AmigaDOS a partir de um ícone.

Para usar ICONX, crie ou copie um ícone de dados (project) para o script. Abra a janela Information do ícone e modifique o Default Tool do ícone para C:ICONX. Em seguida acrescente os ToolTypes WINDOW= e DELAY=, se desejar, e selecione o item Save para gravar o arquivo .info recém-modificado. O script poderá então ser executado clicando-se o ícone duas vezes com o botão esquerdo do mouse.

Quando o ícone é aberto, ICONX muda o diretório corrente para o diretório contendo o ícone project antes de executar o script. Uma janela de entrada/saída para o arquivo script será aberta na tela do Workbench. O ToolType WINDOW= do ícone pode ser usado para especificar o tamanho dessa janela. Você pode ainda acrescentar uma pausa (expressa em segundos) após o término da execução do arquivo, usando o ToolType DELAY=. Isto manterá a janela aberta por alguns segundos, permitindo a visualização do seu conteúdo. Especificando-se 0 para DELAY=, ICONX aguardará que o usuário pressione Ctrl-C para terminar. Pode-se fazer uso de uma seleção estendida a fim de passar arquivos que possuem ícones para o script. Os nomes desses arquivos aparecerão como palavras-chave para o script. Para usar essa facilidade, a palavra-chave .KEY deverá constar no início do script. Nesse caso, o comando EXECUTE do AmigaDOS deverá ser usado para executar o arquivo script.

Veja também: EXECUTE

IF

Formato: IF [NOT] [WARN] [ERROR] [FAIL] [<string> EQ|GT|GE <string>] [VAL] [EXISTS <nome do arquivo>]

Gabarito: NOT/S,WARN/S,ERROR/S,FAIL/S,EQ/K,GT/K,GE/K,VAL/S,EXISTS/K

Propósito: Resolver operações condicionais em arquivos script.

Path: Internal

Especificação: Num arquivo script, quando a condição expressa por um IF é verdadeira, todos os comandos subseqüentes são executados até que um comando ENDIF ou ELSE seja encontrado. Quando a condição não é verdadeira, a execução salta diretamente para o ENDIF ou o ELSE. Os comandos e condições expressos em blocos IF e ELSE podem ser estendidos por várias linhas até seus ENDIFs correspondentes.

Seguem-se algumas maneiras nas quais os comandos IF, ELSE e ENDIF podem ser usados.

```
IF <condição>     IF <condição>     IF <condição>
<comando(s)>      <comando(s)>      <comando(s)>
ENDIF             ELSE              IF <condição>
                  <comando(s)>      <comando(s)>
                  ENDIF             ENDIF
                                    ENDIF
```

O ELSE é opcional, e os IFs aninhados saltam para o ENDIF mais próximo.

As palavras-chave adicionais têm o seguinte significado:

NOT: Inverte a interpretação do resultado.

WARN: Verdadeiro se o código de retorno prévio for maior ou igual a 5.

ERROR: Verdadeiro se o código de retorno prévio for maior ou igual a 10. Disponível somente se FAILAT tiver sido reajustado para um valor maior que 10.

FAIL: Verdadeiro se o código de retorno prévio for maior ou igual a 20. Disponível somente se FAILAT tiver sido reajustado para um valor maior que 20.

<a> EQ <b>: Verdadeiro se o texto de A for idêntico ao de <b> (sem distinção quanto ao uso de letras maiúsculas ou minúsculas).

EXISTS <arquivo>: Verdadeiro se o arquivo existir.

Se mais de uma palavra-chave associada ao teste de flags (WARN, ERROR, FAIL) for especificada, aquela com o valor mais baixo será usada.

IF também admite comparações do tipo "maior que" (GT, de *greater than*) e "maior ou igual" (GE, de *greater than or equal*). Normalmente tais comparações são

efetuadas entre strings. Contudo, utilizando-se a opção VAL, é possível realizar comparações numéricas.

Nota: Pode-se usar NOT GE para "menor que" e NOT GT para "menor ou igual".

Variáveis locais e globais também podem ser usadas em conjunto com IF, desde que antecedidas por um caractere $.

Exemplos:

```
IF EXISTS Work/Prog
TYPE Work/Prog
ELSE ECHO "O arquivo não está aqui."
ENDIF
```

Se o arquivo Work/Prog existir no diretório corrente, o AmigaDOS listará seu conteúdo na tela. Caso contrário será exibida uma mensagem dando conta de que o arquivo não foi encontrado, e o AmigaDOS prosseguirá a execução do script após o ENDIF.

```
IF ERROR
SKIP errlab
ENDIF
ECHO "Sem erros."
LAB errlab
```

Se o comando previamente executado tiver gerado um código de retorno maior ou igual a 10, o AmigaDOS saltará o comando ECHO e desviará a execução para o label errlab.

Veja também: EXECUTE, FAILAT, LAB, QUIT, SKIP

IHELP

Formato: IHELP [CYCLE=<tecla>] [MAKEBIG=<tecla>] [MAKESMALL=<tecla>] [CYCLESCREEN=<tecla>] [ZIPWINDOW=<tecla>] [CX_PRIORITY=<n>]

Gabarito: CYCLE/K,MAKEBIG/K,MAKESMALL/K,CYCLESCREEN/K,ZIPWINDOW/K,CX_PRIORITY/K/N

Propósito: Habilitar o teclado a executar certas operações características do mouse.

Path: Extras2.0:Tools/Commodities/IHelp

Especificação: IHELP é um programa Commodities Exchange que permite associar às teclas determinadas funções normalmente executadas pelos gadgets das janelas. Os argumentos aceitos por IHelp correspondem aos ToolTypes que podem ser introduzidos na janela Information do ícone do programa. Se você quiser especificar mais de uma tecla em algum dos argumentos <tecla>, certifique-se de digitar todo o argumento entre aspas. A descrição dos argumentos de IHelp é a seguinte:

CYCLE: Faz com que qualquer tela de dados (project) ou de programa (tool) aberta seja rotacionada da parte de trás da tela principal para a frente da mesma.
MAKEBIG: Amplia a janela ativa o máximo possível, sem movê-la.
MAKESMALL: Reduz a janela ativa o máximo possível.
CYCLESCREEN: Rotaciona todas as telas abertas.
ZIPWINDOW: Efetua um zoom da janela ativa (equivale a ativar o gadget Zoom da janela).

O argumento CX_PRIORITY=<n> determina a prioridade de IHelp em relação aos demais programas Commodities Exchange. Por default, todos os programas Commodities Exchange possuem o nível de prioridade igual a zero. Atribuindo-se a IHelp uma prioridade igual a 3, por exemplo, este programa interceptará as teclas especificadas nos argumentos antes de qualquer outro programa Exchange.

Exemplo:

```
1> IHELP "CYCLE=Alt F7" "MAKESMALL=Control S" "MAKEBIG=Control B"
```

Ao pressionar as teclas Alt-F7, o usuário fará com que as janelas project ou tool sejam rotacionadas para a parte de trás da tela. A combinação Ctrl-B aumentará a janela ativa enquanto que Crtl-S reduzirá o seu tamanho.

INFO

Formato: INFO [<dispositivo>]
Gabarito: DEVICE
Propósito: Fornecer informações sobre um disco ou partição.
Path: C:INFO
Especificação: INFO exibe na tela uma linha contendo informações sobre cada disco ou partição, incluindo o tamanho máximo do disco, a quantidade de espaço livre e utilizado, o número de soft errors ocorridos e o status do disco.

Seguido pelo argumento <dispositivo>, INFO fornecerá as informações relativas somente ao dispositivo ou volume especificado.

Exemplo:

```
1> INFO

Unit Size Used Free Full Errs Status Name

DF0: 879K 1738   20  98%    0  Read Only Workbench2.0
DF1: 879K  418 1140  24%    0  Read/Write Text-6
Volumes available:
Workbench2.0 [Mounted]
Text-6 [Mounted]
```

INITPRINTER

Formato: INITPRINTER
Gabarito: (nenhum)
Propósito: Inicializar a impressora para as opções de impressão especificadas nos editores Preferences.
Path: Extras2.0:Tools/InitPrinter
Especificação: O comando INITPRINTER, que não possui argumentos, executa o programa InitPrinter. Você ouvirá a impressora ressetar, e o prompt do Shell reaparecerá.

Exemplo:

```
1> INITPRINTER
```

INPUT

Formato: INPUT [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]
Gabarito: FROM,EDIT/S,USE/S,SAVE/S
Propósito: Especificar velocidades diferentes para o mouse e o teclado.
Path: SYS:Prefs/Input
Especificação: O comando INPUT, sem argumentos ou seguido do argumento EDIT, ativa o editor Input. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Input. Caso você tenha gravado uma definição especial do editor Input no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.
Se a chave USE for especificada, a janela do editor não se abrirá, mas a definição gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a definição do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> INPUT Prefs/Presets/Input.fast SAVE
```

carrega e salva a definição armazenada no arquivo Input.fast. Esta definição continuará tendo efeito mesmo que o sistema seja reinicializado. A janela do editor não se abrirá.

INSTALL

Formato: INSTALL [DRIVE] <DF0: | DF1: | DF2: | DF3:> [NOBOOT] [CHECK] [FFS]

Gabarito: DRIVE/A,NOBOOT/S,CHECK/S,FFS/S

Propósito: Gravar o boot block em um disquete formatado, habilitando-o ou não a inicializar o sistema.

Path: C:INSTALL

Especificação: INSTALL limpa a área do boot block de um disquete e grava um novo boot block válido nesse disco. Por default, será instalado o boot block do sistema para gravação de arquivos utilizado durante a formatação desse disco: ou o sistema padrão, o Old Filing System (OFS), ou o Fast Filing System (FFS). Para forçar a instalação do Fast Filing System, use a chave FFS.

A opção NOBOOT remove o boot block de um disco do AmigaDOS. Este disco não poderá mais ser utilizado para dar o boot no sistema.

ATENÇÃO: A opção NOBOOT gravará um boot block num disco que não pertença ao padrão do AmigaDOS. INSTALL utilizará o tipo de DOS default, o OFS, quando efetuar uma gravação num disco não pertencente ao AmigaDOS.

A opção CHECK verifica a existência de um código de boot válido no disco. Ela informará se o disco está ou não habilitado a dar a partida no sistema, bem como se o código de boot padrão Commodore-Amiga está presente no disco. O flag de condição assume o valor 0 quando o código de boot é padrão (ou quando o disco não é "bootável"), ou 5 (WARN) em caso contrário.

Exemplos:

```
1> INSTALL DF0: CHECK
No bootblock installed
```

indica que no drive DF0: há um disquete não habilitado a inicializar o sistema.

```
1> INSTALL DF0:
```

habilita o disco no drive DF0: a dar o boot no sistema.

```
1> INSTALL DF0: CHECK
Appears to be FFS bootblock
```

indica que há um disquete padrão FFS no drive DF0:.

IPREFS

Formato: IPREFS
Gabarito: (nenhum)
Propósito: Comunicar ao Workbench as informações contidas nos arquivos individuais dos editores Preferences.
Path: C:IPREFS
Especificação: IPREFS lê os arquivos individuais do sistema Preferences e passa as informações para o Workbench a fim de que este possa responder de acordo. IPREFS é geralmente executado na Startup-sequence após os arquivos Preferences serem copiados no diretório ENV:. Cada vez que o usuário seleciona uma opção Save ou Use de um editor Preferences, IPREFS é notificado e passa as informações para o Workbench. Se necessário, IPREFS ressetará o Workbench para implementar essas mudanças. Caso alguma janela de dados (project) ou de programa (tool) esteja aberta, IPREFS exibirá uma mensagem solicitando que o usuário feche todas as janelas não drawers.

JOIN

Formato: JOIN {<arquivo | padrão>} AS | TO <nome do arquivo>
Gabarito: FILE/M/A,AS=TO/K/A
Propósito: Concatenar dois ou mais arquivos em um novo arquivo.
Path: C:JOIN
Especificação: JOIN copia todos os arquivos especificados, na ordem indicada, para um novo arquivo. O arquivo destino não poderá ter o mesmo nome de nenhum dos arquivos fonte. É obrigatório indicar o nome do arquivo destino. Os arquivos originais não serão alterados. Qualquer número de arquivos pode ser concatenado em uma única operação.

A palavra-chave TO pode ser usada como sinônimo de AS.

Exemplo:

```
1> JOIN Parte1 Parte2 Parte3 AS ArqTexto
```

KEYSHOW

Formato: KEYSHOW
Gabarito: (nenhum)
Propósito: Exibir o mapa de configuração do teclado em uso.
Path: Extras2.0:Tools/KeyShow
Especificação: O comando KEYSHOW, que não possui argumentos, abre a janela do programa KeyShow. Para abandonar o programa, selecione o gadget que fecha a janela.

Exemplo:

```
1> KEYSHOW
```

LAB

Formato: LAB [<string>]
Gabarito: (nenhum)
Propósito: Especificar um label num arquivo script.
Path: Internal
Especificação: LAB é usado em scripts para definir um label que será procurado pelo comando SKIP. O label <string> pode ser de qualquer tamanho, mas precisa consistir numa cadeia de caracteres alfanuméricos. Não é permitido o uso de símbolos. Caso contenha espaços, o <string> deverá ser escrito entre aspas.

Veja também: EXECUTE, IF, SKIP

LIST

Formato: LIST [{<dir | padrão>}] [P | PAT <padrão>] [KEYS] [DATES] [NODATES] [TO <nome>] [SUB <string>] [SINCE <data>] [UPTO <data>] [QUICK] [BLOCK] [NOHEAD] [FILES] [DIRS] [LFORMAT <string>] [ALL]
Gabarito: DIR/M,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K, SUB/K,SINCE/K,UPTO/K,QUICK/S,BLOCK/S,NOHEAD/S,FILES/S,DIRS/S,LFORMAT/K,ALL/S
Propósito: Listar as informações especificadas sobre diretórios e arquivos.
Path: C:LIST
Especificação: LIST fornece informações sobre o conteúdo do diretório corrente. Especificando-se um argumento <dir>,

<padrão> ou <nome do arquivo>, LIST exibirá informações sobre o diretório indicado, sobre todos os arquivos/diretórios cujos nomes coincidam com um determinado padrão, ou apenas sobre o arquivo especificado, respectivamente.

A menos que outras opções sejam designadas, LIST exibirá as seguintes informações:

Nome: O nome do arquivo ou diretório.

Tamanho: O tamanho do arquivo, em bytes. Se não houver nada no arquivo, este campo aparecerá vazio. No caso dos diretórios, neste registro se lerá "Dir".

Proteção: Os flags de proteção do arquivo que estiverem setados serão representados em forma de letras. Os flags ressetados aparecerão como hífens. A maioria dos arquivos exibe os flags de proteção default, ----rwed (de readable/writable/executable/deletable). Veja a descrição do comando PROTECT para maiores informações sobre os flags de proteção dos arquivos.

Data e hora: A data e hora da criação ou da última alteração do arquivo.

Comentário: O comentário anexado ao arquivo com o comando FILENOTE, se houver, precedido por dois pontos (:).

LIST possui opções que permitem ao usuário modificar a saída dessas informações. Estas opções estão relacionadas a seguir.

KEYS: Exibe o número do bloco de cada cabeçalho do arquivo ou diretório.

DATES: Exibe as datas no formato DD-MMM-AA (a menos que seja especificada a opção QUICK, este é o default).

NODATES: Suprime a exibição das informações de data e hora.

TO <nome>: Especifica um arquivo ou dispositivo de saída para LIST. Normalmente a saída de LIST é dirigida para a janela atual.

SUB<string>: Lista apenas os arquivos que contêm o substring <string>.

SINCE<data>: Lista apenas os arquivos criados após a data indicada.

UPTO<data>: Lista apenas os arquivos criados antes da data indicada.

QUICK: Lista apenas os nomes dos arquivos ou diretórios.

BLOCK: Exibe o tamanho dos arquivos em blocos, ao invés de bytes.

NOHEAD: Suprime a exibição de informações sobre os cabeçalhos dos arquivos.

FILES: Lista apenas os arquivos (sem os diretórios).

DIRS: Lista apenas os diretórios (sem os arquivos).

LFORMAT: Define um string de formatação especial para a saída de LIST.

ALL: Lista todos os arquivos em diretórios e subdiretórios.

A opção LFORMAT, que modifica a saída de LIST, pode ser empregada como um método rápido para a geração de arquivos script. As opções QUICK e NOHEAD serão assumidas automaticamente se LFORMAT for especificado. Em conjunto com LFORMAT o usuário deverá fornecer um string de formatação de saída, o qual será incorporado ao arquivo script. Caso você deseje gravar a saída, basta redirecioná-la para um arquivo, utilizando o operador > ou especificando um arquivo TO.

O formato do string de formatação de saída é LFORMAT=<string>. Para incluir a saída de LIST nesse string deverá ser usado o operador de substituição %S. O path e o nome do arquivo podem ser incorporados ao string através do mesmo procedimento. A ordem em que %S será substituído pelo path ou pelo nome do arquivo é determinada pelo número e ordem das ocorrências de %S na linha de LFORMAT, tal como se segue:

Ocorrências

de %S	1ª	2ª	3ª	4ª
1	nome do arquivo			
2	path	nome do arquivo		
3	path	nome do arquivo	nome do arquivo	
4	path	nome do arquivo	path	nome do arquivo

Algumas novas opções permitem a especificação de campos que serão impressos na saída de LFORMAT. Estas opções são as seguintes:

%A	Imprime os atributos (flags de proteção) dos arquivos.
%B	Imprime o tamanho do arquivo em blocos.
%C	Imprime os comentários anexados ao arquivo.
%D	Imprime a data associada ao arquivo.
%K	Imprime o bloco chave do arquivo.
%L	Imprime o tamanho do arquivo em bytes.
%N	Imprime o nome do arquivo.
%P	Imprime o path matriz do arquivo.
%T	Imprime a hora associada ao arquivo.

Você pode inserir um especificador de tamanho e/ou justificação entre o símbolo de porcentagem (%) e o especificador do campo.

Exemplos:

```
1> LIST Dirs
Monitors    Dir -- rwed 27-June-90    11:43:59
T           Dir -- rwed Wednesday     11:37:43
Trashcan    Dir -- rwed 21-Jun-90     17:54:20
```

Lista somente os diretórios do diretório corrente – no caso, SYS: (uma versão abreviada da saída típica é mostrada acima).

```
1> LIST Li#? TO RAM:Libs.arq
```

LIST irá procurar por todos os arquivos ou diretórios cujos nomes começam com "LI". A saída de LIST será enviada para o arquivo Libs.arq em RAM:.

```
1> LIST DF0:Documentos UPTO 09-Oct-90
```

Lista somente os arquivos ou diretórios do diretório Documentos em DF0: que não foram modificados até 9 de outubro de 1990.

```
1> LIST >RAM:Scriptnotes #? LFORMAT="filenote %S%S Testnote"
```

Um arquivo script chamado Scriptnotes será criado em RAM:. Seu conteúdo consistirá numa lista de todos os arquivos do diretório corrente. Quando Scriptnotes for executado, a cada arquivo será anexado o comentário Testnote. Por exemplo: supondo que S: seja o diretório corrente, o conteúdo de Scriptnotes, tal como produzido pelo comando acima, teria uma aparência similar a esta:

```
filenote s:HDBackup.config Testnote
filenote s:DPat Testnote
filenote s:Ed-startup Testnote
filenote s:PCD Testnote
filenote s:Shell-startup Testnote
filenote s:SPat Testnote
filenote s:Startup-sequence Testnote
```

LOADWB

Formato: LOADWB [-DEBUG] [DELAY] [CLEANUP] [NEWPATH]

Gabarito: -DEBUG/S,DELAY/S,CLEANUP/S,NEWPATH/S

Propósito: Iniciar a operação do Workbench.

Path: C:LOADWB

Especificação: LOADWB inicia a operação do Workbench. Normalmente isto é feito por ocasião do boot, através da introdução do comando LOADWB no arquivo Startup-sequence. Se o usuário tiver abandonado o Workbench, o comando LOADWB poderá ser usado a partir de um Shell a fim de reiniciar a operação do mesmo.

O Workbench tira um snapshot (impressão instantânea de dados contidos na memória) dos paths em efeito quando o comando LOADWB é executado. Esses paths são utilizados em todos os Shells iniciados a partir do Workbench.

Exemplo: Supondo que você tenha abandonado o Workbench e esteja operando através de um Shell, o comando:

```
1> LOADWB
```

trará de volta o Workbench. Digitar LOADWB quando o Workbench já está carregado não produzirá nenhum efeito. A opção LOADWB NEWPATH fará com que o Workbench tire um novo snapshot dos paths atuais. LOADWB -DEBUG iniciará o Workbench com a apresentação de um menu especial que poderá ser utilizado pelo programador para limpar a memória. Esta opção é bastante útil para a depuração de rotinas e programas.

LOCK

Formato: LOCK <drive> [ON | OFF] [<senha>]
Gabarito: DRIVE/A,ON/S,OFF/S,PASSKEY
Propósito: Atribuir a um disco o status de proteção contra gravação.
Path: C:LOCK
Especificação: LOCK é usado para habilitar ou desabilitar o status de proteção contra gravação de um disco ou partição. A proteção permanecerá ativa (LOCK ON) até que o sistema seja reinicializado, ou até a proteção ser desativada com o comando LOCK OFF.

Opcionalmente, o usuário poderá especificar uma senha a fim de proteger uma partição de disco rígido. Neste caso, a mesma senha deverá ser fornecida para que seja possível desproteger essa partição. A senha pode ter qualquer número de caracteres de extensão.

Exemplos:

```
1> LOCK Work: ON SenhaSecreta
```

A partição Work: ficará protegida contra gravação. Pode-se ler o conteúdo de Work: com os comandos DIR, LIST ou MORE, mas não será possível alterar o conteúdo dessa partição. Se o usuário tentar editar um arquivo contido em Work:, surgirá uma mensagem dando conta de que esta partição está protegida contra gravação. Se você tentar criar um novo diretório, por exemplo, a seguinte mensagem será exibida:

```
1> MAKEDIR WORK:Teste
Can't create directory Work:Teste
Disk is write-protected
```

(Impossível criar diretório Work:Teste

Disco protegido contra gravação)
Para desproteger a partição, digite:

```
1> LOCK Work: OFF SenhaSecreta
```

MAKEDIR

Formato: MAKEDIR {<nome>}
Gabarito: NAME/M
Propósito: Criar um novo diretório.
Path: C:MAKEDIR
Especificação: MAKEDIR cria um novo diretório (ou diretórios) vazio(s) com o(s) nome(s) que você especificar. O comando opera em apenas um nível de diretório de

cada vez. Assim, é necessário que os diretórios no path, ou paths indicados, já existam. O comando não será completado caso um diretório ou arquivo com o mesmo nome especificado já exista no nível imediatamente superior da hierarquia de diretórios. MAKEDIR não cria um ícone de drawer para o novo diretório.

Exemplos:

```
1> MAKEDIR Teste
```

cria o diretório Teste no diretório corrente.

```
1> MAKEDIR DF1:Xyz
```

cria o diretório Xyz no diretório raiz do disco contido no drive DF1:.

```
1> CD DF0:
1> MAKEDIR Documentos Textos Fontes
```

cria três diretórios, Documentos, Textos e Fontes, no disco inserido em DF0:.

MAKELINK

Formato: MAKELINK [FROM] <arquivo> [TO] <arquivo> [HARD]
Gabarito: FROM/A,TO/A,HARD/S
Propósito: Criar um link entre arquivos.
Path: C:MAKELINK
Especificação: MAKELINK cria um arquivo no disco que funciona como ponteiro para um outro arquivo (isto é conhecido como link). Toda vez que um comando ou aplicação chamar o arquivo FROM, na realidade o arquivo TO será usado. Por default, MAKELINK suporta hard links, os arquivos FROM e TO deverão estar no mesmo volume.

Nota: Soft links, que são links através de volumes, não são suportados por este comando.

MEMACS

Formato: MEMACS [<nome do arquivo>] [goto <n>] [OPT W]
Gabarito: (nenhum)
Propósito: Criar e editar arquivos de texto.
Path: SYS:Utilities/Tools/MEMACS
Especificação: MEMACS é um editor de texto orientado por telas que permite criar e editar vários arquivos de textos simultaneamente. O MEMACS pode ser usado tanto

para criar novos arquivos como para editar arquivos já existentes. O texto exibido na tela pode ser rolado para cima ou para os lados, utilizando-se as teclas do cursor. Para introduzir o texto, basta digitá-lo na posição desejada. As teclas de deleção permitem remover o texto já existente.

O argumento <nome do arquivo> especifica o arquivo a ser editado. Se este arquivo já existir, a tela do MEMACS aparecerá e o arquivo será carregado automaticamente. Se o arquivo ainda não existir, o MEMACS o criará e apresentará uma tela em branco, pronta para a introdução do texto.

O argumento opcional goto <n> permite especificar o número da linha na qual o cursor deverá ser posicionado quando o MEMACS for executado. O argumento OPT W determina que a janela do MEMACS seja aberta na tela do Workbench. Se este argumento for omitido, o MEMACS abrirá a sua própria tela, o que consome um pouco mais de memória.

Exemplos:

```
1> MEMACS MeuTexto OPT W
```

ativa o MEMACS na tela do Workbench, carregando o arquivo MeuTexto e apresentando-o pronto para ser editado. Se não existir, o arquivo MeuTexto será criado automaticamente.

```
1> MEMACS MeuTexto goto 17
```

o MEMACS é iniciado em sua própria tela, com o arquivo MeuTexto carregado e pronto para ser editado. O cursor será posicionado na linha 17, ao invés de no começo do arquivo. Se não existir, o arquivo MeuTexto será criado automaticamente.

Comandos do MEMACS – Tabela de Referência Rápida

Segue-se uma lista dos comandos aceitos pelo editor de texto MEMACS. O símbolo ^ antes de uma letra indica que devem ser pressionadas simultaneamente a tecla Ctrl e a tecla da letra indicada (por exemplo, ^X significa que devem ser acionadas ao mesmo templo as teclas Ctrl e X). A abreviatura Esc indica que devem ser pressionadas simultaneamente a tecla Esc e a letra indicada (por exemplo, EscW significa que devem ser acionadas ao mesmo tempo as teclas Esc e W).

Comandos de Arquivos (MENU PROJECT)

^X F — Rename. Troca o nome do arquivo em edição. O MEMACS solicitará que você introduza o novo nome do arquivo.

^X^R — Read-file. Este comando substitui o arquivo em edição por um outro arquivo. O MEMACS solicitará o nome do novo arquivo a ser editado.

^X^V — Visit-file. Este comando permite que você trabalhe com vários arquivos ao mesmo tempo. MEMACS solicitará que você introduza o nome do arquivo visita. Digite o nome do arquivo e tecle Return, o MEMACS abrirá mais uma janela, exibindo o arquivo especificado pronto para ser editado.

^X^I Insert-file. O MEMACS solicitará o nome do arquivo a ser inserido. Digite o nome do arquivo e tecle Return, o arquivo especificado será inserido na posição do cursor.

^X^S Save-file. O MEMACS gravará o arquivo, sobrepondo o conteúdo do arquivo já existente.

^X^W Save-file-as. O MEMACS gravará o arquivo, mas antes solicitará um nome para o arquivo a ser gravado.

^X^M Save-mod. Grava o arquivo atual e os arquivos visita somente se eles tiverem sido modificados. Os arquivos originais serão sobrepostos.

^X^F Save-exit. Grava as modificações, se existirem, e encerra o MEMACS. O arquivo original será sobreposto.

^- New-CLI. Abre uma janela do Shell para que você possa executar comandos do AmigaDOS. Digite ENDSHELL ou ENDCLI para fechar a janela do Shell e retornar ao MEMACS.

^X! CLI-command. O MEMACS solicitará que você introduza um comando do AmigaDOS e o passará para o Shell. As mensagens do AmigaDOS serão exibidas na metade de baixo da tela. Use ENDSHELL ou ENDCLI para retornar ao MEMACS (digite ^X 1 para fechar a janela de saída de dados do Shell).

^C Quit. Encerra o MEMACS. O arquivo não será gravado.

Comandos de Edição

^W Kill-region (Cut). Este comando destaca um bloco marcado (ou região) do texto e o copia no kill buffer.

^Y Yank (Paste). Este comando recupera um bloco previamente destacado (Cut) do kill buffer e o copia na posição do cursor.

^@ Set-mark. Este comando marca a posição inicial de um bloco para as operações de cut-and-paste. A posição final do bloco não precisa ser marcada, seguindo a posição do cursor.

EscW Copy-region. Este comando copia o bloco marcado no kill buffer.

^X^L Lower-region. Este comando converte todo o conteúdo do bloco marcado para letras maiúsculas.

^X^U Upper-region. Este comando converte todo o conteúdo do bloco marcado para letras minúsculas.

^X^B List-buffers. O MEMACS exibirá uma lista de todos os arquivos em edição e o tamanho dos buffers utilizados para mantê-los.

^XB Select-buffer. O MEMACS solicitará o nome do buffer a ser editado. Digite o nome desejado e tecle Return.

Esc^Y	Insert-buffer. O MEMACS solicitará o nome do buffer a ser inserido na posição do cursor. Digite o nome desejado e tecle Return.
^XK	Kill-buffer. O MEMACS solicitará o nome do buffer a ser deletado. Digite o nome e tecle Return.
^XJ	Justify-buffer. Remove todos os espaços em branco e marcas de tabulação existentes no lado esquerdo do arquivo de texto atual.
^L	Redisplay. Este comando desenha novamente a tela do MEMACS.
^Q	Quote-char. Utilize esta combinação de teclas antes de digitar os caracteres Control, Escape e Return nos seus textos (ou numa operação de procura de caracteres).
^J	Indent. Move o cursor para a próxima linha e endenta automaticamente o mesmo número de espaços que a linha anterior.
^T	Transpose. Inverte a posição de dois caracteres adjacentes.
^G	Cancel. Encerra a operação de comandos longos, como procura a substituição de palavras com a confirmação do usuário.

Comandos de Janelas

^X1	One-window. Exibe o buffer atual em uma tela cheia.
^X2	Split-window. Divide o buffer atual pela metade de maneira que você possa ver duas partes separadas do texto ao mesmo tempo.
^XN	Next-window. Move o cursor para a janela seguinte.
^XP	Prev-window. Move o cursor para a janela anterior.
^XZ	Expand-window. Numa tela split-window, este comando acrescenta uma linha à janela atual e subtrai uma linha da janela adjacente.
^X^Z	Shrink-window. Numa tela split-window, este comando subtrai uma linha da janela atual e acrescenta uma linha à janela adjacente.
Esc^V	Next-w-page. Numa tela split-window, este comando rola o texto da janela adjacente uma página para cima.
^XV	Prev-w-page. Numa tela split-window, este comando rola o texto da janela adjacente uma página para baixo.

Comandos de Deslocamento

Esc<	Top-of-buffer. Move o cursor para o topo do buffer de texto.
Esc>	End-of-buffer. Move o cursor para a última linha do buffer de texto.
Esc,	Top-of-window. Move o cursor para o topo da janela.
Esc.	End-of-window. Move o cursor para a última linha na janela.
^X^G	Go-to-line. O MEMACS solicitará que você introduza o número de uma linha e moverá o cursor para a linha especificada.

^X^X Swap-dot&mark. Permuta a posição do cursor com uma marca previamente estabelecida. Isto permite alternar rapidamente entre duas posições diferentes no texto.

^V Next-page. Rola o texto uma página para cima.

EscV Prev-page. Rola o texto uma página para baixo.

EscF Next-word. Move o cursor para o fim da próxima palavra.

EscB Previous-word. Move o cursor para a primeira letra da palavra anterior.

^Z Scroll-up. Rola o texto uma linha para baixo.

EscZ Scroll-down. Rola o texto uma linha para cima.

Comandos de Linhas

^O Open-line. Divide a linha na posição do cursor, transformando uma linha em duas. O cursor permanece no final da linha superior.

^X^D Kill-line. Deleta a linha sobre a qual está posicionado o cursor e copia o texto no kill buffer para futuras operações de paste (yank).

^K Kill-to-eol. Deleta o texto existente entre o cursor e o final da linha e o copia no kill buffer para operações de paste (yank).

^A Start-of-line. Posiciona o cursor no início da linha.

^E End-of-line. Posiciona o cursor no final da linha.

^N Next-line. Move o cursor uma linha para baixo.

^P Previous-line. Move o cursor uma linha para cima.

Esc! Line-to-top. Move a linha sobre a qual o cursor está posicionado para o topo da janela.

^X^O Delete-blanks. Deleta todas as linhas em branco existentes entre o cursor e a próxima linha que contém texto.

^X= Show-line#. Exibe a linha e o número da coluna na qual está posicionado o cursor, e o percentual da quantidade relativa de texto existente acima do cursor.

Comandos de Palavras

EscD Delete-forw. Deleta o texto existente entre o cursor e o final da palavra.

EscH Delete-back. Deleta o texto existente entre o cursor e o início da palavra.

EscU Upper-word. Converte o texto entre o cursor e o final da palavra em letras maiúsculas.

EscL Lower-word. Converte o texto entre o cursor e o final da palavra em letras minúsculas.

EscC Cap-word. Converte a letra sob o cursor em maiúscula e todas as outras letras da palavra em minúsculas.

Esc^ Switch-case. Transforma todas as letras maiúsculas existentes entre o cursor e o final da palavra em minúsculas, e vice-versa.

Comandos de Localização

^S Search-forward. O MEMACS solicitará que você introduza o texto do string a ser localizado. Digite a palavra desejada e tecle Return. Se o MEMACS encontrar no texto uma ocorrência da palavra especificada, o cursor será posicionado sobre ela.

^R Search-backward. O MEMACS solicitará que você introduza o texto do string a ser localizado. Digite a palavra desejada e tecle Return. Se o MEMACS encontrar em alguma parte do texto antecedente uma ocorrência da palavra especificada, o cursor será posicionado sobre ela.

EscR Search-replace. O MEMACS solicitará primeiro o string a ser localizado e logo após o string que deverá substituí-lo. Digite as palavras desejadas e tecle Return.

EscQ Query-sr. O mesmo que search-replace, porém a cada ocorrência do string especificado, o MEMACS pedirá que você confirme se deseja substituí-lo ou não.

EscF Fence-match. Move o cursor para a próxima ocorrência do caractere encontrado sob o cursor.

Comandos Extras

^U Set-arg. Possibilita digitar um número para o próximo comando a ser executado. Isto é particularmente útil para o comando Set.

EscS Set. Permite ajustar vários parâmetros do MEMACS. O MEMACS solicitará o nome do parâmetro a ser ajustado, tal como se segue:

Screen	Alterna entre a tela do Workbench e uma tela personalizada.
Interlace	Ativa/desativa o modo de vídeo entrelaçado.
Mode	+cmode e -cmode ativa/destiva os recursos do modo de edição em C. +wrap e -wrap liga/desliga o modo word-wrap (alinhamento para os textos maiores que o tamanho da linha).
Left	Ajusta a margem esquerda.
Right	Ajusta a margem direita.
Tab	Ajusta o incremento de espaços para tabulação.
Incident	Ajusta o nível de endentação para o modo cmode.
Case	Ativa/desativa distinção de maiúsculas durante a localização de strings.
Backup	ON para backups automáticos com <nome do arquivo>.bak;

SAFE para backups com proteção contra regravação acidental; OFF para desativar os backups automáticos.

^X(Start-macro. Determina que o MEMACS grave as próximas seqüências de caracteres digitados.

^X) Stop-macro. Informa ao MEMACS para interromper a gravação das seqüências de caracteres digitados.

^XE Execute-macro. Repete automaticamente todas as seqüências de comandos e operações de menus executadas entre Start-macro e Stop-macro.

^X^K Set-key. Permite definir atribuições para as teclas. O MEMACS solicitará a indicação da tecla a ser definida. Pressione uma tecla de função, de help ou do teclado numérico, e o MEMACS exibirá a definição atual dessa tecla. Você poderá então digitar uma nova definição, ou simplesmente pressionar Return.

EscK Reset-keys. Restaura as definições default das teclas de função, tecla de help e teclado numérico.

EscE Execute-file. Permite executar comandos MEMACS através de um arquivo script. O MEMACS solicitará o nome do arquivo a ser executado. Digite o nome desejado e tecle Return. O arquivo script deverá conter somente comandos do MEMACS, com os nomes dos comandos digitados exatamente como aparecem nos menus.

^[^[Execute-line. Permite executar uma linha de comandos MEMACS. O MEMACS solicitará a introdução da linha de comandos. Digite os comandos do MEMACS desejados, observando a mesma grafia dos nomes dos comandos apresentada nos menus.

MORE

Formato: MORE <nome do arquivo>
Gabarito: FILENAME/K
Propósito: Exibir o conteúdo de um arquivo ASCII.
Path: SYS:Utilities/More
Especificação: MORE exibe o conteúdo do arquivo especificado. Quando o arquivo não estiver no diretório corrente, o usuário deverá fornecer o path completo até o mesmo. Caso o arquivo não seja indicado, MORE exibirá uma mensagem solicitando a introdução do nome do arquivo.

MORE também pode ler dados de um PIPE. Como a entrada padrão obtida através do PipeHandler é de tamanho previamente desconhecido, os comandos Backspace, > e %n são desabilitados quando a entrada de dados de MORE é efetuada a partir de um PIPE (veja o Capítulo 8 para maiores informações sobre o PIPE).

Se a variável de ambiente EDITOR tiver sido definida (veja o comando SETVAR) e

você estiver usando MORE a partir de um Shell, é possível chamar um editor para utilizar com o arquivo que está sendo visualizado (digite Shift-E). A variável EDITOR deverá conter o path completo para o editor especificado (por exemplo, C:ED).

Exemplo:

```
1> MORE DF0:ArqTeste
```

exibe o conteúdo do arquivo ASCII chamado ArqTeste, presente no disco inserido no drive DF0:.

MOUNT

Formato: MOUNT <dispositivo> [FROM <nome do arquivo>]
Gabarito: DEVICE/A,FROM/K
Propósito: Tornar disponível um dispositivo conectado ao sistema.
Path: C:MOUNT
Especificação: MOUNT faz com que o AmigaDOS reconheça os dispositivos conectados ao sistema. Dado o comando, MOUNT irá buscar no arquivo MountList do diretório DEVS: (ou no arquivo opcional FROM) os parâmetros do dispositivo a ser montado. Os comandos MOUNT geralmente são incluídos no arquivo Startup-sequence.

Exemplo: Eis alguns exemplos do uso de MOUNT na Startup-sequence:

```
MOUNT Speak:
MOUNT Aux:
MOUNT Pipe:
```

Estes comandos montam o Speak.handler, o Aux.handler e o Pipe.handler encontrados no diretório L:.

NEWCLI

Formato: NEWCLI
Gabarito: (nenhum)
Propósito: Iniciar um novo processo Shell.
Path: Internal
Especificação: NEWCLI inicia um novo processo Shell. Este comando é equivalente a NEWSHELL. Veja a especificação de NEWSHELL para maiores informações.

NEWSHELL

Formato: NEWSHELL [<especificação da janela>] [FROM <nome do arquivo>]

Gabarito: WINDOW,FROM

Propósito: Abrir uma nova janela interativa do Shell.

Path: C:NEWSHELL

Especificação: NEWSHELL dá início a um novo Shell interativo. A nova janela e o novo processo tornam-se a janela e o processo atual. A nova janela terá o mesmo diretório corrente, a mesma mensagem de prompt e o mesmo tamanho de stack que a janela a partir da qual foi ativada. Entretanto, cada janela do Shell é independente, permitindo a entrada/saída de dados e a execução de programas em separado.

A janela do Shell pode ter o seu tamanho, posição e profundidade ajustados como qualquer outra janela no Amiga. Para criar uma janela personalizada, basta definir o argumento WINDOW. Você pode especificar as dimensões iniciais, a localização e o título da janela observando a seguinte sintaxe de <especificação da janela>:

```
CON:x/y/largura/altura/título/opções
```

Onde:

x	é o número de pixels da extremidade esquerda da tela até a borda esquerda da janela do Shell.
y	é o número de pixels do topo da tela até o topo da janela do Shell.
largura	é a largura da janela do Shell, em pixels.
altura	é a altura da janela do Shell, em pixels.
título	é o texto que aparece na barra de título da janela do Shell.

Por default, NEWSHELL utiliza o arquivo de inicialização S:Shell-startup, a menos que um arquivo FROM seja especificado. Você pode ter vários arquivos S:Shell-startup diferentes, cada qual contendo nomes alternativos de comandos (aliases) diferentes, por exemplo. Para carregar esses ambientes Shell personalizados basta usar a opção FROM.

O comando NEWCLI tem o mesmo efeito que NEWSHELL, ele inicia um novo processo Shell.

Exemplos:

```
1> NEWSHELL
```

Uma nova janela de Shell será aberta.

```
1> NEWSHELL CON:0/0/640/200/Meushell/CLOSE
```

Uma janela começando no canto superior esquerdo da tela e medindo 640 pixels de largura por 200 pixels de altura será aberta. A janela se intitulará Meushell e terá um gadget para fechá-la. Se você adicionar este comando ao arquivo User-startup, a janela do Shell se abrirá automaticamente quando o Amiga for inicializado.

```
1> NEWSHELL FROM S:Programa.startup
```

abrirá uma nova janela, mas em vez do arquivo Shell-startup, o arquivo Programa.startup será executado. Este arquivo pode conter os aliases e prompts de comandos que você tiver definido para utilizar durante as suas sessões de programação, por exemplo.

NOCAPSLOCK

Formato: NOCAPSLOCK [CX_PRIORITY=<n>]
Gabarito: CX_PRIORITY/K/N
Propósito: Desabilitar a tecla Caps Lock.
Path: Extras2.0:Tools/Commodities/NoCapsLock
Especificação: NOCAPSLOCK é um programa Commodities Exchange que desabilita temporariamente a tecla Caps Lock.

CX_PRIORITY=<n> determina a prioridade de NoCapsLock em relação aos demais programas Commodities Exchange. Todos os programas Commodities Exchange possuem um nível de prioridade default igual a zero.

Para eliminar NoCapsLock, pressione Ctrl-E.

Exemplo:

```
1> NOCAPSLOCK
```

NOFASTMEM

Formato: NOFASTMEM
Gabarito: (nenhum)
Propósito: Forçar o Amiga a utilizar somente a Chip RAM residente.
Path: SYS:System/NoFastMem
Especificação: NOFASTMEM desabilita a Fast RAM (ou qualquer expansão de memória) utilizada pelo sistema. A expansão de memória pode ser reativada enviando-se um BREAK a NoFastMem através de um comando BREAK ou então pressionando-se as teclas Ctrl-C. Note que Ctrl-C só funcionará caso você não tenha iniciado o programa com o comando RUN.

Exemplo:

```
1> NOFASTMEM
```

OVERSCAN

Formato: OVERSCAN [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Modificar o tamanho das áreas de exibição de textos e gráficos.

Path: SYS:Prefs/Overscan

Especificação: O comando OVERSCAN, sem argumentos ou seguido do argumento EDIT, ativa o editor Overscan. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Overscan. Caso você tenha gravado uma configuração especial do editor Overscan no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> OVERSCAN Prefs/Presets/Overscan.graphics SAVE
```

carrega e salva as dimensões de Overscan gravadas no arquivo Overscan.graphics.

PALETTE

Formato: PALETTE [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Modificar as cores da tela do Workbench.

Path: SYS:Prefs/Palette

Especificação: O comando PALETTE, sem argumentos ou seguido do argumento EDIT, ativa o editor Palette. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Palette. Caso você tenha gravado uma configuração especial do editor Palette no diretório Presets, por exemplo, a opção

FROM poderá ser usada para carregar esse arquivo. Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> PALETTE Prefs/Presets/Palette.grey USE
```

carrega e utiliza a configuração de cores gravada no arquivo Palette.grey. Se for reinicializado, o sistema utilizará a configuração de cores salva antes deste comando.

PATH

Formato: PATH [{dir}] [ADD] [SHOW] [RESET] [QUIET] [REMOVE]

Gabarito: PATH/M,ADD/S,SHOW/S,RESET/S,QUIET/S,REMOVE/S

Propósito: Controlar a lista de diretórios que o Shell utiliza para localizar os comandos.

Path: Internal

Especificação: PATH permite visualizar e alterar os paths que o AmigaDOS segue a fim de localizar os comandos e programas a serem executados. Quando um diretório se encontra no path de busca do Shell, não é necessário indicar todos os arquivos e/ou subdiretórios que precisam ser percorridos para se chegar até determinado arquivo. Basta especificar o nome do arquivo e o AmigaDOS procurará pelo path até encontrar tal arquivo.

O comando PATH sozinho (ou seguido da opção SHOW) exibirá os nomes dos diretórios no path atual. Normalmente, ao mostrar os nomes dos diretórios, PATH apresentará uma mensagem solicitando a intervenção do usuário caso não consiga encontrar um volume que faça parte do path. Se você, por exemplo, incluir o nome de um disco no path e mais tarde remover esse disco, PATH pedirá que você insira novamente o disco no drive.

A opção QUIET suprime a exibição dessas mensagens de alerta quanto a volumes que não estejam montados. Ao encontrar um volume não montado, PATH simplesmente mostrará o nome desse volume. Os nomes dos diretórios do volume incluídos no path não serão listados.

A opção ADD permite indicar nomes de diretórios a serem acrescentados ao path atual. Podem ser adicionados até 10 diretórios com o comando PATH ADD (a palavra-chave ADD é opcional), os nomes dos diretórios deverão estar separados por

pelo menos um espaço. Depois que o usuário der este comando, o AmigaDOS passará a procurar por cada um dos diretórios adicionais.

Para substituir um path existente por outro, use a opção PATH RESET seguida pelos nomes dos diretórios que irão compor o novo path. O path existente, com exceção do diretório corrente e SYS:C, será apagado e substituído pelo novo.

A opção REMOVE elimina o diretório indicado do path atual.

Exemplos:

```
1> PATH EXTRAS2.0:Tools ADD
```

acrescenta o diretório Tools no disco Extras2.0 ao path de busca do Shell. Se o disco Extras2.0 não estiver num drive, uma mensagem solicitará que o disco seja inserido em qualquer drive.

Se você retirar o disco Extras2.0 do drive e digitar:

```
1> PATH
```

será exibida a lista dos diretórios no path, e em seguida uma mensagem o convidará a reinserir o disco Extras2.0. Entretanto, se você tivesse digitado:

```
1> PATH QUIET
```

a lista dos diretórios no path seria exibida, e quando o path chegasse em Extras2.0:Tools, somente o nome do volume, Extras2.0:, apareceria na lista.

Veja também: ASSIGN

POINTER

Formato: POINTER [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Modificar o formato do apontador da tela.

Path: SYS:Prefs/Pointer

Especificação: O comando POINTER, sem argumentos ou seguido do argumento EDIT, ativa o editor Pointer. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Pointer. Caso você tenha gravado uma configuração especial do editor Pointer no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo. Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> POINTER Prefs/Presets/Pointer.star USE
```

carrega e utiliza a definição do apontador gravada no arquivo Pointer.star. Se o sistema for reinicializado, o apontador cuja definição foi salva antes deste comando aparecerá na tela.

PRINTER

Formato: PRINTER [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Especificar uma impressora e opções de impressão.

Path: SYS:Prefs/Printer

Especificação: O comando PRINTER, sem argumentos ou seguido do argumento EDIT, ativa o editor Printer. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Printer. Caso você tenha gravado uma configuração especial do editor Printer no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> PRINTER Prefs/Presets/Printer.epson SAVE
```

carrega e salva a configuração para impressão armazenada no arquivo Printer.epson.

PRINTERGFX

Formato: PRINTERGFX [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Especificar opções para impressão de gráficos.

Path: SYS:Prefs/PrinterGfx

Especificação: O comando PRINTERGFX, sem argumentos ou seguido do argumento EDIT, ativa o editor PrinterGfx. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido

previamente gravado com a opção Save As do menu do editor PrinterGfx. Caso você tenha gravado uma configuração especial do editor PrinterGfx no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> PRINTERGFX Prefs/Presets/PrinterGfx.halftone USE
```

carrega e utiliza a configuração para impressão gráfica gravada no arquivo PrinterGfx.halftone. Se o sistema for reinicializado, as especificações salvas antes deste comando serão carregadas.

PRINTFILES

Formato: PRINTFILES {[-f] <nome do arquivo>}
Gabarito: -f/S,FILENAME/A/M
Propósito: Enviar arquivos para a impressora.
Path: Extras2.0:Tools/PrintFiles
Especificação: PRINTFILES imprime o arquivo especificado. O flag -f ativa o modo de avanço de formulário (form feed mode). Ao imprimir múltiplos arquivos, certifique-se de especificar o flag antes do nome de cada arquivo.

Exemplo:

```
1> PRINTFILES -f DF0:arqteste -f DF0:arqdoc
```

imprime os arquivos arqteste e arqdoc, armazenados no disco inserido no drive DF0:. O argumento -f introduz um avanço de formulário entre os dois documentos, fazendo com que a impressão de cada arquivo seja iniciada numa nova página.

PROMPT

Formato: PROMPT [<prompt>]
Gabarito: PROMPT
Propósito: Modificar a mensagem de prompt do Shell atual.
Path: Internal
Especificação: PROMPT permite ao usuário personalizar a mensagem de prompt (o string exibida pelo Shell no início de uma linha de comando). O string do prompt

pode conter quaisquer caracteres, inclusive seqüências de códigos de escape. Nos exemplos apresentados neste manual, o string do prompt é um "1> ".

A string de prompt default é:

```
"%N.%S> "
```

o que exibe o número do Shell, um ponto (.), o diretório corrente, um parêntese angular para a direita (>) e um espaço.

As substituições disponíveis para o string <prompt> são as seguintes:

%N Exibe o número do Shell.

%S Exibe o diretório corrente.

%R Exibe o código de retorno da última operação.

Observe que o espaço não é acrescentado automaticamente ao final do string. Para incluir um espaço entre o prompt e o texto digitado, você deverá incluir o espaço no string e digitar todo o argumento entre aspas.

É possível embutir comandos no string do prompt, bastando digitar o comando desejado entre aspas inversas (`).

O comando PROMPT sozinho, sem um string de argumento, restaura o prompt default.

Exemplo:

```
1> PROMPT %N
1
```

Somente o número do Shell será exibido. O > é removido do prompt.

```
1> PROMPT "%N.%S.%R> "
1.SYS:.0>
```

O número do Shell, o diretório corrente e o código de retorno do último comando serão exibidos. Note que um espaço foi incluído após o >.

```
1> PROMPT " `date`>"
Tuesday 11-Sept-90 14:36:39>
```

O comando DATE será executado e exibido como prompt. O prompt não é atualizado à medida que o tempo passa, você terá que executar novamente o comando PROMPT para atualizá-lo.

PROTECT

Formato: PROTECT [FILE] <arquivo | padrão> [FLAGS] [+ | -] [<flags>] [ADD | SUB] [ALL] [QUIET]

Gabarito: FILE/A,FLAGS,ADD/S,SUB/S,ALL/S,QUIET/S

Propósito: Modificar os flags de proteção de um arquivo.

Path: C:PROTECT

Especificação: Todos os arquivos são armazenados com uma série de flags de proteção que controlam os seus atributos. Esses flags podem ser modificados a fim de indicar o tipo do arquivo e as operações que podem ser realizadas com ele. O comando PROTECT é usado para setar ou ressetar os flags de proteção dos arquivos.

Os flags de proteção são representados por letras, de acordo com a convenção que se segue:

r	O arquivo pode ser lido.
w	Pode-se efetuar uma gravação sobre o arquivo (o arquivo pode ser alterado).
e	O arquivo é executável (normalmente, um programa).
d	O arquivo pode ser deletado.
s	O arquivo é um script.
p	O arquivo é um comando puro e pode ser tornado residente.
a	O arquivo possui uma cópia back-up.

Para visualizar os flags de proteção associados a um arquivo, use o comando LIST. No campo de proteção, os flags setados (ON) serão representadas pelas letras correspondentes e os flags ressetados (OFF) aparecerão como hífens (-). Por exemplo: um arquivo que pode ser lido (**r**ead), gravado (**w**rite) e deletado (**d**elete) exibirá ----rw-d no campo de proteção.

Para delinear todo o campo de proteção de uma só vez, basta fornecer como argumento de FLAGS as letras dos flags que você deseja setar, sem especificar nenhuma outra palavra-chave. Os flags indicadas serão setados, e todos os outros serão ressetados.

Os símbolos + e – (ou as palavras-chave equivalentes ADD e SUB) são usados para controlar os flags indicados, sem alterar o estado dos flags não especificadas. Digitando-se as letras dos flags a serem setados ou ressetados após + ou -, respectivamente, somente esses flags serão alteradas (certifique-se de não deixar espaços depois dos símbolos nem entre as letras, a ordem das letras é indiferente). ADD e SUB funcionam de maneira semelhante, mas nesse caso deverá ser digitado um espaço entre a palavra-chave e as letras. Não é permitido setar e ressetar flags num mesmo comando.

A opção ALL acrescenta ou remove os flags de proteção indicados de todos os arquivos no diretório especificado. A opção QUIET suprime a saída na tela.

Exemplos:

```
1> PROTECT DF0:Memo +rw
```

seta apenas os flags de proteção *r* (leitura) e *w* (gravação) do arquivo Memo no drive DF0:. Nenhum outro flag de proteção é afetado.

```
1> PROTECT L:#? e SUB
```

resseta o flag de proteção *e* (execução) de todos os arquivos contidos no diretório L:.

```
1> PROTECT Work:Paint rwed
```

O status de proteção do arquivo Paint será "----rwed".

Veja também: LIST

QUIT

Formato: QUIT <código de retorno>
Gabarito: RC/N
Propósito: Abandonar um arquivo script com o código de retorno especificado.
Path: Internal
Especificação: QUIT é usado para interromper a execução do script com o código de retorno especificado. O código de retorno default é zero. Como regra geral, é recomendável utilizar os códigos de retorno padrão (5, 10 e 20).

Exemplo:

```
ASK "Quer parar agora?"
IF WARN
QUIT 5
ENDIF
ECHO "OK, o script continua."
```

Se o usuário pressionar Y como resposta à pergunta, o script será abortado, uma vez que WARN equivale a um código de retorno igual a 5. Teclando-se N ou Return, a mensagem:

```
OK, o script continua.
```

será exibida na janela do Shell.

RELABEL

Formato: RELABEL [DRIVE] <drive> [NAME] <nome>
Gabarito: DRIVE/A,NAME/A
Propósito: Mudar o nome de volume de um disco.
Path: C:RELABEL
Especificação: RELABEL troca o nome de volume do disco inserido no drive indicado pelo <nome> especificado. O nome do volume é atribuído inicialmente quando o usuário formata o disco.

Se você possui um sistema de disco flexível com apenas um drive, certifique-se de especificar os discos pelo nome de volume, e não pelo nome do drive.

Exemplos:

```
1> RELABEL Workbench2.0: MeuDisco2.0
```

troca o nome do disco Workbench2.0 por MeuDisco2.0. Note que você não precisa digitar os dois pontos (:) após o segundo nome.

```
1> RELABEL DF2:Disco_de_dados
```

muda o nome do disco inserido no drive DF2: para Disco_de_dados.

REMRAD

Formato: REMRAD [<drive>] [FORCE]
Gabarito: DEVICE,FORCE/S
Propósito: Remover o dispositivo recuperável ramdrive.device.
Path: C:REMRAD
Especificação: O comando REMRAD é usado para remover da memória o dispositivo recuperável ramdrive.device (normalmente montado como RAD:) sem que para isso seja necessário desligar o equipamento. Caso você tenha montado mais de um ramdrive.device recuperável, basta especificar o argumento <drive>. REMRAD faz com que o ramdrive.device delete todos os seus arquivos e se torne inativo. Quando o Amiga for reinicializado, o dispositivo ramdrive.device será removido completamente da memória. Se você tentar acionar o comando REMRAD com o dispositivo ainda em uso, a operação será abortada com uma mensagem de erro "Drive in use". Para remover o dispositivo enquanto este se encontra em uso, utilize a opção FORCE.

RENAME

Formato: RENAME [{FROM}] <nome> [TO | AS] <nome> [QUIET]
Gabarito: FROM/A/M,TO=AS/A,QUIET/S
Propósito: Mudar o nome de um arquivo ou diretório.
Path: C:RENAME
Especificação: RENAME substitui o nome o arquivo ou diretório FROM pelo nome TO especificado. FROM e TO deverão estar no mesmo disco. Quando aplicado ao nome de um diretório, RENAME não altera o seu conteúdo (isto é, os subdiretórios e arquivos desse diretório continuarão com os mesmos nomes e conteúdos).

Se você renomear um diretório, ou utilizar RENAME para dar a um arquivo o nome de outro diretório (por exemplo, renomear :João/Carta para :Maria/Carta), o AmigaDOS mudará a posição desse diretório dentro da hierarquia do sistema de arquivo.

Os dois pontos (:) antes do nome do diretório indicam que este se encontra no nível do diretório raiz.

Exemplos:

```
1> RENAME Maria/Prog1 AS :Antonio/Exemplo
```

muda o nome do arquivo Prog1 para Exemplo e move esse arquivo do diretório Maria para o diretório Antonio. É necessário que o diretório Antonio já exista no diretório raiz para que este comando funcione.

```
1> RENAME 7.2Fax 8.16Fax 9.22Fax TO Faxes
```

move os arquivos 7.2Fax, 8.16Fax e 9.22Fax para o diretório Faxes. É necessário que Faxes seja um diretório já existente.

RESIDENT

Formato: RESIDENT [<nome residente>] [<nome do arquivo>] [REMOVE] [ADD] [REPLACE] [PURE | FORCE] [SYSTEM]

Gabarito: NAME,FILE,REMOVE/S,ADD/S,REPLACE/S,PURE=FORCE/S,SYSTEM/S

Propósito: Exibir e modificar a lista de comandos residentes.

Path: Internal

Especificação: RESIDENT é usado para carregar comandos e acrescentá-los à lista residente mantida pelo Shell. Um comando residente é aquele que pode ser executado imediatamente, sem precisar ser carregado do disco. Isto elimina o tempo dispendido no carregamento do comando e reduz o consumo de memória durante um processamento multitarefa.

Para que possa ser tornado residente, um comando precisa ser ao mesmo tempo reentrante e reexecutável. Um comando reentrante é aquele que pode ser utilizado independentemente por dois ou mais programas ao mesmo tempo. O comando é dito reexecutável quando não precisa ser novamente carregado para ser executado uma outra vez. Os comandos que possuem essas características são chamados de puros, e têm o flag de proteção p (*pure*) setado.

LISTe o diretório C: e verifique a presença do flag de proteção p para descobrir quais comandos são puros.

Diversos comandos do diretório C:, bem como MORE do diretório Utilities, são comandos puros e podem ser tornados residentes. Um comando que não possui o flag p setado provavelmente não poderá ser tornado residente com segurança. Observe, no entanto, que apenas setar o flag p não basta para qualificar um comando ou programa como puro.

A opção REPLACE é o default do comando RESIDENT e não precisa ser declarada explicitamente. Caso não seja seguido por nenhuma opção o comando RESIDENT exibirá os programas que fazem parte da lista residente. Se nenhum <nome residente> for indicado (isto é, quando apenas um nome de arquivo for especificado), RESIDENT assumirá a parte do nome do arquivo como o nome na lista residente.

Nota: O path completo até o arquivo deverá ser indicado.

Se um <nome residente> for especificado e um programa com este nome se encontrar na lista, RESIDENT tentará substituir o comando. O <nome residente> deverá então ser usado para se referir à versão residente do comando. A substituição só terá sucesso caso o comando já residente não esteja em uso.

Para evitar este arranjo e tornar residentes várias versões de um comando simultaneamente, use a opção ADD, indicando um <nome residente> diferente para cada versão carregada.

A opção SYSTEM faz com que o comando especificado seja adicionado à parte do sistema da lista residente. Os comandos acrescentados à lista residente através da opção SYSTEM não poderão ser removidos. Para listar os arquivos do sistema contidos na lista residente, use a opção SYSTEM.

A opção PURE força RESIDENT a carregar comandos que não estejam classificados como puros (isto é, comandos que não possuem o flag p setado). Esta opção pode ser usada experimentalmente com o propósito de testar a pureza de outros comandos e programas.

ATENÇÃO: A opção PURE deve ser empregada com cautela. Lembre-se que para ser residente, um comando precisa ser ao mesmo tempo reentrante e reexecutável. Embora isto seja pouco provável, alguns de seus programas podem ser puros o bastante para ser completamente reentrantes e utilizáveis por mais de um processo ao mesmo tempo. Outros programas podem não ser inteiramente reentrantes, porém puros o bastante para serem reexecutáveis. Estes comandos podem ser tornados residentes, mas você deverá ter o cuidado de usar tais comandos em apenas um processo de cada vez.

A disponibilidade dos comandos Internal também pode ser controlada por RESIDENT. Para desativar um comando Internal (no caso de uma aplicação possuir outro

comando com o mesmo nome, por exemplo), utilize a opção RESIDENT <comando> REMOVE. A partir daí, o AmigaDOS não reconhecerá mais esse comando. Para reativar o comando desabilitado por REMOVE, use a opção REPLACE.

Exemplos:

```
1> RESIDENT C:COPY
```

torna o comando COPY residente (substitui qualquer versão já existente).

```
1> RESIDENT Copy2 DF1:C/COPY ADD
```

acrescenta outra versão de COPY à lista residente, com o nome de Copy2.

```
1> RESIDENT Xdir DF1:C/Xdir PURE
```

torna residente uma versão experimental, não pura, do comando DIR.

```
1> RESIDENT CD REMOVE
```

desabilita o comando interno CD.

```
1> RESIDENT CD REPLACE
```

faz com que o comando CD seja novamente reconhecido pelo sistema.

Veja também: PROTECT

RUN

Formato:	RUN <comando> [+{<comando>}]
Gabarito:	COMMAND/F
Propósito:	Executar comandos como um processo de segundo plano
Path:	C:RUN
Especificação:	RUN é utilizado para iniciar processos de segundo plano. Um processo de segundo plano (*background process*) não abre a sua própria janela para entrada/saída de dados nem assume o controle sobre o Shell matriz.

RUN tentará executar o <comando> e quaisquer argumentos indicados na linha de comando. Múltiplos comandos podem ser executados, bastando separá-los por um sinal de adição (+). Se você pressionar Return após o +, RUN interpretará a linha seguinte como uma continuação da mesma linha de comando.

Para possibilitar o fechamento da janela do Shell a partir da qual o processo foi iniciado, deve-se redirecionar a saída de RUN com o comando RUN >NIL: <comando>. O novo Shell de segundo plano terá o mesmo path e um stack de comandos (veja a descrição do comando STACK) do mesmo tamanho que o Shell no qual RUN foi executado.

RUN também é capaz de executar comandos armazenados na lista residente (veja o comando RESIDENT). Para maior velocidade, os comandos na lista residente são

checados antes dos comandos no path. Um Shell iniciado com RUN NEWSHELL utilizará o arquivo de inicialização default (S:Shell-startup).

Exemplos:

```
1> RUN COPY Texto PRT:+
DELETE Texto +
ECHO "Impressão terminada."
```

imprime o arquivo Texto copiando-o para o dispositivo de impressora, deleta o arquivo e exibe a mensagem dando conta de que o trabalho está terminado. Os sinais de adição são usados para concatenar as linhas de comando.

```
1> RUN EXECUTE Comseq
```

executa, em segundo plano, todos os comandos do arquivo Comseq.

SAY

Formato: SAY [-m] [-f] [-r] [-n] [-s<n>] [-p<n>] [-x <nome do arquivo>]

Gabarito: -m/S,-f/S,-r/S,-n/S,-s/K/N,-p/K/N,-x/K

Propósito: Reproduzir com sons de fala o conteúdo de uma frase ou @L3 = arquivo.

Path: SYS:Utilities/Say

Especificação: SAY faz uso das capacidades de fala do Amiga. O comando suporta as mesmas opções de operação disponíveis no Workbench, sendo que através do Shell, o usuário poderá também indicar um arquivo contendo as frases a serem pronunciadas. As opções deste comando são as seguintes:

-m Especifica uma voz masculina.

-f Especifica uma voz feminina.

-r Especifica uma voz de robô.

-n Especifica uma voz natural.

-s<n> Seguida por um número <n> entre 40 e 400, esta opção controla a velocidade da voz. Não digite espaços entre -s e o número.

-p<n> Seguida por um número entre 65 e 230, esta opção controla a entonação (pitch) da voz.

-x <nome do arquivo> Esta opção faz com que o Amiga "leia em voz alta" o conteúdo do arquivo especificado.

Não esqueça de digitar o hífen antes da letra de cada opção.

Exemplo:

```
1> SAY -m -s125 -p65 -x s:startup-sequence
```

O Amiga lerá o conteúdo do arquivo Startup-sequence com uma voz masculina num ritmo moderado.

SCREENMODE

Formato: SCREENMODE [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Selecionar um modo de vídeo.

Path: SYS:Prefs/ScreenMode

Especificação: O comando SCREENMODE, sem argumentos ou seguido do argumento EDIT, ativa o editor ScreenMode. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor ScreenMode. Caso você tenha gravado uma configuração especial do editor ScreenMode no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> SCREENMODE Prefs/Presets/ScreenMode.Hires USE
```

Você será instruído a fechar todas as janelas não drawers. O sistema ressetará e utilizará a configuração gravada no arquivo ScreenMode.Hires. A janela do editor não será aberta. Caso o sistema seja reinicializado, o modo de vídeo retornará à última configuração salva.

SEARCH

Formato: SEARCH [FROM] <nome|padrão> [SEARCH|NAME] <string|padrão> [ALL] [NONUM] [QUIET] [QUICK] [FILE] [PATTERN]

Gabarito: FROM/M,SEARCH/A,ALL/S,NONUM/S,QUIET/S,QUICK/S,FILE/S,PATTERN/S

Propósito: Localizar um string em um arquivo do diretório especificado.

Path: C:SEARCH

Especificação: O comando SEARCH procura em todos os arquivos do diretório FROM pela ocorrência do string SEARCH

especificado (as palavras-chave FROM e SEARCH são opcionais). Com a chave ALL, SEARCH estenderá a procura aos subdiretórios do diretório FROM. SEARCH exibirá o nome do arquivo sendo pesquisado e qualquer linha do mesmo que contenha o texto procurado. Caso o string a ser localizado contenha espaços, certifique-se de digitá-lo entre aspas. SEARCH não faz distinção entre letras maiúsculas e minúsculas no string procurado.

As opções disponíveis para este comando são as seguintes:

NONUM Suprime a impressão de números de linhas junto com os strings.

QUIET Pesquisa os arquivos sem exibir seus nomes na tela.

QUICK Exibe a saída num formato compacto.

FILE Procura um arquivo pelo nome especificado, e não pela ocorrência de um string.

PATTERN Utiliza a comparação de padrões na busca.

SEARCH atribui ao flag de condição o valor zero caso o string seja encontrado, ou 5 (WARN) em caso contrário. Este dado pode ser utilizado para comparações em arquivos scripts. Para abandonar a procura no arquivo atual e continuar no arquivo seguinte (se houver), digite Ctrl-D. Pressionando-se Ctrl-C, SEARCH será abortado.

Exemplos:

```
1> SEARCH DEVS: alternative
(dir)
  Keymaps (dir)
  Printers (dir)
clipboard.device..
MountList..

14 /* This is an example of an alternative type of non-filing
device mount.
narrator.device..
parallel.device..
printer.device..
serial.device..
system.configuration..
```

procura no diretório DEVS: pela ocorrência da palavra "alternative". A palavra é encontrada na linha 14 do arquivo MountList.

```
1> SEARCH Universo: "Vida inteligente" ALL
```

procura por Vida inteligente (ou vida inteligente) em todos os arquivos do diretório Universo.

```
1> SEARCH Work:#?.source SEARCH Progteste.c?? FILE PATTERN
```

localiza todos os arquivos Progteste.c com um sufixo de dois caracteres nos diretórios terminados por .source no volume Work:

SERIAL

Formato: SERIAL [FROM <nome do arquivo>] [EDIT] [USE] [SAVE]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Atribuir as especificações para a comunicação através da porta serial.

Path: SYS:Prefs/Serial

Especificação: O comando SERIAL, sem argumentos ou seguido do argumento EDIT, ativa o editor Serial. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor Serial. Caso você tenha gravado uma configuração especial do editor Serial no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> SERIAL Prefs/Presets/Serial.9600 USE
```

carrega e utiliza a configuração armazenada no arquivo Serial.9600. Caso o sistema seja reinicializado, voltará a ter efeito a última configuração salva.

SET

Formato: SET [<nome>] [<string>]

Gabarito: NAME,STRING/F

Propósito: Criar uma variável local.

Path: Internal

Especificação: O comando SET, seguido dos argumentos <nome> e <string>, cria uma nova variável de ambiente. A primeira palavra após SET será considerada o <nome>. O restante da linha de comando será assumido como o argumento <string>. Não é necessário digitar <string> entre aspas.

O comando SET, sem argumentos, exibe uma lista das variáveis locais em uso.

Uma variável de ambiente atribuída por SET é considerada local ao Shell no qual foi criada. Um novo Shell criado com com o comando NEWSHELL reconhecerá as variáveis estabelecidas no Shell original, mas se você criar um novo Shell utilizando o item Execute Command do menu do Workbench ou "abrindo" o ícone do Shell, as variáveis criadas por SET não serão reconhecidas.

Para chamar uma variável de ambiente num script ou numa linha de comando, digite um cifrão ($) antes do nome da variável. Para remover uma definição de variável local, deve-se usar o comando UNSET.

Exemplos:

```
1> SET origem
```

Este processo foi iniciado a partir de um ícone cria a variável local origem, que armazena um lembrete dando conta de que o Shell foi criado a partir de um ícone, e não por um comando NEWSHELL.

```
1> ECHO $origem
```

Este processo foi iniciado a partir de um ícone

Veja também: GET, UNSET

SETCLOCK

Formato: SETCLOCK LOAD|SAVE|RESET
Gabarito: LOAD/S,SAVE/S,RESET/S
Propósito: Acertar ou ler o relógio de tempo real alimentado a bateria.
Path: C:SETCLOCK
Especificação: SETCLOCK SAVE acerta a data e a hora do relógio do hardware (alimentado a bateria) com a data e a hora do sistema, previamente ajustadas através do editor Time ou com o comando DATE. Geralmente, SETCLOCK SAVE é utilizado após um comando DATE.

A opção SETCLOCK LOAD acerta a hora e a data do sistema utilizando as informações do relógio do hardware. Este comando costuma ser incluído na Start-up-sequence, a fim de acertar a hora automaticamente quando o Amiga é inicializado. A opção RESET simplesmente resseta o relógio. Isto pode ser necessário, por exemplo, se um programa mal escrito (ou contendo bugs) desligar o relógio ou setar o bit de teste do mesmo.

Exemplo:

```
1> DATE 17-Aug-92 05:45:54
1> SETCLOCK SAVE
```

salva a data, 17 de agosto de 1992, e a hora, 05:45 da manhã, no relógio alimentado a bateria. Este relógio continuará a marcar a hora certa mesmo depois que o Amiga for desligado. Quando o sistema é inicializado, o comando SETCLOCK LOAD na

Startup-sequence acerta a hora do sistema de acordo com a hora do relógio de tempo real.

Veja também: DATE

O relógio alimentado a bateria continua marcando a hora certa mesmo depois que o Amiga é desligado. Os modelos Amiga 500 não possuem relógio a bateria, a menos que um cartucho de expansão de memória A501 tenha sido instalado.

SETDATE

Formato: SETDATE <arquivo | padrão> [<dia da semana>] [<data>] [<hora>] [ALL]

Gabarito: FILE/A,WEEKDAY,DATE,TIME,ALL/S

Propósito: Modificar a data e hora da criação ou última alteração de um arquivo.

Path: C:SETDATE

Especificação: SETDATE modifica a data e hora da criação (ou última alteração) de um arquivo ou diretório. O comando SETDATE <arquivo> substitui a data/hora desse arquivo pela data/hora atuais do sistema. SETDATE não afeta o relógio do software nem o relógio alimentado a bateria. A saída do comando DATE pode ser utilizada para fornecer os dados de SETDATE.

Exemplos:

```
1> SETDATE ArqTeste
```

muda a data e hora associadas ao arquivo ArqTeste para a data e hora atuais.

```
1> SETDATE ArqTeste 16-09-89 15:25:52
```

muda a data e hora associadas ao arquivo ArqTeste para 16 de setembro de 1989, 3:25h da tarde.

Veja também: DATE

SETENV

Formato: SETENV [<nome>] [<string>]

Gabarito: NAME,STRING/F

Propósito: Criar uma variável global.

Path: Internal

Especificação: O comando SETENV, seguido pelos argumentos <nome> e <string>, cria uma nova variável de ambiente. A primeira palavra após SETENV será considerada o <nome>. O restante da linha de comando será assumido como o argumento <string>. Não é necessário digitar <string> entre aspas.

O comando SETENV, sem argumentos, exibe uma lista das variáveis globais em uso. As variáveis globais são armazenadas no diretório ENV: e usadas em todos os processos. Entretanto, caso uma variável local (definida por SET) e uma variável global possuam o mesmo nome, a variável local será utilizada.

As variáveis de ambiente podem ser chamadas por scripts ou outros comandos, bastando incluir um cifrão ($) antes do nome da variável.

Para remover uma definição de variável global, deve-se usar o comando UNSETENV.

Exemplos:

```
1> SETENV Editor Extras2.0:Tools/MEmacs
```

cria a variável de ambiente Editor, que pode ser utilizada com o comando MORE. O comando acima determina que o editor seja o MEmacs, localizado no diretório Tools do disco Extras2.0. A variável Editor estará disponível em todos os Shells.

```
1> SETENV Editor C:ED
```

o mesmo que acima, porém aqui o editor especificado é o ED.

Veja também: GETENV, UNSETENV

SETFONT

Formato: SETFONT <fonte> <tamanho> [SCALE] [PROP] [ITALIC] [BOLD] [UNDERLINE]

Gabarito: NAME/A,SIZE/N,SCALE/S,PROP/S,ITALIC/S,BOLD/S, UNDERLINE/S

Propósito: Mudar as fontes do Shell.

Path: C:SETFONT

Especificação: SETFONT permite que o usuário mude as fontes do texto de uma determinada janela do Shell, anulando a configuração System Default Text especificada no editor Font. SETFONT terá efeito somente na janela na qual o comando for executado.

Para utilizar SETFONT, o usuário deverá especificar o nome e o tamanho da nova fonte desejada. As outras opções deste comando são as seguintes:

SCALE — Habilita escalamento mapeado por bits (*bitmap scaling*) da fonte.

PROP	Habilita fontes proporcionais.
ITALIC	Exibe os caracteres da fonte em itálico.
BOLD	Exibe os caracteres da fonte em negrito.
UNDERLINE	Exibe os caracteres da fonte sublinhados.

Quando SETFONT é acionado, a janela do Shell é limpa e um novo prompt, desenhado com a nova fonte, surge no topo da janela.

Exemplo:

```
1> SETFONT Topaz 13 BOLD UNDERLINE
```

A janela do Shell será limpa e um novo prompt será exibido. Os caracteres pertencerão à fonte Topaz, terão um tamanho de 13 pontos, e serão impressos em negrito e sublinhados.

SETMAP

Formato: SETMAP <mapa de teclado>
Gabarito: KEYMAP/A
Propósito: Trocar o mapa de configuração do teclado usado pelo Amiga.
Path: SYS:System/SetMap
Especificação: SETMAP especifica o mapa de configuração do teclado (*keymap*) utilizado pelo Amiga. Os arquivos disponíveis são os seguintes:

Mapa	Teclado
cdn	Francês canadense
ch1	Suíço francês
ch2	Suíço alemão
d	Alemão
dk	Dinamarquês
e	Espanhol
f	Francês
gb	Inglês (Grã-Bretanha)
i	Italiano
is	Islandês
n	Norueguês
s	Sueco
usa0	(Para programas desenvolvidos antes da V1.0)

usa Americano

usa2 Dvorak

Para fazer com que o sistema utilize sempre um mapa de teclado específico, inclua o comando SETMAP no arquivo Startup-sequence.

Exemplo:

Para fazer com que o Amiga utilize um mapa de configuração de teclado espanhol, digite:

```
1> SETMAP e
```

O arquivo contendo o mapa deverá estar em DEVS:Keymaps para que SetMap possa encontrá-lo.

SETPATCH

Formato: SETPATCH [QUIET]
Gabarito: QUIET/S
Propósito: Instalar patches da ROM no software do sistema.
Path: C:SETPATCH
Especificação: SETPATCH instala modificações temporárias no sistema operacional. Quando for necessário utilizá-lo, este comando deverá ser incluído no início do arquivo Startup-sequence. Versões atualizadas de SETPATCH estarão disponíveis sempre que necessário, à medida em que o desenvolvimento do AmigaDOS continue.

SKIP

Formato: SKIP [<label>] [BACK]
Gabarito: LABEL,BACK/S
Propósito: Saltar um label durante a execução de arquivos script.
Path: Internal
Especificação: SKIP é usado em scripts com o propósito de desviar a execução do script para um <label> definido pela declaração LAB. Caso nenhum <label> seja especificado, SKIP saltará para a próxima declaração LAB.

SKIP sempre procura o label à frente da linha atual no arquivo. A opção BACK, contudo, determina que o label seja procurado a partir do começo do arquivo. Isto permite desviar a execução para pontos do script anteriores ao comando SKIP.

A opção SKIP BACK só pesquisará para trás até a última ocorrência da declaração EXECUTE. Se não houver nenhuma declaração EXECUTE no script, a procura se estenderá até o começo do arquivo.

Caso SKIP não consiga encontrar um label com o nome especificado pelo usuário, a seqüência de comandos terminará e a mensagem "Label <label> not found by Skip" será exibida.

Exemplo: Imagine o seguinte script, chamado CheckFile:

```
.KEY name
IF exists <name>
  SKIP mensagem
ELSE
ECHO "<name> não está neste diretório"
ENDIF
LAB mensagem
  ECHO "O arquivo <name> existe."
```

Suponha que você execute o script com o comando:

```
1> EXECUTE CheckFile Documento
```

Se existir um arquivo chamado Documento no diretório corrente, a execução do script saltará o comando LAB, e a mensagem "O arquivo Documento existe" será mostrada na janela do Shell.

Caso o arquivo Documento não se encontre no diretório corrente, a execução do script saltará para a linha após a declaração ELSE, e a mensagem "Documento não está neste diretório" será exibida.

Veja também: EXECUTE, LAB

SORT

Formato: SORT [FROM] <arquivo | padrão> [TO] <nome do arquivo> [COLSTART <n>] [CASE] [NUMERIC]

Gabarito: FROM/A,TO/A,COLSTART/K,CASE/S,NUMERIC/S

Propósito: Ordenar alfabeticamente as linhas de um arquivo.

Path: C:SORT

Especificação: SORT ordenará alfabeticamente o arquivo FROM, linha por linha, enviando o resultado para o arquivo TO. O comando SORT assume que FROM é um arquivo de texto normal, cujas linhas são separadas por Returns ou códigos de alimentação de linha (line-feeds). Normalmente, SORT não faz distinção entre letras maiúsculas e minúsculas. Especificando-se a opção CASE, os itens que começam por letras maiúsculas serão classificados antes daqueles iniciados por minúsculas.

A opção COLSTART permite determinar a coluna de caracteres na qual deverá ser iniciada a comparação. SORT começa a comparar as linhas a partir desse ponto, mas o processo se estenderá até o começo da linha caso as linhas comparadas coincidam até o fim.

A palavra-chave NUMERIC faz com que as linhas sejam interpretadas como números, começando da primeira coluna e seguindo para a direita até ser encontrado o primeiro caractere não numérico. As linhas que não começam por um número serão tratadas como um 0. Nesta opção, as linhas serão classificadas em ordem numérica. Caso seja especificada junto com NUMERIC, a chave CASE será ignorada.

Exemplo:

```
1> SORT DF0:Glossário DF0:Glossário.alfa
```

ordena alfabeticamente as linhas do arquivo Glossário e grava a saída classificada no arquivo Glossário.alfa. Na ordenação não será levada em conta a diferença entre letras maiúsculas e minúsculas.

STACK

Formato: STACK <n>
Gabarito: SIZE/N
Propósito: Exibir ou ajustar o tamanho do stack do Shell atual.
Path: Internal
Especificação: Todo programa requer, para ser executado, uma determinada área de stack (uma área especial da memória alocada para o programa). Em geral, a documentação do programa deve indicar o tamanho do stack exigido para o perfeito funcionamento do mesmo. Entretanto, quando um programa causa uma falha do sistema, pode ser útil experimentar executá-lo com outros tamanhos de stack.

Comandos que executam operações consistindo de múltiplos níveis podem requerer espaço de stack adicional.

O tamanho do stack normalmente varia entre 4.000 e 25.000 bytes. Se o tamanho do stack for muito pequeno, pode ocorrer um erro do tipo software failure. Uma área de stack muito grande, por outro lado, pode acabar roubando memória de outras funções do sistema.

ATENÇÃO: Você poderá receber uma mensagem "Software failure" se ficar sem espaço de stack. Caso você tenha alterado o stack do programa que causou essa mensagem de erro, experimente aumentar o tamanho do stack.

STATUS

Formato: STATUS [<processo>] [FULL] [TCB] [CLI | ALL] [COMMAND <comando>]

Gabarito: PROCESS/N,FULL/S,TCB/S,CLI=ALL/S,COM=COMMAND/K

Propósito: Exibir informações sobre os processos Shell/CLI.

Path: C:STATUS

Especificação: O comando STATUS, sem argumentos, lista os números dos processos Shell/CLI em curso, e os programas ou comandos (se houver) em execução em cada um. O argumento <processo> permite especificar o número do processo sobre o qual STATUS deverá fornecer informações.

Para obter informações sobre o tamanho do stack e dos vetores globais, a prioridade do processo e o(s) comando(s) em execução, use a palavra-chave FULL. A opção TCB é semelhante, porém omite a informação sobre o comando.

A opção COMMAND determina que STATUS procure pelo <comando> especificado. STATUS pesquisará a lista do Shell, e caso o comando seja encontrado, será exibido o número do Shell correspondente. O flag de condição receberá o valor 0, neste caso, ou 5 (WARN) caso a busca não tenha sucesso. Esta opção pode ser usada em arquivos script.

Exemplos:

```
1> STATUS 1
Process 1: Loaded as command: status
1> STATUS 1 FULL
Process 1: stk 4000, gv 150, pri 0 Loaded as command: status
1> STATUS >RAM:Xyz COMMAND=COPY
1> BREAK <RAM:Xyz >NIL: ?
```

envia um BREAK para o processo executando COPY.

Veja também: BREAK

TIME

Formato: TIME [EDIT]

Gabarito: EDIT/S

Propósito: Acertar o relógio do sistema.

Path: SYS:Prefs/Time

Especificação: O comando TIME, sem argumentos ou seguido do argumento EDIT, abre a janela do editor Time.

Exemplo:

TYPE

Formato: TYPE {<arquivo|padrão>} [TO <nome>] [OPT H|N] [HEX] [NUMBER]
Gabarito: FROM/A/M,TO/K,OPT/K,HEX/S,NUMBER/S
Propósito: Exibir o conteúdo de um arquivo de texto.
Path: C:TYPE
Especificação: TYPE exibe o conteúdo do arquivo especificado na janela atual, ou então copia o seu conteúdo no arquivo TO, caso este tenha sido indicado. Enumerando-se mais de um nome de arquivo sem a palavra-chave TO, o comando TYPE exibirá o conteúdo de cada um dos arquivos em seqüência.

As opções OPT H e OPT N equivalem às palavras-chave HEX e NUMBER, respectivamente. A opção HEX faz com que o conteúdo do arquivo seja exibido em colunas de números hexadecimais, com o acréscimo de uma coluna contendo a interpretação dos caracteres em código ASCII. Esta opção é bastante útil para analisar arquivos objeto. A opção NUMBER numera as linhas à medida em que elas são exibidas.

Para fazer uma pausa na saída de TYPE, pressione a barra de espaço. As teclas Backspace, Return ou Ctrl-X dão continuidade à listagem. Para interromper a saída de TYPE, tecle Ctrl-C (a mensagem ***BREAK aparecerá na tela).

Exemplo:

```
1> TYPE DEVS:MountList
```

exibe na tela o conteúdo do arquivo MountList, localizado no diretório DEVS:.

UNALIAS

Formato: UNALIAS [<nome>]
Gabarito: NAME
Propósito: Remover um nome alternativo de comando (alias).
Path: Internal
Especificação: UNALIAS remove da lista de aliases o nome alternativo de comando (ou alias) especificado.

O comando ALIAS, sem argumentos, lista os aliases em uso.

Veja também: ALIAS

UNSET

Formato: UNSET [<nome>]

Gabarito: NAME
Propósito: Remover uma variável local.
Path: Internal
Especificação: UNSET remove da lista de variáveis do processo atual a variável local especificada .

O comando UNSET, sem argumentos, lista as variáveis locais em uso.

Veja também: SET

UNSETENV

Formato: UNSETENV [<nome>]
Gabarito: NAME
Propósito: Remover uma variável global.
Path: Internal
Especificação: UNSETENV remove da lista de variáveis em uso a variável global especificada .

O comando UNSETENV, sem argumentos, lista as variáveis globais em uso.

Veja também: SETENV

VERSION

Formato: VERSION [<biblioteca | dispositivo | arquivo>] [<versão #>] [<revisão #>] [<unidade #>]
Gabarito: NAME,VERSION,REVISION,UNIT,FILE/S,INTERNAL S,RES/S,FULL/S
Propósito: Localizar os números de versão e revisão do software.
Path: C:VERSION
Especificação: VERSION localiza o número da versão e revisão de uma biblioteca, dispositivo, comando ou disco do Workbench. O comando VERSION também pode ser usado para testar a existência de uma versão/revisão específica, ressetando o flag de condição caso o número da mesma seja igual ou maior que o procurado. Este dado poderá ser utilizado para comparações em scripts.

O comando VERSION, sem os argumentos <biblioteca | dispositivo | arquivo>, exibirá os números das versões do Kickstart e do Workbench e setará as variáveis de ambiente. Especificando-se um nome, VERSION tentará "abrir" a biblioteca, dispositivo, drive ou arquivo, para em seguida obter a informação sobre o número da versão.

Quando o usuário solicitar a comparação de uma <versão #> (e eventualmente uma

<revisão #>), VERSION atribuirá ao flag de condição o valor 0 se o número da versão (e revisão) do Kickstart, biblioteca ou driver de dispositivo for igual ou maior que os valores especificados. Caso contrário, o flag de condição valerá 5 (WARN). Nenhuma comparação quanto ao número da revisão será feita se este parâmetro não for especificado na linha de comando.

A opção <unidade #> permite ao usuário indicar o número de uma unidade diferente de zero. Isto pode ser necessário para acessar dispositivos com mais de uma unidade.

Exemplos:

```
1> VERSION
Kickstart version 36.202 Workbench version 36.77

1> VERSION Prefs/Font
Prefs/Font version 36.191
```

WAIT

Formato: WAIT [<n>] [SEC|SECS] [MIN|MINS] [UNTIL <hora>]
Gabarito: /N,SEC=SECS/S,MIN=MINS/S,UNTIL/K
Propósito: Aguardar o intervalo especificado de tempo.
Path: C:WAIT
Especificação: WAIT é usado em seqüências de comandos e após um comando RUN, com o objetivo de fazer uma pausa durante um período de tempo, ou aguardar até uma hora especificada. O tempo de espera default é de 1 segundo.

O argumento <n> especifica os segundos de espera (ou minutos, caso a opção MINS seja usada).

A palavra-chave UNTIL determina que a espera se estenda até a hora indicada (sempre no formato HH:MM).

Exemplo:

```
1> WAIT 10 MINS
```

produz uma pausa de 10 minutos.

```
1> WAIT UNTIL 21:15
```

a pausa se estenderá até 9:15h da noite.

WBCONFIG

Formato: WBCONFIG [FROM <nome do arquivo> [EDIT] [SAVE] [USE]
Gabarito: FROM,EDIT/S,USE/S,SAVE/S

Propósito: Controlar a apresentação da janela de fundo e a opção de trazer as janelas para a frente clicando-as duas vezes com o mouse.

Path: SYS:Prefs/WBConfig

Especificação: O comando WBCONFIG, sem argumentos ou seguido do argumento EDIT, ativa o editor WBConfig. O argumento FROM permite especificar um arquivo a ser carregado. Este arquivo deverá ter sido previamente gravado com a opção Save As do menu do editor WBConfig. Caso você tenha gravado uma configuração especial do editor WBConfig no diretório Presets, por exemplo, a opção FROM poderá ser usada para carregar esse arquivo.

Se a chave USE for especificada, a janela do editor não se abrirá, mas a configuração gravada no arquivo FROM será utilizada. A chave SAVE, quando presente, determina que a configuração do arquivo FROM seja salva. A janela do editor não será aberta.

Exemplo:

```
1> WBCONFIG Prefs/Presets/Backdrop.on USE
```

carrega a configuração contida no arquivo Backdrop.on. Se o sistema for reinicializado, a configuração salva antes deste comando será carregada.

WBPATTERN

Formato: WBPATTERN [FROM <nome do arquivo>] [EDIT] [USE] [SAVE] [WORKBENCH] [WINDOW]

Gabarito: FROM,EDIT/S,USE/S,SAVE/S,WORKBENCH/S,WINDOW/S

Propósito: Criar padrões de fundo para o Workbench e as janelas.

Path: SYS:Prefs/WBPattern

Especificação: O comando WBPATTERN, sem nenhum argumento ou seguido do argumento EDIT, ativa o editor WBPattern.

O argumento FROM deverá ser usado em conjunto com pelo menos uma das duas chaves: WORKBENCH ou WINDOW (mais de uma chave pode ser enumerada). O arquivo FROM deverá ser um arquivo previamente gravado com o item Save As do menu Project do editor WBPattern. As chaves permitem que um padrão particular seja utilizado em qualquer área da tela. Mesmo que tenha sido gravado como configuração específica de uma determinada área da tela, o padrão contido no arquivo FROM poderá ser usado numa área diferente, bastando que o usuário especifique a chave apropriada. Por exemplo: se o arquivo FROM foi criado quando você gravou um padrão para o Workbench, o mesmo padrão poderá ser usado nas janelas bastando especificar a chave WINDOW após o nome do arquivo.

A opção USE fará com que o padrão seja carregado e utilizado na área indicada, tal

como se você tivesse ativado o editor WBPattern, selecionado o botão de rádio apropriado, criado o padrão e ativado o gadget Use. Para salvar esse padrão, basta especificar a opção SAVE.

Se você não especificar USE ou SAVE, a opção EDIT será assumida automaticamente, e o editor WBPattern será ativado. Indicando-se um arquivo FROM e uma chave WORKBENCH ou WINDOW, á janela do editor WBPattern se abrirá, o padrão contido no arquivo FROM será exibido e o botão de rádio será selecionado. Se nenhuma chave for especificada em conjunto com o arquivo FROM, o editor exibirá o último padrão utilizado.

Exemplos:

```
1> WBPATTERN FROM SYS:Prefs/Presets/Diamonds WORKBENCH SAVE
```

carrega e salva o padrão gravado no arquivo Diamonds como o padrão de fundo do Workbench.

```
1> WBPATTERN FROM SYS:Prefs/Presets/Dots WINDOWS
```

abre a janela do editor WBPattern. O padrão gravado no arquivo Dots aparecerá ampliado e o botão de rádio Windows será selecionado. O usuário deverá então selecionar o gadget Save, Use ou Cancel, a fim de executar a operação desejada (salvar ou utilizar o padrão, ou então cancelar a operação).

WHICH

Formato: WHICH <comando> [NORES] [RES] [ALL]
Gabarito: FILE/A,NORES/S,RES/S,ALL/S
Propósito: Localizar o item especificado no path de comandos.
Path: C:WHICH
Especificação: WHICH permite ao usuário localizar um comando, programa ou diretório, bastando indicar o nome do mesmo. Se o item especificado se encontrar no path de busca, WHICH exibirá o path completo até ele. WHICH listará os comandos residentes como RESIDENT e os comandos internos como INTERNAL.

Normalmente, WHICH efetua a busca na lista residente, no diretório corrente, nos paths do comando e no diretório C:. O flag de condição recebe o valor 5 (WARN) caso o item especificado não seja encontrado.

Especificando-se a opção NORES, a lista de comandos residentes não será verificada. A opção RES, por sua vez, determina que a procura seja realizada somente na lista residente.

A chave ALL faz com que a procura se estenda por todo o path de busca, mesmo que o item especificado já tenha sido encontrado e listado uma ou mais vezes. Este procedimento visa garantir que todas as versões de um comando ou programa sejam

localizadas, mas pode acabar produzindo múltiplas listagens de um mesmo comando, caso este comando seja acessado por mais de uma via (através do diretório C: e do diretório corrente, por exemplo).

Exemplos:

```
1> WHICH avail
C: avail

1> WHICH C:
Workbench2.0:C

1> WHICH alias
INTERNAL alias
```

WHY

Formato: WHY
Gabarito: (nenhum)
Propósito: Exibir uma mensagem de erro explicando a razão da falha do comando anterior.
Path: Internal
Especificação: Normalmente, quando um comando falha, é exibida uma breve mensagem de advertência na tela. Em geral, esta mensagem inclui o nome do arquivo (quando este está relacionado com o problema), mas não são fornecidos maiores detalhes.

Como exemplo, imagine que tenha surgido a mensagem "Can't open <nome do arquivo>" (*Impossível abrir <nome do arquivo>*). Isto pode acontecer por vários motivos – o AmigaDOS pode ser incapaz de localizar o arquivo, o arquivo pode ser do tipo errado, ou pode haver espaço insuficiente no disco ou na RAM para executar o comando.

Se a razão não parecer clara para você, use o comando WHY a fim de obter uma explicação mais completa.

Exemplo:

```
1> COPY DF0:
Bad arguments
(Argumento inválido)
1> WHY
Last command failed because required argument missing
(Erro no último comando por falta de argumento obrigatório)
```

A mensagem de WHY revela por que o comando COPY resultou num erro: o destino de COPY não foi especificado.

Comandos do AmigaDOS – Tabela de Referência Rápida

ADDBUFFERS	Adiciona buffers ao sistema de arquivo.
ADDMONITOR	Informa ao Amiga que um monitor não RGB foi ligado ao sistema.
ALIAS	Atribui ou exibe nomes alternativos de comandos.
ASK	Obtém uma resposta do usuário durante a execução de um arquivo script.
ASSIGN	Controla a atribuição de nomes de dispositivos lógicos aos diretórios do sistema de arquivo.
AUTOPOINT	Ativa automaticamente a janela sobre a qual o apontador for posicionado.
AVAIL	Reporta a quantidade de memória (Chip e Fast RAM) disponível.
BINDDRIVERS	Vincula drivers de dispositivos ao hardware.
BINDMONITOR	Associa nomes aos diferentes modos de vídeo.
BLANKER	Apaga a tela do monitor se nenhum dado for recebido dentro de um intervalo especificado de tempo.
BREAK	Seta os flags de atenção no processo especificado.
CALCULATOR	Ativa uma calculadora na tela.
CD	Seleciona, modifica ou exibe o diretório corrente.
CHANGETASKPRI	Muda a prioridade de um processo em execução.
CLOCK	Ativa um relógio na tela.
CMD	Redireciona a saída da impressora para um arquivo.
COLORS	Modifica as cores da tela da frente.
COPY	Copia arquivos ou diretórios.
CPU	Ativa ou desativa os caches da CPU, testa se determinado processador está instalado; carrega a imagem da ROM na memória rápida de 32 bits, seta um handler de acesso ilegal à memória, que enviará informações via porta serial a 9600 bauds se uma tarefa acessar a página zero (os 256 bytes mais baixos) ou a memória acima de 16M.
DATE	Exibe ou acerta a data e/ou hora do sistema.
DELETE	Deleta arquivos ou diretórios.
DIR	Exibe uma lista ordenada dos arquivos contidos em um diretório.
DISKCHANGE	Informa ao Amiga a troca de disco em um drive.
DISKCOPY	Copia o conteúdo de um arquivo para outro.
DISKDOCTOR	Tenta restaurar o conteúdo de um disco danificado.
DISPLAY	Exibe os gráficos gravados com o formato IFF ILBM.
ECHO	Exibe um string.
ED	Edita arquivos de texto (editor full-screen).

EDIT	Edita arquivos de texto processando o arquivo fonte seqüencialmente (editor de linhas).
ELSE	Especifica uma alternativa para uma declaração IF num arquivo script.
ENDCLI	Encerra um processo Shell.
ENDIF	Termina um bloco IF num arquivo script.
ENDSHELL	Encerra um processo Shell.
ENDSKIP	Termina um bloco SKIP num arquivo script.
EVAL	Resolve expressões simples.
EXCHANGE	Monitora e controla os programas Commodities Exchange.
EXECUTE	Executa um script com a substituição opcional de argumentos.
FAILAT	Determina que uma seqüência de comandos não seja executada se o programa devolver um código de retorno igual ou maior que o especificado.
FAULT	Exibe as mensagens correspondentes aos códigos de erro especificados.
FILENOTE	Anexa um comentário a um arquivo.
FIXFONTS	Atualiza os arquivos .font no diretório FONTS:.
FKEY	Atribui strings de texto às teclas de função.
FONT	Especifica as fontes utilizadas pelo sistema.
FORMAT	Formata um disco para uso com o Amiga.
GET	Obtém o valor de uma variável local.
GETENV	Obtém o valor de uma variável global.
GRAPHICDUMP	Imprime uma cópia da tela.
ICONEDIT	Edita o formato e o tipo dos ícones.
ICONTROL	Especifica os parâmetros utilizados pelo Workbench.
ICONX	Permite a execução de um arquivo script a partir de um ícone.
IF	Resolve operações condicionais em arquivos script.
IHELP	Habilita o teclado a executar certas operações características do mouse.
INFO	Fornece informações sobre um disco ou partição.
INITPRINTER	Inicializa a impressora para as opções de impressão especificadas nos editores Preferences.
INPUT	Especifica velocidades diferentes para o mouse e o teclado.
INSTALL	Grava o boot block em um disquete formatado, habilitando-o ou não a inicializar o sistema.
IPREFS	Comunica ao Workbench as informações contidas nos arquivos individuais dos editores Preferences.
JOIN	Concatena dois ou mais arquivos em um novo arquivo.
KEYSHOW	Exibe o mapa de configuração do teclado em uso.

LAB	Especifica um label num arquivo script.
LIST	Lista as informações especificadas sobre diretórios e arquivos.
LOADWB	Inicia a operação do Workbench.
LOCK	Atribui a um disco o status de proteção contra gravação.
MAKEDIR	Cria um novo diretório.
MAKELINK	Cria um link entre arquivos.
MEMACS	Cria e edita arquivos de texto.
MORE	Exibe o conteúdo de um arquivo ASCII.
MOUNT	Torna disponível um dispositivo conectado ao sistema.
NEWCLI	Inicia um novo processo Shell.
NEWSHELL	Abre uma nova janela interativa do Shell.
NOCAPSLOCK	Desabilita a tecla Caps Lock.
NOFASTMEM	Força o Amiga a utilizar somente o Chip RAM residente.
OVERSCAN	Modifica o tamanho das áreas de exibição de textos e gráficos.
PALETTE	Modifica as cores da tela do Workbench.
PATH	Controla a lista de diretórios que o Shell utiliza para localizar os comandos.
POINTER	Modifica o formato do ponteiro da tela.
PRINTER	Especifica uma impressora e opções de impressão.
PRINTERGFX	Especifica opções para a impressão de gráficos.
PRINTFILES	Envia arquivos para a impressora.
PROMPT	Modifica a mensagem de prompt do Shell atual.
PROTECT	Modifica os flags de proteção de um arquivo.
QUIT	Abandona um arquivo script com o código de retorno especificado.
RELABEL	Muda o nome de volume de um disco.
REMRAD	Remover o dispositivo recuperável ramdrive.device.
RENAME	Muda o nome de um arquivo ou diretório.
RESIDENT	Exibe e modifica a lista de comandos residentes.
RUN	Executa comandos como um processo de segundo plano.
SAY	Reproduz com sons de fala o conteúdo de uma frase ou arquivo.
SCREENMODE	Seleciona um modo de vídeo.
SEARCH	Localiza um string em um arquivo do diretório especificado.
SERIAL	Atribui as especificações para a comunicação através da porta serial.
SET	Cria uma variável local.
SETCLOCK	Acerta ou lê o relógio de tempo real alimentado a bateria.

SETDATE	Modifica a data e hora da criação ou última alteração de um arquivo.
SETENV	Cria uma variável global.
SETFONT	Muda as fontes do sistema.
SETMAP	Troca o mapa de configuração do teclado usado pelo Amiga.
SETPATCH	Instala patches da ROM no software do sistema.
SKIP	Salta um label durante a execução de arquivos script.
SORT	Ordena alfabeticamente as linhas de um arquivo.
STACK	Exibe ou ajusta o tamanho do stack do Shell atual.
STATUS	Exibe informações sobre os processos Shell/CLI.
TIME	Acerta o relógio do sistema.
TYPE	Exibe o conteúdo de um arquivo de texto.
UNALIAS	Remove um nome alternativo de comando (alias).
UNSET	Remove uma variável local.
UNSETENV	Remove uma variável global.
VERSION	Localiza os números de versão e revisão do software.
WAIT	Aguarda o intervalo especificado de tempo.
WBCONFIG	Controla a apresentação da janela de fundo e a opção de trazer as janelas para a frente clicando-as duas vezes com o mouse.
WBPATTERN	Cria padrões de fundo para o Workbench e as janelas.
WHICH	Localiza o item especificado no path de comandos.
WHY	Exibe uma mensagem de erro explicando a razão da falha do comando.

Capítulo 3

Mensagens de Erro do AmigaDOS

Este capítulo lista os erros que podem ocorrer no AmigaDOS, seguidos por suas causas prováveis e algumas sugestões para recuperação.

103 Not enough memory
Memória insuficiente.

Causa Provável:
O seu Amiga não tem memória suficiente para completar a execução do comando.

Sugestão para Recuperação:
Feche todas as janelas e aplicações desnecessárias e tente executar novamente o comando. Se isto não funcionar, experimente reinicializar o equipamento, pode ser que haja memória suficiente, mas que ela esteja fragmentada. Eventualmente, pode ser necessário instalar mais RAM no seu sistema.

104 Process table full
Tabela de processos cheia.

Causa Provável:
Existe um limite para o número de processos em andamento.

Sugestão para Recuperação:
Suspenda uma ou mais tarefas.

114 Bad template
Gabarito incorreto.

Causa Provável:
Linha de comando incorreta.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

115 Bad number
Número incorreto.

Causa Provável:
O programa esperava a introdução de um argumento numérico.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

116 Required argument missing
Falta argumento obrigatório.

Causa Provável:
Linha de comando incorreta.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

117 Argument after "=" missing
Falta argumento após "=".

Causa Provável:
Linha de comando incorreta.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

118 Too many arguments
Argumentos em excesso.

Causa Provável:
Linha de comando incorreta.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

119 Unmatched Quotes
Aspas não fecham.

Causa Provável:
Linha de comando incorreta.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

120 Argument line invalid or too long
Linha de argumentos inválida ou muito longa.

Causa Provável:
A linha de comando está incorreta ou contém argumentos em excesso.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

121 File is not executable
Arquivo não é executável.

Causa Provável:
O nome do comando foi digitado incorretamente, o arquivo pode não ser executável (script ou programa).

Sugestão para Recuperação:
Redigite o nome do arquivo e certifique-se de que se trata de um arquivo de programa. Lembre-se de que para a execução de um script, é necessário que o flag s do arquivo esteja setado, ou então que o comando EXECUTE seja utilizado.

122 Invalid resident library
Biblioteca residente inválida.

Causa Provável:
Você está tentando utilizar um comando com uma versão antiga do AmigaDOS (comandos da Versão 2.0 com a Versão 1.0, por exemplo).

Sugestão para Recuperação:
Reinicialize o sistema com a versão atual do AmigaDOS.

202 Object is in use
Objeto em uso.

Causa Provável:
O arquivo ou diretório especificado já está sendo utilizado por uma outra aplicação. Se uma aplicação estiver lendo um arquivo, nenhuma outra poderá gravar nesse arquivo e vice-versa.

Sugestão para Recuperação:
Suspenda a outra aplicação que está utilizando o arquivo ou diretório e execute novamente o comando.

203 Object already exists
Objeto já existente.

Causa Provável:
O nome que você especificou já pertence a um outro arquivo ou diretório.

Sugestão para Recuperação:
Use um outro nome, ou delete o arquivo ou diretório já existente e o substitua.

204 Directory not found
Diretório não encontrado.

Causa Provável:
O AmigaDOS não está conseguindo encontrar o diretório especificado. Você pode ter cometido um engano na grafia do nome ou um erro de digitação.

Sugestão para Recuperação:
Cheque o nome do diretório (use o comando DIR se necessário) e execute novamente o comando.

205 Object not found
Objeto não encontrado.

Causa Provável:
O AmigaDOS não está conseguindo encontrar o dispositivo ou o arquivo especificado. Você pode ter cometido um engano na grafia do nome ou um erro de digitação.

Sugestão para Recuperação:
Cheque o nome do arquivo (usando DIR) ou o nome do dispositivo (usando INFO) e execute novamente o comando.

206 Invalid window description
Descrição de janela inválida.

Causa Provável:
Este erro ocorre na especificação do tamanho das janelas do Shell, ED ou ICONX. Você pode ter indicado dimensões muito grandes ou pequenas demais, ou então ter omitido um argumento. Este erro também ocorre com o comando NEWSHELL se o usuário fornecer um nome de dispositivo que não seja uma janela.

Sugestão para Recuperação:
Corrija a especificação da janela e execute novamente o comando.

209 Packet request type unknown
Tipo de requisição de packet desconhecido.

Causa Provável:
Você solicitou ao processo de manipulação do dispositivo (device handler) uma operação que ele não é capaz de executar. Por exemplo: o processo de manipulação do console não pode renomear arquivos.

Sugestão para Recuperação:
Corrija o código de requisição passado para o processo de manipulação do dispositivo.

210 Object name invalid
Nome de objeto inválido.

Causa Provável:
Existe um caractere inválido no nome do arquivo ou o nome do arquivo é muito longo. Lembre-se de que os nomes dos arquivos não podem ter mais de 30 caracteres, nem conter caracteres de controle.

Sugestão para Recuperação:
Redigite o nome do arquivo, certificando-se de não utilizar caracteres inválidos nem exceder o tamanho máximo permitido.

211 Invalid object lock
Lock de objeto inválido.

Causa Provável:
Você utilizou algo que não é um lock válido.

Sugestão para Recuperação:
Verifique o seu programa e certifique-se de que ele passe apenas locks válidos para as chamadas do AmigaDOS.

212 Object not of required type
Objeto não é do tipo requerido.

Causa Provável:
Você pode ter especificado o nome de um arquivo para uma operação que requer o nome de um diretório, ou vice-versa.

Sugestão para Recuperação:
Consulte a documentação e verifique o formato correto do comando.

213 Disk is not validated
Disco não validado.

Causa Provável:
Se você acabou de inserir o disco, isto pode indicar que o processo de validação do disco está em andamento. Caso contrário, pode significar que o disco está danificado.

Sugestão para Recuperação:
Caso você tenha acabado de inserir o disco, aguarde até que o processo de validação se complete. Isto pode levar menos de um minuto para um disquete, ou vários minutos para um disco rígido. Se estiver danificado, o disco não poderá ser validado. Neste caso, tente recuperar os arquivos do disco (utilizando DISKDOCTOR, por exemplo) e copie-os em um outro disquete.

214 Disk is write-protected
Disco protegido contra gravação.

Causa Provável:
A pastilha de plástico do disquete está na posição de proteção contra gravação.

Sugestão para Recuperação:
Se você tiver certeza de que deseja gravar neste disco, retire-o do drive, mova a pastilha e insira o disquete novamente no drive. Caso contrário, use um outro disco.

215 Rename across devices attempted
Tentativa de renomeação através de dispositivos.

Causa Provável:
O comando RENAME é capaz de modificar somente o nome de arquivos que estejam no mesmo volume. Você pode usar RENAME para mover arquivos de um diretório para outro, mas não de um volume para outro.

Sugestão para Recuperação:
Use COPY para copiar o arquivo no volume destino. Delete o arquivo do volume original, se você desejar, e então use RENAME.

216 Directory not empty
Diretório não está vazio.

Causa Provável:
Este erro ocorre quando se tenta deletar um diretório que contém arquivos ou subdiretórios.

Sugestão para Recuperação:
Se você tiver certeza de que deseja deletar o diretório completo, use a opção ALL do comando DELETE.

217 Too many levels
Níveis em excesso.

Causa Provável:
Você excedeu o limite de 15 soft links.

Sugestão para Recuperação:
Reduza o número de soft links.

218 Device (or volume) not mounted
Dispositivo ou volume não montado.

Causa Provável:
Se o dispositivo for um disquete, indica que o mesmo não está inserido no drive. Para outros dispositivos, significa que estes não foram montados com MOUNT. É também possível que você tenha cometido um erro de digitação ao especificar o nome do dispositivo.

Sugestão para Recuperação:
Insira o disquete correto no drive, verifique a digitação do nome do dispositivo, monte o dispositivo, ou revise o seu arquivo MountList.

219 Seek error
Erro na chamada SEEK.

Causa Provável:
Você tentou chamar SEEK com argumentos inválidos.

Sugestão para Recuperação:
Certifique-se de que a chamada SEEK seja posicionada dentro do arquivo. Você não pode executar SEEK fora dos limites do arquivo.

220 Comment is too long
Comentário muito longo.

Causa Provável:
O comentário que você anexou ao arquivo excede o número máximo de caracteres (79).

Sugestão para Recuperação:
Escreva um comentário mais curto.

221 Disk is full
Disco cheio.

Causa Provável:
Não existe espaço suficiente no disquete para completar a operação requisitada.

Sugestão para Recuperação:
Delete alguns arquivos ou diretórios desnecessários, ou então use um outro disco.

222 Object is protected from deletion
Objeto protegido contra apagamento.

Causa Provável:
O flag de proteção d do arquivo ou diretório está ressetado, impedindo que o mesmo seja deletado.

Sugestão para Recuperação:
Se você tiver certeza de que deseja deletar o arquivo ou diretório, use PROTECT para setar o flag d ou então use a opção FORCE do comando DELETE.

223 File is write protected
Arquivo protegido contra gravação.

Causa Provável:
O flag de proteção w do arquivo está ressetado, impedindo que seja feita uma gravação sobre este arquivo.

Sugestão para Recuperação:
Se você tiver certeza de que deseja sobrepor o conteúdo do arquivo, use PROTECT para setar o flag w.

224 File is read protected
Arquivo protegido contra leitura.

Causa Provável:
O flag de proteção r (contra leitura) do arquivo está ressetado.

Sugestão para Recuperação:
Use PROTECT para setar o flag r do arquivo.

225 Not a valid DOS disk
Disco inválido.

Causa Provável:
O disco inserido no drive não é um disco do AmigaDOS, o disco não foi formatado, ou então o disco está danificado.

Sugestão para Recuperação:
Certifique-se de estar utilizando o disco correto. Se você tiver certeza de que o disco estava funcionando antes, use DISKDOCTOR ou outro utilitário similar para tentar recuperar os arquivos. Se o disco não estiver formatado, use FORMAT para fazê-lo.

226 No disk in drive
Disco não está no drive.

Causa Provável:
O disco não está corretamente inserido no drive especificado.

Sugestão para Recuperação:
Insira o disco apropriado no drive especificado.

232 No more entries in the directory
Não há mais entradas no diretório.

Causa Provável:
Isto indica que a rotina EXNEXT do AmigaDOS não consegue encontrar mais nenhum registro no diretório que você está examinando.

Sugestão para Recuperação:
Não faça mais nenhuma chamada a EXNEXT.

233 Object is soft link
Objeto é um soft link.

Causa Provável:
Você tentou executar uma operação num soft link que deveria ser executada somente em um arquivo ou diretório.

Sugestão para Recuperação:
O AmigaDOS utiliza o packet Action_Read_Link a fim de resolver o soft link e depois tenta executar novamente a operação.

Capítulo 4

Glossário

Apontador de caracteres

Apontador para a extremidade esquerda de uma janela de linhas no editor de textos EDIT, utilizado para indicar a parte da linha que EDIT deverá alterar.

Argumentos

Informações adicionais fornecidas aos comandos.

Arquivo

Um agrupamento de dados relacionados.

Arquivo destino

O arquivo onde os dados serão gravados, durante uma operação de cópia.

Arquivo fonte

O arquivo de onde esses dados serão lidos.

Arquivo seqüencial

Um arquivo que pode ser acessado em qualquer ponto, partindo-se do início e percorrendo-se seqüencialmente o arquivo até alcançar o ponto desejado.

Bloco

Em termos genéricos corresponde, no jargão do Amiga, ao setor de um disco. Cada setor consiste num agrupamento de 512 bytes de dados que são gravados e lidos em conjunto.

Boot

O procedimento pelo qual se dá a partida no computador. Vem da expressão em inglês "*pulling yourself up by the bootstraps*", cuja tradução aproximada é "levantar-se pelos cordões dos sapatos".

Botão de rádio

Objeto existente na interface do Amiga com o usuário que se comporta como um botão de rádio. A função relacionada com o botão se torna ativa quando o mesmo é pressionado, o que é indicado por uma representação gráfica na tela. Apenas um botão num determinado grupo pode ser selecionado de cada vez.

Caracteres delimitadores

Caracteres empregados no início e no fim de um **string de caracteres**.

Chip RAM

Parte da memória RAM reservada para o armazenamento de dados relativos ao processamento de gráficos e sons.

Código objeto

Código binário gerado por um compilador ou assembler, ou ainda, a entrada binária de um linker.

Comando

Qualquer instrução que o usuário dá diretamente ao computador.

Console

Dispositivo de entrada que serve de interface entre a máquina e o usuário. Em geral se refere ao teclado e vídeo.

Default

Valor inicial de um parâmetro, ou em outras palavras, o valor ou atributo assumido se você não especificar nenhum outro.

Diretório

Um agrupamento de **arquivos**.

Diretório corrente

Pode ser tanto o **diretório raiz** como o último **diretório** selecionado com o comando CD.

Diretório raiz

O nível mais alto do sistema de arquivo. Os **arquivos** e **diretórios** do diretório raiz têm os seus nomes precedidos por dois pontos (:).

Disco de sistema

Um disco contendo o Workbench e os comandos do AmigaDOS.

Dispositivo lógico

Nome que o usuário dá a um diretório com o comando ASSIGN, e que depois pode ser usado como o nome de um dispositivo.

Drive corrente

A unidade de disco declarada corrente. O default é SYS:.

Exec

Abreviatura de Executive. Parte do sistema operacional do Amiga encarregada de decidir a próxima ação a ser executada pelo microprocessador. O Exec lida com funções básicas como a alocação de memória para os programas, além de gerenciar o recurso de multitarefa.

Fast RAM

A parte da memória RAM destinada ao armazenamento de dados e programas.

FFS

Abreviatura de Fast Filing System, o sistema utilizado a partir da versão 1.3 do AmigaDOS para armazenar dados nos discos. Neste sistema os blocos (ou setores) são organizados de forma contígua no disco, e cada bloco contém 512 bytes de informações do usuário. *Veja* **OFS**.

Fila de saída

Buffer na memória que retém os dados a serem gravados num **arquivo**.

Filehandle

Valor interno do AmigaDOS que representa um dispositivo ou um arquivo aberto.

Filesystem

Um **processo** do AmigaDOS que é capaz tanto de manipular dispositivos de I/O como efetuar operações com arquivos e diretórios. *Veja* **Handler**.

Gabarito do comando

Método usado para definir a **sintaxe** do comando.

Gadgets

Pequenos símbolos gráficos que aparecem nas bordas das janelas ou no interior de menus. Apontando para os gadgets com o mouse, o usuário pode abrir e fechar as janelas, movê-las, ou modificar o seu tamanho.

Handler

Um **processo** do AmigaDOS encarregado da manipulação de um dispositivo de I/O. Os handlers são um subconjunto dos filesystems, podendo existir vários handlers para um mesmo dispositivo. Os handlers e filesystems estão localizados no ROM ou no diretório L: do Amiga. *Veja* **Filesystem**.

Ícones

Pequenas figuras exibidas na tela com o objetivo de representar graficamente discos, programas e agrupamentos de dados.

IFF

Abreviatura de Interchangeable File Format. Refere-se aos arquivos de dados gráficos intercambiáveis, isto é, arquivos que podem ser carregados e manipulados por mais de um editor gráfico. A sigla **ILBM** (Interleaved Bitmap) designa um sub formato IFF no qual as imagens gráficas são armazenadas.

Interface de linhas de comando (CLI)

Um **processo** que decodifica as instruções do usuário.

Intuition

A parte do sistema operacional do Amiga encarregada da manipulação de janelas, menus, etc.

Lock

Estrutura interna utilizada pelo AmigaDOS nas operações de leitura e gravação de arquivos com o objetivo de evitar que um arquivo seja manipulado por mais de um processo ao mesmo tempo. Os locks também possibilitam a identificação de um arquivo específico.

Multiprocessamento

Execução de dois ou mais **processos** em paralelo, isto é, a um só tempo.

Nome do arquivo

O nome dado a um arquivo com o propósito de permitir a sua identificação.

Nome do dispositivo

O nome exclusivo dado a um dispositivo. Por exemplo: DF0: = drive 0.

Nome do volume

O nome exclusivo associado a um disco.

OFS

Abreviatura de Old Filing System, o sistema utilizado até a versão 1.2 do AmigaDOS para armazenar dados nos discos. Neste sistema apenas 448 bytes de dados do usuário são gravados em um setor, embora todos os setores sejam constituídos de 512 bytes. Informações do sistema ocupam os bytes restantes. *Veja* **FFS**.

Palavra-chave

Argumento do comando que precisa ser declarado explicitamente.

Prioridade

A importância relativa de um **processo**.

Processo

Uma tarefa requisitada pelo usuário ou pelo sistema operacional.

Reinicializar

Dar novamente o **boot** no sistema.

Requester

Pequeno menu, geralmente apresentando uma mensagem e dois gadgets, que o sistema exibe automaticamente na tela ao executar uma operação que exija a intervenção do usuário.

Script

Arquivo contendo seqüências de comandos que o AmigaDOS é capaz de executar. Corresponde aos arquivos batch do MS-DOS.

Sintaxe

O formato, ou "regras de gramática", para escrever o nome de um comando.

Startup-sequence

Arquivo script contendo uma seqüência de comandos que o AmigaDOS executa automaticamente toda vez que o usuário reinicializa o sistema. Equivale ao AUTOEXEC.BAT do MS-DOS.

Stream

Um arquivo aberto ou dispositivo associado com um filehandle. Como exemplo, o stream de entrada pode ser um arquivo, e o stream de saída, o dispositivo do console.

String de caracteres

Uma seqüência de caracteres que podem ser impressos.

String qualificado

Um string precedido por um ou mais qualificadores.

Terminal

Veja **Console**.

Parte II

MANUAL DO PROGRAMADOR

Capítulo 5

Programando no Amiga

Este capítulo introduz o leitor à programação em C e Assembler sob o AmigaDOS.

Introdução

Este manual assume que o leitor já tenha alguma familiaridade com o C ou com o Assembler, não tentando por esse motivo ensinar nenhuma dessas linguagens. Uma introdução ao C pode ser encontrada no livro *The C Programming Language*, de Brian Kernighan e Dennis M. Ritchie, publicado pela editora Prentice-Hall. Existem também vários livros sobre o Assembler do 68000, tais como *Programming the MC68000*, de Tim King e Brian Knight, publicado pela Addison-Wesley.

Desenvolvimento de Programas para o Amiga

Esta seção mostra como desenvolver programas para o Amiga. Ela revela os itens de que você deverá dispor antes de começar a programar, como chamar as rotinas do sistema, e como criar um arquivo que pode ser executado no Amiga.

Primeiros Passos

Antes de começar a escrever programas para o Amiga, você deverá dispor dos seguintes itens:

1. A documentação sobre o AmigaDOS e outras rotinas do sistema que podem ser chamadas pelo seu programa.
2. A documentação sobre a linguagem que você pretende utilizar. Se a sua intenção é trabalhar com o Assembler ou C, este manual mostrará como usar essas ferramentas, embora não contenha qualquer informação específica normalmente encontrada num manual de referência dessas linguagens.
3. Arquivos header contendo as definições de estrutura e os valores apropriados para chamar as rotinas do sistema do Amiga. O Commodore-Amiga fornece os

arquivos header como arquivos include tanto para o C (geralmente terminados em .h) como para o Assembler (terminados em .i). Para utilizar uma biblioteca residente específica, você deverá incluir um ou mais arquivos header contendo as definições adequadas. Para usar o AmigaDOS a partir do C, por exemplo, é necessário incluir (#include) o arquivo "dos.h".

4. Um compilador ou um editor assembler.
5. O linker do Amiga, bem como a biblioteca padrão do Amiga (amiga.lib), contendo funções, rotinas de interface e diversos valores absolutos.

Chamando as Bibliotecas Residentes

O usuário dispõe de duas alternativas para chamar rotinas do sistema a partir de um programa em Assembler. Quem programa em C precisa apenas chamar a função relevante de acordo com os parâmetros especificados. Para chamar uma rotina do sistema em Assembler, o procedimento usual é posicionar um ponteiro para a base da biblioteca residente no registrador A6, e então comandar um salto para o offset negativo apropriado. Os offsets estão disponíveis como externos absolutos na biblioteca do Amiga, com os nomes na forma _LVOnome. Assim, um exemplo de chamada típica seria JSR_LVOnome(A6), onde A6 teria sido carregado com o ponteiro para a base da biblioteca apropriada. Os ponteiros de bases podem ser obtidos através da chamada OpenLibrary( ) para Exec, você pode encontrar o ponteiro de base para Exec na posição 4 (a única posição absoluta usada no Amiga). Esta posição, também conhecida como AbsExecBase, é definida na biblioteca amiga.lib.

Você pode utilizar este procedimento para chamar certas bibliotecas residentes baseadas em RAM e a biblioteca do AmigaDOS, se for preciso (observe que a biblioteca do AmigaDOS se chama "dos.library"). Saiba ainda que não é obrigatório utilizar A6 como o ponteiro para a base da biblioteca, qualquer outro registrador pode ser usado, se necessário. Além disso, você pode chamar o AmigaDOS utilizando o recurso de chamada à biblioteca residente disponível no linker. Neste caso, basta programar um JSR para o entry point desejado e o linker notará o fato de que você usou uma referência a uma biblioteca residente. Quando o seu programa for carregado na memória, o loader abrirá automaticamente a biblioteca, e a fechará para você quando o programa for removido. O loader modifica automaticamente as referências aos entry points do AmigaDOS para que as mesmas se refiram ao offset correto a partir do ponteiro da base da biblioteca.

Criando um Programa Executável

Para produzir um arquivo executável no Amiga, siga os quatro passos a seguir:

1. Introduza o seu programa fonte no Amiga. Para fazer isso você pode digitá-lo diretamente, utilizando um editor, ou então transferi-lo a partir de um outro computador.
2. Monte ou compile o seu programa.
3. Faça a linkagem das diversas partes do programa, incluindo as rotinas de inicialização e as bibliotecas do Amiga (ou quaisquer outras) necessárias para satisfazer as referências externas do mesmo.
4. Carregue o seu programa no Amiga e... veja-o em ação!

Rodando um Programa Sob o CLI

Existem duas alternativas para se rodar um programa: a primeira é executá-lo no ambiente do CLI (também conhecido como Shell), a segunda é executá-lo sob o Workbench. Esta seção descreve a primeira maneira.

Executar um programa sob o CLI (Shell) pode se assemelhar, em certos aspectos, a utilizar um antiquado sistema de teletipo (TTY) – embora você possa considerar o Shell útil, por exemplo, para portar o seu programa para o Amiga numa primeira fase do desenvolvimento do mesmo. Para carregar e introduzir o seu programa, basta digitar o nome do arquivo contendo o código binário, e eventualmente acrescentar mais alguns argumentos.

Ambiente Inicial em Assembler

Quando você carrega um programa no ambiente do Shell, normalmente é necessário digitar o nome do programa e um conjunto de argumentos. Pode-se também redirecionar as operações de entrada e saída por meio dos símbolos ">" e "<". O Shell fornece automaticamente todas essas informações para o programa quando este é executado.

Ao iniciar um programa, o Shell aloca um stack para o mesmo. Este stack tem a princípio um tamanho de 4.000 bytes, que pode ser aumentado ou reduzido com o comando STACK. O AmigaDOS obtém este stack a partir da pilha de memória livre antes da execução do programa (note que o stack do programa não é o mesmo stack utilizado pelos processos do Shell). O AmigaDOS introduz um endereço de retorno apropriado no stack, o qual permite ao Shell retomar o controle e remover o programa. Abaixo disso no stack, em 4(SP), está o tamanho do stack em bytes, o que pode ser útil caso você deseje verificar algum dado contido no stack.

No início do seu programa, o registrador A0 aponta para os argumentos digitados pelo usuário. O AmigaDOS armazena a linha com os argumentos na memória, dentro do stack do CLI, sendo que este ponteiro permanece válido durante toda a execução do programa. O registrador D0 indica o número de caracteres da linha de argumentos. Você pode usar estes valores iniciais para decodificar a linha de argumentos e identificar a operação requisitada pelo usuário. Lembre-se de que todos os registradores podem ser alterados por um programa do usuário.

Para obter os filehandles de entrada e saída, você deverá chamar as rotinas Input( ) e Output( ) do AmigaDOS (lembre-se de abrir a biblioteca do AmigaDOS antes de fazer isto). Estas chamadas retornam os filehandles que irão referenciar as operações de entrada e saída requisitadas pelo usuário. Este padrão de entrada/saída será o definitivo, a menos que o I/O tenha sido redirecionado por meio da inclusão dos símbolos ">" ou "<" na linha de argumentos. Não é preciso fechar esses filehandles em seu programa. Assim como o CLI os abriu para você, ele também os fechará quando for necessário.

Ambiente Inicial em C

Programadores em C devem observar que a rotina de inicialização (startup) seja

sempre o primeiro item na entrada do linker. Isto significa que o linker entrará no seu programa no entry point da rotina de startup. Esta rotina vasculha a lista de argumentos, tornando disponíveis os argumentos em "argv" e o número de argumentos am "argc". Ela também abre a biblioteca do AmigaDOS e chama Input( ) e Output( ) para você, colocando os filehandles resultantes em "stdin" e "stdout". Finalmente, esta rotina chama a função "main" do C.

Falhas de Execução das Rotinas

A maioria das rotinas do AmigaDOS retornam um zero caso sua execução não tenha sido bem sucedida. As exceções são as chamadas Read e Write, que retornam -1 em caso de erro. Você pode chamar IoErr( ) a fim de obter maiores informações sobre o que pode ter ocasionado a falha das chamadas. A função IoErr( ) retorna um inteiro que corresponde ao código do erro, permitindo que você decida pela ação apropriada em função da causa do mesmo. Uma lista completa dos códigos e mensagens de erro é apresentada no final do Capítulo 2 deste manual.

Terminando um Programa

Para sair de um programa, em geral é suficiente comandar um RTS utilizando o stack pointer (SP) inicial. Neste caso, você deve também fornecer um código de retorno no registrador D0. Este código deverá ser zero se a execução do programa tiver sido bem sucedida, ou um número positivo em caso contrário. Quando você retornar um número diferente de zero, o CLI notará a ocorrência de um erro. Dependendo do limite do código de erro (atribuído pelo comando FAILAT), um CLI não interativo – como o que estiver executando um script de comandos através do comando EXECUTE – será encerrado. Um programa escrito em C pode simplesmente retornar de "main", o que devolve o controle à rotina de startup. Isto limpará D0 e executará um RTS.

Alternativamente, um programa em Assembler pode chamar a função Exit( ) do AmigaDOS, que utiliza o código de retorno como um argumento. Isto permite abandonar o programa em qualquer ponto de sua execução, não importando qual seja o valor do stack pointer nesse instante. Um programa em C deverá chamar a rotina de saída fornecida pelo fabricante do compilador. Esta rotina é tipicamente conhecida como exit( ), note o "e" minúsculo.

É importante ressaltar que o AmigaDOS não controla nenhum dos recursos empregados em um programa, esta tarefa é de competência exclusiva do programador. Assim, quaisquer arquivos abertos pelo usuário deverão ser fechados quando o programa terminar. Da mesma maneira, todos os locks obtidos deverão ser liberados, qualquer rotina carregada deverá ser removida, e toda a memória alocada pelo programa deverá ser retornada para o sistema. Naturalmente, nem sempre todas essas providências serão necessárias, É perfeitamente aceitável, por exemplo, que você escreva um programa para carregar na memória uma rotina para uso posterior. O importante é que você disponha de algum mecanismo que possibilite eventualmente efetuar o retorno de memória, locks de arquivos, etc.

Rodando um Programa Sob o Workbench

Para compreender a execução de programas sob o Workbench, é preciso considerar as diferentes maneiras através das quais um programa pode ser executado no Amiga. Sob o CLI, o programa do usuário é executado como parte do processo CLI. Desse modo, ele está habilitado a herdar os streams de I/O, e outras informações como os argumentos fornecidos ao Shell.

Sob o Workbench, entretanto, o AmigaDOS inicia o seu programa como um novo processo, executado ao mesmo tempo que o Workbench. O Workbench carregará o programa e então enviará uma mensagem para iniciar a execução do mesmo. É necessário esperar por essa mensagem inicial antes de poder começar a fazer qualquer coisa. Você deverá guardar essa mensagem e retorná-la ao Workbench quando o programa terminar, a fim de que o Workbench possa remover o código do seu programa. Nos programas em C isto é feito utilizando-se uma parte diferente da rotina de startup. Já os programadores Assembler terão que desenvolver suas próprias rotinas para esta finalidade.

Note ainda que um programa rodando como um novo processo iniciado pelo Workbench não possui streams de entrada e saída defaults. Você terá de abrir todos os canais de I/O de que o seu programa necessitar, e certificar-se de fechá-los quando o programa terminar.

Fundamentos da Programação de I/O

Esta seção descreve os fundamentos das funções de I/O da biblioteca dos.library. Muitos compiladores C fornecem suas próprias funções de I/O, que diferem das rotinas da dos.library aqui apresentadas. Consulte o manual do fabricante para maiores informações sobre as funções padronizadas de I/O disponíveis no seu compilador.

A biblioteca dos.library original foi escrita em BCPL, uma linguagem de programação precursora do C. Embora na versão 2.0 do AmigaDOS a dos.library tenha sido reescrita em C e Assembler, alguns resquícios do BCPL continuaram presentes, a fim de assegurar a compatibilidade com as antigas versões da biblioteca. Um dos elementos remanescentes é o ponteiro BCPL, ou BPTR.

O BCPL pensa apenas em termos de palavras longas de 32 bits. Para se referir a um endereço individual de memória, o BCPL utiliza uma quantidade de quatro bytes de tamanho, ao invés de apenas um byte, como faz a CPU. Assim, para se referir ao endereço de memória 2, o BCPL pensa no segundo (após o 0 e o 1º) conjunto de 4 bytes – os quais a CPU "enxerga" como os endereços de 8 a 11.

Como o AmigaDOS utiliza BPTRs, a programação em certas áreas da memória no AmigaDOS exige a conversão de endereços normais em BPTRs e vice-versa. Para converter um endereço normal (que deverá consistir numa palavra longa alinhada) em um BPTR, basta dividi-lo por 4 (>>2). O arquivo include "dos.h" contém macros para efetuar automaticamente a conversão entre os dois formatos de endereços. Observe que como o BPTR se refere a uma palavra longa (um endereço de 32 bits),

qualquer item por ele endereçado deverá estar no formato de uma palavra longa alinhada.

Outro elemento remanescente é o string BCPL, ou BSTR. Um BSTR consiste num BTPR para um string estilo BCPL. O primeiro byte de um string BCPL contém o tamanho do string, e os bytes restantes correspondem aos caracteres do string propriamente dito.

Um dos recursos básicos do AmigaDOS é a entrada e saída de arquivos. Para executar o I/O de arquivos, o DOS requer o que se chama de *filehandle*. Um filehandle é uma estrutura que o DOS utiliza para manter o controle de um arquivo aberto.

Outra estrutura usada pelo DOS na leitura e gravação de arquivos é o *lock*. O AmigaDOS utiliza um lock para "bloquear" um arquivo, ou prevenir que dois ou mais processos tentem manipular esse arquivo ao mesmo tempo. Quando um lock está ativo sobre um determinado arquivo, os outros processos ficam impedidos de modificá-lo.

Existem dois tipos de locks: os exclusivos e os compartilhados. Os locks compartilhados são usados para garantir o acesso a um determinado arquivo somente para leitura. Podem existir vários locks compartilhados sobre um mesmo arquivo de uma só vez. O propósito de um lock compartilhado é evitar que um processo tente gravar no arquivo que está sendo lido por um outro processo. O lock exclusivo, por sua vez, destina-se a permitir o acesso a esse arquivo para a gravação de dados. Enquanto existirem locks num arquivo (exclusivos ou compartilhados), não é permitido criar um novo lock exclusivo nesse arquivo.

Para abrir um arquivo (e obter um filehandle para ele), use a função Open( ) da biblioteca dos.library:

```
BPTR myfilehandle = Open( UBYTE *filename, LONG accessMode );
```

onde *myfilehandle* é um BPTR para uma estrutura FileHandle, *filename* é um string C especificando o nome do arquivo a ser aberto (relativo ao diretório corrente), e *accessMode* é uma das opções: MODE_OLDFILE, MODE_NEWFILE ou MODE_READWRITE.

MODE_OLDFILE abre um arquivo existente para leitura ou gravação. Neste modo, o DOS cria um lock de leitura no arquivo (que muda para um lock de gravação quando você tenta gravar nesse arquivo). Qualquer tentativa de abrir um arquivo não existente nesse modo causará um erro. Nesta circunstância, Open( ) retornará um código NULL. MODE_NEWFILE, por sua vez, abre um arquivo para leitura e gravação e deleta o arquivo se ele já existir. Neste modo o DOS cria um lock exclusivo no arquivo. Já MODE_READWRITE, um novo modo introduzido na versão 2.0 do AmigaDOS, abre um arquivo (com um lock compartilhado) para leitura e gravação, mas não deleta o arquivo se ele já existir.

As funções Read( ) e Write( ) são usadas para ler e gravar blocos de dados contidos em um buffer:

```
LONG actualcount = Read( BPTR filehandle, APTR buffer,
LONG lenght );
LONG actualcount = Write( BPTR filehandle, APTR buffer,
```

```
LONG lenght );
```

Normalmente estas funções são utililizadas para ler e gravar blocos de dados de grande extensão, não sendo muito eficientes para pequenas operações de gravação e leitura. A versão 2.0 introduziu rotinas de leitura e gravação bufferizadas muito mais eficazes para operações com pequenos blocos de dados, estas rotinas serão discutidas mais adiante.

As chamadas Read( ) e Write( ) requerem os mesmos argumentos: o filehandle para o arquivo em questão, o buffer contendo os dados, e a indicação do número de bytes a serem lidos ou gravados. Ambas as funções retornam o número de bytes efetivamente gravados ou lidos. Se Read( ) retornar um zero, então nenhum caractere terá sido lido e a posição atual do arquivo será o final do mesmo. Se qualquer uma das funções retornar um -1, é sinal de que algum erro ocorreu (você pode usar a função IoErr( ) da dos.library para obter o código do erro do DOS mais recente).

O DOS mantém um registro da posição atual de todos os arquivos abertos, incrementando esta posição a cada byte lido ou gravado. As operações de Read( ) e Write( ) são relativas à posição atual, portanto, se você abrir um arquivo já existente e desejar gravar dados adicionais nele, certifique-se de que a posição atual esteja indicando o final desse arquivo. Quando Open( ) abre um arquivo pela primeira vez, a posição atual é situada no início do arquivo.

Para mover a posição atual de um arquivo, basta empregar a função Seek( ):

```
LONG oldposition = Seek( BPTR filehandle, LONG fileposition, LONG
offset_from_where );
```

O campo *fileposition* corresponde à nova posição do arquivo, que deverá ser relativa a um offset. O *offset_from_where* deverá ser um dos seguintes:

OFFSET_END	a nova posição será relativa ao final do arquivo (*fileposition* deverá ser zero, neste caso).
OFFSET_CURRENT	a nova posição será relativa à posição atual do arquivo.
OFFSET_BEGINNING	a nova posição será relativa ao início do arquivo.

Seek( ) retorna a posição inicial do arquivo, ou -1 para indicar a ocorrência de um erro.

Para fechar um filehandle aberto, use a função Close( ) da dos.library:

```
LONG returnvalue = Close ( BPTR file );
```

Na versão 2.0 do AmigaDOS, Close( ) retorna DOSTRUE (-1) se a execução tiver sido bem sucedida ou DOSFALSE (0) em caso de erro. Antes da versão 2.0, o valor de retorno era indefinido.

O programa a seguir é um exemplo simples de I/O básico de arquivos.

```
;/* - Execute-me para compilar com o Lattice 5.10a
LC -bl -cfistq -v -y -j73 RW.c
```

```
Blink FROM LIB:C.o,RW.o TO RW LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/
/*   Este programa abre o arquivo "s:startup-sequence"
**   e o copia para "ram:qwe".
*/
#include <dos/dos.h>
#include <dos/dosextens.h>
#include <clib/dos_protos.h>
#include <clib/alib_stdio_protos.h>
/*   Para compilar com os #includes 1.3, substitua os
quatro #includes acima por:
#include <libraries/dos.h>
#include <libraries/dosextens.h>
*/
#ifdef LATTICE
int CXBRK(void) {return(0);} /* Desabilita manipulação do CTRL/C
pelo LATTICE */
int chkabort(void) {return(0);}
#endif
/*   Para a maioria das aplicações, o tamanho do buffer abaixo é muito pequeno.
**   Para aplicações em programação, este buffer deverá ser MUITO maior
**   a fim de tornar as operações de gravação e leitura mais eficientes.
*/
#define BUFSIZE 256
UBYTE *vers = "\0$VER: RW 1.0";
UBYTE buffer[BUFSIZE];
void main(void)
{
      BPTR myfile,startup;
      LONG count,actual;
      if (myfile = Open("ram.que",MODE_NEWFILE))
{
      if (startup = Open("a:startup-sequence",MODE_OLDFILE))
      {
          count = 1;
          /* continua a gravar até alcançar o final do arquivo
          ** ou até ocorrer um erro */
          while ((actual = Read(startup,buffer,BUFSIZE))&&(count>0)
          count = Write(myfile,buffer,actual);
          if(actual<0) printf("Erro durante a leitura\n");
          if(count<0) printf("Erro durante a gravação\n");
          Close(startup);
      }
      Close(myfile);
   }
}
```

Usando Handlers de Arquivo

Cada dispositivo do AmigaDOS possui um processo de manipulação associado que se chama *handler*. O processo de manipulação (ou handler) é utilizado pelo AmigaDOS para se comunicar com os dispositivos Exec. Cada handler é responsável pelo processamento do conjunto de comandos padrão que o AmigaDOS envia quando necessita utilizar um dispositivo. O AmigaDOS utiliza handlers para executar tarefas tais como escrever e gravar dados nos arquivos, escrever no console, e enviar a saída de dados para a impressora. Dispositivos como DH0:, RAM;, CON: e SER: são controlados com seus respectivos processos handlers.

Um tipo particular desses processos é o handler de arquivos. O handler de arquivos é usado para a manutenção do sistema de arquivo (isto é, os arquivos e diretórios) num determinado dispositivo. O processo de manipulação responsável por DF0: é um exemplo de handler de arquivos. Este handler permite que o AmigaDOS utilize o dispositivo Exec trackdisk.device. Quando você tenta ler um arquivo em DF0:, o DOS envia uma requisição de leitura para o processo de manipulação de DF0:. O handler de DF0: interpreta o comando de leitura e extrai os dados do arquivo em algum lugar do disco contido no drive DF0:. O DOS não precisa saber nada sobre como o dispositivo subjacente funciona, ele simplesmente solicita os dados ao handler de arquivo, e este se encarrega de fornecer as informações requisitadas. Este sistema torna relativamente simples o acréscimo de novos dispositivos ao AmigaDOS.

Explicar como desenvolver handlers está além do objetivo desta seção, porém vamos mostrar aqui como utilizá-los. O handler permite que você pense em várias formas de I/O como uma operação de I/O de arquivos. Por exemplo, utilizando-se a rotina Open( ), é possível abrir uma janela de console e escrever nela como se fosse um arquivo:

```
consoleFH = Open( "CON:20/20/500/100/Console", MODE_NEWFILE );
```

Isto abre uma janela de console na tela do Workbench. Você pode escrever e ler os dados diretamente nessa janela usando o filehandle retornado pela chamada Open( ) (lembre-se de que na leitura de dados do filehandle de uma janela CON:, o usuário deverá antes pressionar a tecla Return para que os dados possam ser lidos).

A todo processo está associado um filehandle padrão de entrada e um filehandle padrão de saída. Para os programas baseados no Shell, os handles padrões de I/O são normalmente a janela de console do Shell. É possível redirecionar o I/O padrão de um programa iniciado a partir do Shell utilzando-se os operadores de redirecionamento < (entrada) e > (saída). Desta maneira o usuário pode, por exemplo, redirecionar a saída de um programa baseado no Shell para um arquivo, ou para o dispositivo PRT: (impressora). As rotinas Input( ) e Output( ) da dos.library retornam os filehandles padrão atuais de entrada e de saída, respectivamente:

```
BPTR inputFH = Input( void );
BPTR outputFH = Output( void );
```

A versão 2.0 do AmigaDOS introduziu duas novas funções, SelectInput( ) e SelectOutput( ), as quais permitem que um programa modifique os handles padrão de I/O atuais:

```
oldinputFH = SelectInput( BPTR newinputFH );
oldoutputFH = SelectOutput( BPTR newoutputFH );
```

onde *newinputFH/nemoutputFH* é um filehandle aberto válido, e *oldinputFH/oldoutputFH* é o antigo handle padrão de entrada/saída. Preste atenção para não descartar o filehandle antigo, pois ele continua válido e eventualmente você precisará fechá-lo ou instalá-lo novamente.

I/O Bufferizado

A versão 2.0 do AmigaDOS introduziu uma série de rotinas de I/O bufferizado. Estas rotinas melhoram o desempenho de pequenas operações de leitura e gravação, reduzindo o tempo e trabalho envolvidos na gravação e leitura de blocos de dados de menor tamanho.

As funções de I/O bufferizado equivalentes a Read( ) e Write( ) são:

```
LONG actualblocks = FRead( BPTR fh, APTR buffer, ULONG
blocklenght, ULONG numblocks );
LONG actualblocks = FWrite ( BPTR fh, APTR buffer, ULONG
blocklenght, ULONG numblock );
```

Estas duas funções são similares a suas equivalentes não bufferizadas, mas seus argumentos são ligeiramente diferentes. Em vez de requerer o número de bytes a serem lidos ou gravados, FRead( ) e FWrite( ) irão ler/gravar um número especificado de blocos (*numnblock*). O tamanho de cada bloco é definido por *blocklenght.* Estas funções retornam o número de bytes efetivamente lidos/gravados (ou zero se for lido um EOF). Se houver algum erro, as funções retornarão o número de bytes lidos ou gravados, mas este número será diferente do número de blocos requisitado.

Sempre que você intercalar o uso de rotinas de I/O bufferizadas e não bufferizadas, lembre-se de limpar o buffer de arquivos com a rotina Flush( ):

```
void Flush( BPTR fh );
```

Normalmente, o DOS limpa o buffer quando ele está cheio ou quando alguém insere um \n, \0, \r ou \12. Ao fazer uso de I/O bufferizado no seu próprio filehandle de entrada, você deverá chamar Flush( ) antes de proceder à leitura de dados.

Existem ainda algumas rotinas para leitura e gravação bufferizada de caracteres simples e de strings:

```
LONG FGetC( BPTR fh );
UBYTE *FGets( BPTR fh, UBYTE *buf, ULONG buflen );
LONG FPuts( BPTR fh, UBYTE *str );
void FPutC( BPTR fh, ULONG ch );
LONG WriteChars( UBYTE *buf, unsigned long buflen );
LONG PutStr( UBYTE *str );
LONG UnGetC( BPTR fh, long character );
Veja o Capítulo 6 para obter maiores informações sobre estas e
outras funções relacionadas.
```

Existem também algumas funções de I/O bufferizado destinadas a gravar dados formatados num filehandle:

```
LONG VFPrintf( BPTR fh, UBYTE *formatstring, LONG *argarray );
LONG VPrintF( UBYTE *formatstring, LONG *argarray );
```

onde *formatstring* é um string C contendo um gabarito de formatação no estilo pritf( ), com as seguintes opções %:

```
%[flags][largura.limite][tamanho]tipo
```

flags – apenas uma é permitida. '-' especifica justificação à esquerda.

largura – largura do campo. Se o primeiro caractere for um '0', o campo será preenchido com zeros à esquerda.
– deverá seguir a largura do campo, se especificado

limite – número máximo de caracteres da string que serão exibidos (válido somente com a opção %s).

tamanho – tamanho default dos dados de entrada para WORD (tipos d, x e c); "l" converte este parâmetro para palavra longa (32 bits).

tipo – os tipos suportados são:

- b – BSTR: o dado é um BPTR de 32 bits para uma contagem de bytes seguida por um string de bytes, ou um string de bytes terminado por NULL. Um BPTR NULL é tratado como um string vazio (a partir do Exec V36).
- d – decimal.
- x – hexadecimal.
- s – string: um ponteiro de 32 bits para um string de bytes terminado por NULL. A partir da V36, um ponteiro para NULL é tratado como um string vazio.
- c – caractere.

O *argarray* é um ponteiro para o array de argumentos correspondente às entradas no gabarito de formatação.

A única diferença entre estas duas funções é que VPrintf( ) escreve no filehandle de saída atual, enquanto que VFPrintf( ) escreve no filehandle de saída que você fornecer. A vantagem de usar uma destas funções no lugar de uma função como printf( ) é que printf( ) terá que ser linkado (a partir da biblioteca amiga.lib), o que aumenta o tamanho do programa. Como VPrintF( ) e VFPrintf( ) já se encontram nas ROMs, a utilização dessas funções evita a linkagem de código adicional.

Análise de Sintaxe da Linha de Comandos

A versão 2.0 do AmigaDOS introduziu a análise de sintaxe (parsing) da linha de comandos. A rotina ReadArgs( ) é o coração desse recurso:

```
struct RDArgs *rda = ReadArgs( UBYTE *argtemplate, LONG
*argarray, struct RDArgs *myrda );
```

O primeiro argumento, *argtemplate*, é um string em estilo C que descreve opções de programa definíveis a partir da linha de comandos. Cada opção deverá consistir num nome descritivo por extenso ("Quick", por exemplo, e não "Q"). Cada opção pode ser precedida por uma abreviação na forma "abrev=opção" (ou seja, "Q"="Quick"). As opções de *argtemplate* deverão ser delimitadas por vírgulas. Cada opção também pode ser seguida por modificadores que especificam as características de opções individuais. Os modificadores válidos são:

/S–Switch. É tratado como uma variável booleana que será setada se o nome da opção aparecer na linha de comandos. A entrada é booleana (0 para não setado, diferente de zero para setado).

/K–Palavra-chave. Isto significa que a opção não será preenchida a menos que a palavra-chave (*keyword*) apareça. Por exemplo: se o gabarito for "Name/K", a menos que "Name=<string>" ou "Name <string>" apareça na linha de comando, a opção Name não será preenchida.

/N–Número. O parâmetro é considerado um número decimal e será convertido por ReadArgs( ). Caso um número inválido seja especificado, a execução de ReadArgs( ) não terá sucesso. A entrada deverá ser um ponteiro para um número em forma de palavra longa ou NULL (é desse modo que você sabe se um número foi especificado).

/T–Toggle. É similar a um switch, só que alterna o valor booleano.

/A–Obrigatório. Esta palavra-chave determina que a execução de ReadArgs( ) seja abortada se a opção correspondente não for especificada na linha de comandos.

/F–Restante da linha. Quando especificado, faz com que todo o restante da linha de comandos seja tomado como o parâmetro da opção, mesmo que outras palavras-chave apareçam no restante da linha.

/M–Múltiplo. Significa que a opção utilizará qualquer número de argumentos, retornando-os como um array de ponteiros. Quaisquer argumentos que não sejam considerados parte de outra opção serão incluídos nesta opção. Apenas um /M deverá aparecer em um gabarito. Por exemplo: num gabarito "Dir/M,All/S" a linha de comando "foo bar all qwe" setará o booleano "all" e retornará um array consistindo de "foo", "bar" e "qwe". A entrada no array será um ponteiro para um array de ponteiros string, sendo que o último deles será um NULL.

Vale notar que existe uma interação entre os parâmetros /M e /A. Se houver parâmetros /A não preenchidos após a análise da linha, ReadArgs( ) pegará os strings do final da última lista de parâmetros /M para preenchê-los. Este recurso pode ser utilizado em mensagens como Copy ("From/A/M,To/A").

O segundo argumento de ReadArgs( ), argarray, é um array de LONGs utilizado por ReadArgs( ) para armazenar os valores dos argumentos da linha de comando. Antes de passar este array para ReadArgs( ), um programa deverá definir as entradas do array utilizando valores default aceitáveis, ou então zerar essas entradas.

Caso sua execução seja bem sucedida, ReadArgs( ) retornará um ponteiro para uma estrutura RDArgs (de <dos/rdargs.h>). ReadArgs( ) usa essa estrutura internamente para controlar sua operação. É possível passar para ReadArgs( ) uma estrutura RDArgs personalizada (*myrda* no protótipo de ReadArgs( ) apresentado acima).

Todavia, na maioria das vezes *myrda* será NULL, uma vez que a maior parte das aplicações não requer o controle de ReadArgs( ).

```
struct RDArgs {
    struct  CSource RDA_Source;  /* Seleciona fonte de entrada */
    LONG    RDA_DAList;          /* privativa. */
    UBYTE   *RDA_Buffer;         /* Espaço opcional para análise do string */
    LONG    RDA_Bufsiz;          /* Tamanho do buffer RDA (0..n) */
    UBYTE   *RDA_ExtHelp;        /* Help estendido opcional */
    LONG    RDA_Flags;           /* Flags de controle */
};
```

Toda as chamadas bem sucedidas a ReadArgs( ), inclusive aquelas que usam uma estrutura RDArgs personalizada, deverão ser complementadas com uma chamada a FreeArgs( ):

```
void FreeArgs( struct RDArgs *rda );
```

onde *rda* é a estrutura RDArgs utilizada por ReadArgs( ).

Uma aplicação pode fazer uso de uma estrutura RDArgs personalizada a fim de proporcionar uma fonte alternativa para a linha de comandos, um buffer de armazenamento temporário alternativo, ou um string de help estendida. A estrutura RDArgs personalizada deverá ser alocada com AllocDosObject( ) e desalocada com FreeDosObject( ). Veja no Capítulo 6 informações mais detalhadas sobre estas funções.

O campo RDA_Source é usado para suprir ReadArgs( ) com uma linha de comandos alternativa para ser analisada. ReadArgs( ) utilizará esta linha somente se os campos RDA_Source estiverem preenchidos. A estrutura CSource (de <dos/rdargs.h>) é a seguinte:

```
struct CSource {
      UBYTE      *CS_Buffer;
      LONG       CS_Lenght;
      LONG       CS_CurChr;
};
```

Onde *CS_Buffer* é a linha de comandos a ser analisada, *CS_Lenght* é o tamanho de *CS_Buffer*, e *CS_CurChr* é a posição no *CS_Buffer* a partir da qual ReadArgs( ) deverá começar a analisar a linha. Normalmente *CS_CurChr* é inicializado com zero. Note que o buffer deverá terminar com um newline (\n).

A estrutura *RDA_DAList* é privativa e deverá ser definida como NULL antes de ser utilizada por ReadArgs( ).

Os campos *RDA_Buffer* e *RDA_BufSiz* permitem que a aplicação forneça um buffer de tamanho fixo para armazenar os dados analisados. Isto possibilita à aplicação pré-alocar um buffer, em vez de solicitar a ReadArgs( ) para alocar o espaço do buffer. Se *RDA_Buffer* ou *RDA_BufSiz* forem NULL, ReadArgs( ) assumirá que a aplicação não dispõe do buffer em questão.

RDA_ExtHelp é um string de texto que ReadArgs( ) exibe se o usuário solicitar ajuda adicional. Você pode pedir ajuda digitando um ponto de interrogação quando

ReadArgs( ) solicitar a entrada de dados do usuário (o que normalmente só acontece quando um ponto de interrogação é fornecido como o único argumento da linha de comandos).

RDA_Flags é um campo de bits usado para alternar certas opções de ReadArgs( ). Por ora somente uma opção está implementada, RDAF_NOPROMPT. Quando setado, RDAF_NPROMPT evita que ReadArgs( ) apresente um prompt para o usuário.

A rotina a seguir, ReadArgs.c, utiliza uma estrutura RDArgs personalizada a fim de passar uma linha de comandos para ReadArgs( ).

```
;/* ReadArgs.c - Execute-me para compilar com o Lattice 5.10a
LC -bl -cfistq -v -y -j73 ReadArgs.c
Blink FROM LIB:c.o,ReadArgs.o TO ReadArgs LIBRARY LIB:LC.Lib,LIB:Amiga.lib
quit
*/
  #include <dos/dos.h>
  #include <dos/rdargs.h>
  #include <clib/dos_protos.h>
  #include <clib/alib_stdio_protos.h>
  #ifdef LATTICE
  int CXBRK(void) { return(0); } /* Desabilita manipulação do CTRL/C pelo Lattice */
  int chkabort(void) { return(0); }
  #endif
UBYTE *vers = "\0$VER: ReadArgs 1.0";
#define TEMPLATE "S=SourceFiles/A/M,D=DebugLevel/K/N,L=link/S"
#define OPT_SOURCE      0
#define OPT_DEBUG       1
#define OPT_LINK        2
#define OPT_COUNT       3
/*  O array de LONGs onde ReadArgs( ) irá armazenar os dados
**  dos argumentos da linha de comando. O C garante que todas
**  as entradas do array serão definidas como zeros.
*/
LONG result [OPT_COUNT];
/*  Minha RDArgs personalizada */
struct RDArgs *myrda;
ULONG StrLen(UBYTE *);
void main(void)
{
    UWORD x;
    UBYTE **sourcefiles;
/* É preciso solicitar ao DOS uma estrutura RDArgs */
if (myrda = struct RDArgs *)AllocDosObject(DOS_RDARGS, NULL))
{
    /* define meus parâmetros para ReadArgs( ) */
    /* a seguinte linha de comandos é usada */
    myrda->RDA_Source.CS_Buffer = "file1 file2 file3
  D=1 Link file4 file5\n";
    myrda->RDA_Source.CS_Lenght =
              (LONG)StrLen(myrda->RDA_Source.CS_Buffer);
```

```
     /* analisa minha linha de comandos */
if (ReadArgs(TEMPLATE, result, myrda))
{
     /* começa a imprimir os resultados */
    /*  Nós não precisamos verificar se há um valor em
    **  result[OPT_SOURCE] pois o gabarito de ReadArgs( )
    **  exige (usando o modificador /A) a presença de
    **  nomes de arquivos, logo ReadArgs( ) preencherá um
    **  valor ou sua execução não terá sucesso.
    */
    sourcefiles = (UBYTE **)result[OPT_SOURCE];
    /*  VPrintf é bastante semelhante a Printf( ), só que
    **  está em ROM, e os argumentos são referenciados a
    **  partir de um array, em vez de serem extraídos
    **  do stack.
    */
    VPrintf("Arquivos especificados:\n", NULL);
    for (x=0; sourcefiles[x]; x++)
         VPrintf("\t%s\n", (LONG *)&sourcefiles[x]);
    /*  Há alguma coisa na opção "DebugLevel"?
    **  Se houver, imprima.
    */
    if  (result[OPT_DEBUG])
        Vprintf("Nível de Debugging = %ld\n", (LONG
        *)result[OPT_DEBUG]);
    /*  Se o toggle do link estiver presente, informe
    **  algo a respeito.
    /*
    if  (result[OPT_LINK])
        Vprintf("linkando...\n", NULL);
    FreeArgs(myrda);
 }
  FreeDosObject(DOS_RDARGS, myrda);
}
}
ULONG StrLen(UBYTE *string)
{
    ULONG x = 0L;
    while (string[x++]);
    return(x);
}
```

Capítulo 6

Chamando o AmigaDOS

Este capítulo descreve as funções disponíveis na biblioteca residente do AmigaDOS. Para auxiliar o leitor, serão fornecidas uma explicação sobre a sintaxe utilizada no capítulo, uma descrição completa de cada função, e uma tabela de referência rápida das funções disponíveis.

Sintaxe

A sintaxe empregada neste capítulo mostra a chamada da função em C para cada função do AmigaDOS, e os registradores correspondentes que você deverá utilizar quando estiver programando em Assembler.

Valores dos Registradores

A combinação letra/número (D0...Dn) representa os registradores. O texto à esquerda do sinal de igual representa o resultado de uma função. Um registrador (por exemplo, D0) aparecendo debaixo deste texto indica o valor do resultado apontado pelo registrador. O texto exibido à esquerda do sinal de igual representa uma função e seus respectivos argumentos, sendo que o texto entre parênteses relaciona a lista dos argumentos. Um registrador (por exemplo, D2) aparecendo sob um argumento indica o valor desse argumento apontado pelo registrador.

Observe que nem todas as funções retornam um resultado.

Grafia dos Nomes

O tamanho das letras nos nomes das funções (isto é, maiúsculas ou minúsculas) é significativo. Você deve digitar "FileInfoBlock", por exemplo, usando maiúsculas na primeira letra de cada componente da palavra.

Valores de Retorno Booleanos

-1 (TRUE ou SUCCESS), 0 (FALSE ou FAILURE).

Valores

Todos os valores são palavras longas (isto é, valores de quatro bytes ou 32 bits). Valores referenciados como "strings" são ponteiros de 32 bits para uma série de caracteres terminados por NULL (0).

Funções do AmigaDOS

Esta seção descreve as funções da biblioteca residente do AmigaDos. Deven ser usadas (linkadas) com AMIGA.LIB.

AbortPkt

Nome

AbortPkt: Aborta um packet assíncrono, se possível (V36).

Sinopse

```
AbortPkt(port, pkt)
          D1    D2
void AbortPkt (struct MsgPort *, struct DosPacket *)
```

Função

Esta função tenta abortar um packet previamente enviado por intermédio da função SendPkt( ) para um processo de manipulação de dispositivo (handler). Não é possível prever se o handler permitirá que o packet seja abortado, nem se este será abortado após a função requisitada ter sido completada ou não. Após chamar AbortPkt( ), você deverá esperar pelo retorno do packet antes de usá-lo novamente ou desalocá-lo.

Entrada

port: porta para a qual o packet foi enviado.
pkt: packet que você deseja abortar.

Bugs

A partir da versão V37 do dos.library, esta função não faz nada.

Veja também: SendPkt( ), DoPkt( ), WaitPkt( ).

AddBuffers

Nome

AddBuffers: muda o número de buffers de um filesystem.

Sinopse

success = AddBuffers(filesystem, number)

```
D0                D1          D2
BOOL AddBuffers(char *, LONG)
```

Função

Adiciona buffers a um filesystem. Caso a execução seja bem sucedida, o número atual de buffers será retornado por IoErr( ). Note que o argumento *number* pode ser negativo. A quantidade de memória usada por cada buffer e o limite para o número de buffers dependem do filesytem em questão. Se a chamada for bem sucedida, o número de buffers em uso no filesystem será retornado por IoErr( ).

Entrada

filesystem: nome do dispositivo ao qual os buffers serão acrescentados (com ":").
number: número de buffers a serem acrescentados. Pode ser negativo.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

No dos.library V36, o filesystem da ROM (FFS/OFS) não retorna o número correto de buffers, a menos que a chamada seja precedida por um AddBuffers(fs,-1) (os buffers em uso não são contados). Este bug foi corrigido na versão V37.

AddDosEntry

Nome

AddDosEntry: acrescenta uma entrada à lista de dispositivos do DOS.

Sinopse

```
success = AddDosEntry(dlist)
D0                    D1
BOOL AddDosEntry(struct DosList *)
```

Função

Acrescenta um dispositivo, volume ou associação à lista de dispositivos do DOS (DOS List). A execução da rotina pode falhar se houver conflito com uma entrada já existente (como uma outra associação do mesmo nome, ou um outro dispositivo com o mesmo nome). Nós de volumes com diferentes datas e o mesmo nome podem ser acrescentados, ou com nomes que conflitem com dispositivos ou associações.

Nota: Não é necessário ativar um lock sobre a lista de dispositivos do DOS (DOS List) antes de se chamar esta função. Não acesse *dlist* após acrescentar o item, a menos que você tenha ativado um lock sobre o DOS List.

Entrada

dlist: entrada na lista de dispositivos a ser acrescentada.

Resultado

success: indicador de falha/sucesso.

Veja também: RemDosEntry( ), FindDosEntry( ), NextDosEntry( ), LockDosList( ), MakeDosEntry( ), FreeDosEntry( ).

AddPart

Nome

AddPart: acrescenta um arquivo/diretório ao final de um path.

Sinopse

```
success = AddPart( dirname, filename, size )
D0                 D1        D2     D3
BOOL AddPart( UBYTE *, UBYTE *, ULONG )
```

Função

Esta rotina acrescenta um nome de arquivo, diretório ou subpath ao nome de path de um diretório, levando em conta os caracteres de separação especificados. Se for um path inteiramente qualificado, *filename* substituirá por completo o valor atual de *dirname*.

Entrada

dirname: path ao qual deverá ser acrescentado o nome do arquivo/diretório.

filename: nome do arquivo ou diretório a ser acrescentado. Pode ser um nome de path relativo ao diretório corrente (por exemplo: foo/bar). Esta opção é capaz de lidar com barras (/), indicando um nível de diretório acima por cada barra, e também com ":", indicando que o nome é relativo à raiz do volume.

size: tamanho em bytes do espaço alocado para *dirname*. Deve ser obrigatoriamente diferente de zero.

Resultado

success: não zero para OK, ou FALSE caso seja ultrapassada a capacidade de armazenamento do buffer. Se houver um overflow do buffer, *dirname* não será alterado.

Bugs

Esta função não verifica se um subpath é legal (isto é, não checa os ":") e não manipula barras precedentes ("/") nas versões 2.0 até 2.02 do AmigaDOS (dos.library V36). A versão 2.04 (V37) deve consertar isto, permitindo que o *filename* seja qualquer path, inclusive absoluto.

Veja também: Filepart( ), PathPart( ).

AddSegment

Nome

AddSegment: acrescenta um segmento à lista residente.

Sinopse

```
success = AddSegment(name, seglist, type)
D0                   D1       D2      D3
BOOL AddSegment(char *, BPRT, LONG)
```

Função

Esta função acrescenta um segmento à lista residente do DOS, com o *seglist* e *type* especificados (armazenados em seg_UC – normalmente zero). Observe que *types* sem uso no momento podem fazer com que outros registradores (de D4 em diante) sejam interpretados como parâmetros adicionais em futuras versões da rotina.

NÃO construa suas próprias estruturas de segmentos ao chamar esta função!

Entrada

name: nome do segmento.

seglist: lista de segmentos do DOS com o código do segmento.

type: usecount inicial, normalmente zero.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Na versão 1.4 Beta 1, o código de retorno da função era indefinido e uma falha na alocação poderia provocar um Alert sem saída. Na versão 2.0 do AmigaDOS, esta rotina funciona corretamente.

Veja também: FindSegment( ), RemSegment( ), LoadSeg( ).

AllocDosObject

Nome

AllocDosObject: cria um objeto do DOS.

Sinopse

```
ptr = AllocDosObject(type, tags)
D0                   D1    D2
void *AllocDosObjectTagList(ULONG, struct TagItem *)
ptr = AllocDosObjectTags (type, Tag1, ...)
void *AllocDosObjectTags (ULONG, ULONG, ...)
```

Função

Esta rotina cria um dos vários objetos do DOS, inicializando-o e devolvendo-o para você. Observe que DOS_STDPKT retorna um ponteiro para o sp_Pkt da estrutura.

Entrada

type: tipo do objeto requisitado.
tags: ponteiro para a lista de tags com informações adicionais.

Resultado

packet: ponteiro para o objeto, ou NULL.

Veja também: FreeDosObject( ), <dos/dostags.h>, <dos/dos.h>.

AssignAdd

Nome

AssignAdd: acrescenta um lock a uma associação em associações de múltiplos diretórios.

Sinopse

```
success = AssignAdd(name, lock)
D0                  D1   D2
BOOL AssignAdd(char *, BPTR)
```

Função

Esta função acrescenta um lock a uma associação, tornando-a uma associação de múltiplos diretórios ou adicionando-a às associações existentes. Observe que esta chamada só terá sucesso numa associação criada com AssignLock( ), ou numa associação criada com AssignLate( ) que tenha sido resolvida (convertida numa associação AssignLock).

Nota: Você não deverá usar o lock de modo algum após completar esta chamada

com sucesso. O lock se torna parte da associação, e será desativado pelo sistema quando a associação for removida. Se for preciso preservar o lock, passe um lock de DupLock( ) para AssignLock( ).

Entrada

name: nome do dispositivo ao qual o lock deverá ser associado (sem ":" no final).
lock: lock associado ao nome atribuído.

Resultado

success: indicador de falha/sucesso. Em caso de falha na execução da chamada, o lock não será desativado.

Veja também: Lock( ), AssignLock( ), AssignPath( ), AssignLate( ), DupLock( ), RemAssignList( ).

AssignLate

Nome

AssignLate: cria uma associação que será posteriormente vinculada ao path.

Sinopse

```
success = AssignLate(name, path)
D0                   D1    D2
BOOL AssignLate(char *, char *)
```

Função

Cria uma associação que será expandida na PRIMEIRA referência a *name*. O *path* (um string) é vinculado ao nó. Quando o nome for referenciado (Open "FOO:xyzzy"...), o string será usado para determinar onde será definida a associação, e se for possível ativar um lock sobre o diretório. A associação agirá a partir deste ponto tal como se tivesse sido criada por AssignLock( ).

A principal vantagem disto é permitir a atribuição de dados a volumes não montados, que serão requisitados durante o acesso (o que é bastante útil nas seqüências de inicialização).

Entrada

name: nome do dispositivo a ser associado (sem ":" no final).
path: nome da última associação a ser resolvida por ocasião da primeira referência.

Resultado

success: indicador de falha/sucesso da operação.

AssignLock

Nome

AssignLock: cria uma associação para um objeto sobre o qual foi ativado um lock.

Sinopse

```
success = AssignLock(name, lock)
D0                   D1    D2
BOOL AssignLock(char *, BPTR)
```

Função

Estabelece a associação de um nome a um lock especificado. Passando NULL para um lock, você cancelará qualquer associação a ser resolvida para aquele nome. Se já houver na lista uma entrada com uma associação de mesmo nome, a rotina cancelará a antiga entrada e a substituirá pela nova. Se houver uma entrada na lista que conflite com a nova associação, será devolvido um código indicando a falha de execução da rotina.

Nota: Você não deverá usar o lock de modo algum após completar esta chamada com sucesso. O lock se torna parte da associação, e será desativado pelo sistema quando a associação for removida. Se for preciso preservar o lock, passe um lock de DupLock( ) para AssignLock( ).

Entrada

name: nome do dispositivo ao qual o lock deverá ser associado (sem ":" no final).
lock: lock associado ao nome atribuído.

Resultado

success: indicador de falha/sucesso. Em caso de falha na execução da chamada, o lock não será desativado.

AssignPath

Nome

AssignPath: cria uma associação para um path especificado.

Sinopse

```
success = AssignPath(name, path)
D0                   D1    D2
BOOL AssignPath(char *, char *)
```

Função

Estabelece uma associação que será expandida em CADA referência a *name*. Isto é implementado através de um novo tipo de lista de dispositivos (DLT_ASSIGNPATH,

ou similar). O *path* (um string) seria vinculado ao nó. Quando o nome é referenciado (Open "FOO:xyzzy"), o string é usado para determinar onde será efetuada a operação de abertura. Nenhum lock permanente faz parte desse processo. Por exemplo, você pode comandar AssignPath c2: to df2:c, e as referências para c2 iriam para df2:c mesmo que você trocasse os discos.

A outra grande vantagem deste processo é permitir a atribuição de dados a volumes não montados, que serão requisitados durante o acesso (o que é bastante útil nas seqüências de inicialização).

Entrada

name: nome do dispositivo a ser associado (sem ":" no final).
path: nome da última associação a ser resolvida em cada referência.

Resultado

success: indicador de falha/sucesso da operação.

AttemptLockDosList

Nome

AttemptLockDosList: tenta ativar um lock sobre a lista de dispositivos do DOS para utilizá-la.

Sinopse

```
dlist = AttemptLockDosList(flags)
D0                         D1
struct DosList *AttemptLockDosList(ULONG)
```

Função

Esta função ativa um lock sobre a lista de dispositivos do DOS, como preparativo para percorrer a lista. Se a lista estiver "ocupada" (busy), a rotina retornará um NULL. Veja LockDosList( ) para maiores informações a respeito.

Entrada

flags: flags determinando sobre que tipos de nós deverão ser ativados os locks.

Resultado

dlist: ponteiro para o início da lista, ou NULL. Este NÃO é um nó válido!

ChangeMode

Nome

ChangeMode: modifica o modo atual de um lock ou filehandle.

Sinopse

```
success = ChangeMode(type, object, newmode)
D0                   D1     D2       D3
BOOL ChangeMode(ULONG, BPTR, ULONG)
```

Função

Esta função permite que você tente modificar o modo em uso por um lock ou filehandle. Você pode tentar, por exemplo, converter um lock compartilhado num lock exclusivo. O handler, entretanto, pode rejeitar a sua requisição.

ATENÇÃO: Se usar o tipo errado (CHANGE_FH ou CHANGE_LOCK) para o objeto, você poderá provocar uma perda de controle do sistema (*crash*).

Entrada

type: ou CHANGE_FH ou CHANGE_LOCK.
object: um lock ou filehandle.
newmode: o novo modo desejado.

Resultado

success: Booleano.

Bugs

Esta rotina não funcionava nas versões 2.0, 2.01 e 2.02 do AmigaDOS (dos.library V36). Ela deve funcionar na versão 2.04 (V37). Nas versões anteriores, o uso desta rotina pode provocar um *crash*.

CheckSignal

Nome

CheckSignal: verifica sinais de break.

Sinopse

```
signals = CheckSignal(mask)
D0                D1
ULONG CheckSignal(ULONG)
```

Função

Verifica se algum dos sinais especificados na máscara *mask* foi setado e o(s) retorna em caso positivo. Caso contrário, a rotina retornará FALSE. Todos os sinais especificados na máscara serão ressetados.

Entrada

mask: sinais que deverão ser checados.

Resultado

signals: sinais especificados na máscara que estavam setados.

Cli

Nome

Cli: retorna um ponteiro para a estrutura CLI do processo.

Sinopse

```
cli_ptr = Cli( )
D0
struct CommandLineInterface *Cli(void)
```

Função

Retorna um ponteiro para a estrutura CLI do processo atual, ou NULL caso o processo não disponha de nenhuma estrutura CLI.

Resultado

cli_ptr: ponteiro para a estrutura CLI ou NULL.

Close

Nome

Close: fecha um arquivo aberto.

Sinopse

```
success = Close( file )
D0                D1
BOOL Close(BPTR)
```

Função

O arquivo especificado pelo filehandle *file* será fechado. Você deverá fechar todos os arquivos que tiver aberto, porém não deverá fechar os filehandles que o sistema tenha passado para você (cada filehandle deverá ser fechado uma e SOMENTE UMA vez). Se a execução de Close( ) falhar o filehandle continuará desalocado e não deverá ser utilizado.

Entrada

file: ponteiro BCPL para um filehandle.

Resultado

success: retorna se a execução de Close( ) tiver sido bem sucedida. Note que a rotina pode falhar devido a erros de bufferização ou no I/O necessário para fechar um arquivo sendo gravado.

Nota: este valor de retorno não existia antes do dos.library V36.

Veja também: Open( ), OpenFromLock( )

CompareDates

Nome

CompareDates: compara dois indicadores de datas.

Sinopse

```
result = CompareDates(date1, date2)
D0                    D1      D2
LONG CompareDates(struct DateStamp *,struct DateStamp *)
```

Função

Esta rotina compara dois indicadores de datas (*datestamps*) em função de sua magnitude relativa. Retorna <0 se *date1* for posterior a *date2*, 0 se as duas datas forem iguais, ou >0 se *date2* for posterior a *date1*.

Nota: Esta ordenação NÃO é idêntica à da comparação de strings!

Entrada

date1, date2: DateStamps a comparar.

Resultado

result: <0, 0 ou >0, baseado na comparação entre os dois indicadores de datas.

Veja também: DateStamp( ), DateToStr( ), StrToDate( ).

CreateDir

Nome

CreateDir: cria um novo diretório.

Sinopse

```
lock = CreateDir( name )
D0           D1
BPTR CreateDir(char *)
```

Função

CreateDir( ) cria um novo diretório com o nome especificado. A chamada retorna um erro se a sua execução falhar. Lembre-se de que os diretórios podem ser criados somente nos dispositivos capazes de suportá-los, como discos, por exemplo. CreateDir( ) retornará um lock exclusivo sobre o novo diretório, caso sua execução tenha sido bem sucedida.

Entrada

name: ponteiro para um string terminado por NULL.

Resultado

lock: ponteiro BCPL para um lock, ou NULL em caso de falha.

CreateNewProc

Nome

CreateNewProc: cria um novo processo.

Sinopse

```
process = CreateNewProc(tags)
D0                      D1
struct Process *CreateNewProc(struct TagItem *)
process = CreateNewProcTagList (tags)
D0                              D1
struct Process *CreateNewProcTagList (struct TagItem *)
process = CreateNewProcTags (Tag1, ...)
struct Process *CreateNewProcTags (ULONG, ...)
```

Função

Esta função cria um novo processo de acordo com os identificadores (tags) passados para ela (veja o arquivo dos/dostags.h para maiores informações sobre os tags). Você deverá selecionar um tag NP_Seglist ou NP_Entry. A NP_Seglist é passada uma lista de segmentos (como a retornada pela função LoadSeg( )). A NP_Entry é passado um ponteiro para o código da rotina a ser chamada. Diversas opções estão disponíveis, como você pode confirmar examinando o arquivo dos/dostags.h.

Os defaults correspondem a um processo não CLI, com cópias do diretório corrente, diretório residente (usado para PROGDIR:), prioridade, porta da tarefa do console, janela para impressão de mensagens, e variáveis do mesmo. Os defaults dos filehandles de entrada e saída são: abertura no dispositivo NIL:, stack de 4.000 bytes, e outros que podem ser verificados observando-se o arquivo dostags.h. Este é um conjunto de definições default bastante razoável para uso genérico, embora você possa desejar modificá-lo, por exemplo, para dar um nome descritivo ao processo.

CreateNewProc( ) pode ser chamado a partir de uma tarefa, embora neste caso as ações que requeiram I/O do DOS (DupLock( ) do diretório corrente, por exemplo) não sejam executadas.

Entrada

tags: ponteiro para um array TagItem.

Resultado

process: o processo criado, ou NULL em caso de erro.

CreateProc

Nome

CreateProc: cria um novo processo.

Sinopse

```
process = CreateProc( name, pri, seglist, stackSize )
D0                    D1    D2     D3         D4
struct MsgPort *CreateProc(char *, LONG, BPTR, LONG)
```

Função

CreateProc( ) cria um novo processo do AmigaDOS com o nome especificado por *name*. Os processos do AmigaDOS são um superconjunto de tarefas Exec.

Uma lista de segmentos, como a que é retornada por LoadSeg( ), deverá ser passada como *seglist*. Isto representa uma seção de código que deverá ser executada como um novo processo. O código é inserido no primeiro hunk da lista de segmentos (veja o Capítulo 10), o qual deverá conter o código de inicialização apropriado ou um salto para o mesmo. Uma estrutura de controle do processo é alocada na memória e inicializada. Se você quiser simular uma lista da segmentos (a qual nunca deverá ser utilizada com a chamada UnLoadSeg( ) do DOS), utilize o seguinte código:

```
DS.1     0  ;Alinha para palavra longa
DC.L     16 ;"Tamanho" do segmento (simulado)
DC.L     0  ;Ponteiro para o próximo segmento
...início do código...
```

O tamanho do stack raiz do processo por ocasião da ativação é passado como *stackSize*. O argumento *pri* especifica a prioridade desejada do novo processo. O resultado é o endereço da porta de mensagens (MsgPort) do novo processo, ou zero em caso de falha na execução da rotina. O argumento *name* especifica o nome do novo processo. Um código de retorno igual a zero sinaliza a ocorrência de um erro. A lista de segmentos passada para CreateProc( ) não é liberada na saída, compete ao processo liberá-la, ou então ao código remover-se a si mesmo.

Com o dos.library versão V36 ou posterior, provavelmente você deverá utilizar CreateNewProc( ) no lugar desta função.

Entrada

name: ponteiro para um string terminado por NULL.
pri: número em forma de palavra longa com sinal (faixa de -128 a +127).
seglist: ponteiro BCPL para uma lista de segmentos.
stackSize: inteiro (deverá ser um múltiplo de 4 bytes).

Resultado

process: identificador do processo.

CurrentDir

Nome

CurrentDir: torna um lock de diretório o diretório corrente.

Sinopse

```
oldlock = CurrentDir( lock )
D0                    D1
BPTR CurrentDir(BPTR)
```

Função

CurrentDir( ) faz com que um diretório associado com um lock seja transformado no diretório corrente. A função retorna o lock do antigo diretório corrente. Nesta função, o valor zero é um resultado válido: o lock 0 representa a raiz do filesystem a partir do qual foi dado o boot no sistema. Todas as chamadas como Open( ), Lock( ) e outras que precisem acessar um arquivo, requerem que o diretório corrente seja um lock válido ou então zero.

Entrada

lock: ponteiro BCPL para um lock.

Resultado

oldLock: ponteiro BCPL para um lock.

Veja também: Lock( ), Unlock( ), Open( ), DupLock( ).

DateStamp

Nome

DateStamp: obtém a data e a hora no formato interno.

Sinopse

```
ds = DateStamp( ds );
D0              D1
struct DateStamp *DateStamp (struct DateStamp *)
```

Função

DateStamp( ) utiliza uma estrutura de três palavras longas através da qual é definida a hora atual. O primeiro elemento dessa estrutura é um contador do número de dias. O segundo elemento representa o número de minutos transcorridos no dia. O terceiro é o número de *tiques* transcorridos no minuto atual, sendo que um tique ocorre 50 vezes por segundo. DateStamp assegura que o dia e os minutos serão consistentes. Os três elementos valerão zero caso a data não tenha sido acertada. DateStamp( ) por ora retorna somente múltiplos pares de 50 tiques. Assim, o tempo obtido será sempre um número par de tiques. O tempo é contado a partir de 1º de janeiro de 1978.

Entrada

ds: ponteiro para DateStamp.

Resultado

O array é preenchido tal como descrito anteriormente.

DateToStr

Nome

DateToStr: converte um indicador de data em um string.

Sinopse

```
success = DateToStr( datetime )
D0                        D1
BOOL DateToStr(struct DateTime *)
```

Função

DateToStr( ) converte um indicador de data (DateStamp) do AmigaDOS num string ASCII compreensível pelo usuário, de acordo com as definições especificadas para a estrutura DateTime.

Entrada

DateTime: ponteiro para uma estrutura DateTime previamente inicializada. Esta estrutura deverá ser inicializada tal como se segue:

dat_Stamp: cópia do indicador de data (datestamp) que você deseja converter.

dat_Format: byte de formatação que especifica o formato de dat_StrDate (veja mais

abaixo). Pode ser qualquer uma das opções que se seguem (note que caso o valor especificado seja diferente dos relacionados a seguir, o default de FORMAT_DOS será usado).

FORMAT_DOS: formato do AmigaDOS (dd-mmm-aa).

FORMAT_INT: formato internacional (aa-mmm-dd).

FORMAT_USA: formato americano (mmm-dd-yy).

FORMAT_CDN: formato canadense (dd-mm-yy).

FORMAT_DEF: formato default para a localidade.

dat_Flags: byte de flags. O único flag que afeta esta função é:

DTF_SUBST: quando setada, um string como *Today*, *Monday*, etc. será usado no lugar da especificação dat_Format, quando possível.

DTF_FUTURE: este flag é ignorado por esta função.

dat_StrDay: ponteiro para o buffer que irá receber o string correspondente ao dia da semana (Monday, Tuesday, etc). Se dat_StrDay for nulo, este string não será gerado.

dat_StrDate: ponteiro para o buffer que irá receber o string da data no formato especificado por dat_Format, sujeito a modificações pela ação do flag DTF_SUBST. Se dat_StrDate for nulo, este string não será gerado.

dat_StrTime: ponteiro para o buffer que irá receber o string da hora do dia. Se dat_StrTime for nulo, este string não será gerado.

Resultado

success: um zero indica que a estrutura DateStamp era inválida e não pôde ser convertida. Um resultado diferente de zero indica que a execução da chamada foi bem sucedida.

Veja também: StrtoDate( ), libraries/datetime.h, DateStamp( ).

Delay

Nome

Delay: retarda um processo durante o intervalo de tempo especificado.

Sinopse

```
Delay( ticks )
       D1
Void Delay(ULONG)
```

Função

O argumento *ticks* especifica quantos "tiques" do AmigaDOS (50 por segundo) deverão ser transcorridos antes desta rotina retornar o controle.

Entrada

ticks: inteiro (Integer).

Bugs

Devido à existência de um bug no dispositivo timer.device das versões V1.2/V1.3 do AmigaDOS, especificar uma contagem de tempo igual a zero para Delay( ) podia provocar falhas de operação no timer e durante o acesso ao disquete. Este bug foi corrigido nas versões V36 e posteriores do dos.library.

DeleteFile

Nome

DeleteFile: apaga um arquivo ou diretório.

Sinopse

```
success = DeleteFile( name )
D0                    D1
BOOL DeleteFile(char *)
```

Função

Esta rotina tenta deletar o arquivo ou diretório especificado por *name*, retornando um erro em caso de falha da tentativa. Observe que todos os arquivos dentro de um diretório deverão ser apagados antes que o próprio diretório também possa ser deletado.

Entrada

name: ponteiro para um string terminado por NULL.

Resultado

success: booleano.

DeleteVar

Nome

DeleteVar: apaga uma variável local ou de ambiente.

Sinopse

```
success = DeleteVar( name, flags )
D0                        D1     D2
BOOL DeleteVar( UBYTE *, ULONG )
```

Função

Esta rotina apaga uma variável local ou de ambiente.

Entrada

name: ponteiro para o nome da variável. Note que os nomes de variáveis seguem a semântica e a sintaxe dos filesystems.

flags: combinação do tipo de variável a deletar (8 bits inferiores) e flags para o controle do comportamento desta rotina. Os flags definidos até o momento incluem:

GVF_LOCAL_ONLY: apaga uma variável local ao seu processo.

GVF_GLOBAL_ONLY: apaga uma variável global.

O default especifica o apagamento de uma variável local, se esta puder ser encontrada, ou caso contrário uma variável global, se esta for encontrada (apenas para LV_VAR).

Resultado

success: se diferente de zero, a variável foi deletada com sucesso. FALSE indica uma falha na execução da chamada.

Bugs

LV_VAR é o único tipo que pode ser global.

Veja também: GetVar( ), SetVar( ), FindVar( ), DeleteFile( ), dos/var.h.

DeviceProc

Nome

DeviceProc: retorna a estrutura MsgPort para um processo de manipulação de I/O específico.

Sinopse

```
process = DeviceProc( name )
D0                    D1
struct MsgPort *DeviceProc(char *)
```

Função

DeviceProc( ) retorna o identificador do processo que manipula o dispositivo associado com o nome especificado. Se nenhum processo de manipulação puder ser encontrado, o resultado da chamada será zero. Se o nome se referir a uma associação, um lock de diretório será retornado por IoErr( ). Este lock não deverá ser acessado pelas funções UnLock( ) ou Examine( ). Se você precisar fazê-lo, duplique antes o lock com DupLock( ).

Bugs

No dos.library V36, se você tentar executar DeviceProc( ) em algo relativo a uma associação feita com AssignPath( ), a execução da chamada falhará. A razão disto é que o sistema não tem como saber quando desfazer o lock. Se você estiver desenvolvendo programas para a V36, é recomendável utilizar GetDeviceProc( ) no lugar desta função, ou então escrever o seu código de maneira a utilizar condicionalmente GetDeviceProc( )/FreeDeviceProc( ) nesta versão.

DoPkt

Nome

DoPkt: envia um packet do DOS e espera pela resposta (V36).

Sinopse

```
result1 = DoPkt(port,action,arg1,arg2,arg3,arg4,arg5)
D0              D1     D2     D3    D4    D5    D6    D7
LONG = DoPkt(struct MsgPort *,LONG,LONG,LONG,LONG,LONG)
```

Função

Esta função envia um packet para um processo de manipulação de dispositivo (handler) e aguarda pelo retorno do mesmo. Qualquer retorno secundário estará disponível no registrador D1 e a partir de IoErr( ). DoPkt( ) funcionará mesmo se chamado por uma tarefa Exec e não por um processo. Entretanto, sua execução será mais lenta e poderá falhar em função de causas adicionais, como a impossibilidade de alocar um sinal. DoPkt( ) utilizará a sua porta de mensagens (pr_MsgPort) para a resposta, e chamará pr_PktWait (veja abaixo os bugs relativos às tarefas, contudo).

Entrada

port: pr_MsgPort do processo de manipulação de dispositivo ao qual o packet deverá ser enviado.

action: ação solicitada do filesystem/processo de manipulação (handler).

arg1, arg2, arg3, arg4, arg5: argumentos, dependendo da ação, podem não ser necessários.

Resultado

result1: o valor retornado em dp_Res1, ou FALSE caso tenha havido algùm problema na recepção ou envio do packet.

Bugs

Utilizar DoPkt( ) a partir de uma tarefa não funciona no DOS 2.0. Use AllocDosObject( ), PutMsg( ) e WaitPort( )/GetMsg( ) para contornar o problema, ou então chame CreateNewProc( ) a fim de iniciar um processo que efetue o I/O do DOS para você. DoPkt( ) só permite que cinco argumentos sejam especificados. Para utilizar

mais argumentos (os packets suportam um máximo de sete –, veja o Capítulo 11), crie um packet e use as funções SendPkt( )/WaitPkt( ).

DupLock

Nome

DupLock: duplica um lock.

Sinopse

```
newlock = DupLock (lock)
D0                D1
BPTR DupLock(BPTR)
```

Função

À chamada DupLock( ) é passado um lock compartilhado do sistema de arquivo. Esta é a ÚNICA maneira de se obter uma duplicata de um lock – copiá-los simplesmente não é permitido.

Um outro lock para o mesmo objeto será então retornado. Não é possível criar uma cópia de um lock exclusivo.
Um zero indica falha de execução da rotina.

Entrada

lock: ponteiro BCPL para um lock.

Resultado

newlock: ponteiro BCPL para um lock.

Veja também: Lock( ), UnLock( ), DupLockFromFH( ), ParentOfFH( )

DupLockFromFH

Nome

DupLockFromFH: obtém um lock em um arquivo aberto.

Sinopse

```
lock = DupLockFromFH(fh)
D0                   D1
BPTR DupLockFromFH(BPTR)
```

Função

Esta função obtém um lock sobre o objeto associado com *fh*. A rotina só funcionará se o arquivo tiver sido aberto com a utilização de um modo não exclusivo. Outras restrições podem ser impostas pelo filesystem.

Entrada

fh: arquivo aberto para o qual deverá ser obtido o lock.

Resultado

lock: lock obtido ou NULL em caso de falha da rotina.

EndNotify

Nome

EndNotify: encerra uma requisição de notificação.

Sinopse

```
EndNotify(notifystructure)
                  D1
VOID EndNotify(struct NotifyRequest *)
```

Função

Esta rotina remove uma requisição de notificação. É seguro chamá-la mesmo que a execução da função StartNotify( ) tenha falhado. Para NRF_SEND_MESSAGE, EndNotify( ) procura em sua porta por quaisquer mensagens sobre o objeto em questão, removendo-as e respondendo às mesmas antes de retornar.

Entrada

notifystructure: a estrutura passada para StartNotify( ).

ErrorReport

Nome

ErrorReport: exibe um menu com uma mensagem Retry/Cancel após a ocorrência de um erro.

Sinopse

```
status = ErrorReport(code, type, arg1, device)
D0                   D1    D2    D3      D4
BOOL ErrorReport(LONG, LONG, ULONG, struct MsgPort *)
```

Função

Baseada no tipo de requisição (*type*), esta rotina formata o requester* <*$F*Nota: *Requester* é um pequeno menu que o sistema exibe na tela ao executar certas operações que requerem a intervenção do usuário. Um requester normalmente apresenta uma mensagem e dois gadgets, solicitando que o usuário indique se deseja cancelar a operação ou dar continuidade ao processo.> apropriado para ser exibido.

Caso o código não seja compreendido, ErrorReport( ) retornará DOS_TRUE de imediato. A rotina retornará DOS_TRUE quando o usuário selecionar a opção Cancel, se a tentativa de apresentar o requester falhar, ou ainda caso o processo pr_WindowPtr seja -1. Quando o usuário selecionar a opção Retry, a função retornará FALSE. A rotina tentará novamente em DISKINSERTED para os códigos de erro apropriados. Os valores de retorno são opostos àqueles retornados por AutoRequest( ).

Nota: Esta rotina atribui o código de erro (*code*) a IoErr( ) antes de retornar.

Entrada

code: código de erro para a exibição do requester. Os códigos de erro válidos atualmente são:

ERROR_DISK_NOT_VALIDATED

ERROR_DISK_WRITE_PROTECTED

ERROR_DISK_FULL

ERROR_DEVICE_NOT_MOUNTED

ERROR_NOT_A_DOS_DISK

ERROR_NO_DISK

ABORT_DISK_ERROR/* read/write error */

ABORT_BUSY/* you MUST replace... */

type: os tipos de requisição disponíveis no momento são os seguintes:

REPORT_LOCK	*arg1* é um lock (BPTR).
REPORT_FH	*arg1* é um filehandle (BPTR).
REPORT_VOLUME	*arg1* é um nó de volume (ponteiro C).
REPORT_INSERT	*arg1* é o string para o nome do volume (será dividida em um ":"). Com o código ERROR_DEVICE_NOT_MOUNTED, exibirá o requester "Please insert...".

arg1: parâmetro variável (veja *type*).

device (opcional): endereço da tarefa para a qual o relatório deverá ser feito. Necessário apenas para REPORT_LOCK, e somente se arg1 == NULL.

Resultado

status: indicador de Cancel/Retry (0 indica Retry).

ExAll

Nome

ExAll: examina um diretório inteiro.

Sinopse

```
continue = ExAll(lock, buffer, size, type, control)
D0                 D1      D2      D3     D4      D5
BOOL ExAll(BPTR,UBYTE *,LONG,LONG,struct ExAllControl *)
```

Função

Esta rotina examina um diretório inteiro. O *lock* deverá estar sobre um diretório. O argumento *size* corresponde ao tamanho do *buffer* fornecido.

O buffer é preenchido com estruturas ExAllData (parciais), tal como especificado pelo campo *type*.

Type deverá ser um dos valores relacionados abaixo, e determina que informações deverão ser armazenadas no buffer. Cada valor mais alto adiciona novos dados à lista, tal como descrito a seguir:

ED_NAME	nome do arquivo
ED_TYPE	tipo
ED_SIZE	tamanho em bytes
ED_PROTECTION	flags de proteção
ED_DATE	três palavras longas correspondentes à data
ED_COMMENT	comentário (NULL se não houver comentários)

Assim, ED_NAME fornecerá apenas o nome do arquivo, enquanto que ED_COMMENT fornecerá todos os dados sobre o mesmo.

A entrada ead_Next fornece um ponteiro para a próxima entrada no buffer. A última entrada apresentará um NULL em ead_Next.

A estrutura de controle é requisitada de modo que o FFS (Fast Filing System) possa manter o controle no caso de mais de uma chamada a ExAll( ) ser solicitada. Isto acontece toda vez que há mais nomes num diretório do que aqueles que o buffer é capaz de comportar. O formato da estrutura de controle é o seguinte:

Nota: Atente para o fato de que a estrutura de controle deverá ser previamente alocada com AllocDosObject( ).

1. Entries: Este campo informa à aplicação que chama a rotina quantas entradas existem no buffer após a chamada a ExAll( ). Nota: Certifique-se de que o seu programa esteja habilitado a manipular a inexistência de entradas (número de entradas igual a zero), incluindo 0 entradas com valores não zero contínuos.

2. LastKey: este campo deverá OBRIGATORIAMENTE ser inicializado com 0 antes de se chamar ExAll( ) pela primeira vez. Qualquer outro valor fará com que a função apresente comportamentos indesejáveis. Se ExAll( ) retornar um valor diferente de zero, este campo não deverá ser tocado antes de se fazer uma segunda chamada ou chamadas subseqüentes a ExAll( ). Sempre que ExAll( ) retornar um valor diferente de zero, haverá mais chamadas requeridas antes que todos os nomes tenham sido recebidos.

Tão logo um retorno FALSE seja recebido, a chamada ExAll( ) estará completada

(caso IoErr( ) retorne ERROR_NO_MORE_ENTRIES – do contrário será retornado o erro ocorrido, de modo similar ao que acontece com a função ExNext( )).

3. MatchString: Se este campo for NULL, todos os nomes de arquivos serão retornados. Caso seja diferente de NULL, este campo será interpretado como um ponteiro para um string, o qual será utilizado para efetuar a combinação de padrões com todos os nomes de arquivos antes de aceitá-los e inseri-los no buffer. A rotina default de comparação de padrões do AmigaDOS (que não leva em conta a diferença entre letras maiúsculas e minúsculas) é usada. Lembre-se de que este string deverá ter sido previamente analisado pela função ParsePatternNoCase( )!

4. MatchFunc: Contém um ponteiro para um vetor (*hook*) para uma rotina, a fim de decidir se a entrada deverá ser incluída na lista de entradas que é retornada. A entrada é preenchida primeiro, e então passada para o vetor. Se não houver necessidade de chamar MatchFunc( ), esta entrada deverá ser igual a NULL. O vetor é chamado com os seguintes parâmetros (que representam o padrão para os vetores):

```
BOOL = MatchFunc( hookptr, data, typeptr )
                    a0     a1     a2
(a0 = ponteiro para o vetor, a1 = ponteiro para estrutura
ExAllData preenchida, a2 = ponteiro para palavra longa
correspondente ao tipo).
```

MatchFunc( ) deverá retornar FALSE se a entrada não for aceita, e TRUE em caso contrário.

O DOS emulará ExAll( ) utilizando Examine( ) e ExNext( ) caso o handler não suporte o packet ExAll.

Entrada

lock: lock no diretório a ser examinado.

buffer: buffer para o retorno dos dados (deverá ser no mínimo múltiplo de dois bytes e preferivelmente múltiplo de quatro bytes).

size: tamanho em bytes do buffer.

type: tipo dos dados a serem retornados.

control: estrutura de controle (veja os tópicos "Função" e "Nota", acima). Deverá OBRIGATORIAMENTE ter sido alocada com AllocDosObject( )!

Resultado

continue: indica se a execução de ExAll( ) foi ou não completada. Se o retorno for igual a FALSE, ou ExAll( ) foi completado (IoErr( ) == ERROR_NO_MORE_ENTRIES), ou um erro ocorreu (verifique IoErr( )). Caso um valor diferente de zero seja retornado, você deverá chamar ExAll( ) novamente até a função retornar FALSE.

Exemplo

```
eac = AllocDosObject(DOS_EXALLCONTROL,NULL);
if (!eac) ...
...
```

```
eac->eac_LastKey = 0;
do {
  more = ExAll(lock,EAData, sizeof(EAData), ED_FOO, eac);
  if ((!more) && (IoErr( ) != ERROR_NO_MORE_ENTRIES)) {
     /* ExAll falhou em condições anormais */
     break;
  }
  if (eac->eac_Entries == 0) {
     /* ExAll falhou normalmente sem entradas */
     continue;
         /* ("more" *geralmente* é zero) */
  }
  ead = (struct ExAllData *) EAData;
  do {
     /* use ead aqui */
     ...
     /* obtém próximo ead */
     ead = ead->ed_Next;
 } while (ead);
    } while (more);
    ...
    FreeDosObject(DOS_EXALLCONTROL,eac);
```

Bugs

Use somente as versões V37 e posteriores desta função.

Examine

Nome

Examine: examina um diretório ou arquivo associado com um lock.

Sinopse

```
success = Examine( lock, FileInfoBlock )
D0              D1      D2
BOOL Examine(BPTR,struct FileInfoBlock *)
```

Função

Examine( ) preenche o bloco *FileInfoBlock* com as informações concernentes ao arquivo ou diretório associado com o *lock*. Estas informações incluem o nome, tamanho, data de criação e especificação (se é arquivo ou diretório). FileInfoBlock deverá ser alinhado por palavra longa. Em caso de falha de execução, Examine( ) gerará um código de retorno igual a zero.

Você poderá fazer uma cópia local da estrutura FileInfoBlock, desde que tenha o cuidado de nunca passá-la de volta para ExNext( ). FileInfoBlock deverá ser alinhado por palavra longa. A função AllocDosObject( ) alocará esta estrutura corretamente para você.

Entrada

lock: ponteiro BCPL para um lock.

infoBlock: ponteiro para uma estrutura FileInfoBlock (alinhada por palavra longa).

Resultado

success: booleano.

ExamineFH

Nome

ExamineFH: obtém informações sobre um arquivo aberto.

Sinopse

```
success = ExamineFH(fh, fib)
D0                  D1  D2
BOOL ExamineFH(BPTR, struct FileInfoBlock *)
```

Função

Esta rotina examina um filehandle, retornando informações sobre o arquivo na estrutura *FileInfoBlock*. Não há garantia de que o campo fib_Size seja capaz de refletir eventuais mudanças no tamanho do arquivo que foi aberto, embora o filesystem deva tentar fornecer informações atualizadas a respeito.

Entrada

fh: filehandle que você deseja examinar.

fib: FileInfoBlock. Deverá ser alinhado por palavra longa.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Execute

Nome

Execute: executa um comando do CLI.

Sinopse

```
success = Execute( commandString, input, output )
D0                 D1             D2     D3
BOOL Execute(char *, BPTR, BPTR)
```

Função

Esta função tenta executar o string *commandString* como se fosse um comando (e argumèntos) do Shell. O string pode conter qualquer tipo de entrada capaz de ser digitada diretamente no Shell, inclusive o redirecionamento de I/O através dos operadores "<" e ">". A função Execute( ) não retornará até que os comandos na *commandString* tenham retornado.

Normalmente o filehandle de entrada é zero, e neste caso Execute( ) executará as ações que tenham sido especificadas na *commandString* e retornará no final. Caso o filehandle de entrada seja diferente de zero, depois de (possivelmente vazia e não zero) *commandString* ser executado, os dados subseqüentes serão lidos do filehandle de entrada especificado até que o final do arquivo seja encontrado.

Na maioria dos casos será necessário fornecer o filehandle de saída, o qual será utilizado como stream de saída pelos comandos do Shell (a menos que o redirecionamento de saída tenha sido especificado). Se o filehandle de saída for zero, então a janela atual, normalmente especificada como *, será utilizada. Observe que programas rodando sob o Workbench em geral não possuem uma janela atual.

A função Execute( ) também pode ser empregada para a criação de um novo processo Shell interativo, tal como aqueles criados por intermédio da função NEWSHELL. Para fazer isso basta chamar Execute( ) com um *commandString* vazio, e passar um filehandle relativo a uma nova janela como o filehandle de entrada. O filehandle de saída pode ser igual a zero. O Shell lerá os comandos a partir da nova janela, utilizando a mesma janela para efetuar a saída. Esta nova janela do Shell só poderá ser encerrada com um comando ENDSHELL.

No dos.library versão V37, se for passado um filehandle de entrada, e este for um filehandle interativo ou NIL:, a pr_ConsoleTask (porta de tarefa do console) do novo processo será a mesma que a do processo desse filehandle (o mesmo se aplica a SystemTagList( )).

Para que este comando funcione nas versões do dos.library anteriores à V36, o programa RUN deverá estar presente no diretório C:.

Entrada

commandString: ponteiro para um string terminada por NULL.

input: ponteiro BCPL para um filehandle.

output: ponteiro BCPL para um filehandle.

Resultado

success: booleano que indica se Execute( ) teve êxito em encontrar e iniciar a execução do programa especificado. Observe que este NÃO é o mesmo código de retorno dos comandos executados.

Exit

Nome

Exit: abandona um programa.

Sinopse

```
Exit( returnCode )
          D1
void Exit(LONG)
```

Função

Exit( ) foi previsto para ser utilizado com programas escritos como se fossem programas BCPL. Esta função normalmente não possui utilidade para outros propósitos.

De maneira geral, portanto, NÃO CHAME ESTA FUNÇÃO!

Para sair, os programas em C deverão fazer uso da função exit( ) do C (preste atenção no "e" minúsculo). Programas em Assembler deverão posicionar um código de retorno no registrador D0 e executar uma instrução RTS com o stack pointer original.

Implementação

A ação da rotina Exit( ) difere em função do programa que a chamou estar ou não rodando como um comando sob o CLI. Se o programa estiver rodando sob o CLI, o programa terminará e o controle reverterá para o CLI. Neste caso, *returnCode* é interpretado como o código de retorno do programa.

Se o programa estiver rodando como um processo separado, Exit( ) deletará o processo e liberará o espaço associado com o stack, a lista de segmentos e a estrutura do processo.

Entrada

returnCode: inteiro (Integer).

ExNext

Nome

ExNext: examina a entrada seguinte num diretório.

Sinopse

```
success = ExNext( lock, FileInfoBlock )
D0                D1          D2
BOOL ExNext(BPTR, struct FileInfoBlock *)
```

Função

A esta rotina são passados um lock de diretório e uma estrutura *FileInfoBlock*, que deverá ter sido previamente inicializada com uma chamada a Examine( ), ou então atualizada por uma chamada prévia a ExNext( ). Em caso de falha de execução, ExNext( ) devolverá um código de retorno igual a zero. A causa mais comum de falha desta rotina é encontrar o fim da lista de arquivos do diretório em questão. Neste caso IoErr( ) retorna ERR_NO_MORE_ENTRIES, e uma boa saída é o procedimento mais indicado.

Siga então os seguinte passos para examinar um diretório:

1. Passe um lock e uma estrutura FileInfoBlock para Examine( ). O lock deverá ser relativo ao diretório que você deseja examinar.
2. Passe o mesmo lock e FileInfoBlock para ExNext( ).
3. Utilize da maneira que julgar conveniente as informações retornadas em FileInfoBlock. Observe que o campo lib_DirEntryType é positivo para os diretórios e negativo para os arquivos.
4. Continue chamando ExNext( ) até que esta função retorne FALSE. Verifique IoErr( ) para se certificar de que a razão da falha da rotina foi ERROR_NO_MORE_ENTRIES.

Nota: Se você desejar pesquisar recursivamente a árvore de arquivos e encontrar um outro diretório enquanto chama ExNext( ), você deverá ativar um lock sobre esse diretório e examiná-lo com Examine( ) utilizando um novo FileInfoBlock. O uso do mesmo FileInfoBlock para entrar num diretório provocaria a perda de importantes informações de estado, o que impossibilitaria a função de continuar pesquisando o diretório pai. Embora seja lícito empregar Lock( ) e UnLock( ) sobre o diretório pai entre as chamadas ExNext( ), esta NÃO é uma prática recomendável.

Importantes informações de estado estão associadas com o lock matriz, e se estas informações forem liberadas entre as chamadas ExNext( ), será preciso reconstruí-las novamente a cada chamada a ExNext( ). Isto tornaria a pesquisa do diretório significativamente mais lenta.

Note que NÃO é permitido examinar um arquivo com Examine( ) e depois chamar ExNext( ) utilizando o mesmo FileInfoBlock. Você pode fazer uma cópia local do FileInfoBlock, contando que esta estrutura nunca seja passada de volta para o sistema operacional.

Entrada

lock: ponteiro BCPL para o lock originalmente usado na chamada Examine( ).

infoBlock: ponteiro para a estrutura FileInfoBlock utilizada na última chamada Examine( ) ou ExNext( ).

Resultado

success: booleano.

Nota especial: O FileInfoBlock deverá ser alinhado por palavra longa. A função AllocDosObject alocará esta estrutura corretamente para você.

Fault

Nome

Fault: retorna o texto associado com um código de erro do DOS.

Sinopse

```
success = Fault(code, header, buffer, len)
D0              D1      D2      D3      D4
BOOL Fault(LONG, UBYTE *, UBYTE *, LONG)
```

Função

Esta rotina obtém o texto da mensagem de erro relativa ao código de erro especificado. O *header* é anexado ao texto da mensagem de erro, seguido por dois pontos (:). Fault( ) insere no *buffer* um string contendo a mensagem de erro (terminada por NULL). Por convenção, as mensagens de erro não devem exceder 80 caracteres (mais um caractere para término), e preferivelmente não devem ter mais do que 60 caracteres.

O valor retornado por IoErr( ) corresponderá ao código passado para a rotina. Se não existir uma mensagem para o código de erro especificado, a mensagem será "Error is code <número>\n".

Entrada

code: código do erro.

header: cabeçalho a ser exibido antes do texto da mensagem de erro.

buffer: buffer para receber a mensagem de erro.

len: tamanho do buffer.

Resultado

success: código de sucesso/falha de execução da rotina.

FGetC

Nome

FGetC: lê um caractere a partir da fonte de entrada especificada (bufferizado).

Sinopse

```
char = FGetC(fh)
D0           D1
LONG FGetC(BPTR)
```

Função

FGetC( ) lê o próximo caractere de uma stream de entrada. A função retorna -1 quando o final do arquivo (EOF) é encontrado ou em caso de ocorrência de um erro. Esta chamada é bufferizada. Lembre-se de usar Flush( ) entre a execução de operações de I/O bufferizadas e não bufferizadas num filehandle.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

Resultado

char: caractere lido (0-255), ou -1 como descrito acima.

Bugs

No dos.library V36, FGetC( ) só retornava EOF depois de haver lido pela primeira vez um EOF. A partir da versão V37, a função tenta ler um dado do handler novamente a cada tentativa (a menos que UnGetC (fh,-1) tenha sido usado).

Veja também: FPutC( ), UnGetC( ), Flush( ).

FGets

Nome

FGets: lê uma linha a partir da fonte de entrada especificada (bufferizado).

Sinopse

```
buffer = FGets(fh, buf, len)
D0              D1  D2   D3
UBYTE *FGets(BPTR, UBYTE *, ULONG)
```

Função

Esta rotina lê uma linha a partir da fonte de entrada especificada, parando num caractere NEWLINE (\n) ou no final do arquivo (EOF). Em ambos os casos, ATÉ o número de bytes especificados por *len*, menos 1, serão copiados no buffer. Assim, se o tamanho especificado for igual a 50, mas a linha de entrada não tiver mais do que 49 bytes, a função retornará 49 caracteres. FGets( ) retorna o ponteiro do buffer normalmente, ou então NULL, caso um EOF tenha sido o primeiro dado lido.

Caso o string termine com um caractere NEWLINE, este será o último caractere presente no buffer. Esta é uma rotina de leitura bufferizada, o string lido é terminado por NULL.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

buf: área na qual os bytes deverão ser lidos.

len: número de bytes a serem lidos, deverá ser maior que zero.

Resultado

buffer: ponteiro para o buffer passado para a rotina, ou NULL nos casos de leitura imediata de um EOF ou ocorrência de erro. Quando for retornado NULL devido a um EOF, IoErr( ) retornará 0.

FilePart

Nome

FilePart: retorna o último componente de um path.

Sinopse

```
fileptr = FilePart( path )
D0              D1
UBYTE *FilePart( UBYTE * )
```

Função

Esta função retorna um ponteiro para o último componente de um string descritivo de um path, que normalmente será o nome do arquivo. Se houver apenas um componente, FilePart( ) retornará um ponteiro para o início do string.

Entrada

path: ponteiro para o string do path. Este ponteiro poderá ser relativo ao disco ou ao diretório correntes.

Resultado

fileptr: ponteiro para o último componente do path.

Exemplo

FilePart("xxx:yyy/zzz/qqq") retornaria um ponteiro para o primeiro *q*.

FilePart("xxx:yyy") retornaria um ponteiro para o primeiro *y*.

Veja também: PathPart( ), AddPath( ).

FindArg

Nome

FindArg: localiza uma palavra-chave num gabarito.

Sinopse

```
index = FindArg(template, keyword)
D0                 D1         D2
LONG FindArg(UBYTE *, UBYTE *)
```

Função

FindArg( ) retorna o número do argumento da palavra-chave (keyword), ou -1 caso a palavra-chave não se encontre no gabarito. Esta função é capaz de manipular abreviações.

Entrada

keyword: palavra-chave a ser procurada no gabarito.

template: string do gabarito onde deverá ser efetuada a busca.

Resultado

index: número da entrada no gabarito, ou -1 caso a palavra-chave não seja encontrada.

FindCliProc

Nome

FindCliProc: retorna um ponteiro para o processo CLI requisitado.

Sinopse

```
proc = FindCliProc(num)
D0                 D1
struct Process *FindCliProc(LONG)
```

Função

Esta rotina retorna um ponteiro para o processo CLI associado com o número do CLI indicado. Caso o processo não seja um processo ativo do CLI, FindCliProc( ) retornará um NULL.

Nota: Esta rotina deverá ser chamada de dentro de Forbid( ), se for absolutamente necessário utilizar esta função.

Entrada

num: número da tarefa do processo CLI.

Resultado

proc: ponteiro para o processo CLI especificado.

FindDosEntry

Nome

FindDosEntry: localiza uma entrada específica na lista de dispositivos do DOS.

Sinopse

```
newdlist = FindDosEntry(dlist, name, flags)
D0                      D1      D2     D3
struct DosList *FindDosEntry(struct DosList *,UBYTE *,ULONG)
```

Função

Esta rotina localiza uma entrada na lista de dispositivos. A busca será iniciada pela entrada *dlist.*

Nota: FindDosEntry( ) deverá ser chamada com um lock ativo sobre a lista de dispositivos. Nenhuma referência deverá ser feita a *dlist* após o lock ser desativado.

Entrada

dlist: entrada de dispositivo por onde iniciar a busca.

name: nome da entrada de dispositivo (sem ":") a ser localizada.

flags: flags para controle da busca. Use os flags que você passou para LockDosList( ), ou um subconjunto deles. LDF_READ e LDF_WRITE não são exigidos para esta chamada.

Resultado

newlist: a entrada correspondente ao dispositivo, ou NULL caso a entrada não seja localizada.

FindSegment

Nome

FindSegment: localiza um segmento na lista residente (V36).

Sinopse

```
segment = FindSegment(name, start, system)
D0                    D1     D2       D3
struct Segment *FindSegment(char *, struct Segment *, LONG)
```

Função

FindSegment( ) localiza um segmento na lista residente do DOS de acordo com *name* e *type*, iniciando a busca no segmento APÓS *start*, ou no começo da lista caso *start* seja NULL. Se *system* for igual a zero, a rotina só retornará os nós com um seg_UC de 0 ou mais. FindSegment( ) NÃO incrementa seg_UC e NÃO efetua qualquer lock

na lista. Você deverá ativar um lock na lista através de Forbid( ) para utilizar esta chamada.

Para utilizar uma entrada que você tenha localizado, o procedimento é o seguinte: se o seg_UC for igual a zero ou mais, incremente-o, e quando você tiver terminado com a lista de segmentos, decremente-o – sempre sob Forbid( )!

Os outros valores para seg_UC são:

-1: módulo de sistema, como um filesystem ou shell.
-2: comando residente do shell.
-999: comando interno desabilitado, ignore-o.

Valores negativos jamais deverão ser modificados. Todos os outros valores negativos entre 0 e -32767 são reservados para o AmigaDOS, e não devem ser utilizados.

Entrada

name: nome do segmento a ser localizado.

start: segmento após o qual deverá ser iniciada a busca.

system: TRUE para segmentos do sistema, FALSE para segmentos normais.

Resultado

segment: o segmento encontrado ou NULL.

FindVar

Nome

FindVar: localiza uma variável local (V36).

Sinopse

```
var = FindVar( name, type )
D0              D1    D2
struct LocalVar * FindVar(UBYTE *, ULONG)
```

Função

Localiza uma estrutura de variável local.

Entrada

name: ponteiro para o nome da variável. Observe que os nomes das variáveis seguem a semântica e a sintaxe dos filesystems.
type: tipo da variável a ser localizada (veja o arquivo include dos/var.h).

Resultado

var: ponteiro para uma estrutura LocalVar, ou NULL caso a variável não seja encontrada.

Veja também: DeleteVar( ), GetVar( ), SetVar( ), dos/var.h.

Flush

Nome

Flush: limpa buffers para um filehandle bufferizado.

Sinopse

```
success = Flush(fh)
D0        D1
BOOL Flush(BPTR)
```

Função

Esta rotina descarrega qualquer operação de gravação bufferizada pendente para o filehandle *fh*. Todas as operações de gravação bufferizadas serão igualmente descarregadas com o uso da função Close( ). Se o filehandle estiver sendo utilizado para entrada, a rotina abandonará o buffer e tentará excutar um Seek( ) de volta à última posição (deste modo, as operações subseqüentes de gravação e leitura irão ocorrer na posição esperada do arquivo).

Entrada

fh: filehandle a descarregar.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Format

Nome

Format: faz com que um filesystem se inicialize (V37).

Sinopse

```
success = Format(filesystem, volumename, dostype)
D0               D1          D2          D3
BOOL Format(UBYTE *, UBYTE *, ULONG)
```

Função

Esta rotina é utilizada para interfacear a inicialização de uma nova mídia em um dispositivo. Format( ) faz com que o filesystem grave a estrutura para a mídia num disco vazio, o qual deverá então estar pronto para uso. Esta função assume que a mídia já tenha recebido uma formatação de baixo nível (*low-level formatting*), além de ter sido previamente verificada.

Entrada

filesystem: nome do dispositivo a ser formatado. É necessário digitar o ":".

volumename: nome para o volume, caso seja suportado (sem o ":" no final).

dostype: tipo de formato, caso o filehandle suporte múltiplos tipos.

Resultado

success: indicador de sucesso/falha de execução da rotina.

FPutC

Nome

FPutC: grava um caractere na saída especificada (bufferizado) (V36).

Sinopse

```
char = FPutC(fh, char)
D0           D1   D2
LONG FPutC(BPTR, UBYTE)
```

Função

FPutC( ) grava um caractere no stream de saída. Esta chamada é bufferizada. Lembre-se de executar um Flush( ) entre as operações de I/O bufferizadas e não bufferizadas em um filehandle. Em filehandles interativos isto é feito automaticamente mediante a inserção de um newline (\n), return, \0 ou line-feed.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

char: caractere a ser escrito.

Resultado

char: o caractere escrito, ou EOF em caso de erro.

FPuts

Nome

FPuts: grava um string na saída especificada (bufferizado).

Sinopse

```
error = FPuts(fh, str)
D0            D1   D2
LONG FPuts(BPTR, UBYTE *)
```

Função

Esta rotina grava um string não formatada no filehandle *fh*. O caractere de newline (\n) não será anexado ao string. A função retorna o número de bytes do string efetivamente gravados. FPuts( ) é uma rotina bufferizada.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

str: string terminado por NULL a ser escrito na saída default.

Resultado

error: normalmente 0, ou -1 em caso contrário. Note que isto é o oposto da maioria das outras funções do DOS, que geralmente retornam *success*.

FRead

Nome

FRead: lê um determinado número de blocos a partir de uma fonte de entrada (bufferizado).

Sinopse

```
count = FRead(fh, buf, blocklen, blocks)
D0            D1  D2      D2        D3
LONG Fread(BPTR, UBYTE *, ULONG, ULONG)
```

Função

FRead( ) tenta ler um determinado número de blocos (cada qual com o tamanho indicado por *blocklen*) no buffer especificado, a partir de uma stream de entrada. A rotina pode retornar um número de blocos menor que o requisitado, caso encontre um final de arquivo (EOF) ou devido à ocorrência de um erro. Esta chamada é bufferizada.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

buf: área na qual deverão ser lidos os bytes.

blocklen: número de bytes por bloco. Deverá ser maior que zero.

blocks: número de blocos a serem lidos. Deverá ser maior que zero.

Resultado

count: Número de blocos lidos, ou 0 caso seja encontrado um EOF. Na ocorrência de um erro, o número de blocos efetivamente lidos é retornado.

Bugs

A rotina não zera o conteúdo de IoErr( ) antes de começar. Se você quiser verificar IoErr( ) para obter informações sobre erros, use SetErr(0L) antes de chamar esta função.

FreeArgs

Nome

FreeArgs: libera a memória alocada após o uso da função ReadArgs( ).

Sinopse

```
FreeArgs(rdargs)
           D1
void FreeArgs (struct RDArgs *)
```

Função

Libera a memória alocada para o retorno dos argumentos de ReadArgs( ). Caso ReadArgs tenha alocado a estrutura RDArgs, a mesma será liberada.

Entrada

rdargs: estrutura retornada por ReadArgs( ).

Veja também: ReadArgs( ), ReadItem( ), FindArg( ).

FreeDeviceProc

Nome

FreeDeviceProc: libera a porta retornada por GetDeviceProc (V36).

Sinopse

```
FreeDeviceProc(devproc)
                 D1
void FreeDeviceProc(struct DevProc *)
```

Função

Esta rotina libera a estrura criada pela função GetDeviceProc( ), bem como quaisquer locks temporários associados à mesma.

FreeDeviceProc( ) decrementa o contador incrementado por GetDeviceProc( ). O contador se encontra em uma extensão da estrutura de processo 1.3. Após chamar FreeDeviceProc( ), não utilize a porta ou lock novamente. Contudo, é seguro chamar FreeDeviceProc(NULL).

Entrada

devproc: valor retornado por GetDeviceProc.

Bugs

O contador não se encontra ativo na versão 2.0.

FreeDosEntry

Nome

FreeDosEntry: libera uma entrada criada pela função MakeDosEntry( ).

Sinopse

```
FreeDosEntry(dlist)
             D1
void FreeDosEntry(struct DosList *)
```

Função

Libera uma estrutura criada por MakeDosEntry( ). Esta rotina deve ser eliminada e substituída pela passagem de um valor para FreeDosObject( ).

Entrada

dlist: DosList a ser liberada.

FreeDosObject

Nome

FreeDosObject: libera um objeto alocado pela função AllocDosObject( ).

Sinopse

```
FreeDosObject(type, ptr)
D0            D1   D2
void FreeDosObject(ULONG, void *)
```

Função

Esta rotina libera um objeto alocado por AllocDosObject( ). NÃO chame esta função para remover objetos alocados de outra maneira.

Entrada

type: tipo passado para AllocDosObject( ).

ptr: ponteiro retornado por AllocDosObject( ).

Veja também: AllocDosObject( ), dos/dos.h.

FWrite

Nome

FWrite: grava um determinado número de blocos em uma saída (bufferizado).

Sinopse

```
count = FWrite(fh, buf, blocklen, blocks)
D0              D1  D2      D3        D4
LONG FWrite(BPTR, UBYTE *, ULONG, ULONG)
```

Função

FWrite( ) tenta gravar um determinado número de blocos (cada qual com o tamanho indicado por *blocklen*) no stream de saída a partir do buffer especificado. A rotina pode retornar um número de blocos menor que o requisitado, caso ocorra algum erro de leitura/gravação ou se o disco estiver cheio. Esta chamada é bufferizada.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.

buf: área na qual os bytes deverão ser gravados.

blocklen: número de bytes por bloco. Deverá ser maior que zero.

blocks: número de blocos a serem lidos. Deverá ser maior que zero.

Resultado

count: Número de blocos gravados. Na ocorrência de um erro, a função retorna o número de blocos efetivamente gravados.

Bugs

A rotina não zera o conteúdo de IoErr( ) antes de começar. Se você quiser verificar IoErr( ) para obter informações sobre erros, use SetErr(0L) antes de chamar esta função.

GetArgStr

Nome

GetArgStr: retorna os argumentos para o processo (V36).

Sinopse

```
ptr = GetArgStr( )
D0
UBYTE *GetArgStr(void)
```

Função

GetArgStr( ) retorna um ponteiro para os argumentos (terminados por NULL) para o programa (processo). Este é o mesmo string passado num 0 por ocasião da inicialização do CLI.

Entrada

ptr: ponteiro para os argumentos.

GetConsoleTask

Nome

GetConsoleTask: retorna o console default para o processo atual.

Sinopse

```
port = GetConsoleTask( )
D0
struct MsgPort *GetConsoleTask(void)
```

Função

Esta rotina retorna a porta da tarefa do console (pr_ConsoleTask) default para o processo atual.

Resultado

port: pr_MsgPort do processo de manipulação do console, ou NULL em caso de erro.

GetCurrentDirName

Nome

GetCurrentDirName: retorna o nome do diretório corrente.

Sinopse

```
success = GetCurrentDirName(buf, len)
D0                          D1   D2
BOOL GetCurrentDirName(char *, LONG)
```

Função

Esta rotina extrai o nome do diretório corrente da estrutura CLI, inserindo-o em um buffer. Se for muito grande para caber no buffer, o nome será truncado e a função retornará um código de erro. Se não houver uma estrutura CLI presente, um nome nulo será retornado no buffer. Uma falha na execução da rotina será assinalada por IoErr( ) == ERROR_OBJECT_WRONG_TYPE.

Entrada

buf: buffer que irá receber o nome extraído.
len: número de bytes de espaço no buffer.

Resultado

success: indicador de sucesso/falha de execução da rotina.

GetDeviceProc

Nome

GetDeviceProc: localiza o handler ao qual deverá ser enviada uma mensagem.

Sinopse

```
devproc = GetDeviceProc(name, devproc)
D0                      D1       D2
struct DevProc *GetDeviceProc(UBYTE *, struct DevProc *)
```

Função

Esta rotina localiza o handler/filesystem ao qual deverão ser enviados os packets referentes a *name*. Isto pode envolver a obtenção de locks temporários. GetDeviceProc( ) retorna uma estrutura que inclui um lock e uma porta de mensagens (MsgPort) para envio a fim de que você possa tentar esta operação. Também são incluídas informações sobre como manipular associações de múltiplos diretórios (passando a estrutura DevProc de volta para GetDeviceProc( ) até que a rotina retorne um NULL).

A chamada inicial a GetDeviceProc( ) deverá passar NULL para *devproc*. Se após utilizar a estrutura DevProc retornada você obtiver um ERROR_OBJECT_NOT_FOUND, e a condição (devproc->dvp_Flags and DVPF_ASSIGN) for verdadeira, você deverá chamar GetDeviceProc( ) novamente, passando a estrutura DevProc. A função deverá retornar uma estrutura DevProc modificada, ou então NULL (com ERROR_NO_MORE_ENTRIES em IoErr( )). Continue até que a rotina retorne um NULL.

Esta chamada também incrementa o contador que obtém um lock sobre o handler/filesytem na memória. Após chamar FreeDeviceProc( ), não utilize a porta ou o lock novamente.

Entrada

name: nome do objeto que você deseja acessar. Pode ser um path relativo ("foo/bar"), relativo ao volume atual (":foo/bar"), ou relativo a um dispositivo/volume/associação ("foo:bar").

Resultado

devproc: ponteiro para uma estrutura DevProc ou NULL.

Bugs

O contador não se encontra ativo na versão 2.0. Nas versões 2.0 e 2.01 do AmigaDOS, é indispensável checar DVPF_ASSIGN antes de chamar a rotina novamente. Este defeito foi corrigido na versão 2.02.

GetFileSysTask

Nome

GetFileSysTask: retorna o filesystem default para o processo atual.

Sinopse

```
port = GetFileSysTask( )
D0
struct MsgPort *GetFileSysTask(void)
```

Função

Esta rotina retorna a porta da tarefa do filesystem (pr_FileSystemTask) default para o processo atual.

Resultado

port: pr_MsgPort do filesystem, ou NULL em caso de erro.

GetProgramDir

Nome

GetProgramDir: retorna um lock no diretório a partir do qual o programa foi carregado.

Sinopse

```
lock = GetProgramDir( )
D0
BPTR GetProgramDir(void)
```

Função

GetProgramDir( ) retorna um lock compartilhado sobre o diretório a partir do qual o programa foi carregado. Isto pode ser utilizado para permitir que um programa localize arquivos de dados, etc. que são armazenados junto com o programa, ou para localizar o próprio arquivo do programa. Retornos NULL são considerados válidos, e podem ocorrer, por exemplo, se o programa tiver sido executado a partir da lista residente. Você NÃO deve desativar o lock.

Resultado

lock: lock sobre o diretório a partir do qual o programa foi carregado, ou NULL se o programa tiver sido carregado da lista residente, etc.

Bugs

Esta função deveria retornar um lock para programas carregados a partir da lista residente. Talvez a rotina devesse retornar o diretório corrente ao invés de NULL.

GetProgramName

Nome

GetProgramName: retorna o nome do programa atual.

Sinopse

```
success = GetProgramName(buf, len)
D0                       D1   D2
BOOL GetProgramName(char *, LONG)
```

Função

Esta rotina extrai o nome do programa da estrutura CLI, inserindo-o em um buffer. Se for muito grande para caber no buffer, o nome será truncado e a função retornará um código de erro. Se não houver uma estrutura CLI presente, um nome nulo será retornado no buffer e a falha na execução da rotina será assinalada por IoErr( ) == ERROR_OBJECT_WRONG_TYPE.

Entrada

buf: buffer que irá receber o nome extraído.

len: número de bytes de espaço no buffer.

Resultado

success: indicador de sucesso/falha de execução da rotina.

GetPrompt

Nome

GetPrompt: retorna o prompt do processo atual (V36).

Sinopse

```
success = GetPrompt(buf, len)
D0                  D1   D2
BOOL GetPrompt(char *, LONG)
```

Função

Esta rotina extrai o string do prompt da estrutura CLI, inserindo-a em um buffer. Se for muito grande para caber no buffer, o string será truncado e a função retornará um código de erro. Se não houver uma estrutura CLI presente, um string nulo será retornado no buffer e a falha na execução da rotina será assinalada por IoErr( ) == ERROR_OBJECT_WRONG_TYPE.

Entrada

buf: buffer que irá receber o prompt extraído.

len: número de bytes de espaço no buffer.

Resultado

success: indicador de sucesso/falha de execução da rotina.

GetVar

Nome

GetVar: retorna o valor de uma variável local ou global.

Sinopse

```
len = GetVar( name, buffer, size, flags )
D0            D1      D2     D3     D4
LONG GetVar( UBYTE *, UBYTE *, LONG, ULONG )
```

Função

Esta rotina obtém o valor de uma variável local ou de ambiente. É aconselhável utilizar somente strings ASCII dentro de variáveis, embora este procedimento não seja obrigatório. A função evita a inserção de caracteres no destino quando \n é acionado, a menos que o flag GVF_BINARY_VAR tenha sido especificado (o caractere \n não é armazenado no buffer).

Entrada

name: ponteiro para o nome da variável.

buffer: área alocada pelo usuário com a finalidade de armazenar o valor associado à variável.

size: tamanho da região do buffer, em bytes.

flags: combinação do tipo de variável cujo valor deverá ser obtido (8 bits inferiores) e flags para controle do comportamento da rotina. Os flags atualmente implementadas são:

GVF_GLOBAL_ONLY: tenta obter o valor de uma variável de ambiente global.

GVF_LOCAL_ONLY: tenta obter o valor de uma variável local.

GVF_BINARY_VAR: não pára em um \n.

O default desta rotina é tentar obter primeiro uma variável local, e depois uma variável de ambiente global.

Resultado

len: tamanho da variável de ambiente. -1 indica que a variável não foi definida. Se o valor exceder a capacidade do buffer alocado pelo usuário, o buffer será truncado. O buffer retornado termina com NULL (mesmo que o flag GVF_BINARY_VAR tenha sido utilizada). A função retorna o número de caracteres inseridos no buffer (descontando-se o "\0") e IoErr( ) retorna o tamanho da variável.

Bugs

LV_VAR é o único tipo de variável que pode ser global. A documentação dos.library V36 especificava (e a função retornava) o tamanho da variável, e não o número de caracteres transferidos. Na versão V37 isto foi modificado: GetVar( ) retorna o número de caracteres inseridos no buffer, enquanto que o tamanho total da variável é retornado por IoErr( ).

Veja também: DeleteVar( ), FindVar( ), SetVar( ), dos/var.h.

Info

Nome

Info: retorna informações sobre o disco em uso.

Sinopse

```
success = Info( lock, parameterBlock )
D0              D1          D2
BOOL Info(BPTR, struct InfoData *)
```

Função

Info( ) pode ser utilizado para se obter informações sobre qualquer disco em uso. O argumento *lock* se refere ao disco, ou a qualquer arquivo nesse disco. O bloco de parâmetros *parameterBlock* retorna informações relativas ao tamanho do disco, número de blocos livres e ocorrência ou não de soft errors.

Entrada

lock: ponteiro BCPL para um lock.

parameterBlock: ponteiro para uma estrutura InfoData (alinhada por palavra longa).

Resultado

success: booleano.

Nota especial: Atente para o fato de que a estrutura InfoData deverá ser alinhada por palavra longa.

Inhibit

Nome

Inhibit: inibe o acesso a um filesystem (V36).

Sinopse

```
success = Inhibit(filesystem, flag)
D0                    D1        D2
BOOL Inhibit(char *,LONG)
```

Função

Esta rotina envia um packet ACTION_INHIBIT para o processo de manipulação de dispositivo (handler) indicado. Esta ação interrompe toda a atividade desempenhada pelo handler até que o mesmo seja desinibido. Note que quando o handler for desinibido, qualquer evento pode ter ocorrido ao disco no drive, ou o disco pode não estar mais inserido.

Entrada

filesystem: nome do dispositivo a ser inibido (com ":").

flag: novo status. DOSTRUE = inibido, FALSE = desinibido.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Input

Nome

Input: identifica o filehandle de entrada inicial do programa.

Sinopse

```
file = Input( )
D0
BPTR Input(void)
```

Função

Input( ) é utilizado para identificar o stream de entrada inicial alocado quando o programa foi iniciado. Lembre-se de nunca fechar o filehandle retornado pela função Input( ).

Resultado

file: ponteiro BCPL para o filehandle.

Veja também: OutPut( ), SelectInput( ).

InternalLoadSeg

Nome

InternalLoadSeg: rotina de carregamento *low level*.

Sinopse

```
seglist = InternalLoadSeg(fh,table,functionarray,stack)
D0                 D0   A0       A1      A2
BPTR InternalLoadSeg(BPTR,BPTR,LONG *,LONG *)
```

Função

InternalLoadSeg( ) carrega o arquivo indicado pelo filehandle *fh*. *Table* é utilizado para o carregamento de arquivos overlay, devendo ser NULL para outros propósitos. *Functionarray* é um ponteiro para um array de funções. Observe que a posição atual de Seek após o carregamento poderá corresponder a qualquer ponto após o último hunk carregado. O filehandle não será fechado. Se um *stacksize* (tamanho do stack para o processo) estiver codificado no arquivo, o tamanho será incluído no LONG apontado pelo argumento *stack*. Este LONG deverá ser inicializado previamente com o seu valor default. InternalLoadSeg( ) não o alterará caso nenhum *stacksize* seja encontrado. A rotina limpa as porções não usadas dos hunks CODE e DATA (bem como os hunks BSS). Este detalhe também se aplica às funções LoadSeg( ) e NewLoadSeg( ).

Se o arquivo carregado for um overlay, a rotina retornará -(*seglist*). Todos os outros valores de retorno são positivos.

Nota: Os usuários de arquivos overlay devem atentar para o fato de que InternalLoadSeg( ) NÃO retorna a lista de segmentos tanto no registrador D0 quanto em D1, como LoadSeg( ) faz. Os arquivos ovs.asm atuais utilizam LoadSeg( ), e assumem que o retorno se dará no registrador D1. Isto é suportado SOMENTE por LoadSeg( ).

Entrada

fh: filehandle a partir do qual se dará o carregamento.

table: somente para o carregamento de overlays, ignorado em caso contrário.

functionarray: array de funções a ser utilizado pelas funções de leitura, alocação e retorno de memória.

```
FuncTable[0] -> Actual = ReadFunc(readhandle,buffer,lenght),DOSBase
                  D0                D1          A0     D0       A6
FuncTable[1] -> Memory = AllocFunc(size,flags),Execbase
```

```
                D0                      D0   D1     A6
FuncTable[2] -> FreeFunc(memory,size),Execbase
                        A1      D0     A6
```

stack: ponteiro para armazenamento (ULONG) para *stacksize.*

Resultado

seglist: lista de segmentos carregada ou NULL. NÃO é retornado em D1.

Bugs

A rotina na verdade deveria utilizar indicadores do tipo tags.

InternalUnLoadSeg

Nome

InternalUnLoadSeg: remove uma lista de segmentos carregada com InternalLoadSeg( ).

Sinopse

```
success = InternalUnLoadSeg(seglist,FreeFunc)
D0                          D1      A1
BOOL InternalUnLoadSeg(BPTR, void (*) (char *,ULONG))
```

Função

Esta rotina remove uma lista de segmentos utilizando FreeFunc para liberar os segmentos. FreeFunc( ) é chamado tal como em InternalLoadSeg( ). **Nota:** esta função chama Close( ) para listas de segmentos overlay.

Entrada

seglist: lista de segmentos a ser removida.

FreeFunc: função chamada para liberar a memória.

Resultado

success: retorna quando tudo correu bem (uma vez que esta rotina pode fechar arquivos). Também retorna FALSE caso seglist seja NULL.

Bugs

A rotina na verdade deveria utilizar indicadores do tipo tags.

IoErr

Nome

IoErr: retorna informações adicionais do sistema.

Sinopse

```
error = IoErr( )
D0
LONG IoErr(void)
```

Função

A maioria das rotinas de I/O retornam um zero a fim de indicar a ocorrência de um erro. Quando isto acontece, você pode chamar IoErr( ) com o objetivo de obter maiores informações. IoErr( ) também é utilizado por algumas rotinas para passar de volta resultados secundários. Note que não é possível prever que valor IoErr( ) retornará após uma operação bem sucedida, a menos que o retorno seja especificado pela função.

Resultado

error: inteiro.

Veja também: Fault( ), PrintFault( ), SetIoErr( ).

IsFileSystem

Nome

IsFileSystem: confirma se um handler do DOS é um filesystem.

Sinopse

```
result = IsFileSystem(name)
D0                    D1
BOOL IsFileSystem(char *)
```

Função

O retorno desta rotina confirma se o dispositivo especificado é um filesystem ou não. Um filesystem suporta o armazenamento de informações em arquivos separados, além de poder suportar subdiretórios, embora isto não seja obrigatório. Se o filesystem não suportar este novo packet, IsFileSystem( ) utilizará Lock(":",...) como um indicador.

Entrada

name: nome do dispositivo em questão, sem ":" no final.

Resultado

result: flag que indica se o dispositivo é um filesystem.

IsInteractive

Nome

IsInteractive: descobre se um arquivo é um terminal virtual.

Sinopse

```
status = IsInteractive( file )
D0                      D1
BOOL IsInteractive(BPTR)
```

Função

O valor de retorno *status* indica se o arquivo associado com o filehandle *file* está conectado a um terminal virtual.

Entrada

file: ponteiro BCPL para o filehandle.

Resultado

status: booleano.

LoadSeg

Nome

LoadSeg: carrega por partes um arquivo carregável na memória.

Sinopse

```
seglist = LoadSeg( name )
D0              D1
```

Função

O arquivo especificado por *name* deverá ser um módulo de carregamento produzido pelo linker. LoadSeg( ) efetuará uma carga dispersa dos segmentos CODE, DATA e BSS na memória, encadeando os segmentos com BPTRs em suas primeiras palavras. O fim da cadeia é indicado por um zero. O arquivo pode conter qualquer número de segmentos. Toda a realocação necessária será manipulada por LoadSeg( ). Quando ocorrer um evento, quaisquer blocos carregados serão removidos e um resultado NULL retornará.

Caso o módulo seja carregado corretamente, a saída será um ponteiro para o início da lista de blocos. O código carregado poderá ser removido através de uma chamada à função UnLoadSeg( ).

Entrada

name: ponteiro para um string terminado por NULL.

Resultado

seglist: ponteiro BCPL para uma lista de segmentos.

Lock

Nome

Lock: ativa um lock sobre um diretório ou arquivo.

Sinopse

```
lock = Lock( name, acessMode )
D0             D1        D2
BPTR Lock(char *, LONG)
```

Função

Um lock do sistema de arquivo sobre o arquivo ou diretório especificado por *name* será retornado por esta rotina, se possível.

Se o *accessMode* for ACCESS_READ, o lock será um lock compartilhado para leitura, se o *accessMode* for ACCESS_WRITE, será um lock exclusivo para gravação. Não utilize valores aletórios na definição do modo.

Se a execução de Lock( ) falhar (isto é, se não for possível obter um lock sobre o arquivo ou diretório), a função retornará um zero.

Não é aconselhável assumir qualquer hipótese quanto ao formato interno de um lock, nem tampouco tentar utilizar os campos fl_link ou fl_Access.

Entrada

name: ponteiro para um string terminado por NULL.

accessMode: inteiro.

Resultado

lock: ponteiro BCPL para um lock.

LockDosList

Nome

LockDosList: ativa um lock sobre a lista especificada do DOS para utilizá-la.

Sinopse

```
dlist = LockDosList(flags)
D0              D1
struct DosList *LockDosList(ULONG)
```

Função

Esta rotina ativa um lock sobre a lista de dispositivos do DOS como preparativo para percorrer a lista.

Se a lista estiver "ocupada" (busy), a rotina não retornará até que a lista esteja disponível. Esta rotina pode ser "aninhada": é lícito chamá-la múltiplas vezes, desde que depois você desative os locks o mesmo número de vezes. Note que a *dlist* retornada NÃO é uma entrada válida, mas sim um valor especial. A fim de preservar a compatibilidade com a versão 1.3 do DOS, esta chamada também executa um Forbid( ) – este procedimento provavelmente será eliminado no futuro. O ativamento do lock sobre esta lista através do Forbid( ) da versão 1.3 apresentava condições restritivas. O ponteiro retornado por LockDosList( ) NÃO é um ponteiro DosList real – você deverá utilizar uma das chamadas do tipo DosEntry (tal como NextDosEntry( )) a fim de obter os ponteiros reais para as estruturas DosList, passando o valor retornado por LockDosList( ) como *dlist*.

Entrada

flags: flags especificando sobre que tipos de nós o lock deverá ser ativado.

Resultado

dlist: ponteiro para o início da lista. Lembre-se de que este NÃO é um nó válido.

LockRecord

Nome

LockRecord: ativa um lock sobre parte de um arquivo.

Sinopse

```
success = LockRecord(fh,offset,lenght,mode,timeout)
D0                   D1   D2     D3    D4     D5
ULONG LockRecord(BPTR,ULONG,ULONG,ULONG,ULONG)
```

Função

Esta rotina ativa um lock sobre uma parte de um arquivo, a fim de permitir acesso exclusivo à mesma. O contador *timeout* corresponde ao tempo de espera em "tiques" do AmigaDOS (50 por segundo) para que o registro esteja disponível.

Os modos válidos para o argumento *mode* são os seguintes:

REC_EXCLUSIVE

REC_EXCLUSIVE_SHARED

REC_SHARED

REC_SHARED_IMMED

Nos modos IMMED, o intervalo especificado por *timeout* é ignorado. Os locks obtidos por LockRecord( ) são ligados ao filehandle que foi utilizado para criá-los. O mesmo filehandle pode obter qualquer número de locks exclusivos sobre o mesmo registro, por exemplo. Estes locks são cooperativos, afetando somente outras rotinas ou programas que chamarem LockRecord( ).

Entrada

fh: filehandle por meio do qual deverá ser ativado um lock sobre o registro.

offset: posição inicial do registro.

lenght: tamanho do registro em bytes.

mode: especificação do tipo de lock.

timeout: contador de intervalo de tempo em "tiques". 0 é considerado válido.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Nas versões 2.0 a 2.02 do AmigaDOS (dos.library V36), LockRecord( ) só funcionava no ramdisk. Qualquer tentativa de ativar um lock sobre registros num filesystem de disco causava a perda de controle do sistema. Este defeito foi corrigido na versão V37.

LockRecords

Nome

LockRecords: ativa locks sobre uma série de registros.

Sinopse

```
success = LockRecords(record_array,timeout)
D0                            D1          D2
BOOL LockRecords(struct RecordLock *, ULONG)
```

Função

Esta rotina ativa locks sobre vários registros de um arquivo, a fim de permitir acesso exclusivo aos mesmos. O argumento *timeout* corresponde ao período de espera em "tiques" do AmigaDOS para que os registros estejam disponíveis. O tempo de espera se aplica a cada tentativa de ativar um lock sobre cada registro existente na lista. É recomendável ativar os locks sobre um determinado grupo de registros sempre na

mesma ordem, a fim de reduzir a possibilidade de ocorrência de um lock sem saída. O array das estruturas RecordLock é terminado com uma entrada rec_FH igual a NULL.

Entrada

record_array: lista de registros sobre os quais deverão ser ativados os locks.

timeout: contador do intervalo de tempo. 0 é aceito.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Veja também: LockRecord( ).

MakeDosEntry

Nome

MakeDosEntry: cria uma estrutura DosList (V36).

Sinopse

```
newdlist = MakeDosEntry(name, type)
D0                      D1     D2
struct DosList *MakeDosEntry(UBYTE *, LONG)
```

Função

Esta rotina cria uma estrutura DosList, inclusive alocando um nome e terminando apropriadamente (com NULL) o string BSTR. MakeDosEntry( ) também inicializa o campo dol_Type e atribui o valor 0 a todos os outros campos. Esta rotina deve ser eliminada e substituída pela passagem de um valor para a função AllocDosObject( ).

Entrada

name: nome para o nó de dispositivo/volume/associação.

type: tipo do nó.

Resultado

newdisk: a nova entrada de dispositivo ou NULL.

MakeLink

Nome

MakeLink: cria uma ligação de filesystems (V36).

Sinopse

```
success = MakeLink( name, dest, soft )
D0                  D1    D2    D3
BOOL MakeLink( UBYTE *, LONG, LONG )
```

Função

Cria uma ligação (link) do filesystem *name* para *dest*. Para se criar soft links, *dest* deverá ser um ponteiro para o string descritivo do path (terminado por NULL). Para a criação de hard links, é necessário que *dest* seja um lock (BPTR). *Soft* deverá ser FALSE para hard links, ou diferente de zero em caso contrário.

Os soft links são resolvidos em tempo de acesso através de uma combinação do filesystem (retornando-se ERROR_IS_SOFT_LINK para o DOS), e pelo DOS (utilizando-se ReadLink( ) para resolver quaisquer links que estejam ativos).

Os hard links são resolvidos pelo filesystem em questão. Uma série de hard links para um arquivo são equivalentes ao próprio arquivo. Se um dos links ou a entrada original para o arquivo forem deletados, os dados serão preservados até que não haja mais nenhum link.

Entrada

name: nome do link a ser criado.

dest: CPTR para string do path (soft link), ou lock BPTR (hard link).

soft: FALSE para hard links, não zero para soft links.

Resultado

success: booleano.

Bugs

No dos.library V36, os soft links não funcionavam no filesystem da ROM. Este defeito foi corrigido na versão V37.

Veja também: ReadLink( ), Open( ), Lock( ).

MatchEnd

Nome

MatchEnd: libera a área de armazenamento alocada pelas funções MatchFirst( ) e MatchNext( ).

Sinopse

```
MatchEnd(AnchorPath)
              D1
VOID MatchEnd(struct AnchorPath *)
```

Função

Esta rotina retorna toda a área de armazenamento associada com uma operação de busca.

Entrada

AnchorPath: âncora utilizada por MatchFirst( )/MatchNext( ). Deverá ser alinhada por palavra longa.

MatchFirst

Nome

MatchFirst: localiza o arquivo que combina com o padrão especificado.

Sinopse

```
error = MatchFirst(pat, AnchorPath)
D0                 D1         D2
BOOL MatchFirst(UBYTE *, struct AnchorPath *)
```

Função

Esta rotina localiza o primeiro arquivo ou diretório que combina com o padrão especificado. À função MatchFirst( ) deverão ser passados o padrão *pat* (não é preciso passá-lo através de ParsePattern( ), MatchFirst( ) se encarregará disto para você) e a estrutura de controle *AnchorPath*. Normalmente, MatchFirst( ) inicializará a estrutura AnchorPath para você, e retornará o primeiro arquivo encontrado que combine com o seu padrão, ou um código de erro. Observe que MatchFirst( )/MatchNext( ) diferem das demais funções do DOS por retornarem 0 como indicador de sucesso de execução ou o código de erro (veja o arquivo dos/dos.h), ao invés de permitirem que a aplicação obtenha o código de erro a partir de IoErr( ).

Na observação do resultado de MatchFirst( )/MatchNext( ), o campo ap_Info da estrutura AnchorPath conterá os resultados de uma operação Examine( ) sobre o objeto. Normalmente você obtém o nome do objeto a partir de fib_FileName, e o diretório onde o mesmo se encontra em ap_Current->an_Lock. Para acessar este objeto, a seqüência de operações usual consiste em efetuar um CurrentDir( ) temporário sobre o lock, executar alguma operação com o arquivo/diretório, e então comandar um CurrentDir( ) de volta para o seu diretório original. Este procedimento assegura que você estará afetando o objeto correto mesmo quando existirem dois volumes com o mesmo nome no sistema. Você pode utilizar ap_Buf (com ap_Strlen) para obter um nome a ser reportado para o usuário.

Os padrões são razoavelmente abrangentes, e sua capacidade se aproxima à dos padrões de expressão regulares do sistema Unix/grep. Eis alguns dos tokens disponíveis:

?	Compara com um único caractere.
#	Compara com a expressão seguinte 0 ou mais vezes.

(ab|cd)	Compara com qualquer um dos itens separados por '|'.
~	Nega a expressão seguinte. Este padrão efetua uma comparação com todos os strings que *não* combinam com a expressão (ex: aka ~(foo) compara com todas os strings que não sejam precisamente "foo").
[abc]	Classe de caracteres: compara com qualquer um dos caracteres da classe.
a-z	Faixa de caracteres (somente dentro de uma classe de caracteres).
%	Compara sempre 0 caracteres (útil em "(foo|bar|%)").
*	Sinônimo de '#?', não disponível por default na versão 2.0. Disponível como uma opção que pode ser especificada.

O termo "expressão" na lista acima se refere a um caractere simples (ex: "#?"), a uma alternação (ex: "#(ab|cd|ef)"), ou a uma classe de caracteres (ex:"#[a-zA-Z]").

Entrada

pat: padrão para busca.

AnchorPath: estrutura para armazenar informações relativas à busca. Deverá ser alinhada por palavra longa.

Resultado

error: 0 para indicar sucesso de execução, ou código de erro (o oposto da maioria das chamadas do DOS).

MatchNext

Nome

MatchNext: localiza o próximo arquivo ou diretório que combina com o padrão especificado.

Sinopse

```
error = MatchNext(AnchorPath)
D0                D1
BOOL MatchNext(struct AnchorPath *)
```

Função

Esta rotina localiza o próximo arquivo ou diretório que combina com o padrão especificado. Veja o arquivo dos/dosasl.h para maiores informações. Vários bits dos flags permitem à aplicação controlar a operação de MatchNext( ).

Entrada

AnchorPath: estrutura para armazenar informações relativas à busca. Deverá ser alinhada por palavra longa.

Resultado

error: 0 para indicar sucesso de execução, ou código de erro (o oposto da maioria das chamadas do DOS).

MatchPattern

Nome

MatchPattern: procura por um string que combine com o padrão especificado.

Sinopse

```
match = MatchPattern(pat, str)
D0                   D1   D2
BOOL MatchPattern(UBYTE *, UBYTE *)
```

Função

Esta rotina procura por um string que combine com um padrão especificado. O padrão deverá ser um string resultante da saída da função ParsePattern( ). A comparação levará em conta a diferença entre letras maiúsculas e minúsculas. Você deverá ter pelo menos 1.500 bytes livres no stack para chamar esta função, uma vez que MatchPattern( ) utiliza uma grande profundidade de recursão.

Entrada

pat: string de padrões especial para comparação, no formato retornado por Parse-Pattern( ).

str: string a ser comparado com o padrão.

Resultado

match: sucesso ou falha na comparação do padrão. Em caso de falha, IoErr( ) retornará 0 ou ERROR_TO_MANY_LEVELS (V37).

MatchPatternNoCase

Nome

MatchPatternNoCase: procura por um string que combine com o padrão especificado (V36).

Sinopse

```
match = MatchPatternNoCase(pat, str)
D0                         D1    D2
BOOL MatchPatternNoCase(UBYTE *, UBYTE *)
```

Função

Esta rotina procura por um string que combine com um padrão especificado. O padrão deverá ser um string resultante da saída da função ParsePattern( ). A rotina leva em conta a diferença entre letras maiúsculas e minúsculas.

Nota: Esta rotina é altamente recursiva. É necessário que você tenha mais do que 1.500 bytes livres no stack para chamá-la (ela abandonará sua recursão e retornará um código de erro antes de atingir um nível mais profundo do que este). Usualmente, este espaço é suficiente para cerca de 100 níveis de profundidade de #, (, ~ e etc.

Entrada

pat: string de padrões especial para comparação, no formato retornado pela função ParsePatternNoCase( ).

str: string a ser comparado com o padrão.

Resultado

match: sucesso ou falha na comparação do padrão. Em caso de falha, IoErr( ) retornará 0 ou ERROR_TOO_MANY_LEVELS (a partir da versão V37 do dos.library – antes disto não era feita a verificação do stack).

Veja também: ParsePatternNoCase( ), MatchPattern( ), MatchFirst( ), MatchNext( ).

MaxCli

Nome

MaxCli: retorna o mais alto número de processo CLI em uso.

Sinopse

```
number = MaxCli( )
D0
LONG MaxCli(void)
```

Função

Esta rotina retorna o mais alto número de processo CLI que possa estar em uso. Os números dos processos CLI podem ser reutilizados, e normalmente são os mais baixos possíveis. Para localizar todos os CLIs, proceda a uma verificação usando FindCliProc( ) de 1 até MaxCli( ). O número retornado por MaxCli( ) pode mudar à medida em que os processos são criados e destruídos.

Resultado

number: o mais alto número do CLI que pode estar em uso.

NameFromFH

Nome

NameFromFH: obtém o nome de um filehandle aberto (V36).

Sinopse

```
success = NameFromFH(fh, buffer, len)
D0                   D1     D2     D3
BOOL NameFromFH(BPTR, char *, LONG)
```

Função

Esta rotina retorna um path inteiramente qualificado para o filehandle. Em nenhuma hipótese NameFromFH( ) gravará no buffer mais caracteres do que os especificados por *len.* O nome será terminado por NULL. *Veja* NameFromLock para maiores informações.

Entrada

fh: lock do objeto a ser examinado.
buffer: buffer para armazenar o nome.
len: tamanho do buffer.

Resultado

success: indicador de sucesso/falha de execução da rotina.

NameFromLock

Nome

NameFromLock: retorna o nome de um objeto sobre o qual esteja ativado um lock.

Sinopse

```
success = NameFromLock(lock, buffer, len)
D0                     D1     D2     D3
BOOL NameFromLock(BPTR, char *, LONG)
```

Função

Esta rotina retorna um path inteiramente qualificado para o lock. Em nenhuma hipótese NameFromLock( ) gravará no buffer mais caracteres do que os especificados por *len.* O nome será terminado por NULL. Nota: Se o volume não estiver montado, o sistema solicitará que você o faça (a menos, é claro, que você tenha definido pr_WindowPtr com -1). Se o volume não estiver montado ou inserido, a função retornará um erro. Passando um lock igual a NULL, a rotina retornará "SYS:". Caso o buffer seja muito pequeno para comportar o nome, NameFromLock( ) retornará um erro e IoErr( ) retornará ERROR_LINE_TOO_LONG.

Entrada

lock: lock sobre o objeto a ser examinado.

buffer: buffer para armazenar o nome.

len: tamanho do buffer.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

A rotina deveria retornar o nome do volume utilizado para dar o boot, e não SYS:, quando é passado um lock igual a NULL.

NewLoadSeg

Nome

NewLoadSeg: versão aperfeiçoada de LoadSeg( ).

Sinopse

```
seglist = NewLoadSeg(file, tags)
D0              D1      D2
BPTR NewLoadSeg(UBYTE *, struct TagItem *)
seglist = NewLoadSegTagList(file, tags)
D0                              D1    D2
BPTR NewLoadSegTagList (UBYTE *, struct TagItem *)
seglist = NewLoadSegTags(file, ...)
BPTR NewLoadSegTags(UBYTE *, ...)
```

Função

Esta rotina executa um LoadSeg( ) de um arquivo, e toma ações adicionais baseadas nos indicadores (tags) fornecidos. NewLoadSeg( ) limpa porções não usadas dos hunks CODE, DATA e BSS (o mesmo se aplica a InternalLoadSeg( ) e LoadSeg( )). Esta chamada não retorna a lista de segmentos (*seglist*) tanto no registrador D0 quanto em D1, como LoadSeg( ) faz. Além disso, não há tags definidos para NewLoadSeg( ) no momento.

Entrada

file: nome do arquivo a ser carregado.

tags: ponteiro para o array TagItem.

Resultado

seglist: lista de segmentos carregada, ou NULL em caso de erro.

NextDosEntry

Nome

NextDosEntry: obtém a próxima entrada na lista de dispositivos do DOS (V36).

Sinopse

```
newdlist = NextDosEntry(dlist,flags)
D0                      D1      D2
struct DosList *NextDosEntry(struct DosList *,ULONG)
```

Função

Localiza no Dos List a próxima entrada do tipo indicado. É indispensável que você ative um lock sobre os tipos de entrada que estiver procurando. A rotina retornará NULL caso não haja mais nenhuma entrada do tipo procurado na lista.

Entrada

dlist: a entrada de dispositivo atual.

flags: tipo de entrada procurado.

Resultado

newdlist: a próxima entrada de dispositivo do tipo procurado, ou NULL.

Open

Nome

Open: abre um arquivo para entrada ou saída.

Sinopse

```
file = Open( name, accessMode )
D0           D1        D2
BPTR Open(char *, LONG)
```

Função

O arquivo especificado por *name* é aberto e um filehandle é retornado. Se *accessMode* for MODE_OLDFILE, um arquivo já existente será aberto para leitura ou gravação. Se o valor for MODE_NEWFILE, um novo arquivo será criado para gravação. MODE_READ_WRITE abre um arquivo com um lock compartilhado, porém criando-o se este não existir. Os tipos desta função estão documentados nos arquivos include libraries/dos.h e dos/dos.h.

Name pode ser o nome de um arquivo (opcionalmente precedido pelo nome de um dispositivo), um dispositivo único como NIL:, uma especificação de janela como CON: ou RAW: seguida pelos parâmetros para a janela, ou então um asterisco (*) repre-

sentando a janela atual. Observe que a partir do dos.library V36, a especificação * tornou-se obsoleta, devendo ser substituída por CONSOLE:. Se por qualquer motivo o arquivo não puder ser aberto, o valor retornado por Open( ) será zero e um código de erro secundário estará disponível através de uma chamada à rotina IoErr( ).

Entrada

name: ponteiro para um string terminado por NULL.

accessMode: inteiro.

Resultado

file: ponteiro BCPL para um filehandle.

OpenFromLock

Nome

OpenFromLock: abre um arquivo sobre o qual tenha sido ativado um lock.

Sinopse

```
fh = OpenFromLock(lock)
D0                D1
BPTR OpenFromLock(BPTR)
```

Função

Dado um lock, esta rotina efetua a abertura desse lock. Caso a operação de abertura seja bem sucedida, o lock estará efetivamente abandonado e não deverá mais ser utilizado, nem tampouco desativado com a função UnLock( ). Se a execução da chamada falhar, o lock ainda poderá ser utilizado. O lock associado com o arquivo internamente é do mesmo modo de acesso que o lock que você abandonou – o lock compartilhado é similar a MODE_OLDFILE, e o exclusivo é similar a MODE_NEW-FILE.

Entrada

lock: lock sobre o objeto a ser aberto.

Resultado

fh: filehandle aberto ou NULL em caso de erro.

Bugs

A documentação original do dos.library V36 afirmava incorretamente que para esta rotina também deveria ser passado um parâmetro *Mode*. Os protótipos e pragmas desta versão estavam igualmente incorretos.

Output

Nome

Output: identifica o filehandle inicial de saída do programa.

Sinopse

```
file = Output( )
D0
BPTR Output(void)
```

Função

Output( ) é utilizado a fim de identificar o stream inicial de saída alocado quando o programa foi iniciado. Nunca feche o filehandle retornado por Output( ).

Resultado

file: ponteiro BCPL para um filehandle.

ParentDir

Nome

ParentDir: obtém o diretório pai de um arquivo.

Sinopse

```
newlock = ParentDir( lock )
D0              D1
BPTR ParentDir(BPTR)
```

Função

O argumento *lock* é associado a um determinado arquivo ou diretório. A função ParentDir( ) retorna *newlock*, que é associado ao diretório pai de *lock*.

Se você tentar obter ParentDir( ) a partir da raiz do filesystem atual, esta rotina retornará um lock NULL (0). Note que este lock 0 representa a raiz do filesystem a partir do qual foi dado o boot no sistema, o que vem a ser, na prática, o pai de todas as outras raízes de filesystems.

Entrada

lock: ponteiro BCPL para um lock.

Resultado

newlock: ponteiro BCPL para um lock.

ParentOfFH

Nome

ParentOfFH: retorna um lock sobre o diretório pai de um arquivo.

Sinopse

```
lock = ParentOfFH(fh)
D0                D1
BPTR ParentOfFH(BPTR)
```

Função

Esta rotina retorna um lock compartilhado sobre o diretório pai do filehandle.

Entrada

fh: filehandle do qual você deseja o pai.

Resultado

lock: lock sobre o diretório pai do filehandle, ou NULL em caso de erro.

ParsePattern

Nome

ParsePattern: cria um string codificada para uso com a função MatchPattern( ).

Sinopse

```
IsWild = ParsePattern(Source,Dest,DestLenght)
D0                    D1     D2      D3
LONG ParsePattern(UBYTE *, UBYTE *, LONG)
```

Função

Esta rotina codificada (converte em códigos especiais) um padrão para ser utilizado com a função MatchPattern( ), indicando também se existe algum curinga no padrão (isto é, se o mesmo pode ser comparado com mais de um item). Observe que *dest* deverá ter pelo menos o dobro do tamanho de *source*.

Entrada

source: string do curinga (não analisado sintaticamente) a procurar.

dest: string de saída, receberá a versão tokenizada da entrada.

DestLenght: tamanho disponível em *dest*, deverá ser no mínimo o dobro do tamanho de *source*, mais 2 bytes.

Resultado

IsWild: 1 significa que existem curingas no padrão, 0 significa que não há curingas no padrão, e -1 indica um overflow (ou seja, que foi excedida a capacidade de armazenamento do buffer) ou a ocorrência de algum outro tipo de erro.

Bugs

Esta função deveria atribuir um valor útil a IoErr( ) (atualmente nenhum valor é retornado) em caso de erro.

ParsePatternNoCase

Nome

ParsePatternNoCase: cria um string codificada para uso com a função MatchPatternNoCase( ).

Sinopse

```
IsWild = ParsePatternNoCase(Source, Dest, DestLenght)
D0                          D1      D2        D3
LONG ParsePatternNoCase(UBYTE *, UBYTE *, LONG)
```

Função

Esta rotina codifica (converte em códigos especiais) um padrão para uso com a função MatchPatternNoCase( ), indicando também se existe algum curinga no padrão (isto é, se o mesmo pode ser comparado com mais de um item). Observe que *dest* deverá ter pelo menos o dobro do tamanho de *source*.

Para a descrição dos padrões, veja ParsePattern( ).

Entrada

source: string do curinga (não analisado sintaticamente) a procurar.
dest: string de saída, receberá a versão codificada da entrada.
DestLenght: tamanho disponível em *dest*, deverá ser no mínimo o dobro do tamanho de *source*, mais 2 bytes.

Resultado

IsWild: 1 significa que existem curingas no padrão.

0 significa que não há curingas no padrão.

-1 indica que foi excedida a capacidade de armazenamento do buffer ou ocorreu algum outro tipo de erro.

Bugs

Esta função deveria atribuir um valor útil a IoErr( ) (atualmente nenhum valor é retornado) em caso de erro.

Veja também: ParsePattern( ), MatchPatternNoCase( ), MatchFirst( ), MatchNext( ).

PathPart

Nome

PathPart: retorna um ponteiro para o final do penúltimo componente de um path.

Sinopse

```
fileptr = PathPart( path )
D0                  D1
UBYTE *PathPart( UBYTE * )
```

Função

Esta rotina retorna um ponteiro para o caractere após o penúltimo componente da especificação de um path, que deverá ser normalmente o nome do diretório. Se houver apenas um componente, a rotina retornará um ponteiro para o início do string. A única diferença real entre esta função e FilePart( ) é o manuseio de "/".

Entrada

path: ponteiro para um string descritivo de um path. Pode ser relativo ao disco ou ao diretório corrente.

Resultado

fileptr: ponteiro para o final do penúltimo componente do path.

Exemplo

PathPart("xxx:yyy/zzz/qqq") retornaria um ponteiro para a última "/".

PathPart("xxx:yyy") retornaria um ponteiro para o primeiro *y*.

Veja também: FilePart( ), AddPart( ).

PrintFault

Nome

PrintFault: retorna o texto associado a um código de erro do DOS.

Sinopse

```
success = PrintFault(code, header)
D0                   D1      D2
BOOL PrintFault(LONG, UBYTE *)
```

Função

Esta rotina obtém o texto da mensagem de erro correspondente ao código de erro especificado. Isto é semelhante ao que faz a função Fault( ), com a diferença de que PrintFault( ) envia os dados para o canal de saída default, efetuando uma saída bufferizada. O valor retornado por IoErr( ) é o mesmo do código passado para esta rotina.

Entrada

code: código de erro.

header: cabeçalho a ser exibido antes do texto do código de erro.

Resultado

success: indicador de falha/sucesso de execução da rotina.

PutStr

Nome

PutStr: grava um string na saída default (bufferizado).

Sinopse

```
count = PutStr(str)
D0              D1
LONG PutStr(UBYTE *)
```

Função

Esta rotina grava um string não formatado na saída default. O caractere de newline (\n) não será acrescentado ao string. A ocorrência de qualquer erro será retornada. Esta rotina é bufferizada.

Entrada

str: string (terminado por NULL) a ser gravado na saída default.

Resultado

error: 0 para indicar sucesso de execução da rotina, -1 para indicar a ocorrência de um erro. Este retorno é o oposto da maioria das funções do DOS.

Read

Nome

Read: lê bytes de dados de um arquivo.

Sinopse

```
actualLenght = Read( file, buffer, lenght )
D0                   D1     D2      D3
LONG Read(BPTR, void *, LONG)
```

Função

A cópia de dados pode ser efetuada utilizando-se uma combinação das funções Read( ) e Write( ). Read( ) lê os bytes de informações de um arquivo aberto (aqui representado pelo argumento *file*) no buffer especificado. O argumento *lenght* corresponde ao tamanho desse buffer.

O valor retornado pela função indica o tamanho da informação efetivamente lida.

Desse modo, quando *actualLenght* for maior do que zero, o valor de *actualLenght* equivalerá ao número de caracteres lidos. Normalmente a função Read( ) tentará preencher o buffer que você especificou antes de retornar. Um valor igual a zero significa que um EOF (final de arquivo) foi encontrado. Erros são indicados por um retorno igual a -1. Esta rotina não é bufferizada (a requisição passa direto para o filesystem). O I/O bufferizado é mais eficiente para efetuar pequenas operações de gravação e leitura, veja a descrição da função FGetC( ).

Entrada

file: ponteiro BCPL para um filehandle.
buffer: ponteiro para um buffer.
lenght: inteiro.

Resultado

actualLenght: inteiro.

ReadArgs

Nome

ReadArgs: analisa a sintaxe da linha de comando.

Sinopse

```
result = ReadArgs(template, array, rdargs)
D0                D1      D2      D3
struct RDArgs *ReadArgs(UBYTE *, LONG *, struct RDArgs *)
```

Função

ReadArgs( ) efetua a análise da sintaxe (parsing) de um string de argumentos, de acordo com um gabarito especificado. Embora normalmente obtenha os argumentos através da leitura do I/O bufferizado produzido por Input( ), esta rotina também pode ser orientada para analisar um string. É indispensável que uma chamada a ReadArgs( ) seja sucedida por uma chamada à função FreeArgs( ).

ReadArgs( ) analisa a linha de comandos confrontando-a com um gabarito (*template*) que é passado para a rotina. O gabarito especifica as diferentes opções para a linha de comando, bem como seus respectivos tipos. Um gabarito consiste numa lista de opções. As opções são descritas através de seus nomes por extenso, sempre que possível (por exemplo, "Quick" ao invés de "Q"). Abreviações também podem ser especificadas na forma "abrev=opção" (por exemplo, "Q=Quick").

As opções no gabarito deverão ser separadas por vírgulas. Para obter os resultados de ReadArgs( ), você deverá examinar o array de palavras longas passado para a rotina (uma entrada no array para cada opção do gabarito). O array deverá ser inicializado com os valores default apropriados, ou então com zeros, antes de ser passado para ReadArgs( ). O tipo da opção define o que efetivamente será inserido numa determinada entrada por ReadArgs( ). O default é um string (uma seqüência de caracteres sem espaços, ou delimitada por aspas, que serão cortadas por ReadArgs), neste caso, a entrada será um ponteiro.

As opções podem ser seguidas por modificadores que, dentre outras coisas, especificam o tipo da opção. Os modificadores são especificados sucedendo-se a opção com uma barra (/) e um caractere representando o modificador. Múltiplos modificadores podem ser especificados mediante o uso de múltiplas barras. Os modificadores válidos são

/S	Switch. É tratado como uma variável booleana que será setada se o nome da opção aparecer na linha de comandos. A entrada é booleana (0 para não setado, não zero para setado).
/K	Palavra-chave. Isto significa que a opção não será preenchida a menos que uma palavra-chave (keyword) apareça. Por exemplo: se o gabarito for "Name/K", a menos que "Name=<string>" ou "Name <string>" apareça na linha de comando, a opção Name não será preenchida.
/N	Número. O parâmetro é considerado um número decimal e será convertido por ReadArgs( ). Caso um número inválido seja especificado, a execução de ReadArgs( ) não terá sucesso. A entrada deverá ser um ponteiro para um número em forma de palavra longa (é desse modo que você sabe se um número foi especificado).
/T	Toggle. Este modificador é similar a um switch, só que alterna o valor booleano.
/A	Obrigatório. Significa que é indispensável fornecer um valor para a palavra-chave durante o processamento da linha de comandos, caso contrário será retornado um erro.
/F	Restante da linha. Quando especificado, este modificador faz com que todo o restante da linha de comandos seja tomado como o parâmetro da opção, mesmo que apareçam outras palavras-chave no restante da linha.

/M — Múltiplos strings. Significa que o argumento poderá utilizar qualquer número de strings, retornando-os como um array de strings. Quaisquer argumentos que não sejam considerados parte de outra opção serão incluídos nesta opção. Apenas um /M deverá aparecer em um gabarito. Por exemplo: num gabarito "Dir/M,All/S" a linha de comando "foo bar all qwe" setará o booleano "all" e retornará um array consistindo de "foo", "bar" e "qwe". A entrada no array será um ponteiro para um array de ponteiros string, sendo que o último deles será um NULL.

Existe uma interação entre os parâmetros /M e /A. Se houver parâmetros /A não preenchidos após a análise da linha, ReadArgs( ) pegará os strings do final da última lista de parâmetros /M para preenchê-los. Este procedimento é utilizado em mensagens como Copy ("From/A/M,To/A"). ReadArgs( ) retornará uma estrutura RDArgs caso sua execução seja bem sucedida. Esta ação serve como uma "âncora" para possibilitar a FreeArgs( ) liberar a memória associada. Você também pode passar uma estrutura RDArgs a fim de controlar a operação de ReadArgs( ) (normalmente você passa NULL como valor para este parâmetro, e ReadArgs( ) aloca uma estrutura para você).

Este procedimento permite que você especifique diferentes fontes para os argumentos, fornecendo o seu próprio espaço de buffer para armazenamento temporário e para exibir um texto de ajuda (help) estendido. *Veja* dos/rdargs.h para maiores informações sobre este tópico.

Nota: Se passar uma estrutura RDArgs para a rotina, você ainda terá que chamar FreeArgs( ) a fim de liberar a memória utilizada para o armazenamento, mas competirá a você mesmo liberar a estrutura RDArgs.

Entrada

template: string de formatação.

array: array de palavras longas para os resultados (uma para cada entrada no gabarito).

rdargs: estrutura RDArgs opcional. A função AllocDosObject( ) deverá ser utilizada para alocar esta estrutura, caso você deseje passá-la para a rotina.

Resultado

result: estrutura RDArgs, ou NULL em caso de falha na execução da rotina.

Bugs

Algumas combinações de argumentos não funcionam no dos.library V36. Use esta função apenas com a versão V37.

ReadItem

Nome

ReadItem: lê um nome/argumento da linha de comandos.

Sinopse

```
value = ReadItem(buffer, maxchars, input)
D0                D1        D2        D3
LONG ReadItem(UBYTE *, LONG, struct CHSource *)
```

Função

ReadItem( ) lê uma "palavra" através da função Input( ) (bufferizado) ou via CHSource, se a mesma não for NULL (*veja* o arquivo dos/rdargs.h para maiores informações a respeito). Esta rotina é capaz de manipular aspas e algumas substituições indicadas por "*". *Veja* o arquivo dos/dos.h para obter uma lista dos valores retornados por ReadItem(ITEM_XXXX). A "palavra" deverá ser delimitada por espaços em branco, aspas ou EOF (final de arquivo). ReadItem( ) sempre retrocede a leitura do último dado lido (o equivalente a UnGetC(fh,-1)), isto permite à aplicação que chama a rotina descobrir qual era o caractere utilizado como terminador.

Entrada

buffer: buffer no qual a palavra deverá ser armazenada.

maxchars: tamanho do buffer.

input: entrada CHSource, ou NULL para utilizar FGetC(Input( )).

Resultado

value: veja o arquivo dos/dos.h para conhecer os valores de retorno.

ReadLink

Nome

ReadLink: lê o path para o soft link de um filesystem (V36).

Sinopse

```
success = ReadLink(port, lock, path, buffer, size)
D0                 D1    D2    D3      D4      D5
BOOL ReadLink(struct MsgPort *,BPTR,UBYTE *,UBYTE *,ULONG)
```

Função

ReadLink( ) toma um par lock/nome (geralmente usados numa tentativa rejeitada para acessar um objeto com packets), e manda o filesystem localizar o soft link e o buffer associado utilizando o string de path modificado. Você poderá então reiniciar

o processo de resolução chamando a função GetDeviceProc( ) com o novo string de ReadLink( ).

Os soft links são resolvidos em tempo de acesso através de uma combinação do filesystem (retornando-se ERROR_IS_SOFT_LINK para o DOS) e pelo DOS (utilizando-se ReadLink( ) para resolver quaisquer links que estejam ativos).

Entrada

port: porta de mensagens do filesystem.

lock: lock ao qual este path é relativo no filesystem.

path: path que causou o ERROR_IS_SOFT_LINK.

buffer: ponteiro para o buffer que irá receber o novo path do handler.

size: tamanho do buffer.

Resultado

success: booleano.

Bugs

No dos.library V36, os soft links não funcionavam no filesystem da ROM. Este defeito foi corrigido na versão V37.

Veja também: MakeLink( ), Open( ), Lock( ), GetDeviceProc( ).

Relabel

Nome

Relabel: muda o nome de um volume.

Sinopse

```
success = Relabel(volumename, name)
D0                      D1        D2
BOOL Relabel(char *,char *)
```

Função

Esta rotina modifica o nome de um volume, nos filesystems capazes de suportá-lo.

Entrada

volumename: nome completo do dispositivo a renomear (com ":").

newname: novo nome a ser aplicado ao dispositivo (sem ":").

Resultado

success: indicador de sucesso/falha de execução da rotina.

RemAssignList

Nome

RemAssignList: remove uma entrada de uma associação de múltiplos diretórios.

Sinopse

```
success = RemAssignList(name, lock)
D0                      D1    D2
BOOL RemAssignList(char *,BPTR)
```

Função

RemAssignList( ) remove uma entrada de uma associação de múltiplos diretórios. A entrada removida será a primeira na qual o retorno da função SameLock( ) com *lock* especificar que dois locks estavam sobre o mesmo objeto. O lock para a entrada na lista será desativado (e não aquele passado para a rotina).

Entrada

name: nome do dispositivo do qual o lock deverá ser removido (sem ":" no final).

lock: lock associado ao objeto a ser removido da lista.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Veja também: Lock( ), AssignLock( ), AssignPath( ), AssignLate( ), DupLock( ), AssignAdd( ), UnLock( ).

RemDosEntry

Nome

RemDosEntry: remove uma entrada da lista de dispositivos do DOS.

Sinopse

```
success = RemDosEntry(dlist)
D0                    D1
BOOL RemDosEntry(struct DosList *)
```

Função

Esta rotina remove uma entrada da lista de dispositivos do DOS (DOS List). Atente para o fato de que a memória associada com a entrada NÃO será liberada. **Nota:** É indispensável ativar um lock sobre o DOS List com os flags apropriados antes de se chamar esta rotina.

Entrada

dlist: entrada na lista de dispositivos a ser removida.

Resultado

success: indicador de sucesso/falha de execução da rotina.

RemSegment

Nome

RemSegment: remove um segmento da lista residente.

Sinopse

```
success = RemSegment(segment)
D0                   D1
BOOL RemSegment(struct Segment *)
```

Função

Esta rotina remove um segmento residente da lista de segmentos residentes do DOS, efetuando todo o retorno de memória necessário. A execução desta rotina só será bem sucedida caso o seg_UC (*usecount*) seja 0.

Entrada

segment: o segmento a ser removido.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Rename

Nome

Rename: renomeia um diretório ou arquivo.

Sinopse

```
success = Rename( oldName, newName )
D0                D1       D2
BOOL Rename(char *, char *)
```

Função

Rename( ) tenta renomear o arquivo ou diretório especificado por *oldName* com o nome *newName*. Se o arquivo ou diretório *newName* já existir, a execução da rotina falhará e Rename( ) retornará um erro. Tanto *oldName* como *newName* poderão conter uma especificação de diretório. Neste caso, o arquivo será movido de um diretório para outro.

Nota: Rename( ) é incapaz de renomear um arquivo de um volume para outro.

Entrada

oldName: ponteiro para um string terminado por NULL.

newName: ponteiro para um string terminado por NULL.

Resultado

success: booleano.

ReplyPacket

Nome

ReplyPacket: devolve um packet para o processo que o enviou para você.

Sinopse

```
ReplyPacket(packet, result1, result2)
              D1       D2         D3
void ReplyPacket(struct DosPacket *, LONG, LONG)
```

Função

Esta função retorna um packet para o processo que enviou o mesmo para você. Além disso, ela põe o endereço da sua porta de mensagens (pr_MsgPort) em dp_Port. Assim, chamando ReplyPacket( ) novamente, a mensagem será enviada para você (esta função é utilizada para enviar packets de uma direção para outra entre dois processos). ReplyPacket( ) utiliza os resultados primário e secundário para definir os campos dp_Res1 e dp_Res2 do packet.

Entrada

packet: packet a ser retornado, assumindo-se que esteja corretamente definido.

result1: primeiro resultado.

result2: resultado secundário.

RunCommand

Nome

RunCommand: roda um programa utilizando o processo atual (V36).

Sinopse

```
rc = RunCommand(seglist, stacksize, argptr, argsize)
D0                D1         D2         D3        D4
LONG RunCommand(BPTR, ULONG, char *, ULONG)
```

Função

Esta rotina executa um comando no seu processo/CLI. *Seglist* pode ser em qualquer linguagem, inclusive programas BCPL. *Stacksize* é definido em bytes. *Argptr* é um string terminad por NULL, e *argsize* é o tamanho dess string. A função retorna o código de retorno com o qual o programa foi abandonado no registrador D0, ou -1 caso não tenha sido possível alocar o stack. RunCommand( ) também se encarrega de definir o filehandle de entrada atual de maneira que ReadArgs( ) possa ser usado no programa, e restaura o estado dos buffers antes de retornar. A rotina também atribui o valor retornado por GetArgStr( ), restaurando-o antes de retornar.

NOTA: A definição do filehandle foi acrescentada no dos.library versão V37.

Entrada

seglist: seglist de comandos a executar.
stacksize: número de bytes a serem alocados como espaço de stack.
argptr: ponteiro para o string de argumentos de comandos.
argsize: número de bytes do string.

Resultado

rc: código de retorno do comando executado. -1 indica falha de execução da rotina.

SameDevice

Nome

SameDevice: verifica se dois locks estão sobre uma partição do mesmo dispositivo (V37).

Sinopse

```
same = SameDevice(lock1, lock2)
D0              D1        D2
BOOL SameDevice( BPTR, BPTR )
```

Função

O retorno de SameDevice( ) determina quando dois locks se referem a partições que estão sobre o mesmo dispositivo físico (caso seja possível constatar essa condição). Isto pode ser útil para o desenvolvimento de rotinas de cópia capazes de tirar proveito do recurso de cópia assíncrona de múltiplos dispositivos.

Esta rotina existia no dos.library V36, mas sempre retornava zero.

Entrada

lock1, lock2: locks.

Resultado

same: especifica se os locks estão sobre o mesmo dispositivo, até onde o DOS seja capaz de determiná-lo.

SameLock

Nome

SameLock: verifica se dois locks estão sobre o mesmo objeto.

Sinopse

```
value = SameLock(lock1, lock2)
D0                D1        D2
LONG SameLock(BPTR, BPTR)
```

Função

Esta rotina compara dois locks, retornando LOCK_SAME se eles estiverem sobre o mesmo objeto, LOCK_SAME_HANDLER se estiverem sobre objetos diferentes no mesmo handler, e LOCK_DIFFERENT caso estejam em handlers diferentes.

Entrada

lock1: primeiro lock para comparação.

lock2: segundo lock para comparação.

Resultado

value: LOCK_SAME, LOCK_SAME_HANDLER ou LOCK_DIFFERENT.

Bugs

Você deverá fazer verificações mais extensivas para confirmar se NULL é um lock real, checando se este é um lock sobre a raiz do volume utilizado para dar o boot no sistema.

Seek

Nome

Seek: define a posição atual para as operações de leitura e gravação.

Sinopse

```
oldPosition = Seek( file, position, mode )
D0                  D1        D2        D3
LONG Seek(BPTR, LONG, LONG)
```

Função

Seek( ) coloca o cursor de leitura/gravação para o arquivo *file* na posição *position*. Esta posição é utilizada pelas funções Read( ) e Write( ) com a finalidade de definir o lugar onde deverá ser iniciada a operação de leitura ou gravação. O resultado de Seek( ) é a posição absoluta atual no arquivo, ou -1 em caso de erro, situação em que você pode chamar IoErr( ) a fim de obter maiores informações. O argumento *mode*, que pode ser OFFSET_BEGINNING, OFFSET_CURRENT ou OFFSET_END, é usado para especificar a posição inicial relativa. Por exemplo, 20 a partir da posição atual (CURRENT) é uma posição 20 bytes adiante da atual, e -20 é uma posição 20 bytes para trás da atual.

Para descobrir onde você está, execute um Seek( ) de zero até a posição atual. O final do arquivo equivale a um Seek( ) posicionado por zero até o final (END). Não é permitido executar um Seek( ) após o final de um arquivo.

Entrada

file: ponteiro BCPL para um filehandle.

position: inteiro.

mode: inteiro.

Resultado

oldPosition: inteiro.

SelectInput

Nome

SelectInput: seleciona um filehandle como o canal de entrada default.

Sinopse

```
old_fh = SelectInput(fh)
D0                   D1
BPTR SelectInput(BPTR)
```

Função

SelectInput( ) define a entrada atual como a entrada default para o processo. Esta rotina modifica o valor retornado por Input( ). O filehandle *old_fh* deverá ser fechado ou salvo, conforme for preciso.

Entrada

fh: novo filehandle default de entrada.

Resultado

old_fh: antigo filehandle default de entrada.

SelectOutput

Nome

SelectOutput: seleciona um filehandle como o canal de saída default.

Sinopse

```
old_fh = SelectOutput(fh)
D0                    D1
BPTR SelectOutput(BPTR)
```

Função

SelectOutput( ) define a saída atual como a saída default do processo. Esta rotina muda o valor retornado por Output( ). O filehandle *old_fh* deverá ser fechado ou salvo, conforme for preciso.

Entrada

fh: novo filehandle de saída.

Resultado

old_fh: filehandle de saída anterior.

SendPkt

Nome

SendPkt: envia um packet para um processo de manipulação de dispositivo (V36).

Sinopse

```
SendPkt (packet, port, replyport)
          D1     D2       D3
void SendPkt(struct DosPacket *, struct MsgPort *, struct MsgPort *)
```

Função

SendPkt( ) envia um packet para um handler e não espera por uma resposta. Todos os campos do packet deverão ser inicializados antes de se chamar esta rotina. O packet será retornado para *replyport*. Se você desejar utilizar esta função com WaitPkt( ), use o endereço de sua porta de mensagens pr_MsgPort como *replyport*.

Entrada

port: pr_MsgPort do processo de manipulação para o qual o packet deverá ser enviado.

packet: packet a enviar. Deverá ter sido inicializado e possuir uma mensagem.

replyport: MsgPort para o retorno do packet.

SetArgStr

Nome

SetArgStr: define os argumentos do processo atual.

Sinopse

```
oldptr = SetArgStr(ptr)
D0                 D1
UBYTE * void SetArgStr(UBYTE *)
```

Função

Esta rotina define os argumentos do programa atual. É indispensável restaurar o valor original de *ptr* antes de se abandonar o processo.

Entrada

ptr: ponteiro para novo string de argumentos.

Resultado

oldptr: antigo string de argumentos.

SetComment

Nome

SetComment: muda o string de comentários de um arquivo.

Sinopse

```
success = SetComment( name, comment )
D0                    D1     D2
BOOL SetComment(char *, char *)
```

Função

SetComment( ) define um comentário para um arquivo ou diretório. *Comment* é um ponteiro para string terminado por NULL com até 80 caracteres.

Entrada

name: ponteiro para um string terminado por NULL.

comment: ponteiro para um string terminado por NULL.

Resultado

success: booleano.

SetConsoleTask

Nome

SetConsoleTask: define o console default do processo atual.

Sinopse

```
oldport = SetConsoleTask(port)
D0                       D1
struct MsgPort SetConsoleTask(struct MsgPort *)
```

Função

SetConsoleTask( ) define a porta de tarefa do console (pr_ConsoleTask) default do processo atual.

Entrada

port: a pr_MsgPort do handler de console default do processo.

Resultado

oldport: o antigo valor de ConsoleTask.

SetCurrentDirName

Nome

SetCurrentDirName: define o nome do diretório corrente do processo.

Sinopse

```
success = SetCurrentDirName(name)
D0                          D1
BOOL SetCurrentDirName(char *)
```

Função

Esta rotina define o nome do diretório corrente da estrutura CLI. Se o tamanho do nome exceder o limite que pode ser aceito, um indicador de erro será retornado e o nome original será mantido intacto. É aconselhável informar o usuário a respeito desta condição. É seguro chamar esta rotina mesmo quando não houver uma estrutura CLI.

Entrada

name: nome do diretório a ser definido.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Esta função abrevia para um tamanho fixo (compatível com o utilizado na versão 1.3).

SetFileDate

Nome

SetFileDate: define a data de modificação de um arquivo ou diretório (V36).

Sinopse

```
success = SetFileDate(name, date)
D0                    D1    D2
BOOL SetFileDate(char *, struct DateStamp *)
```

Função

SetFileDate( ) define a data de um arquivo ou diretório. Observe que tanto no Old Filing System quanto no Fast Filing System, não é permitido definir a data do diretório raiz. Outros filesystems podem não permitir a definição da data para todos os arquivos ou diretórios.

Entrada

name: nome do objeto.

date: nova data de modificação.

Resultado

success: indicador de sucesso/falha de execução da rotina.

SetFileSize

Nome

SetFileSize: define o tamanho de um arquivo.

Sinopse

```
newsize = SetFileSize(fh, offset, mode)
D0                    D1    D2      D3
LONG SetFileSize(BPTR, LONG, LONG)
```

Função

Esta rotina modifica o tamanho de um arquivo, truncando-o ou estendendo-o de acordo com o necessário. Nem todos os handlers suportam esta função, seja cuidadoso e verifique o código de retorno. Não é possível garantir qual será o valor dos novos bytes quando o arquivo for estendido. Se a nova posição ficar antes da posição atual do filehandle no arquivo, o filehandle terminará com uma posição no final do arquivo.

Se houver outros filehandles abertos no arquivo, o novo tamanho não deixará nenhum filehandle apontando para depois do final do arquivo. Você pode verificar isto observando o novo tamanho.

NÃO assuma a existência de nenhum valor específico na área estendida.

Entrada

fh: arquivo a ser truncado/estendido.

offset: offset a partir da posição determinada por *mode.*

mode: OFFSET_BEGINNING, OFFSET_CURRENT ou OFFSET_END.

Resultado

newsize: posição do novo final do arquivo, ou -1 em caso de erro.

SetFileSysTask

Nome

SetFileSysTask: define o filesystem default do processo atual.

Sinopse

```
oldport = SetFileSysTask(port)
D0                       D1
* struct MsgPort void SetFileSysTask(struct MsgPort *)
```

Função

Esta rotina define a porta de tarefa do filesystem (pr_FileSystemTask) default do processo atual.

Entrada

port: a pr_MsgPort do filesystem default do processo.

Resultado

oldport: o antigo valor de FileSysTask.

SetIoErr

Nome

SetIoErr: define o valor retornado pela função IoErr( ) (V36).

Sinopse

```
oldcode = SetIoErr(code)
D0                 D1
LONG SetIoErr(LONG)
```

Função

Esta rotina define o valor secundário (pr_Result2) de código de retorno (retornado pela função IoErr( )).

Entrada

code: código que deverá ser retornado por uma chamada a IoErr( ).

Resultado

oldcode: o antigo código de retorno.

SetMode

Nome

SetMode: define o comportamento atual de um handler (V36).

Sinopse

```
success = SetMode(fh, mode)
D0                D1   D2
BOOL SetMode(BPTR, LONG)
```

Função

SetMode( ) envia um packet ACTION_SCREEN_MODE para o processo de manipulação de dispositivo (handler) em questão, normalmente com o propósito de modificar o modo de operação de um handler CON: para o modo *raw*, ou vice-versa. Para CON:, use 1 a fim de ir para o modo RAW:, e 0 para o modo CON:.

Entrada

fh: filehandle.

mode: o novo modo desejado.

Resultado

success: booleano.

SetProgramDir

Nome

SetProgramDir: define o diretório retornado pela função GetProgramDir( ) (V36).

Sinopse

```
oldlock = SetProgramDir(lock)
D0                      D1
BPTR SetProgramDir(BPTR)
```

Função

Esta rotina ativa um lock compartilhado sobre o diretório a partir do qual o programa foi carregado. Este lock pode ser utilizado por um programa a fim de localizar arquivos de dados, etc. que são armazenados juntamente com o programa, ou ainda para localizar o próprio arquivo do programa. Para esta rotina, NULL é uma entrada válida. A função pode ser acessada via GetProgramDir( ) ou utilizando-se paths relativos a PROGDIR:.

Entrada

lock: lock sobre o diretório a partir do qual o programa foi carregado.

Resultado

oldlock: o antigo ProgramDir.

SetProgramName

Nome

SetProgramName: define o nome do programa sendo executado.

Sinopse

```
success = SetProgramName(name)
D0                       D1
BOOL SetProgramName(char *)
```

Função

Esta função define o nome do programa na estrutura CLI. Se o tamanho do nome exceder o limite que pode ser aceito, um indicador de erro será retornado e o nome original será mantido intacto. É aconselhável informar o usuário a respeito desta condição, se possível, e/ou definir o nome do programa como um string vazio. É seguro chamar esta rotina mesmo quando não houver uma estrutura CLI.

Entrada

name: nome do programa a usar.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Esta função abrevia para um tamanho fixo (compatível com o utilizado na versão 1.3).

SetPrompt

Nome

SetPrompt: define o prompt do CLI/Shell para o processo atual.

Sinopse

```
success = SetPrompt(name)
D0                  D1
BOOL SetPrompt(char *)
```

Função

SetPrompt( ) define o texto para o prompt da estrutura CLI. Se o tamanho do prompt exceder o limite que pode ser aceito, um indicador de erro será retornado e o prompt original será mantido intacto. É aconselhável informar o usuário a respeito desta condição. É seguro chamar esta rotina mesmo quando não houver uma estrutura CLI.

Entrada

name: nome do prompt a ser definido.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Esta função abrevia para um tamanho fixo (compatível com o utilizado na versão 1.3).

SetProtection

Nome

SetProtection: define os atributos de proteção de um arquivo ou diretório.

Sinopse

```
success = SetProtection( name, mask )
D0                       D1    D2:4
BOOL SetProtection(char *, LONG)
```

Função

SetProtection( ) define os atributos de proteção de um arquivo ou diretório. Os bits inferiores de *mask* antes do dos.library V36 são os seguintes:

bit 4: 1 = arquivo não foi alterado, 0 = arquivo foi alterado.
bit 3: 1 = leitura não permitida, 0 = leitura permitida.
bit 2: 1 = gravação não permitida, 0 = gravação permitida.
bit 1: 1 = execução não permitida, 0 = execução permitida.
bit 0: 1 = apagamento não permitido, 0 = apagamento permitido.

Nas versões V36 e posteriores do dos.library e no Fast Filing System, os bits de leitura e gravação são respeitados.

O bit de arquivo (*archive bit*) é ressetado pelo sistema quando o arquivo é modificado. Utilitários de backup geralmente setam este bit após produzirem uma cópia backup de cada arquivo.

O Fast Filing System verifica os bits de leitura e gravação e o Shell verifica o bit de execução, recusando-se a iniciar a execução de um arquivo tal como um arquivo binário executável caso este bit esteja setado.

Outros bits podem ser definidos através dos arquivos include dos/dos.h e libraries/dos.h. Em vez de se referir aos bits pelos números, você deverá empregar suas definições em dos.h.

Entrada

name: ponteiro para um string terminado por NULL.

mask: máscara especificando os bits de proteção desejados.

Resultado

success: booleano.

SetVar

Nome

SetVar: define uma variável local ou de ambiente (V36).

Sinopse

```
success = SetVar( name, buffer, size, flags )
D0                D1      D2      D3     D4
BOOL SetVar(UBYTE *, UBYTE *, LONG, ULONG)
```

Função

Esta rotina define uma variável local ou de ambiente. É aconselhável utilizar somente strings ASCII dentro de variáveis, mas este procedimento não é obrigatório.

Entrada

name: ponteiro para o nome da variável. Observe que os nomes de variáveis seguem a mesma semântica e sintaxe dos filesystems.

buffer: área alocada pelo usuário contendo o string que representa o valor a ser associado a esta variável.

size: tamanho da região do buffer, em bytes. -1 indica que o buffer contém um string terminado por NULL.

flags: combinação do tipo de variável a ser definida (8 bits inferiores) e flags para controlar o comportamento desta rotina. Os flags atualmente definidos são:

GVF_LOCAL_ONLY: define uma variável local ao seu processo.

GVF_GLOBAL_ONLY: define uma variável de ambiente global.

O default desta rotina é definir uma variável de ambiente local.

Resultado

success: se diferente de zero, a variável foi definida com sucesso. FALSE indica uma falha na execução da rotina.

Bugs

LV_VAR é o único tipo que pode ser global.

Veja também: DeleteVar( ), FindVar( ), GetVar( ), dos/var.h

SetVBuf

Nome

SetVBuf: define os modos de bufferização e tamanho de um buffer.

Sinopse

```
error = SetVBuf(fh, buff, type, size)
D0              D1    D2    D3    D4
LONG SetVBuf(BPTR, UBYTE *, LONG, LONG)
```

Função

Esta rotina modifica os modos de bufferização e o tamanho do buffer de um filehandle. Com *buff* == NULL, o buffer atual será desalocado e um novo buffer com o tamanho (aproximado) indicado por *size* será alocado. Caso seja diferente de NULL, o buffer será utilizado para operações de bufferização e deverá ter no mínimo 208 bytes de tamanho. Se o buffer for NULL e *size* igual a -1, então somente o modo de bufferização será alterado.

Entrada

fh: filehandle.
buff: ponteiro do buffer para a operção de I/O bufferizado.
type: modo de bufferização (veja o arquivo <dos/stdio.h>).
size: tamanho do buffer para o I/O bufferizado (tamanhos inferiores a 208 bytes serão ignorados).

Resultado

error: 0 caso a operação tenha sido bem sucedida. **NOTA:** Isto é o oposto da maioria das funções do DOS.

Bugs

Esta função não se encontra ainda implementada, retornando sempre 0.

Veja também: FPutC( ), FGetC( ), UnGetC( ), Flush( ), FRead( ), FWrite( ), FGets( ), FPuts( ).

SplitName

Nome

SplitName: separa num buffer um componente da descrição de um path.

Sinopse

```
newpos = SplitName(name, separator, buf, oldpos, size)
D0                 D1       D2       D3      D4      D5
WORD SplitName(UBYTE *, UBYTE, UBYTE *, WORD, LONG)
```

Função

Esta rotina extrai o próximo componente de um nome de arquivo especificado. Cada parte é copiada no buffer, sendo truncada a *size*-1 caracteres. A nova posição (*newpos*) é retornada de maneira que possa ser passada para a chamada SplitName( ) seguinte. Se o separador não puder ser encontrado dentro do limite de *size* caracteres, *size*-1 caracteres mais um NULL serão copiados no buffer e a posição do próximo separador será retornada. Se o separador não puder ser encontrado, a rotina retornará um -1, e os caracteres da última posição até o final do string serão copiados no buffer (até um máximo de *size*-1 caracteres). Ambos os strings serão terminados por NULL. Esta função se destina basicamente a dar suporte a handlers.

Entrada

name: nome de arquivo sendo analisado.
separator: caractere que indica onde se dará a separação.
buf: buffer para receber o nome extraído.
oldpos: posição atual no arquivo.
size: tamanho de *buf* em bytes (incluindo-se a terminação NULL).

Resultado

newpos: nova posição para a próxima chamada a SplitName( ). -1 assinala a chamada final.

StartNotify

Nome

StartNotify: inicia a notificação em um arquivo ou diretório.

Sinopse

```
success = StartNotify(notifystructure)
D0                              D1
BOOL StartNotify(struct NotifyRequest *)
```

Função

Esta rotina envia uma requisição de notificação. Não modifique a estrutura de notificação enquanto a mesma estiver ativa. Você será notificado quando o arquivo ou diretório for modificado. No caso dos arquivos, você será notificado depois que o arquivo for fechado. Uma aplicação NÃO deverá requisitar esta função, uma vez que nem todos os filesystems são capazes de suportá-la. Em particular, a maioria dos filesystems de rede não suportam esta função.

Entrada

notifystructure: uma estrutura NotifyStructure preenchida.

Resultado

success: indicador de sucesso/falha da requisição.

Bugs

O filesystem de disquete/HD do dos.library V36 na realidade não envia notificações, embora o handler do dispositivo RAM: seja capaz de enviar notificações sem problemas. Este bug foi corrigido no dos.library versão V37.

StrToDate

Nome

StrToDate: converte um string em um indicador de data.

Sinopse

```
success = StrToDate( datetime )
D0                      D1
BOOL StrToDate( struct DateTime * )
```

Função

StrToDate( ) converte um string ASCII compreensível para o usuário num indicador de data (DateStamp) para uso interno do AmigaDOS.

Entrada

DateTime: ponteiro para uma estrutura DateTime previamente inicializada. A estrutura DateTime deverá ser inicializada tal como se segue.

dat_Stamp: ignorado na entrada.

dat_Format: um byte de formatação que especifica o formato de dat_StrDate (veja mais adiante). Pode ser qualquer uma das opções abaixo (note que caso o valor utilizado seja diferente de qualquer um dos valores a seguir, o default de FORMAT_DOS será adotado):

FORMAT_DOS: formato do AmigaDOS (dd-mmm-yy).

FORMAT_INT: formato internacional (yy-mmm-dd).

FORMAT_USA: formato americano (mm-dd-yy).

FORMAT_CDN: formato canadense (dd-mm-yy).

FORMAT_DEF: formato default para a localidade.

dat_Flags: byte de flags. Os únicos flags que afetam esta função são:

DTF_SUBST: ignorada por esta função.

DTF_FUTURE: quando setada, indica que strings como "Monday" (armazenadas em dat_StrDate) se referem à "próxima segunda-feira". Se ressetada, indica que strings como "Monday" se referem à "segunda-feira passada".

dat_StrDay: ignorado por esta função.

dat_StrDate: ponteiro para um string válido representando a data. Pode ser um string no estilo DTF_SUBST como "Today", "Tomorrow", "Monday", ou um string especificada pelo byte dat_Format. Este string será convertido na parte ds_Day da estrutura DateStamp. Se o ponteiro for NULL, DateStamp->ds_Days não serão afetados.

dat_StrTime: ponteiro para um buffer que contém a hora no formato ASCII: hh:mm:ss. Este dado será convertido nas partes ds_Minutes e ds_Ticks da estrutura DateStamp. Se o ponteiro for NULL, ds_Minutes e ds_Ticks não serão alterados.

Resultado

success: um retorno igual a zero indica que alguma conversão não pôde ser efetuada. Um retorno diferente de zero indica que a variável dat_Stamp contém os valores convertidos.

Veja também: DateStamp( ), DateToStr( ), dos/datetime.h.

StrToLong

Nome

StrToLong: converte um string em um valor decimal de palavra longa.

Sinopse

```
characters = StrToLong(string,value)
D0                     D1      D2
LONG StrToLong(UBYTE *,LONG *)
```

Função

Esta rotina converte um string decimal num valor LONG (32 bits), retornando o número de caracteres convertidos. A função salta os espaços e marcas de tabulação (os quais são entretanto incluídos na contagem dos caracteres). Caso nenhum dígito decimal seja encontrado (após pular os espaços e tabulações), StrToLong( ) retornará -1 como o número de caracteres e 0 como o valor convertido.

Entrada

string: string de entrada.

value: ponteiro para o valor LONG. Será zero caso nenhum dígito seja convertido.

Resultado

result: número de caracteres convertidos, ou -1 como descrito acima.

SystemTagList

Nome

SystemTagList: faz o Shell executar uma linha de comandos (V36).

Sinopse

```
error = SystemTagList(command, tags)
D0                    D1       D2
LONG SystemTagList(UBYTE *, struct TagItem *)
error = System(command, tags)
D0                D1       D2
LONG System(UBYTE *, struct TagItem *)
error = SystemTags(command, Tag1, ...)
LONG SystemTags(UBYTE *, ULONG, ...)
```

Função

Esta função é similar a Execute( ), a diferença é que SystemTagList( ) não lê os comandos do filehandle de entrada. Esta rotina gera um processo Shell para executar o comando e retorna o código de retorno produzido pelo comando, ou -1 caso

por alguma razão não tenha sido possível executá-lo. A rotina não fecha os filehandles de entrada e saída. Você deverá fechar os filehandles (se necessário) após System( ) retornar, caso os tenha especificado via SYS_INPUT ou SYS_OUTPUT.

Por default, o novo processo utilizará os seus filehandles de Input( ) e Output( ) atuais. A análise de sintaxe (parsing) da linha de comandos do Shell será efetuada normalmente, incluindo-se o redirecionamento de comandos. O path e diretório corrente serão herdados do seu processo. O seu path será utilizado para localizar os comandos, caso nenhum outro path seja especificado.

Quando utilizada com o flag SYS_Asynch, a rotina fechará os filehandles de entrada e saída após a execução do comando (mesmo que estes sejam os *seus* filehandles de Input( ) e Output( )!).

Normalmente esta função utiliza o shell ativo por ocasião do boot (ou seja, o Shell da ROM), mas outros shells podem ser especificados através de SYS_UserShell e SYS_CustomShell. De maneira geral, você deverá enviar o que for escrito pelo usuário para o UserShell. O default do UserShell é o mesmo shell do boot.

Os tags (indicadores) da função são passados para CreateNewProc( ) (tags que conflitem com SystemTagList( ) serão filtrados). Isto possibilita a definição de características como prioridade, etc. do novo processo.

Entrada

command: programa e argumentos.

tags: veja o arquivo <dos/dostags.h>. Note que tanto tags específicos de SystemTagList( ) como tags de CreateNewProc( ) podem ser passados para esta função.

Resultado

error: O para execução bem sucedida, resultado do comando, ou -1 em caso de erro. Observe que em caso de erro, a aplicação que chama a rotina será responsável por quaisquer filehandles ou outros parâmetros passados através de tags.

Veja também: Execute( ), CreateNewProc( ), <dos.dostags.h>, Input( ), Output( ).

UnGetC

Nome

UnGetC: torna um caractere disponível para ser lido novamente (bufferizado).

Sinopse

```
value = UnGetC(fh, character)
D0            D1         D2
LONG UnGetC(BPTR, LONG)
```

Função

Esta rotina repõe um caractere especificado no buffer de entrada. Sempre que

utilizar uma rotina de leitura bufferizada, você poderá recuperar um caractere. É possível recuperar mais de um caractere, mas esta prática não é recomendada pois não é possível prever quantos caracteres é possível retroceder num determinado momento. Passando-se -1 como o argumento de *character* fará com que o último caractere lido fique novamente disponível para leitura.

Nota: UnGetC( ) pode ser usado para confirmar se determinado filehandle foi definido como um filehandle de leitura. Isto só tem utilidade prática se você estiver escrevendo um Shell e desejar manipular o buffer do filehandle.

Entrada

fh: filehandle a ser utilizado para o I/O bufferizado.
character: caractere a retroceder, ou -1 para recuperar o último caractere.

Resultado

value: caractere recuperado, ou FALSE caso não tenha sido possível recuperar o caractere.

Bugs

No dos.library V36, uma chamada UnGetC(fh,-1) após um EOF não fazia com que o caractere lido a seguir fosse um EOF. Este bug foi corrigido na versão V37.

UnLoadSeg

Nome

UnLoadSeg: remove da memória uma lista de segmentos carregada por LoadSeg( ).

Sinopse

```
success = UnLoadSeg( seglist )
D0                   D1
BOOL UnLoadSeg(BPTR)
```

Função

Esta rotina remove da memória uma lista de segmentos carregada pela função LoadSeg( ). *Seglist* pode ser zero. Para os segmentos overlay a rotina efetua todo o retorno de memória necessário, incluindo-se o fechamento de arquivos.

Entrada

seglist: ponteiro BCPL para um identificador de segmento.

Resultado

success: retorna zero se um *seglist* NULL foi passado para a rotina, ou caso não tenha sido possível fechar um arquivo overlay.
Nota: Esta função retornava um valor aleatório antes do dos.library versão V36.

UnLock

Nome

UnLock: desativa um lock sobre um diretório ou arquivo.

Sinopse

```
UnLock( lock )
          D1
void UnLock(BPTR)
```

Função

Esta rotina remove e desaloca um lock do sistema de arquivo, obtido através das funções Lock( ), DupLock( ) ou CreateDir( ).

Entrada

lock: ponteiro BCPL para um lock.

Nota: É seguro passar zero para UnLock( ).

UnLockDosList

Nome

UnLockDosList: desativa um lock sobre a lista de dispositivos do DOS.

Sinopse

```
UnLockDosList(flags)
              D1
void UnLockDosList(ULONG)
```

Função

Esta rotina desbloqueia o acesso sobre a lista de dispositivos do DOS (DOS List). É indispensável passar para a função os mesmos flags utilizados para ativar o lock sobre a lista.

Entrada

flags: OBRIGATORIAMENTE, os mesmos flags passados para (Attempt)LockDos-List.

UnLockRecord

Nome

UnLockRecord: desativa um lock sobre um registro.

Sinopse

```
success = UnLockRecord(fh, offset, lenght)
D0                     D1     D2       D3
BOOL UnLockRecord(BPTR,ULONG,ULONG)
```

Função

Esta rotina libera o lock especificado sobre o registro de um arquivo. Observe que você deverá empregar o mesmo filehandle, *offset* e *lenght* usados para ativar o lock sobre o registro. Toda chamada a LockRecord( ) deverá sempre ser balanceada com uma chamada a UnLockRecord( ).

Entrada

fh: filehandle do arquivo sobre o qual o lock está ativo.

offset: posição inicial do registro.

lehght: tamanho do registro em bytes.

Resultado

success: indicador de falha/sucesso da operação.

Bugs

Veja LockRecord( ).

UnLockRecords

Nome

UnLockRecords: desativa um lock sobre uma lista de registros.

Sinopse

```
success = UnLockRecords(record_array)
D0                      D1
BOOL UnLockRecords(struct RecordLock *)
```

Função

Esta rotina libera um array de locks sobre registros obtidos por intermédio da função LockRecords( ). Você NÃO deve modificar o *record_array* enquanto os locks sobre os registros estiverem ativos. Toda chamada a LockRecords( ) deverá sempre ser balanceada com uma chamada a UnLockRecords( ).

Entrada

record_array: lista de registros cujos locks deverão ser desativados.

Resultado

success: indicador de sucesso/falha de execução da rotina.

Bugs

Veja LockRecords( )

VFPrintf

Nome

VFPrintf: formata e grava um string em um arquivo (bufferizado).

Sinopse

```
count = VFPrintf(fh, fmt, argv)
D0               D1  D2    D3
LONG VFPrintf(BPTR, char *, LONG *)
count = FPrintf(fh, fmt, ...)
LONG FPrintf(BPTR, char *, ...)
```

Função

VFPrintf( ) grava um string e valores formatados no arquivo especificado. É assumido que esta rotina seja capaz de manipular toda a bufferização interna, de modo que o string de formatação e os valores formatados resultantes podem ter um comprimento arbitrário. Qualquer erro secundário será retornado por IoError( ). Esta rotina é bufferizada.

Entrada

fh: filehandle onde os dados serão gravados.

fmt: string de formatação no estilo RawDoFmt.

argv: ponteiro para um array de valores de formatação.

Resultado

count: número de bytes gravados, ou -1 (EOF) em caso de erro.

VFWritef

Nome

VFWritef: grava um string formato BCPL em um arquivo (bufferizado).

Sinopse

```
count = VFWritef(fh, fmt, argv)
D0               D1  D2    D3
```

```
LONG VFWritef(BPTR, char *, LONG *)
count = Fwrite(fh, fmt, ...)
LONG Fwrite(BPTR, char *, ...)
```

Função

VFWritef( ) grava um string e valores formatados na saída default. É assumido que esta rotina seja capaz de manipular toda a bufferização interna, de modo que o string de formatação e os valores formatados resultantes podem ter um comprimento arbitrário. Os formatos deverão estar na forma BCPL. Esta rotina é bufferizada.

Os formatos suportados são relacionados abaixo (**Nota:** *x* está na base 36!).

%S–string (CSTR).
%Tx–grava um string alinhado à esquerda num campo com o tamanho mínimo de x bytes.
%C–grava um único caractere.
%Ox–grava um número octal com no máximo x caracteres de tamanho.
%Xx–grava um número hexadecimal com no máximo x caracteres de tamanho.
%Ix–grava um número decimal com no máximo x caracteres de tamanho.
%N–grava um número decimal de qualquer tamanho.
%Ux–grava um número sem sinal com no máximo x caracteres de tamanho.
%$–ignora o parâmetro.

Entrada

fmt: string de formatação estilo BCPL.

argv: ponteiro para um array de valores de formatação.

Resultado

count: número de bytes gravados, ou -1 em caso de erro.

VPrintf

Nome

VPrintf: formata e imprime um string (bufferizado).

Sinopse

```
count = VPrintf(fmt, argv)
D0              D1    D2
LONG VPrintf(char *, LONG *)
count = Printf(fmt, ...)
LONG Printf(char *, ...)
```

Função

VPrintf( ) grava um string e valores formatados em Output( ). É assumido que esta rotina seja capaz de manipular toda a bufferização interna, de modo que o string de

formatação e os valores formatados resultantes podem ter um comprimento arbitrário. Qualquer erro secundário será retornado por IoError( ). Esta rotina é bufferizada.

Nota: RawDoFmt assume inteiros de 16 bits, logo você usualmente irá necessitar de 'l's nos seus formatos (exemplo: %ld no lugar de %d).

Entrada

fmt: string de formatação no estilo de RawDoFmt( ) da biblioteca exec.library.
argv: ponteiro para um array de valores de formatação.

Resultado

count: número de bytes gravados, ou -1 (EOF) em caso de erro.

WaitForChar

Nome

WaitForChar: determina se os caracteres estarão disponíveis para leitura dentro de um determinado limite de tempo.

Sinopse

```
status = WaitForChar(file, timeout)
D0               D1    D2
BOOL WaitForChar(BPTR, LONG)
```

Função

Se um caractere estiver disponível no arquivo *file* para ser lido dentro do período de tempo (em microssegundos) especificado por *timeout*, WaitForChar( ) retornará -1 (TRUE). Caso esteja disponível, você poderá utilizar a função Read( ) para ler esse caractere. Note que WaitForChar( ) só é válido quando a stream de I/O estiver conectada a um dispositivo de terminal virtual. Se o caractere não estiver disponível dentro de *timeout* microssegundos, a rotina retornará um 0 (FALSE).

Entrada

file: ponteiro BCPL para um filehandle.
timeout: inteiro.

Resultado

status: booleano.

Bugs

Devido a um bug no dispositivo timer.device nas versões V1.2/V1.3, especificar uma contagem de tempo igual a zero para WaitForChar( ) pode provocar falhas no funcionamento do timer e durante o acesso ao disquete.

WaitPkt

Nome

WaitPkt: espera pela chegada de um packet à sua pr_MsgPort.

Sinopse

```
packet = WaitPkt( )
D0
struct DosPacket *WaitPkt(void);
```

Função

Esta rotina espera até que um packet chegue à sua pr_MsgPort. Se uma função de espera tiver sido instalada no campo pr_PktWait do packet, esta rotina será chamada. A mensagem é automaticamente obtida através de GetMsg( ), de modo que a mesma não estará mais na porta. A rotina assume que a mensagem seja um packet do DOS. Note que NÃO é possível garantir que o sinal seja removido da porta.

Resultado

packet: o packet que chegou à porta (a partir do campo In_Name da mensagem).

Write

Nome

Write: grava bytes de dados em um arquivo.

Sinopse

```
returnedLenght = Write(file, buffer, lenght)
D0                     D1     D2      D3
LONG Write(BPTR, void *, LONG)
```

Função

Write( ) grava bytes de dados no arquivo aberto especificado por *file*. O argumento *lenght* indica o tamanho dos dados a serem transferidos, enquanto que *buffer* é um ponteiro para o buffer. O valor retornado pela função corresponde ao número de bytes efetivamente gravados. Desse modo, quando *lenght* for maior que zero, seu valor equivalerá ao número de caracteres gravados. Erros são indicados por um valor de retorno igual a -1.

Entrada

file: ponteiro BCPL para um filehandle.

buffer: ponteiro para o buffer.

lenght: inteiro.

Resultado

returnedLenght: inteiro.

WriteChars

Nome

WriteChars: grava bytes de dados na saída default (V36) (bufferizado)..

Sinopse

```
count = WriteChars(buf, buflen)
D0                 D1
LONG PutStr(UBYTE *, LONG)
```

Função

Esta rotina grava um determinado número de bytes na saída default, retornando o número de bytes efetivamente gravado. Esta rotina é bufferizada.

Entrada

buf: buffer de caracteres a serem gravados.

buflen: número de caracteres a gravar.

Resultado

count: número de bytes gravados, ou -1 (EOF) em caso de erro de execução da rotina.

Veja também: FPuts( ), FPutC( ), FWrite( ), PutStr( ).

Funções do AmigaDOS – Tabela de Referência Rápida

AbortPkt	Aborta um packet assíncrono, se possível (V36).
AddBuffers	Muda o número de buffers de um filesystem.
AddDosEntry	Acrescenta uma entrada à lista de dispositivos do DOS.
AddPart	Acrescenta um arquivo/diretório ao final de um path.
AddSegment	Acrescenta um segmento à lista residente.
AllocDosObject	Cria um objeto do DOS.
AssignAdd	Acrescenta um lock a uma associação de múltiplos diretórios.
AssignLate	Cria uma associação que será posteriormente vinculada ao path.
AssignLock	Cria uma associação para um objeto sobre o qual foi ativado um lock.
AssignPath	Cria uma associação para um path especificado.
AttemptLockDosList	Tenta ativar um lock sobre a lista de dispositivos do DOS para utilizá-la.
ChangeMode	Modifica o modo atual de um lock ou filehandle.
CheckSignal	Verifica sinais de break.
Cli	Retorna um ponteiro para a estrutura CLI do processo.
Close	Fecha um arquivo aberto.
CompareDates	Compara dois indicadores de datas.
CreateDir	Cria um novo diretório.
CreateNewProc	Cria um novo processo.
CreateProc	Cria um novo processo.
CurrentDir	Torna um lock de diretório o diretório corrente.
DateStamp	Obtém a data e a hora no formato interno.
DateToStr	Converte um indicador de data em um string.
Delay	Retarda um processo durante o intervalo de tempo especificado.
DeleteFile	Apaga um arquivo ou diretório.
DeleteVar	Apaga uma variável local ou de ambiente.
DeviceProc	Retorna a estrutura MsgPort para um processo de manipulação de I/O específico.
DoPkt	Envia um packet do DOS e espera pela resposta (V36).
DupLock	Duplica um lock.
DupLockFromFH	Obtém um lock em um arquivo aberto.
EndNotify	Encerra uma requisição de notificação.
ErrorReport	Exibe um menu com uma mensagem Retry/Cancel após a ocorrência de um erro.
ExAll	Examina um diretório inteiro.
Examine	Examina um diretório ou arquivo associado com um lock.

ExamineFH	Obtém informações sobre um arquivo aberto.
Execute	Executa um comando do CLI.
Exit	Abandona um programa.
ExNext	Examina a entrada seguinte num diretório.
Fault	Retorna o texto associado com um código de erro do DOS.
FGetC	Lê um caractere a partir da fonte de entrada especificada (bufferizado).
FGets	Lê uma linha a partir da fonte de entrada especificada (bufferizado).
FilePart	Retorna o último componente de um path.
FindArg	Localiza uma palavra-chave num gabarito.
FindCliProc	Retorna um ponteiro para o processo CLI especificado.
FindDosEntry	Localiza uma entrada específica na lista dispositivos do DOS.
FindSegment	Localiza um segmento na lista residente (V36).
FindVar	Localiza uma variável local (V36).
Flush	Limpa buffers para um filehandle bufferizado.
Format	Faz com que um filesystem se inicialize (V37).
FPutC	Grava um caractere na saída especificada (bufferizado) (V36).
FPuts	Grava um string na saída especificada (bufferizado).
FRead	Lê um determinado número de blocos a partir de uma fonte de entrada (bufferizado).
FreeArgs	Libera a memória alocada após o uso da função ReadArgs( ).
FreeDeviceProc	Libera a porta retornada por GetDeviceProc (V36).
FreeDosEntry	Libera uma entrada criada pela função MakeDosEntry( ).
FreeDosObject	Libera um objeto alocado pela função AllocDosObject( ).
FWrite	Grava um determinado número de blocos em uma saída (bufferizado).
GetArgStr	Retorna os argumentos para o processo (V36).
GetConsoleTask	Retorna o console default para o processo atual.
GetCurrentDirName	Retorna o nome do diretório corrente.
GetDeviceProc	Localiza o handler ao qual deverá ser enviada uma mensagem.
GetFileSysTask	Retorna o filesystem default para o processo atual.
GetProgramDir	Retorna um lock no diretório a partir do qual o programa foi carregado.
GetProgramName	Retorna o nome do programa atual.
GetPrompt	Retorna o prompt do processo atual (V36).
GetVar	Retorna o valor de uma variável local ou global.
Info	Retorna informações sobre o disco em uso.

Inhibit	Inibe o acesso a um filesystem (V36).
Input	Identifica o filehandle de entrada inicial do programa.
InternalLoadSeg	Rotina de carregamento *low level*.
InternalUnLoadSeg	Remove uma lista de segmentos carregada com InternalLoadSeg( ).
IoErr	Retorna informações adicionais do sistema.
IsFileSystem	Confirma se um handler do DOS é um filesystem.
IsInteractive	Descobre se um arquivo é um terminal virtual.
LoadSeg	Carrega por partes um arquivo carregável na memória.
Lock	Ativa um lock sobre um diretório ou arquivo.
LockDosList	Ativa um lock sobre a lista especificada do DOS para utilizá-la.
LockRecord	Ativa um lock sobre parte de um arquivo.
LockRecords	Ativa locks sobre uma série de registros.
MakeDosEntry	Cria uma estrutura DosList (V36).
MakeLink	Cria uma ligação entre filesystems (V36).
MatchEnd	Libera a área de armazenamento alocada pelas funções MatchFirst( ) e MatchNext( ).
MatchFirst	Localiza o arquivo que combina com o padrão especificado.
MatchNext	Localiza o próximo arquivo ou diretório que combina com o padrão especificado.
MatchPattern	Procura por um string que combine com o padrão especificado.
MatchPatternNoCase	Procura por um string que combine com o padrão especificado (V36).
MaxCli	Retorna o mais alto número de processo CLI em uso.
NameFromFH	Obtém o nome de um filehandle aberto (V36).
NameFromLock	Retorna o nome de um objeto sobre o qual esteja ativado um lock.
NewLoadSeg	Versão aperfeiçoada de LoadSeg( ).
NextDosEntry	Obtém a próxima entrada na lista de dispositivos do DOS (V36).
Open	Abre um arquivo para entrada ou saída.
OpenFromLock	Abre um arquivo sobre o qual tenha sido ativado um lock.
Output	Identifica o filehandle inicial de saída do programa.
ParentDir	Obtém o diretório matriz de um arquivo.
ParentOfFH	Retorna um lock sobre o diretório matriz de um arquivo.
ParsePattern	Cria um string codificado para uso com a função MatchPattern( ).
ParsePatternNoCase	Cria um string codificado para uso com a função MatchPatternNoCase( ).
PathPart	Retorna um ponteiro para o final do penúltimo

	componente de um path.
PrintFault	Retorna o texto associado a um código de erro do DOS.
PutStr	Grava um string na saída default (bufferizado).
Read	Lê bytes de dados de um arquivo.
ReadArgs	Analisa a sintaxe da linha de comando.
ReadItem	Lê um nome/argumento da linha de comandos.
ReadLink	Lê o path para o soft link de um filesystem (V36).
Relabel	Muda o nome de um volume.
RemAssignList	Remove uma entrada de uma associação de múltiplos diretórios.
RemDosEntry	Remove uma entrada da lista de dispositivos do DOS.
RemSegment	Remove um segmento da lista residente.
Rename	Renomeia um diretório ou arquivo.
ReplyPacket	Devolve um packet para o processo que o enviou.
RunCommand	Roda um programa utilizando o processo atual (V36).
SameDevice	Verifica se dois locks estão sobre uma partição do mesmo dispositivo (V37).
SameLock	Verifica se dois locks estão sobre o mesmo objeto.
Seek	Define a posição atual para as operações de leitura e gravação.
SelectInput	Seleciona um filehandle como o canal de entrada default.
SelectOutput	Seleciona um filehandle como o canal de saída default.
SendPkt	Envia um packet para um processo de manipulação de dispositivo.
SetArgStr	Define os argumentos do processo atual.
SetComment	Muda o string de comentários de um arquivo.
SetConsoleTask	Define o console default do processo atual.
SetCurrentDirName	Define o nome do diretório corrente do processo.
SetFileDate	Define a data de modificação de um arquivo ou diretório (V36).
SetFileSize	Define o tamanho de um arquivo.
SetFileSysTask	Define o filesystem default do processo atual.
SetIoErr	Define o valor retornado pela função IoErr( ) (V36).
SetMode	Define o comportamento atual de um handler (V36).
SetProgramDir	Define o diretório retornado pela função GetProgramDir( ) (V36).
SetProgramName	Define o nome do programa sendo executado.
SetPrompt	Define o prompt do CLI/Shell para o processo atual.
SetProtection	Define os atributos de proteção de um arquivo ou diretório.
SetVar	Define uma variável local ou de ambiente (V36).
SetVBuf	Define os modos de bufferização e tamanho de um buffer.

SplitName	Separa num buffer um componente da descrição de um path.
StartNotify	Inicia a notificação em um arquivo ou diretório.
StrToDate	Converte um string em um indicador de data.
StrToLong	Converte um string em um valor decimal de palavra longa.
SystemTagList	Faz o Shell executar uma linha de comandos (V36).
UnGetC	Torna um caractere disponível para ser lido novamente (bufferizado).
UnLoadSeg	Remove da memória uma lista de segmentos carregada por LoadSeg( ).
UnLock	Desativa um lock sobre um diretório ou arquivo.
UnLockDosList	Desativa um lock sobre a lista de dispositivos do DOS.
UnLockRecord	Destiva um lock sobre um registro.
UnLockRecords	Desativa um lock sobre uma lista de registros.
VPrintf	Formata e grava um string em um arquivo (bufferizado).
VFWritef	Grava um string formato BCPL em um arquivo (bufferizado).
VPrintf	Formata e imprime um string (bufferizado).
WaitForChar	Determina se os caracteres estarão disponíveis para leitura dentro de um determinado limite de tempo.
WaitPkt	Espera pela chegada de um packet à sua pr_MsgPort.
Write	Grava bytes de dados em um arquivo.

Capítulo 7

O Linker

O **linker** é uma ferramenta utilizada para a criação de um arquivo de carregamento binário a partir de um ou mais arquivos objeto gerados por intermédio do C, Assembler ou outra linguagem de programação do Amiga. Este capítulo descreve o linker do AmigaDOS, o ALINK.

O linker ALINK é fornecido pela Commodore, mas diversas linguagens de programação vêm acompanhadas pelo seu próprio linker, produzido pelo fabricante do compilador. Consulte o manual do seu compilador a fim de obter maiores informações sobre o linker utilizado com a linguagem de programação com a qual você trabalha.

Introdução

O ALINK produz um único arquivo de saída binário a partir de um ou mais arquivos de entrada. Esses arquivos de entrada, conhecidos como **arquivos objeto**, podem conter informações simbólicas externas. Para produzir os arquivos objeto, normalmente é utilizado um editor assembler ou um outro tipo de tradutor de linguagens. Antes de produzir a saída, ou o **arquivo de carregamento**, o linker precisa resolver todas as referências a esses símbolos.

O linker também pode produzir um mapa de ligações e uma tabela de referência cruzada dos símbolos utilizados.

Associado com o linker está um **supervisor de overlay**. Você pode utilizar o supervisor de overlay a fim de criar arquivos overlay de programas escritos em diversas linguagens. O linker produz arquivos de carregamento apropriados para serem utilizados desta maneira.

Você pode orientar a operação do linker de duas formas:

1. Como uma **linha de comando**. Você pode especificar a maioria das informações necessárias para executar o linker através dos parâmetros de uma linha de comando.

2. Como um **arquivo de parâmetros**. Alternativamente, se o programa estiver sendo ligado de forma repetitiva, você pode utilizar um arquivo de parâmetros com o propósito de especificar os dados para o linker.

Veja a descrição da opção WITH do ALINK, mais adiante neste capítulo.

Estes dois métodos podem aceitar três tipos de arquivos de entrada:

1. **Entrada primária binária**. Este termo se refere a um ou mais arquivos objeto que formam a entrada binária inicial para o linker. Estes arquivos serão sempre copiados no arquivo de carregamento. A entrada primária nunca poderá ser vazia.

2. **Arquivos overlay**. No caso de ser utilizado o recurso de overlay, a entrada primária constituirá a raiz da árvore de overlay, e os arquivos overlay formarão o restante da estrutura.

3. **Bibliotecas**. Referem-se a um tipo de código especial que o linker é capaz de incorporar automaticamente. As bibliotecas podem ser residentes ou de consulta. Uma **biblioteca residente** consiste num arquivo de carregamento que pode permanecer residente na memória, ou então ser carregado como parte do chamado *open library* do sistema operacional. Uma **biblioteca de consulta** é um arquivo objeto dentro de um arquivo em formato de fichário. O linker só carregará o arquivo se houver alguma referência externa a ser resolvida pela biblioteca.

O linker trabalha em duas passagens.

Na primeira passagem, o linker lê todos os arquivos primários, overlays e bibliotecas, registrando os segmentos de código e as informações simbólicas externas. No final desta passagem, o linker produz o mapa e a tabela de referência cruzada, quando solicitado.

Se você especificar um arquivo de saída, o linker efetuará então uma segunda passagem sobre os dados de entrada. Primeiramente ele copia os arquivos de entrada primários para a saída, resolvendo as referências simbólicas durante o processo. Na seqüência são igualmente copiados os segmentos de código da biblioteca requisitada. Note que os segmentos de código da biblioteca formam parte da raiz da árvore de overlay. Em seguida o linker produz os dados para o supervisor de overlay e, por fim, são gravados os arquivos overlay.

Na primeira passagem, após ler os arquivos de entrada primários e overlays, o linker inspeciona sua tabela de símbolos, e caso ainda haja alguma referência não resolvida, ele lê os arquivos (se existirem) que você especificou como uma entrada de biblioteca. O linker marcará então quaisquer segmentos de código contendo definições externas para essas referências não resolvidas, para subseqüente inclusão no arquivo de carregamento. O linker só incluirá os segmentos de códigos das bibliotecas que você tiver referenciado.

Usando o Linker

Para utilizar o linker é necessário conhecer a sintaxe de seus comandos, o tipo de entrada e saída que ele utiliza, e os erros que podem ocorrer durante o processo da linkagem. Esta seção procura explicar cada um destes detalhes.

Sintaxe da Linha de Comando

O comando ALINK possui os seguintes parâmetros:

```
ALINK [FROM | ROOT] arqs [TO arq] [WITH arq] [VER arq] [LIBRARY |
LIB arqs] [MAP arq] [XREF arq] [WIDTH n]
```

O gabarito das palavras-chave é:

```
"FROM = ROOT,TO/K,WITH/K,VER/K,LIBRARY =
LIB/K,MAP/K,XREF/K,WIDTH/K"
```

Nas linhas acima, *arq* significa o nome de um arquivo, *arqs* indica zero ou mais nomes de arquivos separados por uma vírgula ou sinal de adição (+), e *n* é um número inteiro.
Seguem-se alguns exemplos de usos válidos do comando ALINK:

```
ALINK a
ALINK ROOT a + b + c + d MAP map-file WIDTH 120
ALINK a,b,c TO output LIBRARY :flib/lib,obj/newlib
```

Quando você fornece uma lista de arquivos, o linker os lê na ordem em que você os especificou. Os parâmetros do ALINK possuem o seguinte significado.

FROM: Especifica os arquivos objeto que serão utilizados como a entrada primária binária. O linker sempre copia o conteúdo desses arquivos no arquivo de carregamento para formar parte da raiz de overlay. Pelo menos uma entrada primária binária deverá obrigatoriamente ser especificada. ROOT é um sinônimo de FROM.

TO: Especifica o destino para o arquivo de carregamento. Se este parâmetro for omitido, o linker não efetuará a segunda passagem.

WITH: Especifica arquivos contendo parâmetros para o linker (por exemplo, linhas de comando normais). Em geral você irá utilizar apenas um arquivo aqui, mas para aumentar a abrangência do comando, uma lista de arquivos pode ser fornecida. Note que os parâmetros da linha de comando cancelam os dos arquivos WITH. Você encontra uma descrição completa da sintaxe desses arquivos no tópico "Arquivos WITH", logo a seguir neste capítulo.

VER: Especifica o destino para as mensagens do linker. Se você não indicar um destino com esta opção, o linker enviará todas as mensagens para a saída default (normalmente, o console).

LIBRARY: Especifica os arquivos que deverão ser examinados como uma biblioteca. O linker incluirá somente referências a segmentos de código. LIB é uma forma alternativa para LIBRARY.

MAP: Especifica o destino para o mapa de ligações.

XREF: Especifica o destino para a saída da referência cruzada.

WIDTH: Especifica a largura que o linker utiliza para produzir a saída do mapa de ligações e da tabela de referência cruzada. Se enviar a saída para uma impressora, você poderá precisar usar este parâmetro.

Arquivos WITH

Os arquivos WITH contêm parâmetros para orientar o funcionamento do linker. Você pode utilizar esses arquivos a fim de poupar a digitação de uma longa e complexa linha de comando do ALINK repetidas vezes.

Um arquivo WITH consiste numa série de parâmetros (um parâmetro por linha), cada qual consistindo em uma palavra-chave seguida por dados. Você pode terminar uma linha com um ponto e vírgula (;) e o linker ignorará o restante da linha, que poderá ser utilizada para a inclusão de um comentário. O linker ignora também as linhas em branco.

As palavras-chave disponíveis são as seguintes:

```
FROM (ou ROOT) arqs
TO arq
LIBRARY arqs
MAP  [arq]
XREF [arq]
OVERLAY
especificação de árvore
#
WIDTH n
```

Onde *arq* significa um nome de arquivo, *arqs* indica o nome de um ou mais arquivos, *[arq]* é um nome de arquivo opcional e *n* é um número inteiro. Você pode usar um asterisco (*) para dividir linhas longas. Incluindo-o no final de uma linha, a impressora interpretará a linha seguinte como uma continuação da primeira. Se o nome do arquivo após MAP ou XREF for omitido, a saída será direcionada para o arquivo VER (por default, o terminal).

Os parâmetros indicados na linha de comando cancelam os parâmetros existentes num arquivo WITH, de maneira que você pode introduzir pequenas variações em links padrões combinando os parâmetros de uma linha de comando com um arquivo WITH. Além disso, se você especificar um parâmetro mais de uma vez num arquivo WITH, o linker utilizará a primeira ocorrência.

Nota: No segundo exemplo abaixo, isto se aplica mesmo que o primeiro valor indicado para um parâmetro seja nulo.

Veja uma comparação de alguns arquivos WITH com suas respectivas chamadas ALINK:

```
ALINK WITH arqlinq
```

onde o arquivo *arqlinq* contém:

```
FROM obj/main,obj/s
TO bin/test
LIBRARY obj/lib
MAP
XREF xo
```

é o mesmo que especificar:

```
ALINK FROM obj/main,obj/s TO bin/test LIBRARY obj/lib XREF xo
```

O comando:

```
ALINK WITH link LIBRARY ""
```

onde *link* contém:

```
FROM bin/prog,bin/subs
LIBRARY nag/fortlib
TO linklib/prog
```

equivale à linha de comando:

```
ALINK FROM bin/prog,bin/subs TO linklib/prog
```

Nota: No exemplo acima, o parâmetro nulo ("") para LIBRARY na linha de comando cancela o valor *naglib /fortlib* no arquivo WITH. Com isto, o linker não lerá nenhuma biblioteca.

Erros e Outras Exceções

Vários erros podem ocorrer durante a execução do linker. A maioria das mensagens são auto-explicativas e se referem a falhas ocorridas na tentativa de abrir arquivos, ou a erros no formato do comando ou do arquivo binário. Após um erro, a execução do linker se encerrará.

Há algumas mensagens que servem apenas como um aviso. As mais importantes se referem a símbolos não definidos, ou que foram definidos mais de uma vez. A execução do linker normalmente não é encerrada após a ocorrência de um aviso.

Se ainda existirem quaisquer símbolos não definidos ao final da primeira passagem, o linker produzirá um aviso e uma tabela relacionando esses símbolos. Durante a segunda passagem, as referências a esses símbolos se transformarão em referências à **locação zero**.

Se encontrar mais de uma definição para um mesmo símbolo durante a primeira passagem, o linker emitirá um aviso e ignorará a última definição. O linker não produzirá esta mensagem caso a segunda definição ocorra num arquivo de biblioteca. Assim, você poderá substituir rotinas de bibliotecas sem obter mensagens espúrias. Um erro sério ocorrerá caso o linker encontre referências simbólicas inconsistentes, neste caso, a linkagem será imediatamente encerrada.

Uma vez que o linker utiliza somente a primeira definição de qualquer símbolo, é importante compreender a seguinte ordem em que os arquivos são lidos:

1. Entrada primária (FROM ou ROOT).

2. Arquivos overlay.

3. Arquivos de bibliotecas (LIBRARY).

Dentro de cada grupo, o linker lerá os arquivos na ordem em que você os tiver especificado na lista de arquivos. Desse modo, as definições na entrada primária cancelarão aquelas existentes nos arquivos overlay, enquanto as definições das bibliotecas terão a mais baixa prioridade.

Saídas MAP e XREF

O mapa de ligações, produzido pelo linker após a primeira passagem, lista todos os segmentos de código que serão copiados no arquivo de carregamento por ocasião da segunda passagem, na ordem em que estes deverão ser gravados.

Para cada segmento, o linker grava um header contendo o nome do arquivo (truncado para oito letras), o número de referência, o tipo (isto é, DATA, CODE, BSS ou COMMON), e o tamanho do segmento de código. Se o segmento de código estiver num arquivo overlay, o linker fornecerá também o nível e a ordenada do overlay.

Após o header, o linker imprime cada símbolo definido no segmento de código, junto com seu valor correspondente. Os símbolos são exibidos na ordem ascendente de seus valores, sendo anexado um asterisco (*) aos valores absolutos.

O valor do parâmetro WIDTH determina o número de símbolos que poderão ser impressos em cada linha. Se esse valor for muito pequeno, o linker exibirá um símbolo em cada linha.

A saída de referência cruzada também lista cada segmento de código, com o mesmo header incluído no mapa.

O header é seguido por uma lista de símbolos com suas referências. Cada referência consiste num par de números inteiros, indicando o offset da referência e o número do segmento de código na qual ela ocorre. O número do segmento de código se refere ao número atribuído em cada um dos headers.

Overlays

O sistema de overlay automático e o supervisor de overlay disponível no linker permitem que os programas ocupem menos memória ao serem executados, sem a necessidade de se fazer alterações na estrutura dos programas.

Ao empregar o recurso de overlay, você deverá considerar o programa como uma estrutura de **árvore**, cuja **raiz** será a entrada primária binária, junto com os segmentos de código das bibliotecas e os blocos COMMON. Esta raiz será sempre residente na memória. Os arquivos overlay formarão então os outros nós da árvore, de acordo com as especificações da diretiva OVERLAY.

Normalmente, quando for utilizado o recurso de overlay, a saída do linker será um arquivo binário que irá consistir de todos os segmentos de código junto com as informações indicando a posição dentro do arquivo de cada nó da árvore de overlay. Quando você carregar o programa, somente a raiz será carregada na memória. Um supervisor de overlay se encarregará de carregar e remover os segmentos overlay automaticamente. O linker inclui este supervisor de overlay no arquivo de saída produzido durante uma linkagem utilizando overlays. O supervisor de overlay é invisível para o programa sendo executado.

A Diretiva OVERLAY

Para especificar a estrutura de árvore de um programa para o linker, é necessário utilizar a diretiva OVERLAY. Esta diretiva é excepcional, uma vez que você só poderá usá-la com arquivos WITH. Como acontece com os outros parâmetros, o linker levará em conta a primeira diretiva OVERLAY que você fornecer.

O formato desta diretiva é:

```
OVERLAY

Xarqs

.

.

.

#
```

Nota: A diretiva OVERLAY pode ocupar várias linhas. O linker reconhece um símbolo # ou um final-de-arquivo (EOF) como o término da diretiva.

Cada linha após OVERLAY especifica um nó da árvore, e consiste de um contador "X" e uma lista de arquivos.

O nível de um nó especifica a "profundidade" do mesmo na árvore, a começar de zero, que é o nível da raiz (ROOT). O contador "X", especificado na diretiva, consiste de zero ou mais asteriscos, o nível de overlay do nó será X+1.

Assim como o nível, cada nó (exceto o da raiz) possui um valor de **ordenada**. Este valor se refere à ordem na qual o linker deverá ler os descendentes de cada nó, a começar de 1, que indica o primeiro descendente de um nó "pai".

Nota: Podem existir nós com o mesmo nível e ordenada, porém com pais (isto é, origens) diferentes.

Ao ler a diretiva OVERLAY, o linker recorda o nível atual e, para cada novo nó, compara o nível especificado com esse valor. Se o resultado for menor, então o novo nó será considerado um descendente do nó anterior. Se for igual, o novo nó terá o mesmo pai que o nó atual. Se for maior, o novo nó será um descendente direto do atual, e o novo nível deverá ficar um nível acima do valor presente.

Alguns exemplos ajudarão a esclarecer melhor isto:

Os valores de nível e ordenada mostrados acima se referem ao nó especificado na mesma linha. Note que todos os arquivos indicados no exemplos acima poderiam ser também listas de arquivos. Foram usadas letras simples unicamente com o propósito de clareza. Como exemplo, a figura 7-B:

Diretiva	Nível	Ordenada	Árvore
OVERLAY			ROOT
a	1	1	/|\
b	1	2	a b c
c	1	3	
#			
OVERLAY			ROOT
a	1	1	/\
b	1	2	a b
*c	2	1	/|
*d	2	2	c d
#			
OVERLAY			ROOT
a	1	1	
b	1	2	/|\\
*c	2	1	a b e f 1
*d	2	2	/| /|\
e	1	3	c d g h k
f	1	4	/|
*g	2	1	i j
*h	2	2	
**i	3	1	
**j	3	2	
*k	2	3	
1	1	5	
#			

Figura 7-A

```
ROOT bin/mainaaa
OVERLAY
bin/mainbbb,bin/mainccc,bin/mainddd
*bin/makereal
*bin/trbblock,bin/transint,bin/transr
*bin/transcri
bin/outcode
#
```

Figura 7-B

especifica a árvore exibida na seguinte figura:

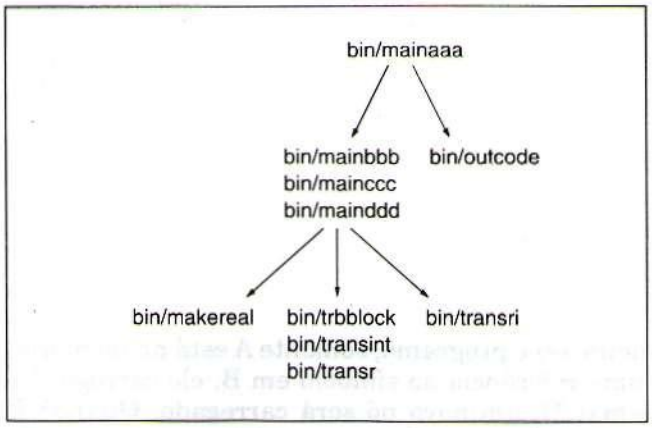


Figura 7-C

Durante a linkagem, o linker lê os arquivos overlay na ordem em que você os especificou na diretiva, linha por linha. O linker preserva esta ordem no mapa e saída de referência cruzada, de maneira que você pode deduzir a exata estrutura da árvore a partir do nível e ordenada do overlay que o linker imprime junto com cada segmento de código.

Referências a Símbolos

Durante a linkagem de um programa que utiliza overlays, o linker verifica a validade de cada referência a um símbolo.

Suponha que a referência esteja no nó "R" de uma árvore, e o símbolo em um nó "S". Esta referência será considerada legal caso uma das seguintes condições seja verdadeira:

1. R e S são o mesmo nó.
2. R é um descendente de S
3. R é o pai de S.

Referências do terceiro tipo acima são conhecidas como **referências de overlay**. Neste caso, o linker entra no supervisor de overlay quando o programa é executado. O supervisor de overlay verifica então se o segmento de códigos contendo o símbolo já está na memória. Caso não esteja, primeiramente o segmento de código (se existir) nesse nível e todos os seus descendentes são removidos, e em seguida o nó contendo o símbolo é carregado na memória. Um segmento de código overlay retorna diretamente para a aplicação que o chamou, e assim não é removido da memória até que outro nó seja carregado por cima dele.

Como exemplo, suponha que árvore seja:

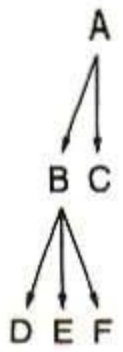


Figura 7-D

Quando o linker carrega pela primeira vez o programa, somente A está na memória. Quando o linker encontra em A uma referência ao símbolo em B, ele carrega B e entra nele. Se por sua vez B chamar D, um novo nó será carregado. Quando B retornar para A, tanto B quanto D serão mantidos na memória, e o linker não precisará carregá-los novamente caso o programa venha a requisitá-los mais tarde. Agora suponha que A chame C. Primeiramente o linker remove os segmentos de código que não são necessários e podem ser sobrepostos – neste caso, B e D. Uma vez recuperada a memória, o linker poderá então carregar C.

Assim, quando o linker executa um determinado nó, todos os "ancestrais" desse nó até a raiz permanecem na memória – e possivelmente também alguns de seus descendentes.

Advertências Sobre Alguns Pontos

O linker assume que todas as referências de overlay são saltos ou chamadas para sub-rotinas, e as direciona através do supervisor de overlay. Deste modo, você não deverá utilizar símbolos de overlay como labels de dados.

Procure evitar códigos impuros ao utilizar overlays, pois o linker nem sempre carrega um nó recente do arquivo de carregamento.

O linker atribui a cada símbolo que possui uma referência de overlay um **número de overlay**. Ele utiliza este valor, que pode ser igual a zero ou maior, para construir o label de entrada do supervisor de overlay associado com este símbolo. Este label obedece ao formato "OVLYnnnn", onde *nnnn* é o número do overlay. Você não deverá utilizar símbolos com este formato em nenhuma outra parte.

O linker une todas as seções do programa com o mesmo nome de seção. Ele faz isto de maneira a poder carregá-las continuamente na memória.

Mensagens e Códigos de Erro

Os erros descritos a seguir deverão ocorrer raramente. Se eles ocorrerem, o erro provavelmente estará no compilador, e não no seu programa. Contudo, você deverá se certificar primeiro de que enviou para o linker um programa apropriado (um programa de entrada, por exemplo, precisa ter uma unidade introdutória que diga ao linker para esperar um programa).

Módulos de Objeto Inválidos

2 Uso inválido de símbolo de overlay.
3 Uso inválido de símbolo.
4 Uso inválido de COMMON.
5 Uso inválido de referência de overlay.
6 Referência de overlay não zero.
7 Relocação de bloco externo inválida.
8 Relocação BSS inválida.
9 Relocação de unidade de programa inválida.
10 Offset inadequado durante relocação de 32 bits.
11 Offset inadequado durante relocação de 6/8 bits.
12 Offset inadequado com referência de 32 bits.
13 Offset inadequado com referência de 6/8 bits.
14 Final de arquivo não esperado.
15 Falta HUNK_END.
16 Terminação de arquivo inválida.
17 Terminação de arquivo prematura.
18 Terminação de arquivo prematura.

Erros Internos

19 Tipo inválido na lista de hunks.
20 Erro interno durante varredura da biblioteca.
21 Argumento inválido.
22 Símbolo não definido na segunda passagem.

Capítulo 8

Entrada e Saída dos Dispositivos do AmigaDOS

O AmigaDOS utiliza *handlers* e *filesystems* com a finalidade de proporcionar um método padronizado de interação com dispositivos físicos de I/O. Os processos handlers e filesystems são similares. Os handlers são subconjuntos dos filesystems, sendo capazes de suportar apenas algumas operações de I/O, ao passo que os filesystems podem suportar também operações com arquivos e diretórios. Os handlers e filesystems residem na ROM ou no diretório L: do Amiga.

Os handlers e filesystems são freqüentemente chamados de "dispositivos do AmigaDOS", mas tenha em mente que um dispositivo do AmigaDOS é diferente de um dispositivo Exec. Os dispositivos do AmigaDOS recebem nomes que aparecem dentro do espaço de nomes do DOS, tais como SER:, RAM: ou DF0:, o que difere das denominações como serial.device ou trackdisk.device que se aplicam aos dispositivos Exec. Os dipositivos do AmigaDOS são freqüentemente montados sobre os dispositivos Exec, utilizando os dispositivos Exec para executar as operações de baixo nível.

Exemplos desta categoria incluem:

- O handler de portas (SER:, PAR: e PRT:), que é montado sobre o serial.device, parallel.device e printer.device.
- O filesystem DF0:, DF1:, que é montado sobre o trackdisk.device.
- O handler do console (CON:), que é montado sobre o console.device.

Um handler ou filesystem não precisa necessariamente ser montado sobre dispositivo Exec. Em alguns casos, o handler pode manejar os seus próprios recursos. Por exemplo: no caso do RAM-handler, o recurso sendo mantido é a memória RAM. Apesar de a memória utilizada pelo RAM-handler ser alocada pelo Exec, não existe nenhum dispositivo Exec sob este handler.

Note que ao contrário do que acontece com um dispositivo Exec, cada handler ou filesystem em execução deve possuir o seu próprio processo.

Os Dispositivos do AmigaDOS

A seguir você tem uma lista dos dispositivos do AmigaDOS implementados como handlers. Observe que alguns handlers possuem mais de um nome (RAW: e CON: são o mesmo handler, porém com nomes diferentes. Os handlers de portas SER:, PAR: e PRT: também são implementados como um único handler com mais de um nome).

AUX: O handler AUX: proporciona entrada e saída serial não bufferizadas. Basicamente este é um handler de console que utiliza a porta serial em vez da tela ou teclado do Amiga. O comando NEWSHELL AUX:, por exemplo, permite que você execute um Shell através da porta serial.

CON: Este handler proporciona entrada e saída bufferizadas para a tela e o teclado, e permite a definição de uma nova janela para a saída. Através de CON:, as teclas digitadas são bufferizadas e ** da aplicação até que o usuário pressione Return. A entrada do teclado é filtrada: as teclas de função e do cursor não são transmitidas. As outras teclas são automaticamente ecoadas na janela CON:.

A janela pode ser criada através da especificação x/y/largura/altura/título, onde *x* e *y* correspondem à distância do topo e da extremidade esquerda da tela a partir da qual a janela deverá ser aberta. Por exemplo: o comando TYPE >CON:5/5/100/100/Saída DEVS:mountlist exibe o conteúdo do arquivo mountlist em uma nova janela intitulada Saída, que terá 100 X 100 pixels e será posicionada 5 pixels abaixo e à direita do canto superior esquerdo da tela.

Ao invés de usar uma nova janela, você pode enviar a saída para a janela atual utilizando um asterisco (*) no lugar da especificação CON:x/y/largura/altura/título.

As versões 2.0 e posteriores do AmigaDOS incluem novas palavras-chave que possibilitam uma maior personalização da janela CON:. Estas novas palavras-chave podem aparecer em qualquer ordem após o string de título na especificação CON: (use uma barra para separá-las). As novas palavras-chave são:

AUTO Não abre a janela até ou a menos que uma operação de I/O ocorra.

CLOSE Insere um gadget Close para fechar a janela. Se o usuário fechar a janela, uma tentativa de leitura em CON: retornará -1L, uma tentativa de leitura a partir de RAW: (ou seja, CON: no modo *raw*) retornará o string de escape Raw Event para um gadget Close.

WAIT Aguarda até que o usuário clique o gadget para fechar a janela ou digite Control-\.

WINDOW Oxendr Utiliza a janela apontada por *endr* (pode ser uma janela personalizada).

SCREEN nome Abre a janela especificada por *nome*.

As palavras-chave adicionais BACKDROP, NODRAG, NOBORDER, NOSIZE, SIMPLE e SMART controlam os mesmos atributos da janela que os flags Intuition de mesmo nome.

Outros novos recursos foram acrescentados ao handler CON: nas versões 2.0 e posteriores do AmigaDOS. A linha de comandos pode ser editada com o auxílio das teclas de cursor, Delete e Backspace. O handler CON: V2.0 suporta um buffer de 2K para manutenção de um histórico dos comandos, permitindo recuperar uma linha previamente digitada pressionando-se a tecla do cursor para cima. As teclas Shift-cursor para cima (ou Control-R) procuram no buffer a ocorrência da última linha digitada que combine com um string parcialmente digitado. Shift-cursor para baixo (ou Control-B) movem para o final do buffer com o histórico dos comandos. As operações adicionais de edição são as seguintes:

Control K	Deleta todos os caracteres do cursor até o fim da linha.
Control U	Deleta todos os caracteres do cursor até o início da linha.
Control X	Deleta a linha inteira.
Control W	Move o cursor para a próxima posição de tabulação.
Control A	Move o cursor para o início da linha (Shift-cursor para a esquerda faz a mesma coisa).
Control Z	Move o cursor para o fim da linha (Shift-cursor para a direita faz a mesma coisa).

Além dos recursos de edição de linhas, alguns recursos de cópia e *paste* foram acrescentados ao handler de console na versão 2.0 do AmigaDOS. O usuário pode selecionar um bloco de texto numa janela do console com o mouse e copiá-lo num buffer interno pressionando as teclas Right-Amiga-C (uma seleção estendida é também suportada utilizando-se as teclas Shift). O texto selecionado pode então ser inserido (*paste*) em outra janela do console por meio das teclas Right-Amiga-V. O texto é inserido no stream de leitura tal como se tivesse sido digitado manualmente.

Um utilitário especial chamado Conclip (parte da Startup-sequence na versão V2.0) proporciona um clipboard para dar suporte às operações de cópia e *paste*. Quando o Conclip é executado, o texto do console copiado com Right-Amiga-C é inserido no dispositivo clipboard.device. Operações de *paste* com Right-Amiga-V fazem com que um código especial (<CSI>0 v) seja inserido na stream de leitura ao invés do texto. O handler CON: lê os dados do clipboard quando este código é recebido, de maneira que as aplicações que usam CON: podem dispor do suporte ao clipboard automaticamente. Aplicações que usam o handler RAW: (veja abaixo) deverão dispor do seu próprio suporte para a leitura do clipboard.

Note que durante a utilização dos handlers CON: e RAW:, se o flag SMART for usado na especificação da janela, somente as operações de *paste* com o texto serão suportadas. A operação de seleção do texto (*cut*) não funciona com o flag SMART.

RAW: Proporciona entrada e saída não bufferizadas para tela e teclado e permite a especificação de uma nova janela para saída, tal como CON: (na verdade, RAW: e CON: são implementados como um único handler com dois nomes e correspondentes modos de operação). Com RAW: as teclas digitadas não são bufferizadas, e podem ser lidas por uma aplicação imediatamente. Além disso a entrada/saída do teclado não é filtrada, o que possibilita o processamento de todas as teclas. As teclas digitadas não são automaticamente ecoadas na janela RAW:.

NEWCON: (Obsoleto) Este handler, implementado apenas na versão V1.3 do sistema operacional do Amiga, funcionava como uma alternativa para o handler CON: original. O handler de console original não possuía funções de edição de linhas, mas estas funções foram incorporadas a CON: nas versões V2.0 e posteriores do AmigaDOS.

SER: O handler SER: proporciona uma interface orientada por streams para a porta serial (uma interface deste tipo permite que você trate o dispositivo físico como se fosse um arquivo).

PAR: O handler PAR: proporciona uma interface orientada por streams para a porta paralela (uma interface deste tipo permite que você trate o dispositivo físico como se fosse um arquivo).

PRT: O handler PRT: proporciona uma interface orientada por streams para a impressora, aceitando também os códigos padrão de impressoras e traduzindo-os na seqüência de comandos utilizada pelo driver da impressora atualmente selecionada.

NIL: O handler NIL: proporciona um destino conveniente para a saída de comandos na qual você não está interessado. Como exemplo, MOUNT>NIL: AUX: monta o dispositivo AUX: sem imprimir nenhuma mensagem de diagnóstico na tela. Note que NIL: na verdade é um handler simulado mantido dentro do AmigaDOS, e não um processo separado.

PIPE: O handler PIPE: é um mecanismo destinado a proporcionar um sistema conveniente de comunicação de I/O bufferizado entre programas. Quando uma operação de saída é efetuada tendo PIPE: como destino, até 4 Kbytes de dados são bufferizados antes que o processo de gravação dos dados seja bloqueado. Depois que um processo escrever os dados, qualquer outro processo poderá acessar PIPE: e ler esses dados. Este recurso pode ser útil, por exemplo, quando você está utilizando duas aplicações e deseja transferir uma grande quantidade de dados de uma (escrita) para outra (leitura), sem criar um arquivo temporário em RAM: ou no disco.

SPEAK: O handler SPEAK: proporciona a saída de voz sintetizada do Amiga. Com SPEAK você pode literalmente fazer com que o Amiga "leia em voz alta" o conteúdo de um arquivo. Como exemplo, o comando COPY DEVS:mountlist SPEAK:OPT/f/s160 lerá o conteúdo do arquivo mountlist com uma voz feminina numa velocidade moderada. Além de todas as opções disponíveis para o comando SAY, SPEAK também aceita o0 e o1 (habilita e desabilita o processamento de opções no stream de entrada), a0 e a1 (articula o modo de fonema direto) , e d0 e d1 (habilita uma pausa na sentença quando for lido um CR ou LF).

Comunicação com os Dispositivos do AmigaDOS

Para se comunicar com os handlers e filesystems, normalmente você deverá utilizar as funções de I/O de arquivos do AmigaDOS, tais como Open( ), Read( ) e Write( ). Um método de comunicação de mais baixo nível é utilizar a interface de packet do DOS, o método de comunicação básico entre os diferentes processos. Montada sobre o sistema de passagem de mensagens Exec, a interface de packet proporciona meios padronizados de comunicação interprocessos.

Esta comunicação pode ter lugar de maneira síncrona ou assíncrona (usualmente através do emprego de uma rotina chamada DoPkt( ) que se encarrega de localizar o endereço da tarefa, enviando-se a mensagem via PutMsg( ), e aguardando por intermédio da função Wait( ) o retorno do sinal reservado do packet do DOS). As chamadas da biblioteca do DOS que se comunicam com os handlers – Read( ), Write( ), Open( ) – utilizam a interface de packet.

O dos.library converte estas chamadas em packets, envia-os para o handler apropriado e retorna o resultado para a chamada. Em comparação com o procedimento para usar a interface de packet diretamente, muito pouco trabalho adicional é necessário para se utilizar as chamadas da biblioteca. Perde-se, entretanto, a capacidade de executar operações de I/O assíncrono com facilidade, de sorte que você pode preferir utilizar a interface de packet ao invés da interface de funções, neste caso. Para maiores informações sobre os packets, veja a seção intitulada "Os Packets do AmigaDOS", no Capítulo 11.

Parte III

MANUAL DE REFERÊNCIAS TÉCNICAS

Capítulo 9

O Sistema de Arquivo

Este capítulo descreve o sistema de arquivo (*filing system*) do AmigaDOS, incluindo informações que revelam como recuperar o conteúdo de um disco danificado por erros do hardware.

Estrutura dos Arquivos do AmigaDOS

O processo de manipulação de arquivos do AmigaDOS utiliza um disco formatado com blocos (ou setores) de igual tamanho. O sistema possibilita uma hierarquia indefinidamente profunda de diretórios, onde cada diretório pode conter arquivos e outros diretórios, ou simplesmente arquivos. Esta é uma estrutura de árvore pura – ou seja, laços não são permitidos.

Existe suficiente redundância no mecanismo para permitir que você possa recuperar a maior parte (senão a totalidade) do conteúdo de um disco após um sério erro de hardware. Para poder recuperar o conteúdo de um disco, contudo, antes você precisa saber como o mesmo é organizado. As subseções abaixo descrevem a organização dos blocos de um disco.

Existem dois tipos básicos de blocos de discos no Amiga, uma vez que o AmigaDOS possui dois diferentes sistemas de arquivo: o OFS e o FFS. O OFS (de *Old Filing System*) é o sistema de arquivo utilizado nas versões V1.2 e anteriores do AmigaDOS. O FFS (de *Fast Filing System*) é um sistema mais recente e mais rápido, implementado nas versões V1.3 e posteriores do AmigaDOS. O FFS preserva a compatibilidade com o OFS, o que significa que o novo sistema pode ler e gravar dados normalmente em discos criados com o antigo sistema de arquivo. Este capítulo lista a estrutura de cada bloco de um disco, tanto do OFS como do FFS.

Root Block

A raiz da árvore é o *root block* (bloco raiz), que fica num lugar fixo do disco. Esta raiz é semelhante à de qualquer outro diretório, com a diferença de que não possui pai e o seu tipo secundário é diferente. O AmigaDOS armazena o nome do volume do

disco no campo DISK NAME do root block.

Cada bloco do sistema de arquivo contém um checksum (somatório de verificação), onde a soma (ignorando-se o "vai 1") de todas as palavras no bloco é zero.

A figura abaixo descreve a estrutura do root block do Old Filing System (OFS) e do Fast Filing System (FFS).

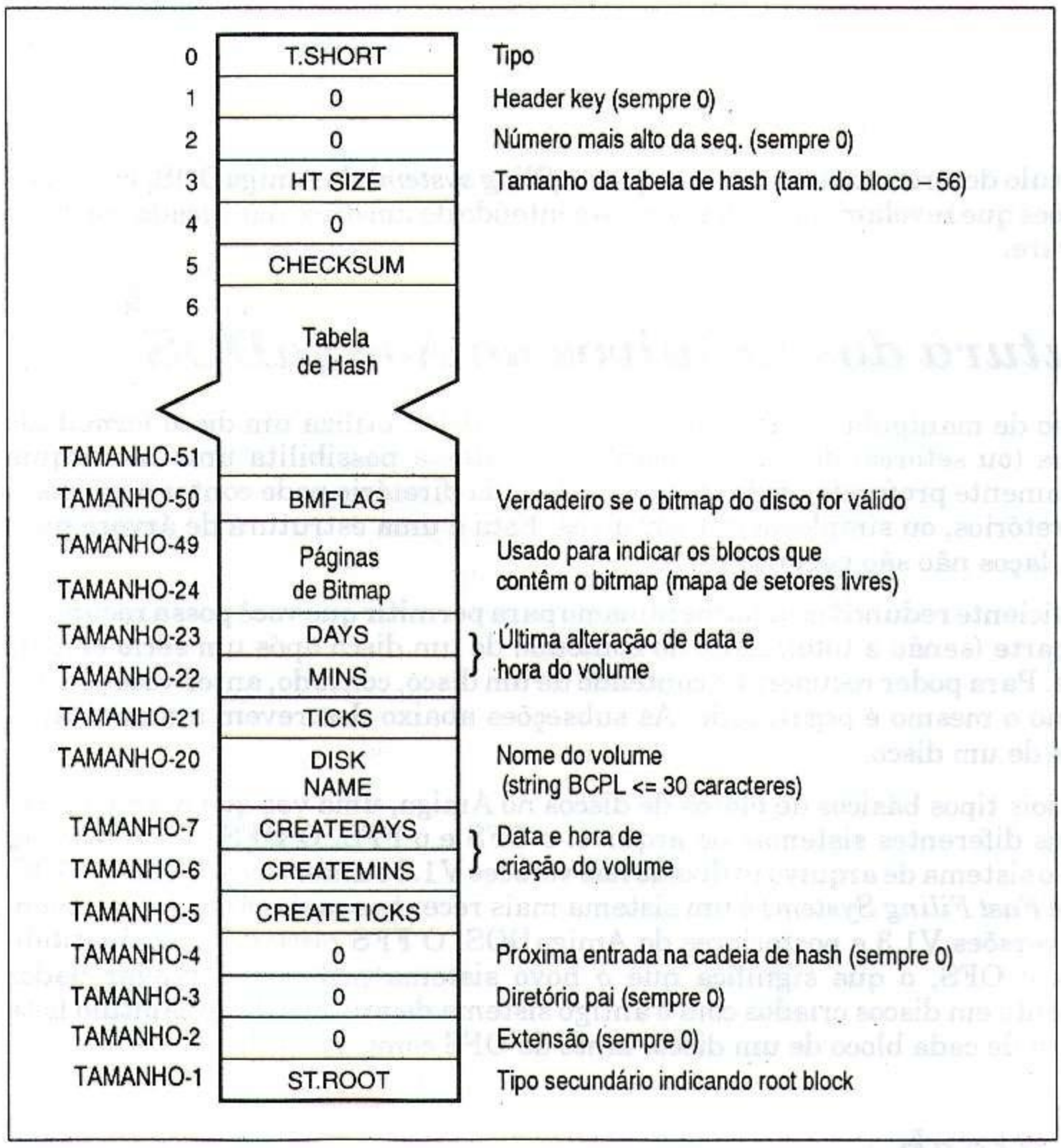


Figura 9-A Root block do OFS.

Root Block do FFS

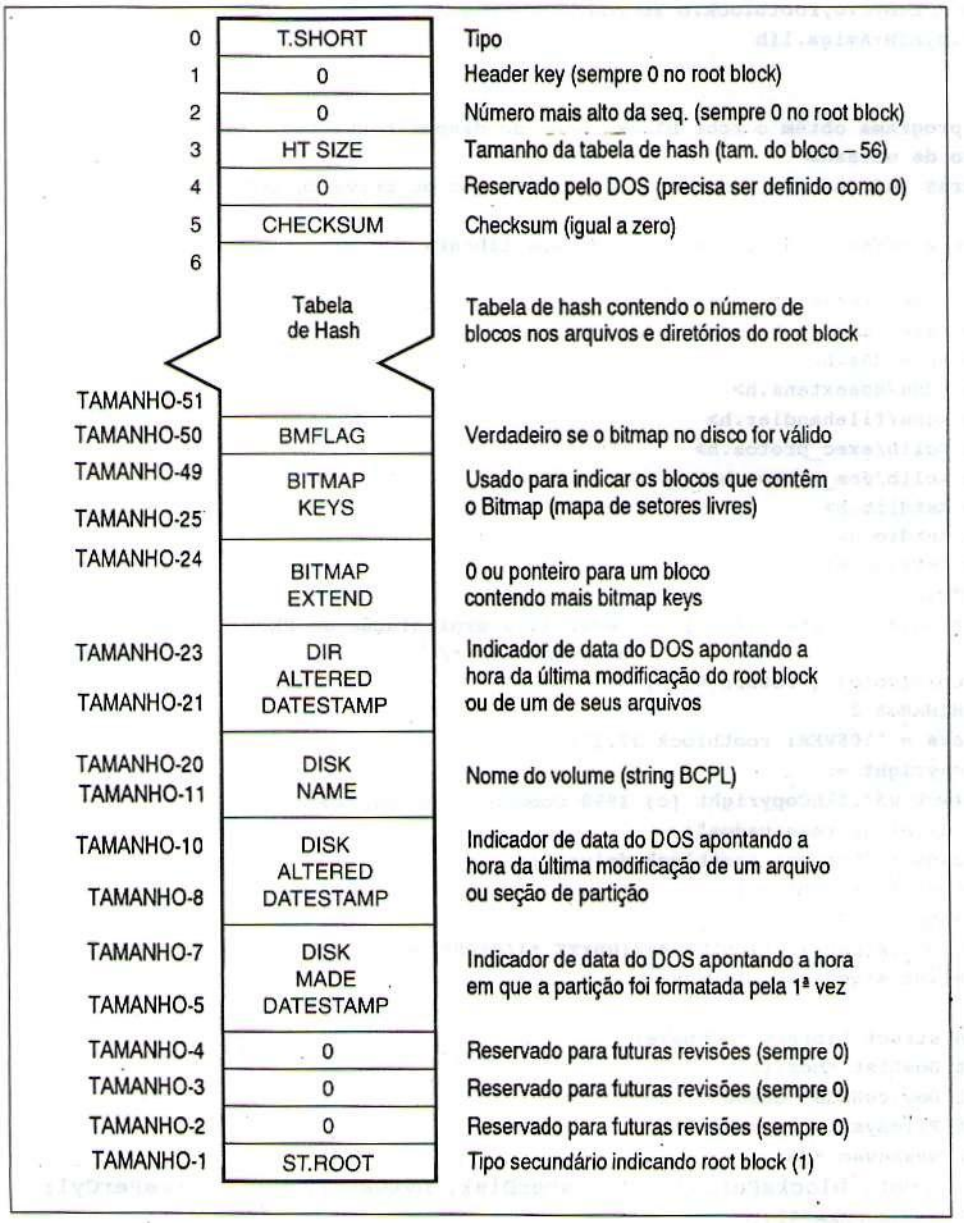


Figura 9-B Root block do FFS.

O programa em C listado abaixo mostra como calcular a localização do root block num disco do Amiga. Este programa também demonstra como extrair outras informações úteis das listas internas mantidas pelo AmigaDOS.

```
;/* rootblock.c - Execute-me para compilar com o SAS C 5.10
LC -bf -cfistq -v -y -j73 rootblock.c
Blink FROM LIB:c.o,rootblock.o TO rootblock LIBRARY
LIB:LC.lib,LIB:Amiga.lib
quit
*/
/* Este programa obtém o root block, nome do dispositivo Exec,
 * número de unidade
 * e outras informações para um nome de partição ou drive do DOS.
 *
 * Requer a versão V36 ou posterior do dos.library
 */
#include <exec.types.h>
#include <exec/memory.h>
#include <dos/dos.h>
#include <dos/dosextens.h>
#include <dos/filehandler.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
ifdef LATTICE
int CXBRK(void) { return(0); } /* Desabilita manipulação do CRTL-C
                                  pelo LATTICE */
int chkabort(void) { return(0); }
#define MINARGS 2
UBYTE *vers = "\0$VER: rootblock 37.1";
UBYTE *Copyright =
  "rootblock v37.1\nCopyright (c) 1990 Commodore-Amiga, Inc.
Todos os direitos reservados";
UBYTE *usage = "Usagem: rootblock drive:";
void bye(UBYTE *s, int e);
void cleanup(void);
#define BTOCSTR(bstr)  ((UBYTE *)((UBYTE *)(BADDR(bstr)) +1))
void main(int argc, char **argv)
   {
   extern struct Library *DOSbase;
   struct DosList *doslist;
   struct DeviceNode *dosdev;
   struct FileSysStartupMsg *fss;
   struct DosEnvec *de;
   ULONG  root, blocksPerCyl, blocksPerDisk, bytesPerBlock, bytesPerCyl;
   UBYTE  dosdevname[32];
   int 1;
   if(((argc)&&(argc<MINARGS))||(argv[argc-1][0]=='?'))
     {
```

```
    printf("%s\n%s\n",Copyright,usage);
    bye("",RETURN_OK);
    }
  if(DOSBase->lib_Version < 36)
    bye("Este programa requer a dos.library V36 ou posterior\n",RETURN_FAIL);
  /* elimina os dois pontos (:), se houver */
  strcpy(dosdevname,argv[1]);
  l = strlen(dosdevname);
  if(dosdevname[l-1]==':') dosdevname[l-1]='\0';
  if(!(doslist = AttemptLockDosList(LDF_DEVICES|LDF_READ)))
    bye("Não pude ativar lock sobre a dos list\n",RETURN_FAIL);
  if(!(dosdev=(struct DeviceNode *)
    FindDosEntry(doslist,dosdevname,LDF_DEVICES)))
      {
      UnLockDosList(LDF_DEVICES|LDF_READ);
      bye("Dispositivo não encontrado\n",RETURN_FAIL);
      }
  /* dosdev é inicializado para apontar para um DeviceNode do DOS */
  fss = (struct FileSysStartupMsg *)BADDR(dosdev->dn_Startup);
  de = (struct DosEnvec *)BADDR(fss->fssm_Environ);
  blocksPerCyl = de->de_BlocksPerTrack * de->de_Surfaces;
  blocksPerDisk = blocksPerCyl * (de->de_HighCyl - de->LowCyl +1);
  root = (blocksPerDisk - 1 + de->de_Reserved) >> 1;
  /* de_SizeBlock é expresso em palavra longa */
  bytesPerBlock = de->de_SizeBlock << 2;
  bytesPerCyl = bytesPerBlock * bytesPerCyl;
  printf("Dispositivo do DOS: %s\n",dosdevname);
  printf("root block: %ld\n",root);
  printf("LowCyl=%ld   HighCyl=%ld   Reserved=%ld\n",
       de->de_LowCyl, de->de_HighCyl, de->de_Reserved);
  printf("Surfaces=%ld   BlockSize=ld%(longs)   BlocksPerTrack=%ld\n",
       de->de_Surfaces, de->de_SizeBlock, de->de_BlocksPerTrack);
  printf("Nome do dispositivo Exec:  %s\n",BTOCSTR(fss->fssm_Device));
  printf("Unidade do dispositivo Exec:  %ld\n",fss->fssm_Unit);
  printf("Flags do dispositivo Exec:  %ld\n",fss->fssm_Flags);
  /* desativa lock sobre o dos list assim que terminar */
  UnLockDosList(LDF_DEVICES|LDF_READ);
  bye("",RETURN_OK);
  }
void bye(UBYTE *s, int e)
  {
  if(*s) printf(s);
  cleanup( );
  exit (e);
  }
void cleanup( )
  {
  /* nada */
  }
```

User Directory Blocks

Os *user directory blocks* (blocos de diretório do usuário) possuem o tipo T.SHORT e o tipo secundário ST.USERDIR. As seis palavras de informações no início do bloco indicam também a chave OWN KEY do bloco (ou seja, o número do mesmo) como uma verificação de consistência, e o tamanho da tabela de hash. As 50 palavras de informações no final do bloco contêm a data e hora da criação, o nome do diretório, um ponteiro para o próximo arquivo ou diretório na cadeia de hash, e um ponteiro para o diretório acima.

Para localizar um arquivo ou subdiretório, é necessário primeiramente aplicar uma função hash ao nome do mesmo. Essa função terá como resultado um offset na tabela de hash, que corresponde à chave (ou número) do primeiro bloco na cadeia onde ficam ligados os blocos com o mesmo valor de hash (ou zero, caso não exista nenhum outro bloco com o mesmo valor). O AmigaDOS lê o bloco com essa chave e compara o nome do bloco com o nome solicitado. Se os nomes não coinciderem, ele lê o próximo bloco na cadeia, e assim por diante.

Algoritmo de Hashing

O algoritmo de hashing, que é utilizado para determinar a localização de um cabeçalho de arquivo na tabela de hash de um diretório, é idêntico tanto para o Old Filing System quanto para o Fast Filing System. A maneira mais fácil de descrever este algoritmo é através de uma pequena função em C. Dado um ponteiro CPTR para um string BCPL, esta rotina retornará o índice do array dentro da tabela de hash. Este valor é utilizado para inserir o cabeçalho do arquivo com aquele nome na entrada apropriada da tabela de hash do diretório pertinente.

```
Hash( name )
unsigned char *name;
{
int val,len,i;
     val = len = (int)*name++;
     for(i=0; i<len; i++) val = ((val*13) + (int)toupper( *name++ ))&0x7ff;
     return(val % 72);
}
```

Uma grande diferença entre o OFS e o FFS é que no Fast Filing System as cadeias de hash precisam ser ordenadas. Quando uma colisão de hashing ocorre, os arquivos são ligados utilizando-se a entrada HASHCHAIN do bloco de cabeçalho de arquivo ou do bloco de diretórios do usuário. O Old Filing System insere o novo cabeçalho no início da lista, enquanto que o Fast Filing System intercala o novo cabeçalho na lista na ordem ascendente dos blocos.

As figuras abaixo mostram a organização de um bloco de diretórios do usuário no Old Filing System (OFS) e no Fast Filing System (FFS).

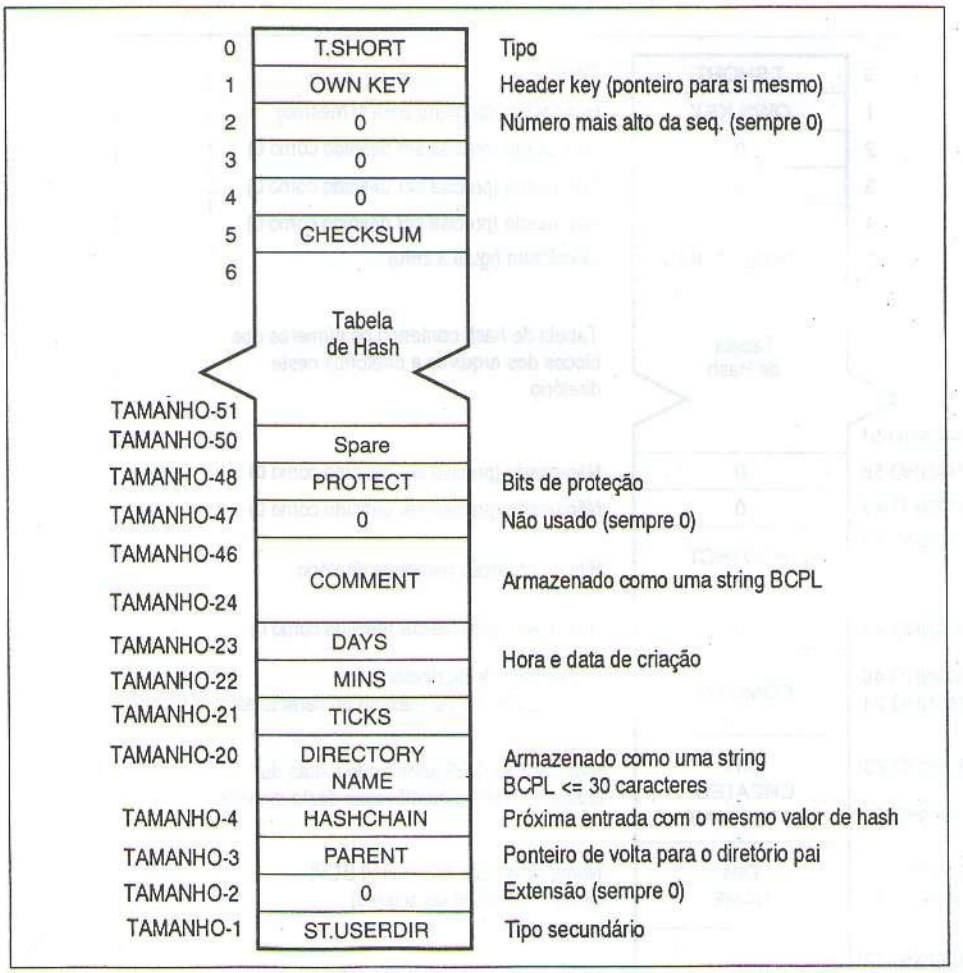


Figura 9-C User directory block do OFS.

User Directory Blocks do FFS

0	T.SHORT	Tipo
1	OWN KEY	Header key (ponteiro para si mesmo)
2	0	Não usado (precisa ser definido como 0)
3	0	Não usado (precisa ser definido como 0)
4	0	Não usado (precisa ser definido como 0)
5	CHECKSUM	Checksum (igual a zero)
6	Tabela de Hash	Tabela de hash contendo os números dos blocos dos arquivos e diretórios neste diretório
TAMANHO-51		
TAMANHO-50	0	Não usado (precisa ser definido como 0)
TAMANHO-49	0	Não usado (precisa ser definido como 0)
TAMANHO-48	PROTECT BITS	Bits de proteção para este diretório
TAMANHO-47	0	Não usado (precisa ser definido como 0)
TAMANHO-46 TAMANHO-24	COMMENT	Comentário deste diretório (string BCPL de no máximo 80 caracteres)
TAMANHO-23 TAMANHO-21	DIR CREATED DATESTAMP	Indicador do DOS apontando a data da criação ou última modificação deste diretório
TAMANHO-20 TAMANHO-12	DIR NAME	Nome deste diretório (string BCPL de no máximo 30 caracteres)
TAMANHO-11 TAMANHO-5	0	Não usados (precisam ser definidos como 0)
TAMANHO-4	HASH CHAIN	0 ou número do bloco do próximo arquivo na cadeia em caso de uma colisão de hashing
TAMANHO-3	PARENT	Número do bloco do pai deste diretório
TAMANHO-2	0	Não usado (precisa ser definido como 0)
TAMANHO-1	ST.USERDIR	Tipo secundário indicando diretório do usuário (2)

Figura 9-D User directory block do FFS.

File Header Block

Cada arquivo terminal inicia com um *file header block* (bloco de cabeçalho de arquivo) que possui o tipo T.SHORT e o tipo secundário ST.FILE. O início e o final do bloco contêm informações de nome, hora e diversas outras, tal como ocorre com um bloco de diretório. O corpo do arquivo consiste de blocos de dados com seqüências de números de 1 em diante. O AmigaDOS armazena os endereços desses blocos em palavras consecutivas, em ordem descendente, a partir do offset [tamanho do bloco] - 51. Em geral o AmigaDOS não utiliza todo o espaço para esta lista, e o último bloco de dados não se encontra preenchido.

File Header Block do OFS e FFS

A figura a seguir descreve a estrutura do file header block.

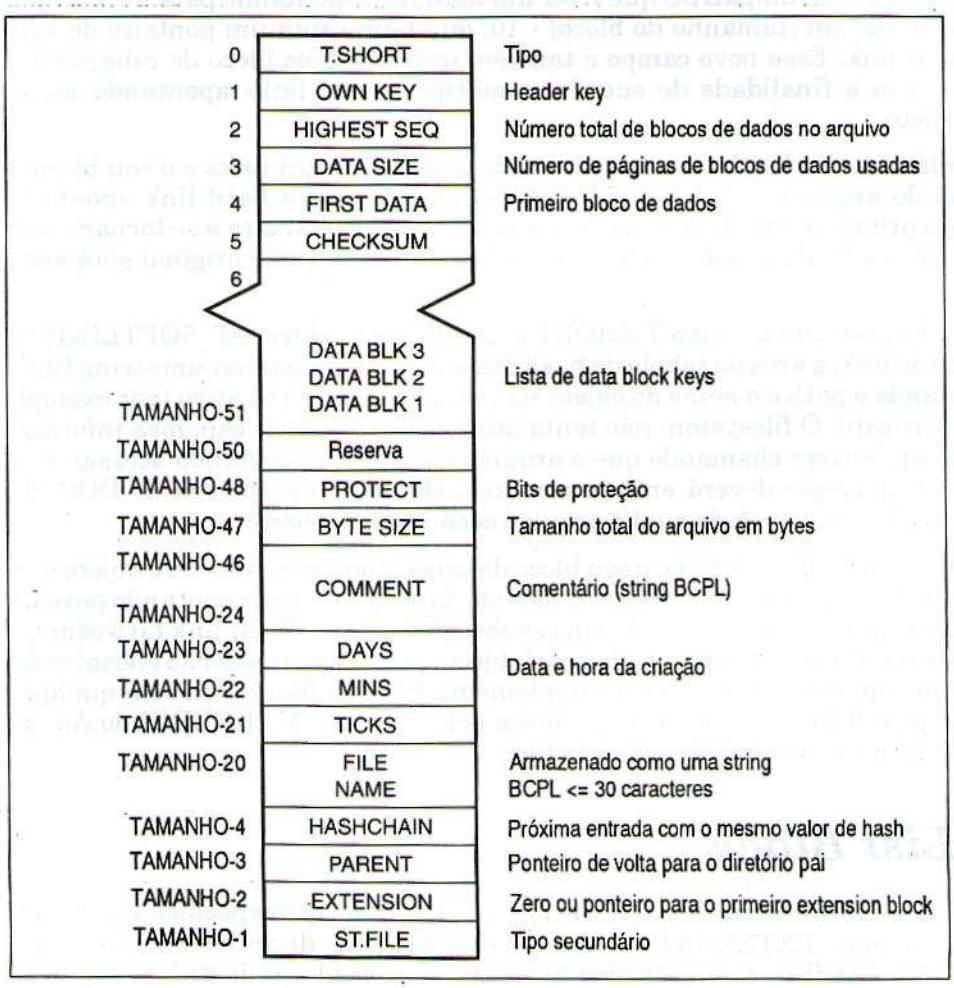


Figura 9-E File header block do OFS e FFS.

Hard Links e Soft Links

Os hard links e soft links constituem um novo recurso acrescentado ao AmigaDOS na versão V2.0. Esses links permitem que um usuário do AmigaDOS se refira a um único arquivo ou diretório através de mais de um nome. Um **hard link** associa um novo nome a um arquivo ou diretório por meio de uma ligação com sua posição física no disco. Já um **soft link** associa um novo nome a um arquivo ou diretório por meio de uma ligação com sua descrição de path. Os hard e soft links são implementados no filesystem como *file header blocks* (blocos de cabeçalho de arquivo) modificados, um novo bloco de cabeçalho de arquivo é acrescentado para cada link criado.

O bloco de cabeçalho para um hard link possui um tipo T.SHORT (2) e um tipo secundário ST_LINKFILE (-4) ou ST_LINKDIR (4), dependendo se a ligação se aplicar a um arquivo ou a um diretório. Os hard links apontam para seu objeto através de um ponteiro de número de bloco armazenado em [tamanho do bloco] - 11.

Qualquer objeto do AmigaDOS que tiver um hard link apontando para ele receberá um novo campo em [tamanho do bloco] - 10, que representa um ponteiro de volta para o hard link. Esse novo campo é também utilizado pelo bloco de cabeçalho do hard link com a finalidade de encadear múltiplos hard links apontando para o mesmo objeto.

Ao ser deletado, um hard link é removido da cadeia de hard links e o seu bloco de cabeçalho de arquivo é liberado. Se o objeto para o qual o hard link aponta for deletado, o primeiro hard link da cadeia será alterado de maneira a se tornar o novo bloco de cabeçalho de arquivo, o bloco de cabeçalho de arquivo original será então liberado.

Os soft links possuem um tipo T.SHORT e um tipo secundário ST_SOFTLINK (3). Neste tipo de link, a área da tabela de hash é usada para armazenar um string BCPL representando o path e o nome do objeto sobre o qual o link está ativo (por exemplo: work:foo/bar/cap). O filesystem não tenta acessar work:foo/bar/cap, mas informa à aplicação que estiver chamando que o arquivo que ela está tentando acessar é um soft link. A aplicação deverá então executar a chamada apropriada do DOS, ReadLink( ), a fim de descobrir qual o arquivo correto a ser aberto.

Quando um soft link é deletado, o seu bloco de cabeçalho é liberado. Se o objeto para o qual o soft link aponta for deletado, o sistema deixa o soft link apontando para um arquivo inexistente. Qualquer referência subseqüente a esse soft link fará com que o AmigaDOS retorne a mensagem de erro "object not found" (objeto não encontrado). Observe que apesar de poderem ser implementados pelo filesystem sem qualquer problema, os soft links não eram suportados pelo comando MAKELINK do AmigaDOS até o término da elaboração deste livro.

File List Block

Se existirem mais blocos no arquivo do que os que podem ser especificados na lista de blocos, o campo EXTENSION terá um valor diferente de zero e apontará para um outro bloco do disco contendo uma lista adicional de blocos de dados.

File List Block do OFS e FFS

A figura a seguir descreve a estrutura do *file list block* (bloco de lista de arquivos).

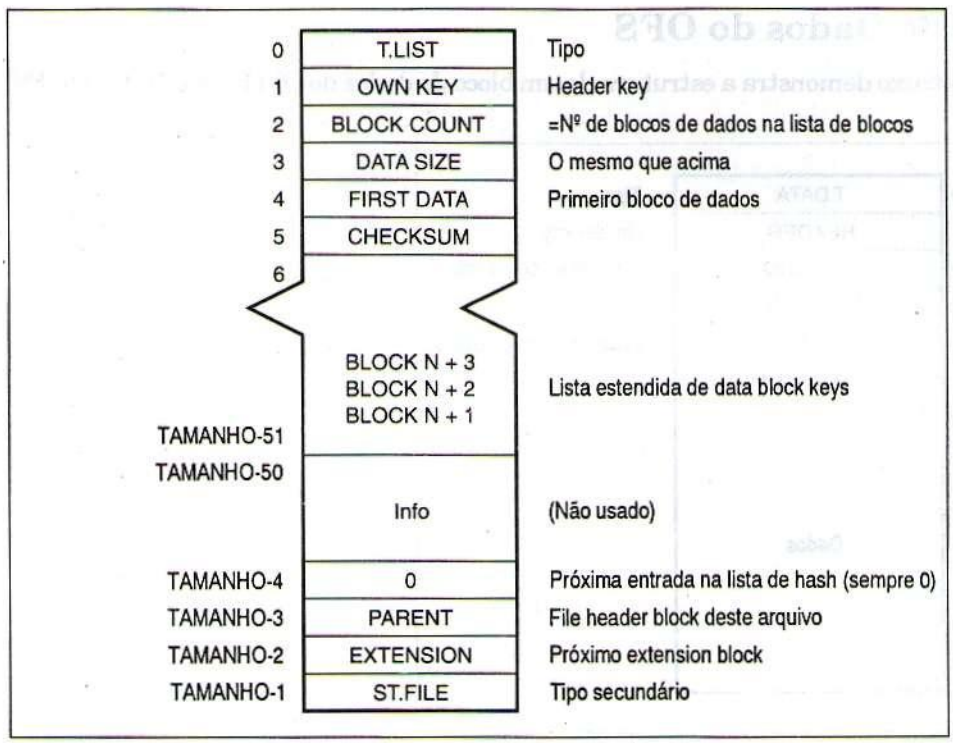


Figura 9-F File list block do OFS e FFS.

Existem tantos blocos de extensões de arquivos quanto necessário para listar os blocos de dados que constituem o arquivo. A estrutura do bloco é bastante similar à do bloco de cabeçalho de arquivo. Apenas o tipo é diferente e os campos referentes à data e ao nome do arquivo não são utilizados.

Blocos de Dados

Os *data blocks* (blocos de dados) contêm apenas seis palavras de informações do sistema de arquivo. Estas seis palavras se referem ao seguinte:

- Tipo (T.DATA)
- Ponteiro para o bloco do cabeçalho do arquivo.
- Número de seqüência do bloco de dados.
- Número de palavras de dados.
- Ponteiro para o próximo bloco de dados.
- Checksum.

Normalmente todos os blocos de dados, com exceção do último, se encontram

preenchidos (isto é, possuem um tamanho igual a [tamanho do bloco] - 6). O último bloco de dados possui um ponteiro para a frente cujo valor é zero. Os blocos de dados do FFS contêm apenas apenas dados do usuário.

Bloco de Dados do OFS

A figura abaixo demonstra a estrutura de um bloco de dados do Old Filing System (OFS).

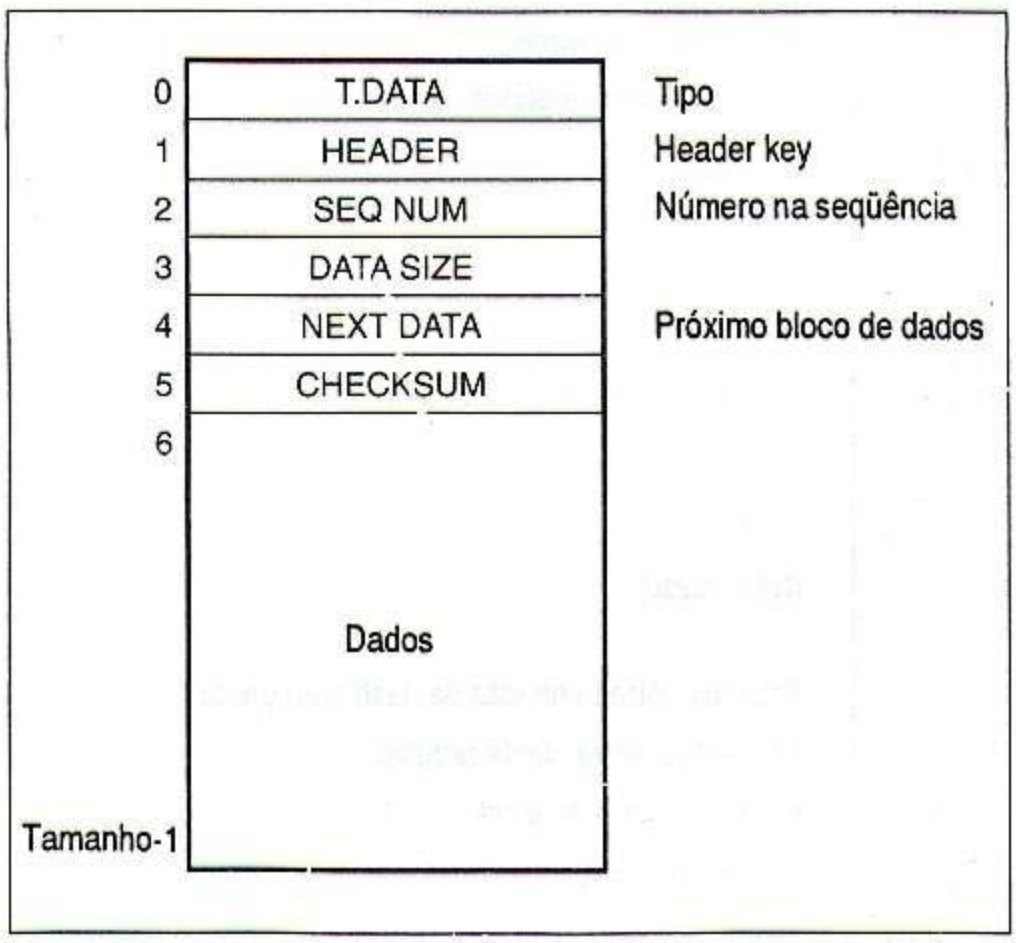


Figura 9-G Bloco de dados do OFS.

Bloco de Dados do FFS

Os blocos de dados utilizados pelo Fast Filing System (FFS) contêm somente dados.

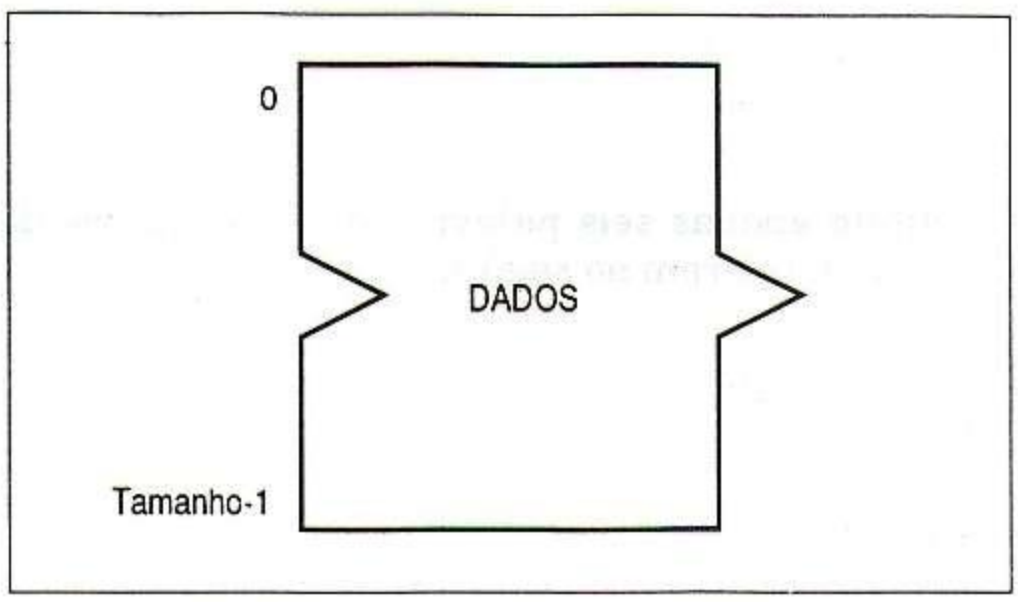


Figura 9-H Bloco de dados do FFS.

Capítulo 10

Estrutura dos Arquivos Binários do Amiga

Introdução

Este capítulo detalha a estrutura dos arquivos binários do Amiga, como os arquivos objeto produzidos pelos assemblers e compiladores. Também será discutido aqui o formato dos arquivos de carregamento binários, que são gerados pelo linker e carregados na memória pelo loader. O formato desses arquivos permite a utilização do recurso de overlay. Além de descrever o formato dos arquivos de carregamento, este capítulo explica a utilização de símbolos comuns, referências absolutas externas e unidades de programa.

Terminologia

Alguns dos termos técnicos empregados neste capítulo são explicados a seguir.

Referências Externas

Você pode utilizar um nome para especificar uma referência entre unidades de programa separadas. A estrutura de dados possibilita a atribuição de nomes com mais de 16 Mbytes, embora o linker restrinja o tamanho dos nomes a 255 caracteres. Ao ligar os vários arquivos objeto em um único arquivo de carregamento, você deverá se certificar de que todas as referências externas são compatíveis com as definições externas. A referência externa pode ter o tamanho de um byte, palavra ou palavra longa. As definições externas se referem a variáveis relocáveis, valores absolutos ou bibliotecas residentes. As referências relocáveis de byte e palavra se referem aos modos de endereçamento relativos ao PC (program counter) do microprocessador, sendo manipuladas exclusivamente pelo linker. Contudo, se você tiver um programa contendo referências relocáveis de palavra longa, a relocação poderá se dar por ocasião do carregamento do programa.

Note que esses tamanhos se referem somente à extensão do campo de relocação. É possível carregar uma palavra a partir de um endereço externo longo, por exemplo,

sem que o linker efetue qualquer verificação quanto à consistência das suas referências e definições externas.

Arquivo Objeto

Os assemblers e compiladores produzem uma imagem binária chamada de *arquivo objeto*. Um arquivo objeto contém uma ou mais unidades do programa, podendo conter também referências externas a outros arquivos objeto.

Arquivo de Carregamento

O linker produz uma imagem binária a partir de vários arquivos objeto. Esta imagem binária é chamada de *arquivo de carregamento*. Um arquivo de carregamento não contém nenhuma referência externa não resolvida.

Unidade de Programa

A *unidade de programa* é o menor elemento com o qual o linker pode lidar. Uma unidade de programa pode conter um ou mais hunks, enquanto que um arquivo objeto pode conter uma ou mais unidades de programa. Se o linker encontrar uma referência externa adequada em uma unidade de programa, ele inspecionará as bibliotecas de consulta e incluirá toda essa unidade de programa no arquivo de carregamento. Para exemplificar, um assembler normalmente produz uma única unidade de programa após uma assemblagem (contendo um ou mais hunks), um compilador como o FORTRAN produz uma unidade de programa para cada sub-rotina, programa principal ou bloco de dados. Os hunks são contados a partir de zero dentro de cada unidade de programa. A única maneira de se referenciar outras unidades de programa é através do uso de referências externas.

Hunks

Um *hunk* é constituído de um bloco de código ou de dados, informações de relocação, e uma lista de símbolos externos definidos ou referenciados. Os hunks de dados podem especificar dados inicializados ou não inicializados. Embora possa conter definições externas, um hunk de dados não inicializados (ou hunk BSS) não pode conter referências externas nem quaisquer valores que necessitem de relocação. Se você utilizar blocos de dados inicializados em overlays, o linker normalmente não alterará esses blocos de dados, uma vez que ele os carrega novamente do disco durante o procedimento de overlay. Os hunks podem ou não ser nomeados. Um hunk também pode conter uma tabela de símbolos que fornece informações especiais para debugging, podendo ainda conter informações adicionais para uso com ferramentas de depuração em linguagens de alto nível. Cada hunk dentro de uma unidade de programa possui um número a partir de zero.

Bibliotecas Residentes

Os arquivos de carregamento são também conhecidos como *bibliotecas residentes*. Tais arquivos permanecem residentes na memória ou, alternativamente, podem ser carregados como parte de uma chamada Open Library( ) do sistema operacional do Amiga. Você pode referenciar bibliotecas residentes por meio de referências externas, as definições estão em um hunk que não contém códigos, mas somente uma

lista de definições de biblioteca residente. O procedimento usual para produzir esses hunks consiste em assemblar um arquivo contendo somente definições absolutas externas, e em seguida passar esse arquivo para uma ferramenta de software especial a fim de converter as definições absolutas em definições de biblioteca residente. O linker adota o nome do hunk como o nome da biblioteca residente e o passa para o arquivo de carregamento, de maneira que o loader possa abrir a biblioteca residente antes de utilizá-la.

Bibliotecas de Consulta

Uma *biblioteca de consulta* é constituída de arquivos objeto contendo unidades de programa que são carregadas somente se houver quaisquer referências externas a serem resolvidas pelas mesmas. Você pode utilizar arquivos objeto como bibliotecas e fornecê-los como a entrada primária para o linker, esse caso a entrada incluirá todas as unidades de programa contidas nos arquivos objeto. Note que é permitido concatenar arquivos objeto.

Nó

Um nó consiste de pelo menos um hunk. Um arquivo de carregamento utilizado para overlay contém um nó raiz, que permanece residente na memória durante toda a execução do programa, e um determinado número de nós de overlay que serão carregados na memória à medida em que forem requisitados.

Estrutura dos Arquivos Objeto

Um arquivo objeto consiste na saída de um assembler ou de um outro programa tradutor de linguagens. Para utilizar um arquivo objeto, é necessário primeiramente resolver todas as referências externas, para isso você deve passar o arquivo objeto pelo linker. Um arquivo objeto é constituído de uma ou mais unidades de programa. As unidades de programa iniciam com um cabeçalho, que é seguido por uma série de hunks encadeados do princípio ao fim. Cada um desses hunks, por sua vez, contém um determinado número de "blocos" de vários tipos, cada bloco inicia com uma palavra longa que define seu tipo, seguido por zero ou outras palavras longas adicionais. Note que cada bloco é arredondado para o próximo limite de palavra longa. O próprio cabeçalho da unidade de programa é um bloco com este formato.

O formato de uma unidade de programa é o seguinte:

- Bloco de cabeçalho da unidade de programa.
- Hunks.

O formato básico de um hunk é o seguinte:

- Bloco de nome do hunk.
- Bloco relocável.
- Bloco de informações de relocação.
- Bloco de informações simbólicas externas.
- Bloco de tabela de símbolos.

- Bloco de debug.
- Bloco final.

Você pode omitir todos esses tipos de bloco, exceto o bloco final.

As subseções a seguir descrevem o formato de cada um desses blocos. O valor da palavra que define o tipo do hunk aparece em notação decimal e hexadecimal após o nome do mesmo. Por exemplo: hunk_unit possui o valor 999 em decimal e 3E7 em hexa.

hunk_unit(999/3E7)

Este bloco especifica o início de uma unidade de programa. Ele consiste em uma palavra que define o tipo, seguida pelo tamanho do nome da unidade em palavras longas e pelo nome propriamente dito, arredondado para um limite de palavra longa e preenchido com zeros, caso necessário. Esquematicamente, o formato deste bloco é o seguinte:

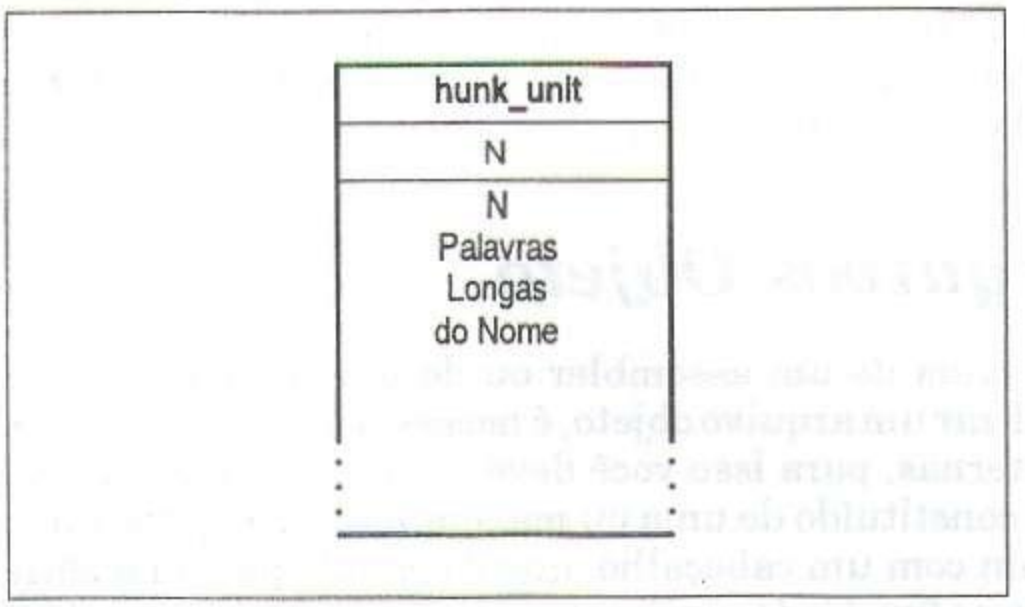


Figura 10-A hunk_unit (999/3E7).

hunk_name (1000/3E8)

Este bloco define o nome de um hunk. Os nomes são opcionais, se encontrar dois ou mais hunks com o mesmo nome, o linker os combinará em um único hunk. Note que referências externas de 8 ou 16 bits relativas ao program counter só poderão ser resolvidas entre hunks com o mesmo nome. Todas as referências externas num arquivo com formato de carregamento estão entre hunks diferentes, requerendo assim referências relocáveis de 32 bits. Entretanto, como o loader dispersa os hunks ao carregá-los na memória, você não deve assumir que eles ficarão dentro de um limite de 32K um do outro. Observe que o tamanho é expresso em palavras longas e o bloco do nome, como todos os blocos, é arredondado para um limite de palavra longa e completado com zeros. O formato deste bloco de hunk é o seguinte:



Figura 10-B hunk_name (1000/3E8).

hunk_code (1001/3E9)

Este bloco define um bloco de código que deverá ser carregado na memória e possivelmente relocado. Seu formato é o seguinte:

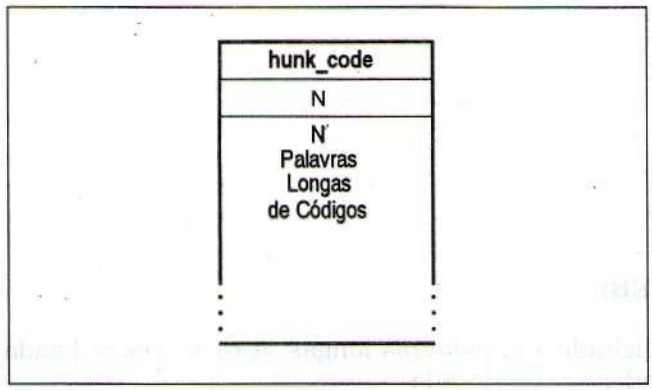


Figura 10-C hunk_code (1001/3E9).

hunk_data (1002/3EA)

Este bloco define um bloco de dados inicializados que deverá ser carregado na memória e possivelmente relocado. O linker normalmente não altera tais blocos se eles fizerem parte de um nó de overlay, uma vez que pode precisar lê-los novamente do disco durante a manipulação do overlay. O formato deste bloco é o seguinte:

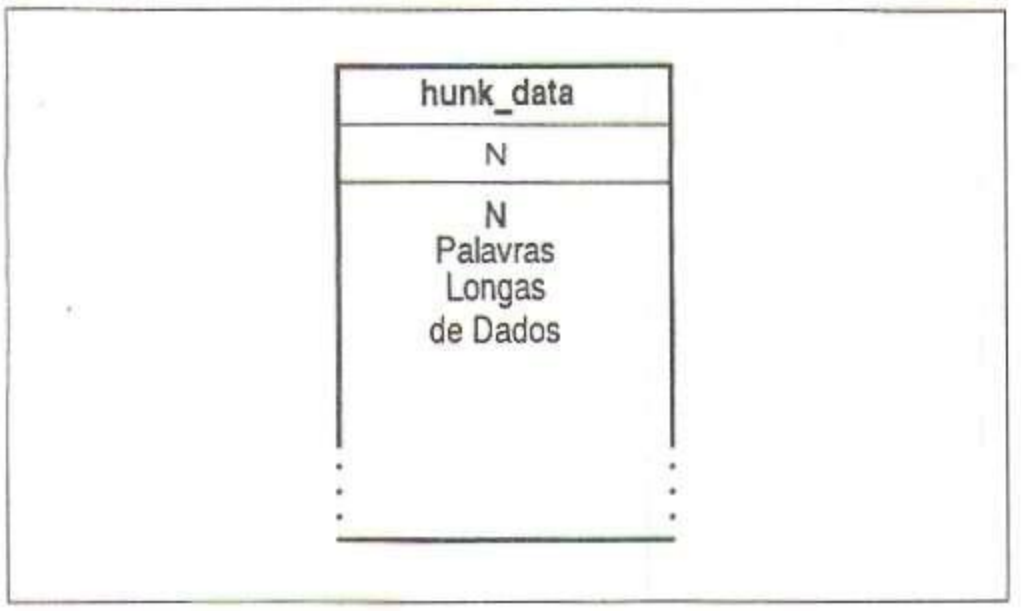


Figura 10-D hunk_data (1002/3EA).

hunk_bss (1003/3EB)

Este bloco especifica um bloco de espaço de trabalho não inicializado que é alocado pelo loader. Os blocos hunk_bss são usados, dentre outras coisas, para stacks e blocos COMMON do FORTRAN. Embora a relocação dentro desse tipo de bloco não seja permitida, você pode definir símbolos dentro de um bloco bss. Seu formato é o seguinte:

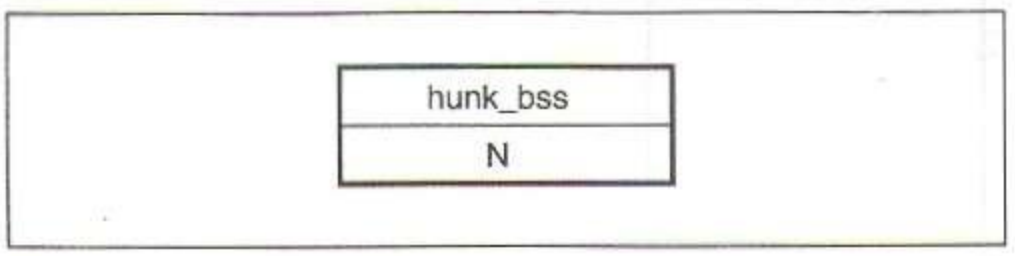


Figura 10-E hunk_bss (1003/3EB).

onde N é o tamanho do bloco solicitado, em palavras longas. A memória utilizada pelos blocos bss é zerada pelo loader ao ser alocada.

O bloco relocável dentro de um hunk deverá ser do tipo hunk_code, hunk_data ou hunk_bss. Um hunk_code contém código de máquina executável. Um hunk_data contém dados inicializados (constantes, etc.) e um hunk_bss contém dados não inicializados (arrays, variáveis, etc.).

Nesses três tipos de hunk, a palavra longa correspondente ao tamanho do hunk é interpretada de uma maneira especial, de acordo com seus dois bits mais significativos:

```
+------------Bit 31 MEMF_FAST
1 +----------Bit 30 MEMF_CHIP
```

0	0	Se nenhum bit estiver setado, o loader obterá o tipo de memória (Chip ou Fast memory) que estiver disponível. A preferência será dada à Fast memory.
1	0	O loader deverá utilizar a Fast memory, caso contrário acusará um erro.
0	1	O loader deverá utilizar a Chip memory, caso contrário acusará um erro.
1	1	Os bits 30 e 31 setados indicam que existem informações adicionais disponíveis na próxima palavra longa. Os 24 bits inferiores da próxima palavra longa representam um tipo que será passado para a função AllocMem( ); os 8 bits superiores estão reservados para futura expansão, não sendo utilizados atualmente.

hunk_reloc32 (1004/3EC)

O bloco hunk_reloc32 especifica uma relocação de 32 bits que o linker deverá efetuar dentro do bloco relocável atual. A informação de relocação consiste em uma referência a uma locação dentro do hunk atual, ou a qualquer outra dentro da unidade de programa. Cada hunk dentro da unidade é numerado a partir de zero. O linker adiciona o endereço da base do hunk especificado a cada uma das palavras longas no bloco relocável precedente, de acordo com o indicado pela lista de offsets. A lista de offsets inclui apenas hunks referenciados, e um valor igual a zero indica o final da lista. O formato deste bloco é o seguinte:

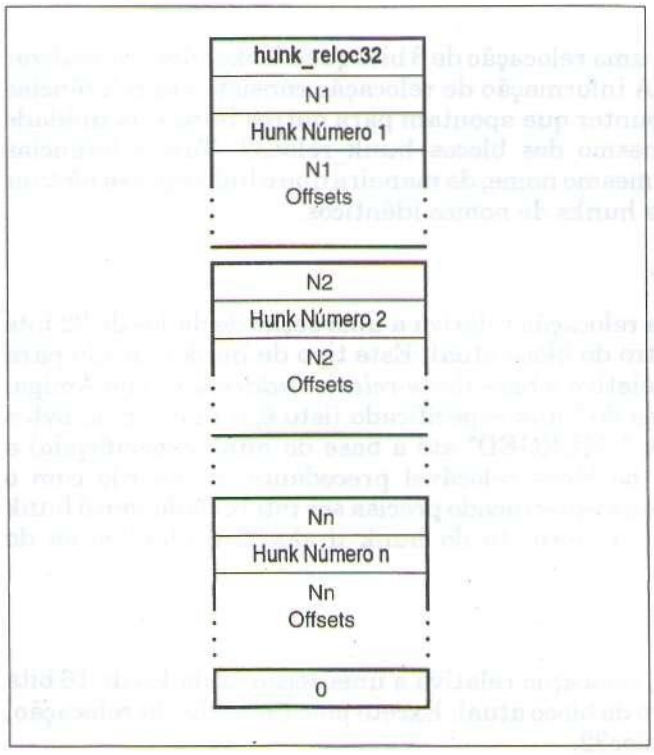


Figura 10-F hunk_reloc32 (1004/3EC).

hunk_reloc32short (1020/3FC)

O bloco hunk_reloc32short especifica uma relocação de 32 bits que o linker deverá efetuar dentro do bloco relocável atual, utilizando quantidades de 16 bits. O formato é idêntico ao do hunk_reloc32 (isto é, os campos a serem modificados são de 32 bits), mas os offsets efetivos e os números de hunks têm 16 bits a fim de poupar espaço e tornar o carregamemto mais rápido.

Esta é uma maneira mais eficiente de codificar a informação de relocação em um arquivo (o hunk_reloc 32 /e composto na maior parte de zeros, uma vez que todos os hunks têm menos de 64K) e é capaz de servir como uma alternativa a hunk_reloc32 para a saída final do linker. Este é um novo hunk que está disponível somente nas versões V2.0 e posteriores do AmigaDOS.

hunk_reloc16 (1005/3ED)

Um bloco hunk_reloc16 especifica uma relocação de 16 bits que o linker deverá realizar dentro do bloco relocável atual. A informação de relocação consiste em referências de 16 bits relativas ao program counter que apontam para outros hunks na unidade de programa. O formato é o mesmo dos blocos hunk_reloc32. Tais referências deverão se aplicar a hunks com o mesmo nome, de maneira que o linker possa efetuar a relocação enquanto aglutina (ou seja, une) os hunks de nomes idênticos.

hunk_reloc8 (1006/3EE)

Um bloco hunk_reloc8 especifica uma relocação de 8 bits que o linker deverá realizar dentro do bloco relocável atual. A informação de relocação consiste em referências de 8 bits relativas ao program counter que apontam para outros hunks na unidade de programa. O formato é o mesmo dos blocos hunk_reloc32. Tais referências deverão se aplicar a hunks com o mesmo nome, de maneira que o linker possa efetuar a relocação enquanto aglutina os hunks de nomes idênticos.

hunk_dreloc32 (1015/3F7)

O bloco hunk_dreloc32 especifica relocação relativa a uma seção de dados de 32 bits que o linker deverá efetuar dentro do bloco atual. Este tipo de hunk é usado para implementar o endereçamento relativo à base (*base-relative addressing*) no Amiga. O linker adiciona o offset da base do hunk especificado (isto é, o número de bytes existentes desde a base do hunk "_MERGED" até a base do hunk especificado) a cada uma das palavras longas no bloco relocável precedente, de acordo com o indicado pela lista de offsets. O hunk especificado precisa ser intercalado com o hunk de dados chamado "_MERGED". O formato do hunk_dreloc32 é idêntico ao do hunk_reloc32.

hunk_dreloc16 (1016/3F8)

O bloco hunk_dreloc16 especifica relocação relativa a uma seção de dados de 16 bits que o linker deverá efetuar dentro do bloco atual. Exceto pelo tamanho da relocação, este bloco é idêntico ao hunk_dreloc32.

hunk_dreloc8 (1017/3F9)

O bloco hunk_dreloc8 especifica relocação relativa a uma seção de dados de 8 bits que o linker deverá efetuar dentro do bloco atual. Exceto pelo tamanho da relocação, este bloco é idêntico ao hunk_dreloc32.

hunk_ext (1007/3EF)

Este bloco contém informações simbólicas externas. Nele estão contidas entradas que definem os símbolos e as listas que se referem aos mesmos. O formato deste bloco é o seguinte:

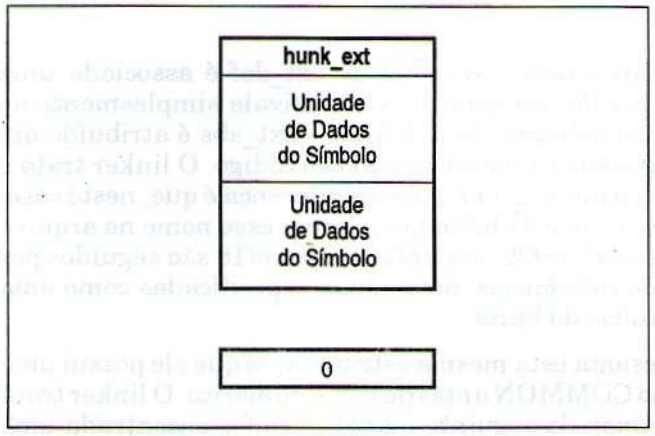


Figura 10-G hunk_ext (1007/3EF).

onde existe uma Unidade de Dados do Símbolo para cada símbolo utilizado. O bloco termina com uma palavra igual a zero.

Cada unidade de dados do símbolo consiste num byte que define o tipo, o tamanho do nome do símbolo (3 bytes), o nome do símbolo propriamente dito, e mais dados adicionais. Você deve especificar o tamanho do nome do símbolo em palavras longas, e preencher o campo do nome com zeros até atingir o próximo limite de palavra longa.

O byte do tipo especifica se o símbolo é uma definição ou referência, e assim por diante. O AmigaDOS utiliza os valores entre 0 e 127 para as definições de símbolos, e os de 128 a 255 para as referências.

Até o momento, os valores dos símbolos externos são os seguintes:

Nome	Valor	Significado
ext_simb	0	Tabela de símbolos–veja bloco de símbolos abaixo.
ext_def	1	Definição relocável.
ext_abs	2	Definição absoluta.
ext_res	3	Definição de biblioteca residente.
ext_ref32	129	Referência de 32 bits a um símbolo.
ext_common	130	Referência de 32 bits a COMMON.
ext_ref16	131	Referência de 16 bits a um símbolo.
ext_ref8	132	Referência de 8 bits a um símbolo.
ext_dref32	133	Referência de 32 bits (relativa à base) a um símbolo.
ext_dref16	134	Referência de 16 bits (relativa à base) a um símbolo.
ext_dref8	135	Referência de 8 bits (relativa à base) a um símbolo.

Tabela 10-A Símbolos Externos

O linker não reconhece os demais valores. À definição ext_def é associada uma palavra de dados que expressa o valor do símbolo, ela equivale simplesmente ao offset do símbolo a contar do início do hunk. Já à definição ext_abs é atribuído um valor que corresponde ao valor absoluto a ser adicionado ao código. O linker trata o valor de ext_res da mesma maneira que ext_def. A única diferença é que, neste caso, ele assume o nome do hunk como o nome da biblioteca e copia esse nome no arquivo de carregamento. Os bytes de tipo ext_ref32, ext_ref16 e ext_ref18 são seguidos por um valor numérico e uma lista de referências, novamente especificadas como uma contagem de offsets a partir do início do hunk.

O tipo ext_common também apresenta esta mesma estrutura, só que ele possui uma especificação de bloco de tamanho COMMON antes do valor numérico. O linker trata símbolos especificados como *common* da seguinte maneira: se for encontrada uma definição para um símbolo referenciado como *common* (uma definição deste tipo deverá ocorrer somente com o uso do Block Data do FORTRAN), o linker utilizará essa especificação. Caso nenhuma definição seja encontrada, ele alocará um espaço bss apropriado utilizando o maior tamanho que você indicou para cada referência simbólica *common*.

O linker manipula referências externas de diferentes maneiras, dependendo do tipo de definição correspondente. Normalmente ele adiciona valores absolutos à palavra longa ou campo de byte, e acusa um erro se o valor com sinal não for do tamanho apropriado. Nas referências relocáveis de 32 bits, o valor do símbolo é adicionado ao campo e um registro de relocação é produzido para o loader. Referências de 16 e 8 bits são manipuladas como referências relativas ao PC, e podem ser feitas somente para hunks com os mesmos nomes, a fim de que os hunks possam ser aglutinados pelo linker antes de serem carregados. As referências relativas ao PC podem resultar em erro se a referência e a definição estiverem muito distantes uma da outra. O linker só é capaz de acessar definições de biblioteca residente com referências de 32 bits, as quais ele passa então a manipular como referências relocáveis de 32 bits. Os formatos das unidades de dados dos símbolos são mostrados a seguir.

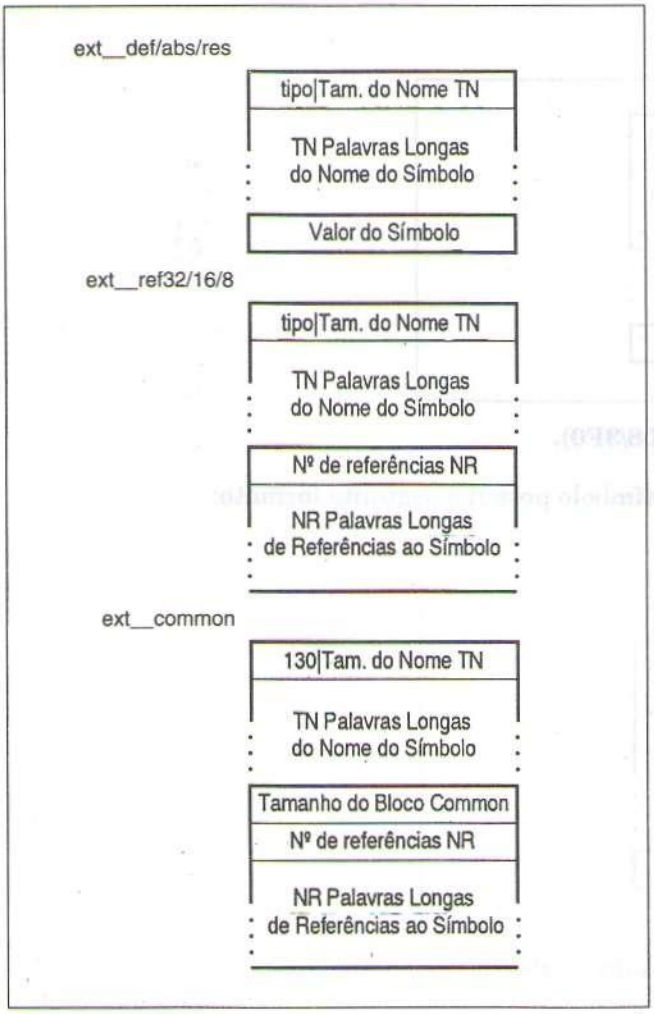


Figura 10-H Unidades de dados dos símbolos.

hunk_symbol (1008/3F0)

Você pode utilizar este bloco para anexar uma tabela de símbolos a um hunk, de maneira a possibilitar a utilização posterior de um depurador de símbolos com o código. O linker passa pelos blocos da tabela de símbolos anexada ao hunk, e se os hunks estiverem aglutinados, ele aglutina as tabelas de símbolos. O loader não carrega blocos da tabela de símbolos na memória, quando isto for necessário, espera-se que o programa depurador se encarregue de ler o arquivo de carregamento. O formato do bloco da tabela de símbolos é o mesmo do bloco de informação simbólica externa, com uma unidade de tabela simbólica para cada nome utilizado. Um tipo cujo código é igual a zero é usado nas unidades de dados simbólicos. O valor do símbolo corresponde ao offset existente entre o símbolo e o início do hunk. O formato deste bloco é o seguinte:

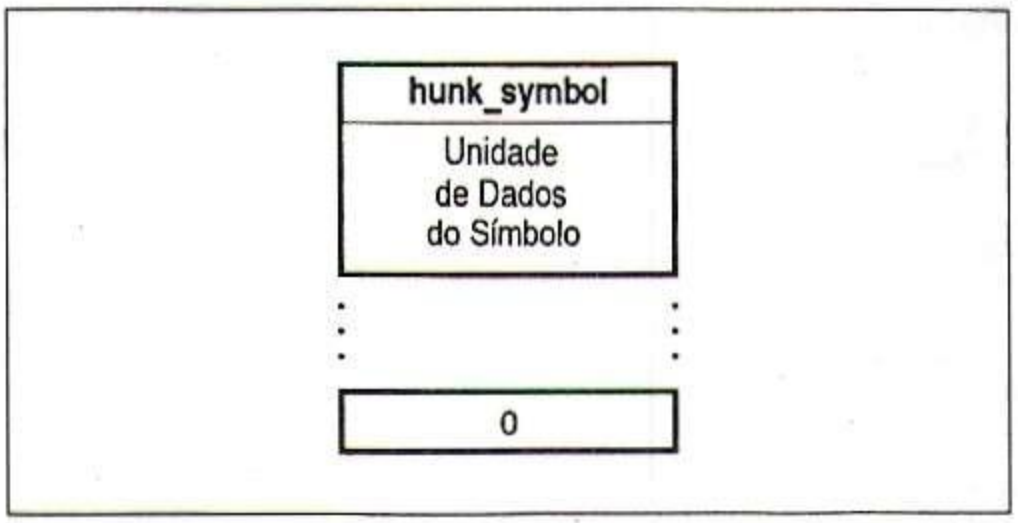


Figura 10-I hunk_symbol (1008/3F0).

onde cada unidade de dados do símbolo possui o seguinte formato:

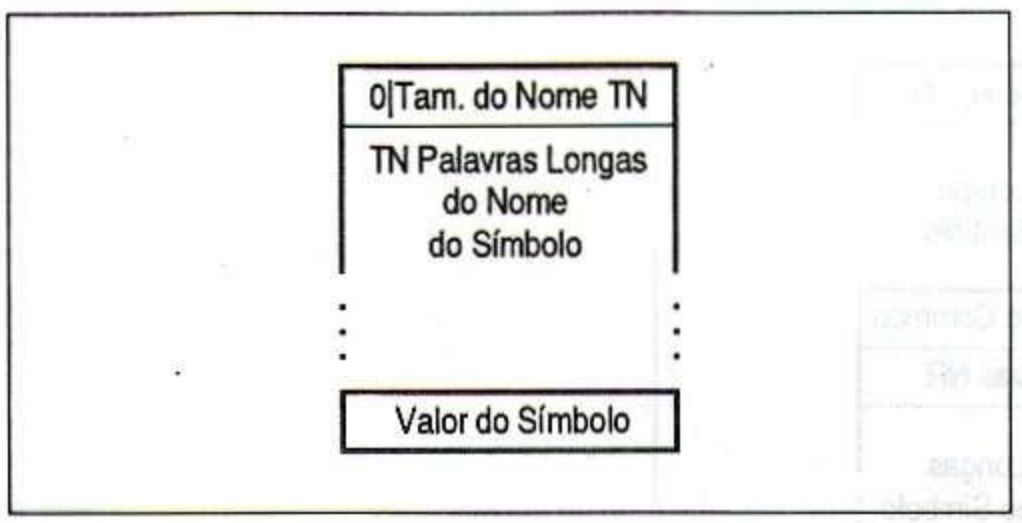


Figura 10-J Unidade de dados do símbolo

hunk_debug (1009/3F1)

O bloco de debug é um recurso que o AmigaDOS oferece para que um arquivo objeto possa suprir informações de depuração posteriores. Compiladores de linguagens de alto nível, por exemplo, podem precisar manter descrições de estruturas de dados para serem usadas com programas depuradores de alto nível. Nesse caso, o bloco de debug poderá armazenar essas informações. O AmigaDOS não impõe um formato específico para o bloco de debug. Ele deverá apenas iniciar com a palavra longa hunk_debug e ser seguido por uma palavra longa que indica o tamanho do bloco, em palavras longas. O formato deste bloco é o seguinte:

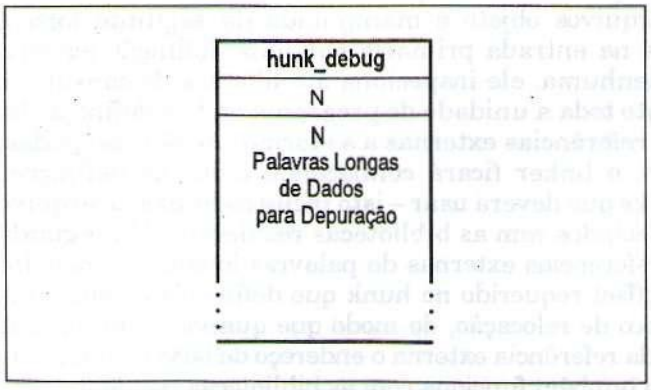


Figura 10-K hunk_debug (1009/3FH).

hunk_end (1010/3F2)

Este bloco especifica o final de um hunk. Ele consiste simplesmente em uma palavra longa, hunk_end.

Arquivos de Carregamento

O formato de um arquivo de carregamento (a saída do linker) é similar ao de um arquivo objeto. Os arquivos de carregamento consistem de um determinado número de hunks, similares àqueles existentes em um arquivo objeto. A principal diferença é que esses hunks nunca contêm um bloco de informação simbólica externa, uma vez que, no arquivo de carregamento, todos os símbolos externos foram resolvidos. Além disso, as informações referentes às unidades de programa não são incluídas em um arquivo de carregamento. Um arquivo de carregamento simples, que não será usado em overlay, possui um bloco de cabeçalho que indica o número total de hunks no arquivo e as bibliotecas residentes referenciadas pelo programa. Esse bloco é seguido pelos hunks, que podem resultar da aglutinação de vários hunks de entrada que possuíam o mesmo nome. A esta estrutura completa se dá o nome de nó. Os arquivos de carregamento também podem conter informações de overlay, neste caso, uma tabela de overlay sucederá o nó primário. Um bloco especial de interrupção (*break block*) separa os vários nós de overlay. A estrutura de um arquivo de carregamento pode ser resumida, portanto, da seguinte maneira (os itens marcados com um asterisco são opcionais):

- Nó primário.
- Bloco de tabela de overlay (*).
- Nós de overlay separados por blocos de interrupção (*).

Lembre-se de que os blocos de relocação nos hunks são sempre do tipo hunk_reloc32. Eles indicam que relocação deverá ser executada na hora do carregamento. Isto abrange tanto a relocação de 32 bits especificada com blocos hunk_reloc32 no arquivo objeto, como a relocação extra requerida para a resolução de símbolos externos.

Cada referência externa nos arquivos objeto é manipulada da seguinte forma: primeiramente o linker procura na entrada primária por uma definição externa coincidente. Se não encontrar nenhuma, ele inspeciona a biblioteca de consulta e inclui no arquivo de carregamento toda a unidade de programa onde a definição foi efetuada. Isto pode levar outras referências externas a se tornarem não resolvidas. Ao final da primeira passagem, o linker ficará conhecendo todas as definições externas e o número total de hunks que deverá usar – isto inclui os hunks no arquivo de carregamento e os hunks associados com as bibliotecas residentes. Na segunda passagem, o linker modifica as referências externas de palavras longas, de maneira que elas passem a se referir ao offset requerido no hunk que define o símbolo. Uma entrada extra é produzida no bloco de relocação, de modo que quando os hunks são carregados, o linker adiciona a cada referência externa o endereço de base do hunk que define o símbolo. Este mecanismo também funciona com as bibliotecas residentes.

Para poder produzir essas referências cruzadas através de hunks, o linker precisa antes conhecer o número e o tamanho de hunks existentes nos nós. Esta informação é fornecida pelo bloco de cabeçalho, que será descrito a seguir. O arquivo de carregamento também pode conter informações de overlay, essas informações são agrupadas num bloco de tabela de overlay. Blocos especiais de interrupção separam os vários nós de overlay.

hunk_header (1011/3F3)

Este bloco fornece informações sobre o tamanho e o número de hunks que deverão ser carregados. Ele também contém os nomes das bibliotecas residentes que precisarão ser abertas quando o nó for carregado.

O formato do hunk_header é ilustrado na figura 10-L. A primeira parte do bloco de cabeçalho contém os nomes das bibliotecas residentes que o loader precisará abrir quando o nó for carregado. Cada nome de biblioteca consiste de uma palavra longa indicando o tamanho do nome, em palavras longas, e o texto do nome, preenchido com zeros até um limite de palavra longa. A lista do nome é encerrada por uma palavra longa igual a zero. Os nomes aparecem na ordem em que o loader deverá abri-los.

Ao carregar um nó primário, o loader aloca uma tabela na memória que servirá para manter um controle de todos os hunks que ele carregar. Esta tabela precisa ter espaço suficiente para todos os hunks do arquivo de carregamento, incluindo os hunks de overlays. O loader também utiliza esta tabela para manter uma cópia das tabelas de hunks associadas com as bibliotecas residentes. A próxima palavra longa no bloco de cabeçalho é por conseguinte o tamanho dessa tabela, o qual equivale ao número total de hunks referenciados mais um.

A próxima palavra longa P se refere à primeira área contígua (ou página) na tabela de hunks que o loader deverá utilizar quando estiver efetuando o carregamento. Para um nó primário que não referencia uma biblioteca residente, este valor será zero, caso contrário, ele será igual ao número de hunks nas bibliotecas residentes. O loader copia essas entradas a partir da tabela de hunks associada com a biblioteca, na seqüência de uma chamada Open Library( ) do sistema operacional. Para um nó de overlay, este valor corresponde ao número de hunks nas bibliotecas residentes, mais o número de hunks já carregados nos nós ancestrais.

A próxima palavra longa U se refere à última página na tabela de hunks que o loader deverá utilizar como parte desta chamada. O número total de hunks carregados pode ser calculado portanto como U-P+1.

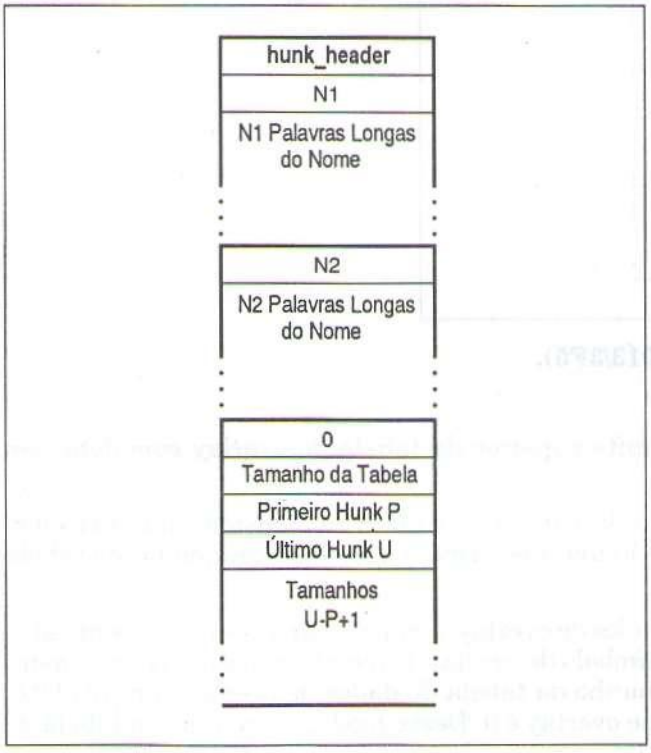


Figura 10-L hunk_header (1011/3F3).

O bloco do cabeçalho continua com U-P+1 palavras longas que indicam o tamanho de cada hunk que deverá ser carregado como parte dessa chamada. Isto habilita o loader a pré-alocar o espaço para os hunks e, a partir daí, efetuar a relocação entre os hunks que é requerida enquanto os mesmos são carregados.

hunk_overlay (1013/3F5)

O bloco de tabela de overlay indica para o loader que ele está carregando um programa que utiliza o recurso de overlay. Este bloco contém todos os dados para a tabela de overlay. Ao encontrá-lo, o loader define a tabela e retorna, deixando o canal de entrada para o arquivo de carregamento aberto. O formato deste bloco é o seguinte:

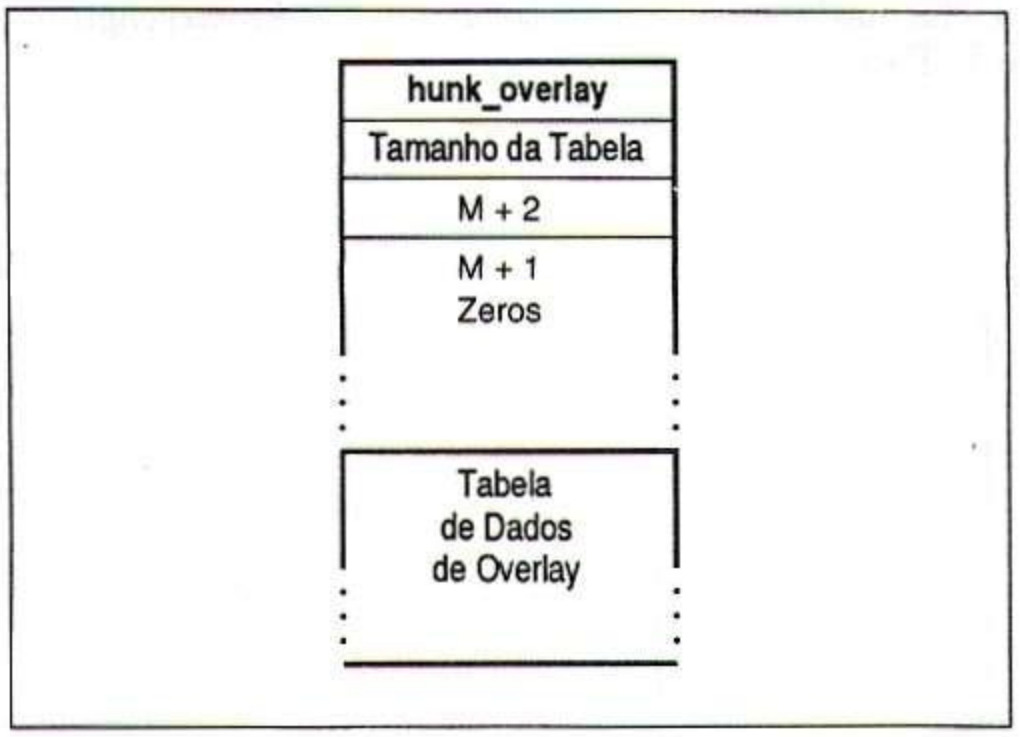


Figura 10-M hunk_overlay (1013/3F5).

A primeira palavra longa é o limite superior da tabela de overlay completa (em palavras longas).

M equivale ao nível máximo utilizado pela árvore de overlay, considerando-se o nível raiz igual a zero. As próximas M+1 palavras formam a seção de ordenadas da tabela de overlay.

O restante do bloco é a tabela de dados de overlay, formada por uma série de entradas de oito palavras, uma para cada símbolo de overlay. Supondo-se que o maior número de overlay usado seja zero, o tamanho da tabela de dados de overlay será (0+1)*8, uma vez que o primeiro número de overlay é 0. Desse modo, o tamanho da tabela de overlay será igual a (0+1)*8+M+1.

hunk_break (1014/3F6)

Um bloco de interrupção (*break block*) indica o final de um nó de overlay. Este bloco consiste de apenas uma palavra longa, hunk_break.

Exemplos

As rotinas simplificadas a seguir mostram como o loader e o linker manipulam símbolos externos. Considere o código a seguir:

```
                        IDNT    A
                        XREF    BILLY, JOHN
                        XDEF    MARY
* A próxima palavra longa requer relocação
0000'0000 0008          DC.L    FRED
0004'123C 00FF          MOVE.B  #$FF,D1
0008'7001       FRED    MOVEQ   #1,D0
* Entry point externo
```

```
000A'4E71       MARY    NOP
000C'4EB9 0000 0000     JSR         BILLY     Chamada externa
0012'2239 0000 0000     MOVE.L      JOHN,D1   Referência externa
                        END
```

Ele produzirá o seguinte arquivo objeto:

```
hunk_unit
00000001  Tamanho em palavras longas
41000000  Nome, preenchido para palavra longa
hunk_code
00000006  Tamanho em palavras longas
00000008  123C00FF  70014E71  4EB90000  00002239  00000000
hunk_reloc32
00000001  Número no hunk 0
00000000  Hunk 0
00000000  Offset a ser relocado
00000000  Zero para marcar o final
hunk_ext
01000001  XDEF, tamanho de 1 palavra longa
4D415259  MARY
0000000A  Offset da definição
81000001  XREF, tamanho de 1 palavra longa
4A4F484E  JOHN
00000001  Número de referências
00000014  Offset da referência
81000002  XREF, tamanho de 2 palavras longas
42494C4C  BILLY
59000000  (zeros para preencher)
00000001  Número de referências
0000000E  Offset da referência
00000000  Final de bloco externo
hunk_end
```

O programa correspondente ao primeiro é o seguinte:

```
                        IDNT        B
                        XDEF        BILLY, JOHN
                        XREF        MARY
0000'2A3C AAAA AAAA     MOVE.L      #$AAAAAA,D5
* Entry point externo
0006'4E71       BILLY   NOP
* Entry point externo
0008'7201       JOHN    MOVEQ       #1,D1
* Chamada a referência externa
000A'4EF9 0000 0000     JMP         MARY
                       END
```

O que produzirá o seguinte código de saída:

```
hunk_unit
00000001  Tamanho em palavras longas
42000000  Nome da unidade
hunk_code
00000004  Tamanho em palavras longas
2A3CAAAA  AAAA4E71  72014EF9  00000000
hunk_ext
01000001  XDEF, tamanho de 1 palavra longa
4A4F484E  JOHN
00000008  Offset da definição
01000002  XDEF, tamanho de 2 palavras longas
42494C4C  BILLY
59000000  (zeros para preencher)
00000006  Offset da definição
```

```
81000001  XREF, tamanho de 1 palavra longa
4D415259  MARY
00000001  Número de referências
0000000C  Offset da referência
00000000  Final de bloco externo
hunk_end
```

Depois que você passar este arquivo pelo linker, o arquivo de carregamento terá o seguinte formato:

```
hunk_header
00000000  Nenhum nome de hunk
00000002  Tamanho da tabela de hunk
00000000  Primeiro hunk
00000001  Último hunk
00000006  Tamanho do hunk 0
00000004  Tamanho do hunk 1
hunk_code
00000006  Tamanho do código em palavras longas
00000008  123C00FF  70014E71  4EB90000  00062239  00000008
hunk_reloc32
00000001  Número no hunk 0
00000000  Hunk 0
00000000  Offset a ser relocado
00000002  Número no hunk 1
00000001  Hunk 1
00000014  Offset a ser relocado
0000000E  Offset a ser relocado
00000000  Zero para marcar o final
hunk_end
hunk_code
00000004  Tamanho do código em palavras longas
2A3CAAAA  AAAA4E71  72014EF9  0000000A
hunk_reloc32
00000001  Número no hunk 0
00000000  Hunk 0
0000000C  Offset a ser relocado
00000000  Zero para marcar o final
hunk_end
```

Ao carregar este código na memória, o loader lê o bloco do cabeçalho e aloca um tabela de hunks de duas palavras longas. Em seguida ele aloca espaço chamando uma rotina do sistema operacional e requisitando duas áreas com o tamanho de 6 e 4 palavras longas, respectivamente. Assumindo que as duas áreas retornadas por esta rotina estejam nos endereços 3000 e 7000, a tabela de hunks irá conter 3000 e 7000.

O loader lê o primeiro hunk e posiciona o primeiro código no endereço 3000, passando então a proceder à relocação. Como o primeiro item especifica relocação com respeito ao hunk 0, o loader adiciona 3000 à palavra longa no offset 0, convertendo o valor ali armazenado de 00000008 para 00003008. O segundo item especifica relocação com respeito ao hunk 1. Embora não seja carregado, nós sabemos que este hunk deverá ser carregado no endereço 7000, assim, esta quantidade é adicionada aos valores armazenados em 300E e 3014. Note que o linker já inseriu os offsets 00000006 e 00000008 nas referências no hunk 0, de maneira que elas se referem ao offset correto no hunk 1 para a definição. Dessa maneira, as palavras longas especificando as referências externas terminam contendo os valores 00007006 e 00007008, o que corresponde ao local correto assim que o segundo hunk for carregado.

De modo semelhante, o loader carrega o segundo hunk na memória na locação 7000,

e a informação de relocação especificada altera a palavra longa em 700C de 0000000A (o offset de MARY no primeiro hunk) para 0000300A (o endereço de MARY na memória).

O loder trata as referências a bibliotecas residentes da mesma maneira. A única diferença é que, após abrir a biblioteca, as locações dos hunks da biblioteca são copiadas no início da tabela de hunks. Em seguida o loader modifica as referências à biblioteca residente para que passem a se referir ao local correto, adicionando a tais referências a base dos hunks da biblioteca.

Estrutura dos Arquivos de Bibliotecas do Amiga

Existem dois tipos de estruturas de arquivos de biblioteca no Amiga: o formato original usado pelos linkers ALINK e Blink, e o novo formato indexado usado com as versões 7.2 e posteriores do Blink.

A estrutura original dos arquivos de biblioteca do Amiga consiste essencialmente em um ou mais módulos de objeto concatenados em um único arquivo. Esta estrutura tem a vantagem da simplicidade. Mais módulos de objetos podem ser acrescentados à biblioteca bastando anexá-los ao final do arquivo da biblioteca.

Neste formato a passagem inicial executada pelo linker precisa processar o arquivo da biblioteca seqüencialmente, a fim de encontrar as unidades de programa necessárias para se efetuar a ligação.

Exemplo de um Arquivo de Biblioteca

Desse modo, uma biblioteca típica teria a seguinte aparência:

```
HUNK_UNIT,              2,  "First PU"
HUNK_NAME,              3,  "First Hunk"
HUNK_CODE,             20,  20 palavras longas de código...
HUNK_RELOC32,           3,  3, 12, 22, 48
                        2,  2, 4, 34
                        0
HUNK_EXT,     EXT_DEF|2,  "FirstDef"
              EXT_DEF|3,  "SecondDef"
           EXT_REF32|2,  "ThirdDef", 2, 12, 48
           EXT_REF32|3,  "FourthDef", 1, 4
                        0
HUNK_DEBUG,             7, 7 palavras longas de informações de depuração...
HUNK_END
HUNK_NAME,              3,  "Second Hunk"
HUNK_DATA,             30,  30 palavras longas de dados...
HUNK_EXT,     EXT_DEF|3,  "FirstConst", 0
              EXT_DEF|3,  "FourthDef", 4
              EXT_DEF|3,  "LongString", 8
                        0
HUNK_END
HUNK_BSS,              40
HUNK_EXT,     EXT_DEF|2,  "workStr", 0
                        0
HUNK_END
```

```
HUNK_UNIT,              3,   "Second PU"
HUNK_NAME,              3,   "Third Hunk"
HUNK_CODE,             64,   64 palavras longas de código...
HUNK_RELOC32,           2,   0, 14, 54
                        4,   1, 4, 22, 28, 44
                        3,   2, 10, 38, 100
                        0
HUNK_EXT,    EXT_REF32|2,   "FirstDef", 2, 14, 54
           EXT_REF32|3,   "LongString", 3, 22, 28, 44
             EXT_DEF|2,   "ThirdDef", 0
                        0
HUNK_END
```

Estrutura da Nova Biblioteca

O novo formato de arquivos de biblioteca é bastante similar ao antigo. A principal diferença é a existência de um nível extra de encapsulamento, proporcionada pelo uso de dois novos tipos de hunks. Os usuários podem continuar intercalando bibliotecas através da concatenação de arquivos, e bibliotecas com o formato novo e antigo podem ser anexadas conjuntamente.

O novo formato é mais compacto e seu processamento pelo linker é mais rápido. Ele alcança esse desempenho e flexibilidade graças à adição de dois tipos extras de hunks, conhecidos como hunk_lib e hunk_index. Como todos os tipos de hunks do AmigaDOS, esses hunks consistem em um valor de palavra longa que define o tipo, o qual é seguido por um valor de 32 bits que indica o número de palavras longas subseqüentes no hunk. Além disso, eles sempre ocorrem em pares, hunk_lib primeiro, e hunk_index depois. Nada fica entre eles.

hunk_lib (1019/3FB)

O formato do hunk_lib é mostrado na figura 10-N.

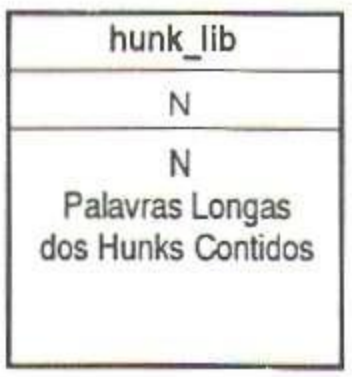


Figura 10-N hunk_lib (1019/3FB).

O campo que descreve o tamanho (N) da estrutura do hunk_lib precisa ser uma soma de TODAS as palavras longas pertencentes à estrutura, excluindo-se o tipo do hunk e o campo do tamanho. A soma das palavras longas indicada no campo do tamanho pode ser maior do que 65.535. Entretanto, é importante observar que o o offset, em palavras longas, para o último hunk code, data ou bss, não pode ser maior que 65.535 (veja hunk_index, a seguir). Se os hunks contidos (ou seus hunks constituintes) se

estenderem além deste ponto, a soma no campo do tamanho de hunk_lib deverá incluí-los da mesma maneira.

hunk_index (1020/3FC)

O hunk_index fornece um índice para todos os hunks concatenados no hunk_lib. O formato do hunk_index é descrito na figura 10-O.

Exemplo de hunk_lib

Eis um exemplo de uma biblioteca com o novo formato, baseada no exemplo descrito anteriormente no tópico "Exemplo de um Arquivo de Biblioteca". Esta biblioteca é formada dispondo-se em pares um hunk_lib e um hunk_index. Este é o hunk_lib:

```
HUNK_LIB,              191
HUNK_CODE,              20,  20 palavras longas de código...
HUNK_RELOC32,            3,  3, 12, 22, 48
                         2,  2, 4, 34
                         0
HUNK_EXT,    EXT_REF32|2,  "ThirdDef", 2, 12, 48
             EXT_REF32|3,  "FourthDef", 1, 4
                         0
HUNK_DEBUG,              7, 7 palavras longas de informações de depuração...
HUNK_END
HUNK_DATA,              30,  30 palavras longas de dados...
HUNK_END
HUNK_BSS,               40
HUNK_END
HUNK_CODE,              64,  64 palavras longas de código...
HUNK_RELOC32,            2,  0, 14, 54
                         4,  1, 4, 22, 28, 44
                         3,  2, 10, 38, 100
                         0
HUNK_EXT,    EXT_REF32|2,  "FirstDef", 2, 14, 54
             EXT_REF32|3,  "LongString", 3, 22, 28, 44
                         0
HUNK_END
```

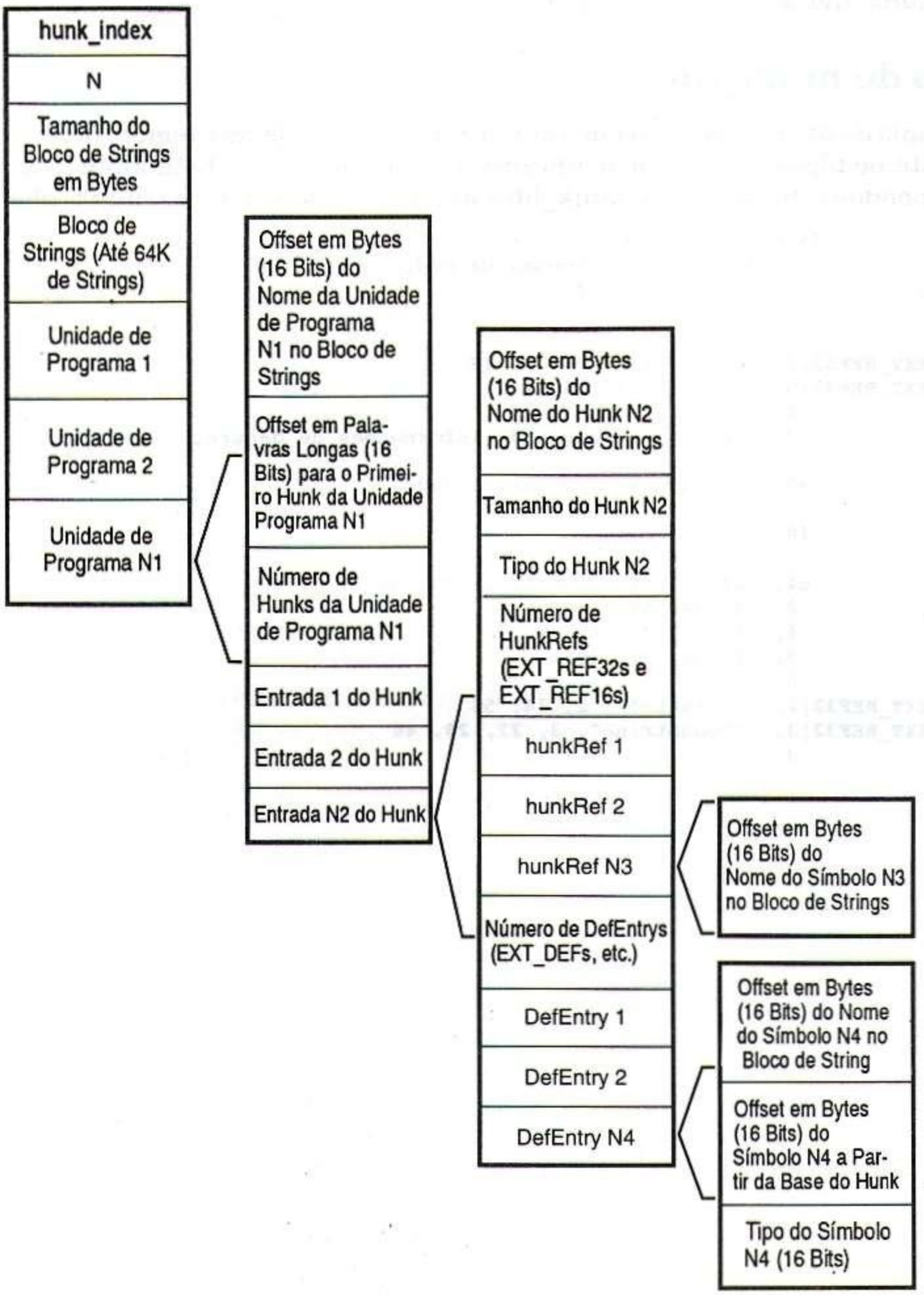


Figura 10-O formato do hunk_index.

Exemplo de hunk_index

O hunk_index para a biblioteca é mais complicado. Ele obedece ao seguinte formato genérico:

- hunk_index.
- Tamanho.
- Bloco de strings alinhado por palavra de 16 bits.
- Uma ou mais estruturas PUNIT.

Onde o bloco de strings consiste em um valor de palavra de 16 bits representando o tamanho do restante do bloco, em bytes, e o resto do bloco consiste de strings em estilo C, terminadas por NULL. O primeiro string precisa ser um string nulo. Os strings NÃO são alinhados para o limite de uma palavra. O bloco, se necessário, deverá ser preenchido no final com um único byte 0, a fim de alinhá-lo para o limite de uma palavra. Desta maneira, um bloco de strings para o exemplo acima teria a seguinte aparência:

```
122
""              no offset     0
"First PU"      no offset     1
"First Hunk"    no offset    10
"FirstDef"      no offset    21
"SecondDef"     no offset    30
"ThirdDef"      no offset    40
"FourthDef"     no offset    49
"Second Hunk"   no offset    59
"FirstConst"    no offset    71
"LongString"    no offset    82
"workStr"       no offset    93
"Second PU"     no offset   101
"Third Hunk"    no offset   111
```

Este bloco não precisou do acréscimo de nenhum byte 0 no final a fim de proporcionar o alinhamento para um limite de palavra de 16 bits. Note que este bloco, excluindo-se o campo que descreve o tamanho, não pode ser maior do que 65.534 bytes (64K – 2 bytes). O byte de preenchimento no final, quando presente, deverá ser incluído no campo de tamanho do bloco.

Sucedendo o bloco de strings no hunk_index, aparecem uma ou mais estruturas PUNIT com o seguinte formato:

- Um cabeçalho PUNIT.
- Uma ou mais entradas de hunks.
- Se necessário, um valor 0 para preenchimento a fim de realinhar o hunk para um limite de palavra longa.

Onde o cabeçalho PUNIT consiste de:

1. Um offset de 16 bits para o string que descreve o nome da unidade de programa no bloco de strings precedente (0 é o offset para a primeira string; -2 é o offset para o tamanho do bloco). Este offset é expresso em bytes e com sinal. Assim, o espaço total de strings disponível para os nomes de símbolos de um hunk_lib corresponde a 65.534 bytes.

2. Um offset de 16 bits para o primeiro hunk (code, data ou bss) de uma unidade de programa dentro da estrutura hunk_lib precedente. Este offset é expresso em

palavras longas, significando que nenhum hunk na hunk_lib correspondente pode começar além de um offset de 262.140 bytes.

3. Um somatório de 6 bits do número de hunks na estrutura hunk_lib precedente (code, data e bss).

E uma entrada de hunk consiste em:

1. Um offset de 16 bits para o string que descreve o nome do hunk no bloco de strings (ou então 0 – o string nulo).
2. Um valor de 16 bits expressando o tamanho do hunk, em palavras longas.
3. Um valor de 16 bits expressando o tipo do hunk (hunk_code, hunk_data, hunk_bss), com a definição de Chip ou Fast memory sinalizada pelos flags nos 2 bits superiores da palavra correspondente ao tipo.
4. Uma soma de 16 bits do número de referências. Esta informação é duplicada a partir das referências EXT_REF do hunk_ext que estiver associado com o hunk. Este campo é seguido pelos offsets de 16 bits para os strings dos símbolos sendo referenciados no bloco de strings (o string em si é uma referência de 32 bits).
5. Uma soma de 16 bits do número de definições. Esta informação é retirada dos hunk_exts do hunk (o que explica por que eles são menores no exemplo do hunk_lib acima). Este campo é seguido por 0 ou mais entradas de três palavras:
   a) Um offset de 16 bits para o string que contém o nome do símbolo definido no bloco de strings (bits mais significativos sempre ressetados).
   b) Um offset de 16 bits (em bytes) para o símbolo a partir da base do hunk.
   c) Um tipo de 16 bits correspondente à definição do símbolo. Observe que o tipo foi estendido. Em alguns tipos de valores EXT_ABS, mais notadamente as referências _CIA na biblioteca amiga.lib, o valor ABS tem bits significativos que consomem até 25 bits para serem armazenados.

Como o valor do campo que descreve o tipo cabe confortavelmente em 1 byte, o byte superior é reservado para os bits 16-23 dos valores EXT_ABS. Já o bit 6 do byte do tipo é usado para anotar o estado dos 8 bits superiores do valor original de 32 bits de EXT_ABS (isto é, todos 1's ou todos 0's). Isto permite que 25 bits de informações EXT_ABS sejam armazenados nas estruturas existentes. Assim, para dados EXT_ABS, o formato é o seguinte:

```
valores EXT_ABS originais:     abs1 = $c709d3
                               abs2 = -14872941 ($ff1d0e93)
valores EXT_ABS resultantes:   abs1 = $09d3       (palavra)
                                      $c7            (byte)
                                      EXT_ABS        (byte)
                               abs2 = $0e93       (palavra)
                                      $1d            (byte)
                                      EXT_ABS | 64   (byte)
```

Note que em todas as estruturas hunk_index, quando um valor de 16 bits que expressa a contagem dos elementos de um determinado tipo num array for igual a zero, NENHUM elemento desse tipo estará presente na estrutura.

Assim, o hunk_index para o hunk_lib mostrado anteriormente é:

```
hunk_index,  57
122
""              no offset      0
```

```
"First PU"                      1
"First Hunk"                   10
"FirstDef"                     21
"SecondDef"                    30
"ThirdDef"                     40
"FourthDef"                    49
"Second Hunk"                  59
"FirstConst"                   71
"LongString"                   82
"workStr"                      93
"Second PU"                   101
"Third Hunk"                  111
1, 0, 3         unidade de programa c/ 3 hunks...
  10, 20, HUNK_CODE     informação do hunk
      2, 40, 49               2 refs...
      2, 21, 0, EXT_DEF       2 defs...
      30, 38, EXT_DEF
  59, 30, HUNK_DATA     informação do hunk
      0                     nenhuma ref
      3, 71, 0, EXT_DEF       3 defs...
         49, 4, EXT_DEF
         82,4,  EXT_DEF
   0, 40, HUNK_BSS      informação do hunk
      0                     nenhuma ref
      1, 93, 0, EXT_DEF       1 def...
 101, 92, 1      unidade de programa c/ 1 hunk...
 111, 64, HUNK_CODE     informação do hunk
   2, 21, 82                 2 refs...
   1, 40, 0, EXT_DEF         1 def...
      0          preench. 16 bits p/ alinhar hunk p/ palavra longa
```

Observe nos exemplos que a estrutura hunk_lib ainda continha os hunk_ends, os hunk_reloc32, o hunk_debug e parte de alguns dos hunk_exts. Se um hunk_symbol estivesse presente, ele também precisaria estar no hunk_lib com seus correspondentes hunks code, data ou bss.

Estes hunks – hunk_code, hunk_data, hunk_bss, hunk_reloc32, hunk_reloc16, hunk_reloc8, hunk_symbol, hunk_debug e hunk_end – precisam estar presentes exatamente como se não estivessem em um hunk_lib. Já o hunk_unit e o hunk_name precisam ser removidos, sendo substituídos por entradas de unidades de programa no hunk_index associado com o hunk_lib. Do hunk_ext precisam ser removidas quaisquer EXT_DEFs. Esse tipo de informação deverá constar nas entradas de hunk do hunk_index. EXT_REF32s e/ou EXT_REF16s precisam estar presentes em um hunk_ext para que este possa permanecer no hunk_lib. As EXT_REF32s e EXT_REF16s precisam também ser anotadas na lista de referências na entrada de hunk encontrada no hunk_index para o hunk. Já as referências EXT_REF8s não são suportadas.

Capítulo 11

Estrutura de Dados do AmigaDOS

Este capítulo descreve as estruturas de dados relativas à memória e aos arquivos do AmigaDOS. A organização dos discos é discutida no Capítulo 9.

Introdução

O AmigaDOS dispõe de um mecanismo de entrada e saída independente do dispositivo físico. Este recurso é implementado através da criação de um processo de manipulação (ou *handler*) para cada dispositivo utilizado. O processo de manipulação aceita um conjunto padrão de requisições de I/O e as converte nas requisições específicas para o dispositivo solicitado. Todos os clientes do AmigaDOS se referem ao processo de manipulação ao invés de se referirem diretamente ao dispositivo, embora seja possível acessar um dispositivo sem utilizar o seu handler, se isto for necessário. Este capítulo descreve a estrutura de dados do AmigaDOS, incluindo o formato de um processo, estruturas de dados centrais compartilhadas, e a estrutura das requisições dos handlers.

Além dos valores usualmente empregados no Amiga como LONGs e APTRs, o AmigaDOS também usa BPTRs. Um BPTR nada mais é do que um ponteiro BCPL, ou seja, um ponteiro para um bloco de memória alinhado por palavra longa e dividido por 4. Assim, para ler um BPTR em C, basta executar um shift left do mesmo duas vezes. Para criar um BPTR, você deve usar a memória obtida através de uma chamada para a função AllocMem( ) do AmigaDOS. Também é lícito utilizar a memória de stack, mas você deverá se certificar de que o endereço inicial está num limite de palavra longa.

O AmigaDOS também usa strings BCPL, conhecidos como BSTRs. Um BSTR consiste em um BPTR para uma posição da memória que contém o tamanho do string, no primeiro byte, e os bytes que formam o string logo em seguida.

Algumas referências ao Vetor Global (*Global Vector*) aparecem neste capítulo. O Vetor Global é uma tabela de jumps usada pelo BCPL, e que pode ser compreendida

como um ponteiro para um Vetor Global padrão compartilhado. Alguns processos, como o que manipula os arquivos, usam um vetor global privativo. Em ambos os casos, você jamais deverá acessar o Vetor Global a partir de seu programa, uma vez que ele se destina ao uso exclusivo do AmigaDOS. Nas versões 2.0 e posteriores do AmigaDOS, o Vetor Global é usado somente para manter a compatibilidade com os antigos programas em BCPL. O novo AmigaDOS foi escrito em C e Assembler e não utiliza o Vetor Global.

Estruturas de Dados dos Processos

Os valores discutidos a seguir são criados como parte de um processo do AmigaDOS. Existe um conjunto completo de valores para cada processo.

Um processo pode ser descrito como uma tarefa Exec ao qual foi acrescentado um determinado número de estruturas de dados adicionais. A estrutura do processo consiste de:

- Estrutura da tarefa Exec.
- Porta de mensagens Exec.
- Valor do processo para o AmigaDOS.

O identificador do processo que o AmigaDOS utiliza internamente é um ponteiro para a porta de mensagens Exec (pr_MessagePort) a partir da qual a tarefa Exec pode ser obtida.

Os valores dos processos do AmigaDOS são os seguintes:

Valor	Nome	Descrição
BPTR	pr_SegList	Array das listas de segmentos utilizados pelo processo.
LONG	pr_StackSize	Tamanho do stack do processo, em bytes.
APTR	pr_GlobVec	Vetor global para o processo (BCPL).
LONG	pr_TaskNum	Número da tarefa do CLI.
BPTR	pr_StackBase	Ponteiro para a posição final do stack do processo na memória.
LONG	pr_Result2	Valor do resultado secundário da última chamada.
BPTR	pr_CurrentDir	Lock associado com o diretório corrente.
BPTR	pr_CIS	Stream de entrada do CLI atual.
BPTR	pr_COS	Stream de saída do CLI atual.
APTR	pr_ConsoleTask	Processo de manipulação do console para a janela atual.
APTR	pr_FileSystemTask	Processo de manipulação de arquivos para o drive atual.
BPTR	pr_CLI	Ponteiro para o CLI.
APTR	pr_ReturnAdd	Ponteiro para o elemento anterior do stack.
APTR	pr_PktWait	Função a ser chamada enquanto é aguardada uma mensagem.
APTR	pr_WindowPtr	Janela para a impressão de mensagens de erro.

As definições a seguir foram acrescentadas à estrutura de dados dos processos nas versões V2.0 e posteriores do AmigaDOS:

```
Valor       Nome            Descrição
BPTR        pr_HomeDir      Diretório residente do programa em execução.
LONG        pr_Flags        Flags contendo informações sobre o processo
                            para o DOS.
LONG        (*pr_ExitCode)  Rotina a chamar na saída do programa,
            (LONG returncode,ou NULL.
            LONG pr_ExitData)
LONG        pr_ExitData     Passado como um argumento para pr_ExitCode.
UBYTE *     pr_Arguments    Argumentos passados para o processo no
                            início do mesmo.
struct MinListpr_LocalVars  Variáveis de ambiente locais.
ULONG       pr_ShellPrivate Para o uso do Shell atual.
BPTR        pr_CES          Stream de erro, se NULL, o stream pr_COS é usado.
```

Para identificar os segmentos usados por um determinado processo, você deverá utilizar o array pr_SegList. O tamanho deste array de palavras longas é apontado por pr_SegList[0], e seus outros elementos serão zero ou ponteiros BPTR para uma lista de segmentos. As funções CreateProc( ) e CreateNewProc( ) do AmigaDOS criam este array, com os dois primeiros elementos do array apontando para o código residente, e o terceiro elemento (a SegList) sendo passado como um argumento. Ao terminar o processo, a função FreeMem( ) deverá ser chamada a fim de retornar o espaço de memória alocado para pr_SegList. Veja o Capítulo 9 para maiores informações sobre estas funções.

O campo pr_StackSize indica o tamanho do stack do processo, tal como indicado pelo usuário nos argumentos das chamadas CreateProc( ) e CreateNewProc( ). Não confunda o stack do processo com o stack de comandos que o CLI utiliza ao chamar um programa. O CLI obtém o stack de comandos antes de executar um programa, e você pode alterar o tamanho desse stack com o comando STACK do AmigaDOS. Quando você cria um processo, o AmigaDOS obtém o stack do processo e armazena o tamanho do mesmo em pr_StackSize. O ponteiro para o espaço destinado a acomodar o stack e o bloco de controle do processo é também armazenado no campo MemEntry da estrutura da tarefa. Quando o processo termina, esse espaço é retornado através de uma chamada à rotina FreeVec( ). Você pode encadear toda a memória que obtiver dentro desta estrutura, de maneira que ela também seja retornada quando a terefa terminar. Mas tenha cuidado: lembre-se de que este método não funciona com programas executados a partir do CLI, pois neste caso a memória não é liberada até que o processo seja encerrado.

Admitindo-se que o processo tenha sido criado por uma chamada a CreateProc( ) ou CreateNewProc( ), GlobVec será um ponteiro para o Vetor Global compartilhado. Alguns processos de manipulação internos do AmigaDOS, contudo, usam um vetor global particular.

O valor de pr_TaskNum é normalmente igual a zero: um processo CLI armazena o menor inteiro que identifica a invocação do CLI neste campo.

O ponteiro pr_StackBase aponta para a posição final do stack do processo na memória. Note que este valor corresponde ao final do stack quando se usam linguagens como o C ou Assembler, porém equivale à base do stack para linguagens como a BCPL. Observe ainda que o valor de pr_StackBase eventualmente pode não

ser o mesmo que a sua aplicação utiliza (por exemplo, se o seu programa for iniciado a partir do CLI).

Os campos pr_Result2 e pr_CurrentDir são manipulados pelas funções IoErr( ) e CurrentDir( ) do AmigaDOS, respectivamente. Os campos pr_CIS e pr_COS são os valores que Input( ) e Output( ) retornam, e se referem aos filehandles que você deverá utilizar ao rodar um programa sob o CLI. Nunca acesse pr_CIS e pr_COS diretamente, use sempre as funções do AmigaDOS disponíveis para este fim.

O campo pr_ConsoleTask indica o processo de manipulação do console (*console handler*) para a janela atual. O campo pr_FileSystemTask se refere ao processo de manipulação de arquivos (*file handler*) para o dispositivo de boot. Esses valores são utilizados quando se tenta abrir o dispositivo *, ou então abrir um arquivo por meio de uma descrição de path relativo quando o campo pr_CurrentDir for nulo.

O valor do ponteiro pr_CLI é não zero apenas para processos CLI. Neste caso ele se refere a uma estrutura adicional utilizada pelo CLI, e que possui o seguinte formato:

Valor	Nome	Descrição
LONG	cli_Result2	Valor de IoErr( ) após o último comando.
BSTR	cli_SetName	Nome do diretório corrente.
BPTR	cli_CommandDir	BPTR para um path do CLI.
LONG	cli_ReturnCode	Código de retorno do último comando.
BSTR	cli_CommandName	Nome do comando atual.
LONG	cli_FailLevel	Limite do erro (definido por FAILAT).
BSTR	cli_Prompt	Prompt atual (definido por PROMPT).
BPTR	cli_StandardInput	Entrada default do CLI (terminal).
BPTR	cli_CurrentInput	Entrada atual do CLI.
BSTR	cli_CommandFile	Nome do arquivo de comandos EXECUTE.
LONG	cli_Interactive	Booleano, verdadeiro se forem requeridos prompts.
LONG	cli_Background	Booleano, verdadeiro se o CLI foi criado por RUN.
BPTR	cli_CurrentOutput	Saída atual do CLI.
LONG	cli_DefaultStack	Tamanho do stack a ser obtido em palavras longas.
BPTR	cli_StandardOutput	Saída default do CLI (terminal).
BPTR	cli_ModuleSegList	do comando atualmente carregado.

A função Exit( ) do AmigaDOS utiliza o valor do campo pr_ReturAdd, que aponta para a posição imediatamente acima do endereço de retorno no stack atualmente ativo. Se a saída de um programa se dá através da execução de um RTS num stack vazio, o controle passa para o endereço da rotina introduzido no stack por CreateProc( ) ou pelo CLI. Caso o programa termine com uma chamada a Exit( ), então o AmigaDOS utiliza esse ponteiro para extrair o mesmo endereço de retorno. Note que a função Exit( ) do AmigaDOS é inadequada para a saída da maioria dos programas, os quais deverão de preferência empregar a função exit fornecida pelo fabricante do compilador.

O valor do campo pr_PktWait é normalmente zero. Caso contrário, o AmigaDOS chamará a função apontada por pr_PktWait toda vez que um processo estiver aguardando um sinal que indica a chegada de uma mensagem. Da mesma maneira que GetMsg( ), essa função deverá retornar uma mensagem assim que a mesma estiver disponível. Normalmente esta função é usada para filtrar mensagens privativas que chegam à porta de mensagens padrão do processo, e não são dirigidas para o AmigaDOS.

O ponteiro pr_WindowPtr é usado quando o AmigaDOS detecta um erro que normalmemte requer a intervenção do usuário. Um exemplo desse tipo de erro é a tentativa de gravar dados num disco protegido contra gravação, ou então quando o disco está cheio. Se o valor do campo pr_WindowPtr for -1, o erro será retornado para o programa que chamou, na forma de um código de erro da rotina Open( ), Write( ), ou qualquer outra desse gênero. Se o valor de pr_WindowPtr for zero, o AmigaDOS apresentará na tela do Worbench um pequeno menu (conhecido como *requester*) informando a ocorrência do erro e proporcionando ao usuário a oportunidade de tentar novamente a operação ou cancelá-la. Caso o usuário decida cancelar a operação, o AmigaDOS retornará o código de erro ao programa que chamou. Se ele preferir tentar de novo, ou inserir um outro disco, então o AmigaDOS tentará executar a operação mais uma vez. Nas versões V2.0 e posteriores do AmigaDOS, os requesters serão apresentados na tela pública default.

Se você inserir um valor positivo no campo pr_WindowPtr, o AmigaDOS assumirá que este é um ponteiro para a estrutura de uma janela. Normalmente você introduziria aqui a estrutura Window referente à janela que estiver sendo utilizada. Neste caso, o AmigaDOS exibirá a mensagem de erro na janela que você tiver especificado, ao invés de usar a tela do Workbench. Observe que mesmo que você esteja utilizando uma outra tela, não há problema em manter o campo pr_WindowPtr igual a zero. Porém, as memsagens exibidas pelo AmigaDOS aparecerão na tela do Workbench – a qual possivelmemte estará encoberta pela sua tela.

O valor inicial de pr_WindowPtr é herdado do processo que criou o processo atual. Se você decidir efetuar uma alteração em pr_WindowPtr de dentro de um programa que roda sob o CLI, você deverá salvar o valor original e restaurá-lo quando terminar, do contrário, o processo CLI irá conter um pr_WindowPtr apontando uma janela que não se encontra mais presente.

Todos os demais campos da estrutura de processo são novos e aparecem somente nas versões V2.0 e posteriores do AmigaDOS. O campo pr_HomeDir indica o diretório a partir do qual o programa associado com o processo foi carregado. Este campo é referenciado quando o recurso PROGDIR: da versão V2.0 é utilizado. O campo pr_Flags é privativo, contendo flags para uso exclusivo do AmigaDOS.

O campo (*pr_ExitCode)( ) é um LONG que aponta para a rotina de limpeza da memória a ser chamada após o término do programa. Este campo recebe como parâmetro o código de retorno do programa, e pode devolver um código de retorno modificado quando o processo termina. O campo pr_ExitData é oferecido como uma conveniência, permitindo que você passe informações adicionais para a rotina pr_ExitCode automaticamente.

Outro campo novo é pr_Arguments. Ele corresponde a um string, terminado por NULL, com os argumentos a nível de registro passados para o processo no início do programa. Você pode modificar este campo usando a função SetArguments( ), mas se o fizer, certifique-se de restaurar seus valores originais antes de abandonar o programa.

O campo pr_LocalVars é usado para implementar variáveis locais ao processo. Não acesse esses dados diretamente, use as novas funções GetVar( ), SetVar( ) e DeleteVar( ) com esta finalidade.

O campo pr_ShellPrivate é para uso privativo do Shell associado com o processo. Nunca acesse este campo.

O valor em pr_CES aponta para um stream de erro que o processo poderá utilizar separadamente de pr_CIS e pr_COS. Este campo ainda não estava completamente implementado no AmigaDOS por ocasião da elaboração deste livro.

Redirecionando os Requesters do Sistema

No Amiga, quando um usuário ou programa requisitam um arquivo num volume que não esteja montado, o AmigaDOS apresenta um pequeno menu intitulado System Requester na tela do Workbench, solicitando que o usuário insira um disco.

O campo pr_WindowPtr da estrutura Process determina onde a maioria dos requesters do sistema relacionados com o processo deverão aparecer. O valor normal é 0L, e faz com que os requesters sejam exibidos na tela do Workbench (a tela pública default).

Se a sua aplicação dispuser de uma tela própria personalizada, e você precisar efetuar qualquer operação que possa causar o surgimento de um System Request (tal como carregar arquivos ou imprimi-los), você deverá redirecionar os requesters para o seu processo, de maneira que os requesters relacionados com o seu processo apareçam na tela da sua aplicação.

Para fazer isso, localize primeiramente o seu processo:

```
proc = (struct Process *)FindTask(NULL)
```

Salve o antigo valor de proc->pr_WindowPtr e substitua-o por um ponteiro para uma das janelas Intuition na sua tela personalizada. Os requesters surgirão na mesma tela (e com o mesmo título) que essa janela. Certifique-se de restaurar o valor original de pr_WindowPtr antes de fechar a sua janela ou encerrar o programa.

Outras aplicações podem desejar desabilitar temporariamente os requesters, de maneira que tentativas de executar chamadas como Open( ) ou Lock( ) com volumes não montados simplesmente retornem um erro sem a apresentação de um requester. Para fazer isso, salve o antigo valor de pr_WindowPtr e armazene -1L em seu lugar. Antes de encerrar o seu programa, restaure o valor original de pr_WindowPtr.

Estrutura das Bibliotecas do DOS

Embora exista somente uma vez, esta estrutura de dados é utilizada por todos os processos do AmigaDOS. Fazendo uma chamada a OpenLibrary( ), você poderá obter um ponteiro para a base da biblioteca. A base da estrutura de dados consiste num offset positivo para o ponteiro da base da biblioteca. O ponteiro da base da biblioteca aponta para a seguinte estrutura:

- Estrutura Node da biblioteca.
- APTR para estrutura RootNode do DOS.
- APTR para Vetor Global Compartilhado do DOS.
- Dump do registro privativo do DOS.

Muitas chamadas internas do AmigaDOS utilizam o Vetor Global Compartilhado, que consiste em uma tabela de jumps. Você não deverá utilizá-lo normalmente, exceto através das chamadas de interfaceamento disponíveis. O Vetor Global está sujeito a modificações sem aviso prévio.

A estrutura RootNode consiste no seguinte:

```
Valor       Nome            Descrição
BPTR        rn_TaskArray    Array de processos CLI.
BPTR        rn_ConsoleSegment SegList para o CLI.
struct
DateStamp   rn_Time         Hora atual.
LONG        rn_RestartSeg   SegList para o processo de validação do disco.
BPTR        rn_Info         Ponteiro para estrutura Info.
```

Os campos descritos a seguir foram acrescentados à estrutura RootNode nas versões V2.0 e posteriores do AmigaDOS.

```
BPTR        rn_FileHandler  Segmento para um handler de arquivos.
            Segment
struct
MinList     rn_CliList      Nova lista de todos os processos CLI, o
                            primeiro cpl_Array é igual a rn_TaskArray.
struct
MsgPort     *rn_BootProc    Ponteiro privativo para a porta de mensagens
                            do filesystem a partir do qual foi dado o boot.
BPTR        rn_ShellSegment SegList para o Shell (para o uso de NEWSHELL).
LONG        rn_Flags;       Flags para uso do DOS.
```

O array rn_TaskArray, cujo tamanho é armazenado em rn_TaskArray[0], armazena os identificadores (ID) de processo – em outras palavras, as portas de mensagens associadas com os processos – para cada CLI. O ID do processo para o CLI *n* é armazenado em rn_TaskArray [*n*]. Uma área contígua vazia no array é preenchida com zeros. No AmigaDOS versão 2.0, a tabela TaskArray é duplicada e estendida com a estrutura rn_CliList. Os comandos RUN e NEWCLI inspecionam a tabela rn_TaskArray à procura da próxima área contígua livre, e a utilizam para o novo CLI criado. Você não deve acessar a tabela rn_TaskArray diretamente a partir de seu programa, ao invés disso, use as funções do AmigaDOS disponíveis para esse fim.

O campo rn_ConsoleSegment contém a lista de segmentos para o código do CLI. Os comandos RUN e NEWCLI usam este valor para criar uma nova instância de um CLI.

A estrutura RootNode armazena a data e hora atuais. Normalmente você deverá utilizar a função DateStamp( ) do AmigaDOS a fim de retornar um conjunto de valores consistentes. Os valores de Days, Mins e Ticks especificam a data e a hora. O valor de Days é o número de dias transcorridos desde 1º de janeiro de 1978. O valor de Mins é o número de minutos contados desde a meia-noite. Um Tick equivale a 1/50 de segundo, mas a hora só será atualizada quando a função DateStamp( ) for chamada.

O campo RestartSeg é uma lista de segmentos para a rotina de validação do disco,

que consiste num processo que o AmigaDOS cria toda vez que você insere um novo disco em um drive. Nas versões V2.0 e posteriores do AmigaDOS este campo é nulo, já que o processo de validação do disco não é mais separado.

Os demais campos da estrutura RootNode são novos, e aparecem apenas nas versões V2.0 e posteriores do AmigaDOS.

O campo chamado rn_FileHandlerSegment é uma lista de segmentos do filesystem da ROM.

O campo rn_CliList armazena uma lista de tabelas de ponteiros CLI. Isto suplanta o procedimento do rn_TaskArray usado nas versões anteriores do AmigaDOS e elimina o limite quanto ao número de CLIs que podem ser executados ao mesmo tempo. Note que a primeira tabela na lista rn_CliList é também armazenada em TaskArray, com o propósito de preservar a compatibilidade com as versões mais antigas do AmigaDOS. Esta lista deve ser acessada somente utilizando-se as funções FindCli( ) e MaxCli( ). Jamais a acesse diretamente de seu programa.

O campo rn_BootProc é um ponteiro privativo para o processo do filesystem a partir do qual foi inicializado o sistema. Este não é necessariamente o mesmo filesystem da ROM.

O campo rn_ShellSegment aponta a lista de segmentos para o Shell ativo na inicialização do sistema. Já o campo rn_Flags contém novos flags usados para futuras expansões pelo AmigaDOS. Atualmente ele contém apenas um flag, que determina se o curinga do AmigaDOS deverá ser um caractere * ou #?.

A Subestrutura Info

Para acessar a estrutura Info, que possui o formato mostrado a seguir, você deverá utilizar o ponteiro rn_Info.

Valor	Nome	Descrição
BPTR	di_McName	Ponteiro para a lista residente.
BPTR	di_DevInfo	Lista de dispositivos.
BPTR	di_Devices	Atualmente zero.
BPTR	di_Handlers	Atualmente zero.
APTR	di_NetHand	Atualmente zero.

A maioria dos campos da subestrutura Info se encontra vazia no momento o Commodore-Amiga deverá utilizá-los para expandir o sistema.

A estrutura DevInfo consiste numa lista de ligações. Você pode utilizá-la a fim de identificar todos os nomes de dispositivos que o AmigaDOS é capaz de reconhecer, incluindo os nomes de associações ASSIGN e os nomes de volume dos discos. Para acessar as informações na estrutura DevInfo nas versões V1.3 e anteriores do AmigaDOS, é necessário primeiramente chamar a função Forbid( ). Isto significa que chamadas Wait( ), bem como a passagem de mensagens, não são permitidas enquanto você acessar a estrutura. Além disso, você precisará também chamar a função Permit( ) quando tiver terminado.

Nas versões V2.0 e posteriores do AmigaDOS, o método é diferente: você deverá chamar a função LockDosList( ) antes acessar a estrutura DevInfo. Isto permite

chamar Wait( ) e efetuar a passagem de mensagens no código que acessa a estrutura DevInfo. Ao terminar, a função a ser chamada é UnLockDosList( ). Para manter a compatibilidade com os programas da versão V1.3, LockDosList( ) chama Forbid( ) e UnLockDosList( ) chama Permit( ). Este procedimento, contudo, será eliminado numa próxima versão do AmigaDOS.

Existem três formatos possíveis para as entradas da lista DevInfo, dependendo se a entrada se refere a um volume de disco, a uma associação, ou a um dispositivo ou diretório. Para uma entrada descrevendo um dispositivo ou diretório (via ASSIGN), o formato é o seguinte:

Valor	Nome	Descrição
BPTR	dvi_Next	Ponteiro para próxima entrada na lista ou zero.
LONG	dvi_Type	Tipo da entrada no lista (dispositivo ou diretório).
APTR	dvi_Task	Processo de manipulação ou zero.
BPTR	dvi_Lock	Lock sobre o filesystem ou zero.
BSTR	dvi_Handler	Nome do arquivo do handler ou zero.
LONG	dvi_StackSize	Tamanho do stack para o processo de manipulação.
LONG	dvi_Priority	Prioridade do processo de manipulação.
LONG	dvi_Startup	Valor de inicialização a ser passado para o processo de manipulação.
BPTR	dvi_SegList	SegList para o processo de manipulação ou zero.
BPTR	dvi_GlobVec	Vetor Global para o processo de manipulação ou zero.
BSTR	dvi_Name	Nome do dispositivo ou nome associado via ASSIGN.

O campo dvi_Next liga todas as entradas da lista em conjunto, e o nome do dispositivo lógico é armazenado no campo dvi_Name. Apesar de o campo dvi_Name ser um string BSTR, note que ele precisa terminar com um byte zero, e que este byte extra não deverá ser incluído na contagem do tamanho do string.

O campo dvi_Type conterá 0 (dt_device) ou 1 (dt_dir). Você pode criar uma entrada de diretório com o comando ASSIGN do AmigaDOS. Este comando aloca um nome para um diretório, o qual passará então ser chamado por um nome de dispositivo. Se a entrada na lista se referir a um diretório, o campo Task referencia o processo do filesystem que manipula o disco, e o campo dvi_Lock contém um ponteiro para um lock sobre o diretório em questão.

Se a entrada na lista se referir a um dispositivo, então o dispositivo pode ou não ser residente. Caso o dispositivo seja residente, o campo dvi_Task identifica o processo de manipulação desse dispositivo e dvi_Lock será normalmente zero. Se o dispositivo não for residente, o campo dvi_Task valerá zero e o AmigaDOS utilizará o restante da estrutura da lista.

O valor zero no campo dvi_SegList indica que a rotina para manipulação do dispositivo não se encontra na memória. O campo Handler é um string que especifica o arquivo contendo a rotina (por exemplo, SYS:L/ram-handler). Uma chamada a LoadSeg( ) carregará a rotina contida no arquivo e inserirá o resultado no campo SegList.

O AmigaDOS atualmente cria um novo processo de manipulação utilizando os valores dos campos dvi_SegList, dvi_StackSize e dvi_Priority. O novo processo será

um processo BCPL, requerendo portanto um Vetor Global. Este pode ser o valor que você especificou no campo dvi_GlobVec, ou um novo vetor global privativo se dvi_GlobVec for zero. Se dvi_GlobVec for -1, então o novo processo não é um processo BCPL, tendo sido criado por CreateProc( ).

Ao novo processo é passada uma mensagem contendo o nome originalmente especificao, o valor armazenado em Startup, e a base da entrada na lista. O novo processo de manipulação pode então decidir se modifica o ID do processo dentro da área Task ou não, de acordo com o solicitado. Se a área dvi_Task for modificada, então referências subseqüentes ao nome do dispositivo utilizarão a mesma tarefa do handler, isto é o que o dispositivo RAM: faz. Caso contrário, as referências posteriores ao dispositivo resultarão em novas invocações de processos, isto é o que o dispositivo CON: faz.

Se o campo dvi_Type na entrada da lista for igual a 2 (dt_volume), o formato da estrutura da lista será ligeiramente diferente:

Valor	Nome	Descrição
BPTR	dvi_Next	Ponteiro para próxima entrada na lista ou zero.
LONG	dvi_Type	Tipo de entrada na lista (volume).
APTR	dvi_Task	Processo de manipulação ou zero.
BPTR	dvi_Lock	Lock sobre o filesystem.
struct DateStamp	dol_VolumeDate	Data de criação do volume.
BPTR	dol_LockList	Lista de locks ativos para o volume.
LONG	dol_DiskType	Tipo do disco.
LONG	Reserva	Não usado.
BSTR	dvi_Name	Nome do volume.

Neste caso, o campo dvi_Name contém o nome do volume, e o campo Task aponta o processo de manipulação, caso o volume esteja inserido no drive. Se o volume não estiver inserido, este campo valerá zero. Para distinguir discos com o mesmo nome, o AmigaDOS grava um indicador de data durante a criação do volume, e salva a indicação de data na estrutura da lista. O AmigaDOS pode então comparar os indicadores de data de diferentes volumes, quando necessário.

Caso o volume não esteja inserido no drive, o AmigaDOS salva a lista dos locks ativos naquele momento no campo dol_LockList. Note que este campo é privativo e nunca deverá ser acessado diretamente pelo seu programa. Nem todos os filesystems suportam o campo dol_LockList. O AmigaDOS usa o campo dol_LockList para identificar o tipo do disco. O tipo do disco é compactado em até quatro caracteres numa palavra longa e preenchido à direita com NULLs.

Se o campo dvi_Type na entrada da lista for igual a 3 ou 4 (dt_nonbinding ou dt_late), a estrutura da lista terá o seguinte formato:

Valor	Nome	Descrição
BPTR	dvi_Next	Ponteiro para próxima entrada na lista, ou zero.
LONG	dvi_Type	Tipo de entrada na lista (associação sem vínculo ou com vínculo posterior).
APTR	dvi_Task	Processo de manipulação ou zero.
BPTR	dvi_Lock	Lock sobre o filesystem.

```
UBYTE *     dol_AssignName  Nome para associação sem vínculo ou com
                            vínculo posterior.
struct
AssignList * dol_list       Para associações de múltiplos diretórios (regular).
LONG        Reserva         Não usado.
LONG        Reserva         Não usado.
LONG        Reserva         Não usado.
LONG        Reserva         Não usado.
BSTR        dvi_Name        Nome do volume.
```

Para associações de múltiplos diretórios, os locks adicionais são retirados do campo dol_Llist. Para associações sem vínculo ou com vínculo posterior, o campo dol_AssignName contém o string para a associação (path). Associações de vínculo posterior tornam-se associações regulares no momento em que a vinculação passa a ter efeito. Note que qualquer associação normal pode ter mais locks retirados de dol_List.

Alocação da Memória

Toda a memória alocada pelo AmigaDOS é obtida através de chamadas à função AllocMem( ) do Exec. É desta maneira que o AmigaDOS obtém estruturas como locks e filehandles. Normalmente o AmigaDOS restitui estas estruturas para a área de memória livre chamando a função FreeMem( ). Nas versões V2.0 e posteriores do AmigaDOS, as funções AllocVec( ) e FreeVec( ) podem ser utilizadas com a mesma finalidade. Em ambos os casos, cada segmento de memória alocado pelo AmigaDOS é identificado por um BPTR para a segunda palavra longa na estrutura. A primeira palavra longa sempre contém o tamanho total do segmento, em bytes. A estrutura da memória alocada é a seguinte:

```
Valor       Nome            Descrição
LONG        BlockSize       Tamanho do bloco de memória.
LONG        FirstData       Primeiro segmento de dados. o BPTR para o
                            bloco aponta para cá.
```

Listas de Segmentos

Para obter uma lista de segmentos você deverá chamar a função LoadSeg( ). O resultado é um BPTR para a memória alocada, sendo que o tamanho do bloco de memória contendo cada entrada na lista é armazenado a -4 bytes do BPTR. Este tamanho possui 8 bytes a mais que o tamanho da entrada na lista de segmentos, dando espaço para o campo do link e o próprio campo do tamanho.

A SegList é uma lista ligada por BPTRs e terminada por zero. O restante de cada entrada na lista de segmentos contém o código carregado. Desse modo, o formato pode ser assim descrito:

```
Valor       Nome            Descrição
LONG        NextSeg         BPTR para o próximo segmento ou zero.
LONG        FirstCode       Primeiro valor do arquivo binário.
```

File Handles

Os filehandles, que são criados pela função Open( ) do AmigaDOS, são geralmente utilizados como argumentos para outras funções, como Read( ) e Write( ). O AmigaDOS retorna um filehandle como um BPTR para a seguinte estrutura:

Valor	Nome	Descrição
struct Message *	fh_Link	Reservado para o AmigaDOS.
struct Message *	fh_Key	Porta de resposta para o packet.
struct Message *	fh_Port	ID de processo do handler.
LONG	fh_Buf	Buffer para uso interno.
LONG	fh_Pos	Posição de caractere para uso interno.
LONG	fh_End	Posição final para uso interno.
LONG	fh_Func1	Função chamada na exaustão da capacidade do buffer.
LONG	fh_Func2	Função chamada quando o buffer está cheio.
LONG	fh_Func3	Função chamada quando o filehandle está fechado.
LONG	fh_Arg1	Argumento (depende do tipo do filehandle).
LONG	fh_Arg2	Argumento (depende do tipo do filehandle).

A maioria dos campos se destina exclusivamente ao uso interno do AmigaDOS. Normalmente, as funções de leitura e gravação de dados como Read( ) e Write( ) utilizam o filehandle a fim de indicar o processo de manipulação e os argumentos a serem passados. Os valores internos do filehandle não devem ser alterados por programas do usuário. De maneira geral, você também não deverá ler esses valores (com exceção do campo fh_Arg1, que é utilizado em operações de I/O direto de packets).

Locks

O sistema de arquivo utiliza extensivamente uma estrutura de dados chamada de *lock*. Esta estrutura serve a dois propósitos. Em primeiro lugar, ela funciona como um mecanismo para a abertura de arquivos para múltiplas operações de leitura, ou uma única operação de gravação. Note que obter um lock compartilhado de leitura em um diretório não faz com que esse diretório deixe de ser atualizado.

Em segundo lugar, o lock é capaz de proporcionar uma identificação exclusiva para um determinado arquivo. Embora um arquivo possa ser especificado de vários modos, o lock constitui uma maneira bastante simples de se lidar com arquivos. O lock contém a localização efetiva do bloco do diretório ou do bloco de cabeçalho de um arquivo no disco, representando assim um meio rápido de se especificar um objeto de filesystem particular. A estrutura de um lock é a seguinte:

Valor	Nome	Descrição
BPTR	fl_Link	BPTR para próximo lock na cadeia, ou zero.
LONG	fl_Key	Número do bloco do diretório ou de cabeçalho do arquivo.

```
LONG        fl_Access       Acesso exclusivo ou compartilhado.
struct
MsgPort *   fl_Task         ID do processo de manipulação.
BPTR        fl_Volume       BPTR para a entrada dlt_volume na lista do DOS.
```

Como o AmigaDOS utiliza o campo fl_Link com a finalidade de encadear os locks, você deverá ter o cuidado de não alterá-lo. O sistema de arquivo preenche o campo fl_Key de modo a apontar a localização no disco do bloco do diretório (*directory block*) ou do bloco do cabeçalho do arquivo (*file header block*). Um valor igual a -2 no campo fl_Access indica um lock compartilhado para leitura, quando este valor for -1, o lock será exclusivo para gravação. O campo fl_Task contém um ponteiro para o processo de manipulação do dispositivo contendo o arquivo ao qual o lock está associado. Finalmente, o campo fl_Volume aponta para o nó na estrutura DevInfo que identifica o volume ao qual o lock se refere. As entradas de volume na estrutura DevInfo permanecem mesmo que outro disco seja inserido ou que hajam quaisquer locks abertos neste volume.

Note que um lock também pode ser zero. O caso especial do lock zero indica que esse lock se refere à raiz do filesystem inicial, além disso, o campo pr_FileSystemTask na estrutura de dados do processo indica o processo de manipulação desse dispositivo. Nenhum dos campos na estrutura do lock deverá ser modificado por um programa do usuário. A estrutura Lock é estritamente privativa dos handlers, e destinada somente para leitura.

Os Packets do AmigaDOS

A passagem de packets é responsável por toda a comunicação efetuada entre os vários processos pelo AmigaDOS. O diagrama funcional abaixo mostra como os packets se relacionam com os outros componentes do sistema operacional do Amiga.

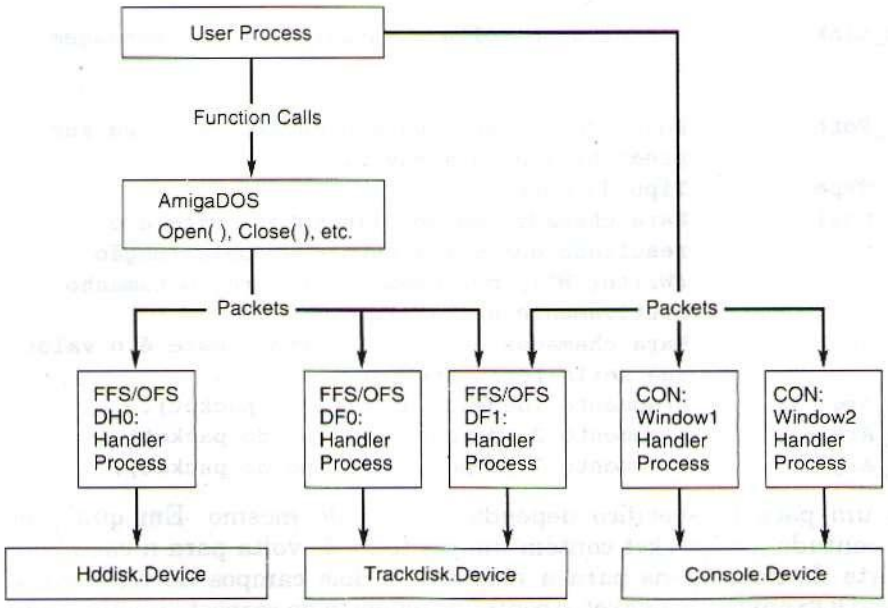


Figura 11-1

A estrutura StandardPacket (definida no arquivo dos/dosextens.h) é utilizada para enviar comandos de packets para uma porta de mensagens de um processo. Esta estrutura contém uma estrutura ExecMessage e uma estrutura DOSPacket do AmigaDOS.

```
struct StandardPacket  {
struct Message    sp_Msg;
struct DOSPacket  sp_Pkt;
};
```

Esta estrutura precisa ser alinhada por palavra longa, e inicializada para ligar as seções Message e DosPacket uma à outra.

```
packet->sp_Msg.mn_Node.ln_Name = (char *)&(packet->sp_Pkt);
packet->sp_Pkt.dp_Link         = &(packet->sp_Msg);
```

Os packets precisam também ser inicializados com uma porta ReplyPort que pode ser criada com a função CreatePort( ) da biblioteca amiga.lib:

```
if (replyport = (struct MsgPort *) CreatePort (NULL,0))
{
     packet->sp_Pkt.dp_Port        = replyport;
}
```

A parte DOSPacket da estrutura StandardPacket é usada tanto para passar o tipo e os argumentos do packet como para receber os resultados do mesmo. O número e os tipos de argumentos, assim como os resultados, variam de acordo com os diferentes tipos de packets, e são documentados na seção a seguir juntamente com a descrição de cada packet. Uma subestrutura DOSPacket precisa ser alinhada por palavra longa, e possui o seguinte formato:

Valor	Nome	Descrição
struct Message *	dp_Link	Ponteiro de volta para estrutura de mensagem Exec.
struct MsgPort *	dp_Port	Porta de resposta para o packet. Precisa ser preenchida a cada envio.
LONG	dp_Type	Tipo do packet
LONG	dp_Res1	Para chamadas de um filesystem, este é o resultado que seria retornado pela função (Write("W"), por exemplo, retorna o tamanho efetivamente gravado).
LONG	dp_Res2	Para chamadas de um filesystem, este é o valor que seria retornado por IoErr( ).
LONG	dp_Arg1	Argumento (depende do tipo do packet).
LONG	dp_Arg2	Argumento 2 (depende do tipo do packet).
LONG	dp_Arg7	Argumento 7 (depende do tipo do packet).

O formato de um packet específico depende do tipo do mesmo. Em qualquer circunstância, contudo, um packet contém um ponteiro de volta para a estrutura Message, a porta de mensagens para a resposta, e dois campos de resultados. Quando o AmigaDOS envia um packet, o conteúdo da porta de resposta é sobreposto

com a identificação do processo do remetente, de maneira que o packet possa ser retornado para o mesmo. Assim, todas as vezes que você enviar um packet para um processo de manipulação de dispositivo do AmigaDOS, é indispensável preencher a porta de mensagens a cada envio; caso contrário, quando o packet retornar, o AmigaDOS terá sobreposto o conteúdo da porta original. O AmigaDOS mantém todos os outros campos, exceto os campos dos resultados.

Todos os packets do AmigaDOS são enviados à porta de mensagens criada como parte de um processo. Essa porta de mensagens é inicializada da tal forma que as mensagens, ao chegarem, fazem com que o bit de sinal 8 seja setado. Todo processo do AmigaDOS, ao aguardar uma mensagem, espera que o bit de sinal 8 seja setado. Quando o processo "acorda" porque este evento ocorreu, GetMsg( ) pega a mensagem na porta e extrai o endereço do packet. Se o processo for um handler (isto é, um processo de manipulação de dispositivo do AmigaDOS), o packet conterá um valor no campo PktType que indica uma ação a ser executada, tal como a leitura de algum dado. Os campos de argumentos do packet contêm informações específicas, tais como o tamanho do buffer onde os caracteres deverão ser copiados.

Quando o processo de manipulação de dispositivo do AmigaDOS tiver completado o trabalho necessário para satisfazer a requisição, o packet retorna para o remetente, utilizando a mesma estrutura de mensagem. Tanto a estrutura de mensagem como a estrutura do packet precisam ser alocadas pelo cliente, e não deverão ser desalocadas antes que a resposta seja recebida. Normalmente o AmigaDOS é chamado pelo cliente para enviar o packet, tal como acontece quando você faz uma chamada à função Read( ). Entretanto, existem circunstâncias nas quais pode ser necessária uma operação de I/O assíncrona, e neste caso o cliente pode enviar packets diretamente para o handler. Lembre-se que as estruturas de mensagem e packet precisam ser alocadas, e o campo de identificação do processo preenchido com a porta de mensagens para a qual o packet deverá retornar. Uma chamada à função PutMsg( ) poderá então ser utilizada para enviar a mensagem ao destinatário. Note que diversos packets podem ser enviados, retornando para as mesmas portas de mensagens ou para outras diferentes.

Tipos de Packets

Os packets enviados para um filesystem ou handler podem ser divididos em diversas categorias básicas:

- Entrada/Saída Básica. Estas ações se ocupam da transferência de dados entre os objetos controlados pelo handler.
- Informação/Manipulação de Arquivos/Diretórios. Estas ações são usadas para se conseguir acesso sobre estruturas de alto nível do filesystem e manipulá-las.
- Informação/Manipulação de Volumes. Estas ações permitem o acesso a um volume específico controlado pelo filesystem.
- Controle e Manutenção do Handler. Estas ações possibilitam o controle sobre o próprio handler/filesystem, independente do volume ou estrutura subjacente.
- Ações Internas do Handler. Estas ações nunca são enviadas diretamente ao handler. Em sua maior parte, elas costumam ser respostas a requisições de I/O feitas pelo handler. O handler molda essas respostas como packets com o propósito de simplificar o processamento.

- Packets Obsoletos. Estes packets não são mais considerados válidos para o uso com handlers e filesystems.
- Packets Somente de Console. Estes packets são específicos para processos de manipulação do console. Os filesystems podem ignorar tais packets.

Cada tipo de packet documentado nesta seção é listado com o nome da ação que executa, o seu número correspondente, as rotinas do AmigaDOS que utilizam esse packet e a lista dos parâmetros usados pelo packet. As variáveis C para os parâmetros do packet deverão ser de um dos seguintes tipos:

BPTR	Refere-se a um ponteiro BCPL (o endereço do objeto em questão sobre o qual é efetuado um *shift right* duas vezes). **Nota:** Isto significa que o objeto precisa ser alinhado para um limite de palavra longa.
LOCK	Um BPTR para uma estrutura FileLock retornada por uma ação ACTION_LOCATE_OBJECT. Um lock igual a zero é considerado legal, indicando a raiz do volume para o handler.
BSTR	BPTR para um string na qual o primeiro byte indica o número de caracteres existentes no string. Um byte desse tamanho não leva sinal, mas como a informação é armazenada em um byte, os strings são limitados a 255 caracteres de extensão.
BOOL	Um valor de 32 bits booleano contendo DOSTRUE (-1) ou DOSFALSE (0). **Nota:** Comparações de igualdade com DOSTRUE devem ser evitadas.
CODE	Um código de erro de 32 bits, tal como definido no arquivo include dos/dos.h. Os handlers não devem retornar códigos de erro, em razão da existência dos códigos definidos em dos/dos.h.
ARG1	O campo FileHandle->fh_Arg1.
LONG	Um valor inteiro de 32 bits.

Entrada/Saída Básica

As ações de Entrada/Saída Básica são suportadas pelos handlers e pelos filesystems. Desse modo, a aplicação pode ter acesso a nível de stream tanto aos dispositivos quanto aos arquivos. A principal diferença é que um handler não suporta necessariamente uma ação ACTION_SEEK, enquanto que em geral se espera que um filesystem seja capaz de fazê-lo.

Estas ações funcionam baseadas em um FileHandle que é preenchido com uma destas três modalidades de abertura:

```
ACTION_FINDINPUT     1005    Open( .., MODE_OLDFILE)
ACTION_FINDOUTPUT    1006    Open(..., MODE_NEWFILE)
ACTION_FINDUPDATE    1004    Open(..., MODE_READWRITE)
ARG1:   BPTR    Filehandle a preencher.
ARG2:   LOCK    Lock sobre o diretório ao qual ARG3 é relativo.
ARG3:   BSTR    Nome do arquivo a ser aberto (relativo a ARG1).
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Todas as ações usam o lock (ARG2) como a posição base do diretório onde deverá ser aberto o arquivo. Se este lock for NULL, o nome do arquivo (ARG3) é relativo à raiz do volume atual. Por este motivo, os arquivos não são definidos apenas através de seus nomes, podendo incluir também um nome de volume (seguido por ":") e múltiplas barras, que permitem ao filesystem a resolução completa do nome. Isto evita que o AmigaDOS ou a aplicação precisem analisar a sintaxe dos nomes antes de enviá-los para o filesystem. Note que o lock em ARG2 precisa ser associado com o filesystem em questão, não é permitido utilizar um lock de outro filesystem.

O programa que chama a ação é que tem a posse do filehandle (ARG1). Desse modo, o programa precisa inicializar o filehandle antes de tentar efetuar qualquer tentativa de abertura de um objeto (no caso de uma chamada a Open( ), o AmigaDOS aloca automaticamente o filehandle, liberando-o por ocasião de uma chamada a Close( )). Todos os campos precisam ser inicializados com zero, exceto fh_Pos e fh_End, que devem ser definidos como -1. Se a abertura for bem sucedida, caberá ao handler preencher o campo fh_Type com um ponteiro para a porta de mensagens do processo. Por último, compete ao handler inicializar fh_Arg1 com algum valor que lhe permita caracterizar a posição única do objeto sendo aberto (normalmente um arquivo). Este valor é específico para implementação, sendo passado para operações READ/WRITE/SEEK/END/TRUNCATE, e não para o filehandle em si.

FINDINPUT e FINDUPDATE têm em comum o fato de que sua execução só será bem sucedida se o arquivo já existir. Tanto FINDINPUT como FINDUPDATE abrem o objeto com um lock compartilhado, a diferença é que FINDUPDATE criará esse arquivo se ele ainda não existir. FINDOUTPUT sempre abrirá o arquivo (deletando qualquer arquivo já existente) com um lock exclusivo.

```
ACTION_READ      'R'     Read(...)
ARG1:   ARG1     Campo fh_Arg1 do FileHandle aberto.
ARG2:   APTR     Buffer para inserir os dados.
ARG3:   LONG     Número de bytes a serem lidos
RES1:   LONG     Número de bytes lidos. 0 indica EOF, -1 indica um erro.
RES2:   CODE     Código de erro se RES1 = -1.
```

Esta ação extrai dados do arquivo (ou do canal de entrada) a partir de sua posição atual. Se restarem menos bytes no arquivo do que o solicitado, somente os bytes existentes serão retornados, o número de bytes será armazenado em RES1. Para indicar um erro, o handler atribui -1 a RES1 e retorna o código de erro em RES2. Se a tentativa de leitura falhar, a posição atual no arquivo permanecerá inalterada. Note que um handler pode retornar um número de bytes menor do que o solicitado, mesmo que não esteja próximo o final do arquivo. Isto acontece com filehandles do tipo interativo que podem retornar uma linha de cada vez quando o usuário aciona a tecla Return – como sucede, por exemplo, com o handler do console (CON:).

```
ACTION_WRITE     'W'     Write(...)
ARG1:   ARG1     Campo fh_Arg1 do FileHandle aberto.
ARG2:   APTR     Buffer de dados a gravar no filehandle.
ARG3:   LONG     Número de bytes a serem gravados.
RES1:   LONG     Número de bytes gravados.
RES2:   CODE     Código de erro se RES1 for diferente de ARG3.
```

Esta ação copia dados na posição atual do arquivo (ou canal de saída). O arquivo

será automaticamente estendido se a gravação ultrapassar o final do arquivo. O handler indica uma falha da ação retornando uma soma de bytes em RES1 diferente do número de bytes requisitados em ARG3. Em caso de erro, o handler não atualiza a posição atual do arquivo (embora eventualmente o arquivo possa ter sido estendido e alguns dados sobrepostos), de maneira que a aplicação possa tentar novamente a operação com segurança.

```
ACTION_SEEK        1008      Seek(...)
ARG1:    ARG1     Campo fh_Arg1 do FileHandle aberto.
ARG2:    LONG     Nova posição
ARG3:    LONG     Modo: OFFSET_BEGINNING, OFFSET_END ou OFFSET_CURRENT.
RES1:    LONG     Antiga posição, -1 indica um erro.
RES2:    CODE     Código de erro se RES1 = -1.
```

Este packet define a posição atual para leitura/gravação no arquivo. A nova posição (ARG2) é relativa ao início do arquivo (OFFSET_BEGINNING), ao final do arquivo (OFFSET_END) ou à posição atual no arquivo (OFFSET_CURRENT), dependendo do modo especificado em ARG3. Note que ARG2 pode ser negativo. O handler retorna a posição anterior do arquivo em RES1. Qualquer tentativa de executar SEEK após o final do arquivo causará um erro e deixará a posição atual do arquivo numa locação desconhecida.

```
ACTION_END         1007      Close(...)
ARG1:    ARG1     Campo fh_Arg1 do FileHandle aberto.
RES1:    LONG     DOSTRUE
```

Este packet fecha um filehandle aberto. Esta função geralmente retorna DOSTRUE, uma vez que há pouco que uma aplicação possa fazer para corrigir um erro na tentativa de fechar um arquivo. Na versão 2.0 do AmigaDOS, quando um erro é retornado o filehandle não é desalocado. Nas versões 1.3 e anteriores, o DOS não checa o resultado.

```
ACTION_SET_FILE_SIZE      1022          SetFileSize(file,off,mode)
ARG1:    BPTR     FileHandle do arquivo aberto a ser modificado.
ARG2:    LONG     Nova posição do final do arquivo baseada no modo.
ARG3:    LONG     Modo: OFFSET_BEGINNING, OFFSET_END ou OFFSET_CURRENT.
RES1:    BOOL     Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE     Código de erro se RES1 = DOSFALSE.
```

Esta função é usada para modificar o tamanho físico de um arquivo aberto. ARG2, a nova posição do final do arquivo, é relativa ao início do arquivo (OFFSET_BEGINNING), ao final do arquivo (OFFSET_END) ou à posição atual no arquivo (OFFSET_CURRENT), dependendo do modo especificado em ARG3. A posição atual no arquivo não será modificada a menos que fique após a nova posição do final do arquivo (EOF). Neste caso, a nova posição do arquivo será movida para o novo final do arquivo. Se houver outros filehandles abertos no arquivo, ACTION_SET_FILE_SIZE ajusta os EOFs desses filehandles alternativos para sua respectiva posição atual no arquivo, ou então para a nova posição do EOF do filehandle apontado por ARG1 – a alternativa que fizer o arquivo parecer maior.

Somente 2.0

```
ACTION_LOCK_RECORD      2008              LockRecord(fh,pos,len,mod,tim)
```

```
ARG1:   BPTR    FileHandle do registro sobre o qual ativar o lock.
ARG2:   LONG    Posição inicial (em bytes) do registro no arquivo.
ARG3:   LONG    Tamanho (em bytes) do registro sobre o qual o lock
                deve ser ativado.
ARG4:   LONG    Modo
                0 = Exclusivo.
                1 = Exclusivo Imediato (espera é ignorada).
                2 = Compartilhado.
                3 = Compartilhado Imediato (espera é ignorada).
ARG5    LONG    Período de espera em ticks do AmigaDOS (0 é válido).
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta função ativa um lock sobre uma área de um arquivo em modo compartilhado (indicando somente leitura) ou exclusivo (indicando leitura/gravação). Vários locks de registro compartilhados de diferentes filehandles podem existir simultaneamente sobre uma área particular de um arquivo, mas apenas um filehandle pode ter locks exclusivos de registros numa área particular de cada vez. A "exclusividade" de um lock de arquivo exclusivo se aplica apenas a locks de registros de outros filehandles, não a locks de registros do próprio filehandle. Um filehandle pode ter qualquer número de locks de registro exclusivos sobrepostos. No caso de locks sobrepostos sobre intervalos, deverá ser possível ativar um lock sobre todo o intervalo para que a requisição tenha sucesso. O período de espera (ARG5) corresponde ao número de *ticks* do AmigaDOS (50 por segundo) que deverão ser aguardados até que a operação seja completada com sucesso, ou abandonada com o retorno de um código de erro.

```
Somente 2.0
ACTION_FREE_RECORD      2009        FreeRecord(file,pos,len)
ARG1:   BPTR    FileHandle do registro sobre o qual desativar o lock.
ARG2:   LONG    Posição inicial (em bytes) do registro no arquivo.
ARG3:   LONG    Tamanho (em bytes) do registro sobre o qual o lock deve
                ser desativado.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta função desfaz qualquer lock previamente ativado sobre um registro. Se não houver um lock sobre o intervalo especificado do arquivo, ACTION_FREE_RECORD retornará um erro. Na eventualidade de existirem múltiplos locks sobre uma determinada área, somente um lock será liberado.

Informação/Manipulação de Arquivos/Diretórios

As ações de diretórios/arquivos possibilitam a uma aplicação fazer consultas a respeito de modificações efetuadas nos objetos do handler. Esses packets executam funções tais como criar subdiretórios, resolver links e preencher estruturas FileInfoBlock para arquivos específicos.

```
ACTION_LOCATE_OBJECT    8           Lock(...)
ARG1:   LOCK    Lock sobre o diretório ao qual ARG2 é relativo.
```

```
ARG2:    BSTR    Nome (possivelmente com um path) do objeto sobre o qual
                         o lock deve ser ativado.
ARG3:    LONG    Modo: ACCESS_READ/SHARED_LOCK,
                 ACCESS_WRITE/EXCLUSIVE_LOCK.
RES1:    LOCK    Lock sobre o objeto requisitado ou 0 para indicar a
                 ocorrência de um erro.
RES2:    CODE    Código de erro se RES1 = 0.
```

A função Lock( ) do AmigaDOS utiliza esta ação para criar seus locks. Dados o nome do objeto, que pode incluir um path (ARG2), e um lock sobre o diretório no qual procurar pelo nome (e path), ACTION_LOCATE_OBJECT localizará o objeto dentro do filesystem e criará uma estrutura FileLock associada com esse objeto. Se o lock do diretório em ARG1 for NULL, o nome será considerado relativo à raiz do volume (também conhecido como ":") do filehandler. A memória para a estrutura FileLock retornada em RES1 é mantida pelo handler e liberada por uma ação ACTION_FREE_LOCK. Embora isto não seja obrigatório, se estiver habilitado a suportar o comando FORMAT pré-versão 1.3, um handler precisa aceitar qualquer modo ilegal como ACCESS_READ.

Um handler só poderá criar um lock exclusivo se não houver outros locks a serem resolvidos sobre o objeto em questão. Uma vez criado, um lock exclusivo impedirá a criação de quaisquer outros locks para aquele objeto. Em geral, os handlers utilizam o campo FileLock->fl_Key a fim de estabelecer uma identificação única para um objeto. Note que algumas aplicações se baseiam nesse dado (embora não seja obrigatória a sua implementação pelo handler).

O campo fl_Volume da estrutura FileLock retornada deve apontar para a entrada do volume, na lista de dispositivos do DOS, no qual existe o lock. Além disso, existem vários programas de diagnóstico que esperam que todos os locks para um volume estejam encadeados fora do campo dl_LockList na entrada referente ao volume. Note que não é seguro confiar nessa condição, e que se basear nisto pode causar sérios problemas, inclusive uma perda de controle do sistema (*crash*). Nenhuma aplicação deve utilizar essa referência.

```
ACTION_COPY_DIR     19      DupLock(...)
ARG1:    LOCK    Lock a duplicar.
RES1:    LOCK    Lock duplicado ou 0 para indicar a ocorrência de um erro.
RES2:    CODE    Código de erro se RES = 0.
```

O nome desta ação (ACTION_COPY_DIR) pode induzir a um equívoco, já que ela não manipula diretórios, mas sim cria uma cópia de um lock compartilhado. A cópia será subseqüentemente liberada por meio da ação ACTION_FREE_LOCK. Note que é válido passar um lock NULL. Na versão atual, a função DupLock( ) do AmigaDOS pode retornar sempre 0 se for passado um 0 para ela, embora não seja obrigatório para o handler retornar um 0.

```
ACTION_FREE_LOCK     15     UnLock(...)
ARG1:  LOCK    Lock a ser liberado.
RES1:  BOOL    TRUE
```

Esta ação libera o lock passado para ela. A função Lock( ) do AmigaDOS usa este packet. Se for passado um lock NULL, o handler deve ignorar o packet e retornar uma indicação de operação bem sucedida (TRUE).

```
ACTION_EXAMINE_OBJECT    23          Examine(...)
ARG1:    LOCK    Lock sobre objeto a ser examinado.
ARG2:    BPTR    FileInfoBlock para ser preenchido.
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE
```

Esta ação preenche a estrutura FileInfoBlock com informações sobre o objeto que possui um lock. A função Examine( ) do AmigaDOS utiliza este packet. ACTION_EXAMINE_OBJECT é utilizado, na prática, por dois diferentes tipos de operações: este packet pode ser chamado tanto para a obtenção de informações sobre um determinado objeto, como em preparação para que uma seqüência de operações EXAMINE_NEXT percorra um diretório.

Esta operação, aparentemente simples, possui algumas características peculiares. Uma delas é o campo FileinfoBlock->fib_Comment. Este campo tinha um tamanho de 116 bytes, mas foi modificado para 80 bytes na versão 1.2 do AmigaDOS, os 36 bytes adicionais foram introduzidos no campo fib_Reserved. Outro detalhe peculiar desse packet é que os campos fib_EntryType e fib_DirEntryType precisam ser definidos com o mesmo valor, já que alguns programas consultam o primeiro campo, enquanto que outros programas verificam apenas o segundo.

Os filesystems deverão adotar os mesmos valores para fib_DirEntryType utilizados pelo filesystem da ROM e o RAM-handler. Estes valores são os seguintes:

ST_ROOT	1	
ST_USERDIR	2	
ST_SOFTLINK	3	**Nota:** Aparece como um diretório a menos que seja checado explicitamente.
ST_LINKDIR	4	
ST_FILE	-3	
ST_LINKFILE	-4	

Note também que os handlers deverão usar números maiores que zero para os diretórios, uma vez que alguns programas checam se fib_DirEntryType é maior que zero, ignorando a possibilidade deste campo ser *igual* a zero. Logo, os handlers devem evitar usar 0 porque isto não é interpretado de maneira consistente.

```
ACTION_EXAMINE_NEXT        24    ExNext(...)
ARG1:    LOCK    Lock sobre o diretório sendo examinado.
ARG2:    BPTR    BPTR para FileInfoBlock.
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.
```

A função ExNext( ) do AmigaDOS utiliza este packet para obter informações sobre todos os objetos em um diretório. A ação ACTION_EXAMINE preenche uma estrutura FileInfoBlock com o primeiro arquivo (ou diretório) armazenado no diretório referenciado pelo lock em ARG1. ACTION_EXAMINE_NEXT é usado para obter dados sobre os demais arquivos e diretórios armazenados no diretório apontado por ARG1. ARG2 contém um ponteiro para um campo FileInfoBlock válido que tenha sido preenchido por uma ACTION_EXAMINE ou por uma chamada AC-

TION_EXAMINE_NEXT anterior. Essa estrutura é utilizada pelo packet para localizar a próxima entrada no diretório. Este packet grava sobre o antigo FileInfoBlock as informações referentes ao próximo arquivo (ou diretório) no diretório ARG2. ACTION_EXAMINE_NEXT retorna um código de erro ERROR_NO_MORE_ENTRIES quando não existirem mais diretórios ou arquivos para serem examinados. Infelizmente, tal como ACTION_EXAMINE, este packet também tem as suas peculiaridades. Eis alguns dos detalhes que ACTION_EXAMINE_NEXT precisa levar em conta:

- A situação na qual uma aplicação chama ACTION_EXAMINE_NEXT uma ou mais vezes e então pára de invocá-la antes de encontrar o final do diretório.
- A situação em que o FileInfoBlock passado para ACTION_EXAMINE_NEXT não é o mesmo que aquele passado para ACTION_EXAMINE ou mesmo para a operação EXAMINE_NEXT anterior, mas sim uma cópia do FileInfoBlock com apenas o campo fib_DiskKey e os primeiros 30 bytes dos campos fib_FileName copiados. Este procedimento já foi lícito, mas agora é considerado ilegal e não deverá funcionar no futuro. Qualquer novo programa deverá ser escrito levando este dado em conta.
- Como um handler pode receber outros tipos de packets entre operações ACTION_EXAMINE_NEXT, a função ACTION_EXAMINE_NEXT precisa lidar com todos os casos especiais que possam resultar deste fato.
- O lock passado para ACTION_EXAMINE_NEXT não é sempre o mesmo lock usado nas operações anteriores, embora seja necessariamente um lock sobre o mesmo objeto.

Por causa desses problemas, ACTION_EXAMINE_NEXT é provavelmente a ação mais complicada de ser desenvolvida para qualquer handler. A negligência em lidar com qualquer uma das circunstâncias descritas acima pode ter resultados desastrosos.

```
ACTION_CREATE_DIR          22    CreateDir(...)
ARG1:    LOCK    Lock ao qual ARG2 é relativo.
ARG2:    BSTR    Nome do novo diretório (relativo a ARG1).
RES1:    LOCK    Lock sobre o novo diretório.
RES2:    CODE    Código de erro se RES1 = DOSFALSE.

ACTION_DELETE_OBJECT       16    DeleteFile(...)
ARG1:    LOCK    Lock ao qual ARG2 é relativo.
ARG2:    BSTR    Nome do objeto a ser deletado (relativo a ARG1).
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.

ACTION_RENAME_OBJECT       17    Rename(...)
ARG1:    LOCK    Lock ao qual ARG2 é relativo.
ARG2:    BSTR    Nome do objeto a ser renomeado (relativo a ARG1).
ARG3:    LOCK    Lock associado com o diretório alvo.
ARG4:    BSTR    Novo nome requisitado para o objeto.
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.
```

Estas três ações executam a maior parte do trabalho por trás dos comandos do AmigaDOS MAKEDIR, DELETE e RENAME (para um só arquivo). Estes packets

recebem como parâmetros um lock descrevendo onde o arquivo está localizado e um nome relativo a esse lock. É de responsabilidade do filesystem assegurar que a operação não cause efeitos adversos. A ação RENAME_OBJECT, em particular, permite mover arquivos através de um diretório, e como tal é preciso se certificar de que ela não criará loops de diretório ocultos renomeando um diretório como filho de si mesmo.

Para objetos Directory, a ação DELETE_OBJECT necessita da garantia de que o diretório esteja vazio antes permitir a execução da operação.

```
ACTION_PARENT     29    Parent(...)
ARG1:   LOCK    Lock sobre o objeto do qual deverá ser obtido o pai.
RES1:   LOCK    Lock parent
RES2:   CODE    Código de erro se RES1 = 0.
```

Esta ação recebe um lock sobre um objeto e cria um lock compartilhado sobre o pai do objeto. Se o objeto original não tiver pai, será retornado um lock igual a zero. Observe que esta operação é usada tipicamente no processo de construção da descrição do path absoluto de um determinado objeto.

```
ACTION_SET_PROTECT    21    SetProtection(...)
ARG1:   Não usado
aRG2:   LOCK    Lock ao qual ARG3 é relativo.
ARG3:   BSTR    Nome do objeto (relativo a ARG2).
ARG4:   LONG    Máscara dos novos bits de proteção.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta ação permite a uma aplicação modificar os bits de proteção de um objeto. Os 4 bits inferiores [*read, write, execute, delete* (RWED)] são um tanto peculiares: se o seu respectivo bit estiver setado, a operação correspondente *não* será permitida (isto é, se o bit *delete* de um arquivo estiver setado, o arquivo não poderá ser deletado). Por default, os arquivos são criados com os bits RWED setados e todos os outros bits ressetados. Além disso, qualquer operação que modifique um arquivo deverá zerar o bit A (*archive*) do mesmo. *Veja* o arquivo include dos/dos.h para obter as definições dos campos de bits.

```
ACTION_SET_COMMENT    28    SetComment(...)
ARG1:   Não usado
ARG2:   LOCK    Lock ao qual ARG3 é relativo.
ARG3:   BSTR    Nome do objeto (relativo a ARG2).
ARG4:   BSTR    Novo string de comentários.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta ação permite à aplicação definir o string de comentários de um objeto. Se o objeto não existir, RES1 retornará DOSFALSE e um código de erro será retornado em RES2. O string de comentários é limitado a 79 caracteres.

```
ACTION_SET_DATE    34    SetFileDate(...)
ARG1:   LOCK    Objeto
ARG2:   BPTR    DateStamp
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
```

```
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta ação permite a uma aplicação definir a data de criação de um objeto.

```
Somente 2.0
ACTION_FH_FROM_LOCK     1026    OpenFromLock(lock)
ARG1:   BPTR    BPTR para o filehandle.
ARG2:   PTR     Lock sobre o arquivo a ser aberto.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE)
RES2:   CODE    Código de erro se RES1 = NULL.
```

Esta ação abre um arquivo especificado por um lock. Se a ação for bem sucedida, o filesystem essencialmente "rouba" o lock, de maneira que um programa não deverá usar esse lock novamente. Se a execução de ACTION_FH_FROM_LOCK falhar, o lock ainda poderá ser usado pela aplicação.

```
Somente 2.0
ACTION_SAME_LOCK    40    SameLock(lock1,lock2)
ARG1:   BPTR    Lock 1 a comparar.
ARG2:   BPTR    Lock 2 a comparar.
RES1:   LONG    Resultado da comparação: LOCK_SAME (0) se os
                locks estiverem sobre o mesmo objeto,
                LOCK_SAME_HANDLER (1) se os locks estiverem sobre
                objetos diferentes do mesmo handler, ou
                LOCK_DIFFERENT (-1) em caso contrário.
RES2:   CODE    Código de erro se RES1 = LOCK_DIFFERENT.
```

Esta ação compara os alvos de dois locks. Se eles apontarem para o mesmo objeto, ACTION_SAME_LOCK deverá retornar LOCK_SAME.

```
Somente 2.0
ACTION_MAKE_LINK     1021    MakeLink(name,targ,mode)
ARG1:   BPTR    Lock sobre o diretório ao qual LOCK2 é relativo.
ARG2:   BSTR    Nome do link a ser criado (relativo a ARG1).
ARG3:   BPTR    Lock sobre o objeto alvo do nome (para soft links).
ARG4:   LONG    Modo do link: LINK_SOFT ou LINK_HARD.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Este packet faz com que o filesytem crie um link para um arquivo ou diretório já existente. Existem dois tipos de links: os hard links e os soft links. A diferença básica é que, no primeiro caso, o filesystem resolve o link por si mesmo, enquanto que no segundo caso, o filesystem passa um string de volta para o DOS dizendo onde encontrar o arquivo ou diretório relacionado com o soft link. Para quem programa a nível de packet, não há essencialmente nenhuma diferença entre referenciar um arquivo através de seu nome original ou por meio de seu nome de hard link. No caso dos hard links, ARG3 é um lock sobre o arquivo ou diretório ao qual o link é associado, para os soft links, ARG3 é um ponteiro para um string estilo C.

Num modelo bastante simplificado do filesystem da ROM, ao ser solicitado a localizar um arquivo o sistema inspeciona o disco à procura de um cabeçalho de arquivo (*file header*) com um nome de arquivo específico. Este cabeçalho aponta para os dados presentes no arquivo em algum lugar do disco. Se forem usados hard links, mais de um *file header* poderá apontar para os mesmos dados do arquivo. Assim, os

dados podem ser referenciados por mais de um nome. Quando o usuário tenta deletar um hard link de um arquivo, o sistema verifica se existe algum outro hard link para o mesmo arquivo. Caso positivo, apenas o hard link é deletado e os dados a que o hard link se referia permanecem inalterados, a fim de que os hard links existentes ainda possam usá-los. Caso o link original (não um hard ou soft link) de um arquivo seja deletado, o filesystem transformará um dos hard links no novo link "real" para o arquivo. Os hard links também podem existir em diretórios. Uma vez que os hard links "ligam" diretamente a mídia subjacente, hard links de um filesystem não podem referenciar objetos de um outro filesystem.

Os soft links são resolvidos através de chamadas do DOS. Ao inspecionar um disco à procura de um nome de arquivo ou diretório e descobrir que esse nome é um soft link, o filesystem retorna um código de erro (ERROR_IS_SOFT_LINK). Quando isto ocorre, cabe à aplicação indagar ao filesystem a que o link se refere, chamando para isso ACTION_READ_LINK. Os soft links são armazenados na própria mídia, mas em vez de apontar diretamente para os dados no disco, um soft link contém um path para o seu objeto. Este path pode ser relativo ao lock em ARG1, relativo ao volume (onde o string será prefaciado por um ":") ou ainda um path absoluto. O path absoluto contém o nome de um outro volume, de maneira que um soft link pode referenciar arquivos e diretórios em outros discos.

Somente 2.0

ACTION_READ_LINK 1024 ReadLink(port,lock,name,buf,len)

ARG1:	BPTR	Lock sobre o diretório ao qual ARG2 é relativo.
ARG2:	CPTR	Path e nome do link (relativo a ARG1). **Nota:** Este é um string C, e não um BSTR.
ARG3:	APTR	Buffer para o novo string do path.
ARG4:	LONG	Tamanho do buffer, em bytes.
RES1:	LONG	Tamanho efetivo do string retornado, -2 se não houver espaço suficiente no buffer, ou -1 para indicar outros erros.
RES2:	CODE	Código de erro.

Esta ação lê um link e retorna uma descrição de path para o objeto do link. O nome do link (mais o path, quando necessário) é passado como um ponteiro CPTR (ARG2) que aponta para um string estilo C, ao invés de um string BSTR. Este é um caso excepcional. Somente o packet ACTION_READ_LINK usa um ponteiro CPTR ao invés de um string BSTR. ACTION_READ_LINK retorna a descrição do path em ARG3. O tamanho do string alvo é retornado em RES1 (ou -1, indicando um erro).

ACTION_CHANGE_MODE 1028 ChangeMode(type,obj,mode)

ARG1:	LONG	Tipo do objeto a ser modificado: CHANGE_FH ou CHANGE_LOCK.
ARG2:	BPTR	Objeto a ser modificado.
ARG3:	LONG	Novo modo para o objeto. *Veja* ACTION_FINDINPUT e ACTION_LOCATE_OBJECT.
RES1:	BOOL	Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:	CODE	Código de erro se RES1 = DOSFALSE.

Esta ação solicita que o handler modifique o modo do lock ou filehandle especificado para o modo indicado por ARG3. Esta requisição deve falhar quando o handler não puder modificar o modo tal como foi solicitado (no caso de uma requisição exclusiva

para um objeto que possui múltiplos usuários, por exemplo).

```
Somente 2.0
ACTION_COPY_DIR_FH     1030     DupFromLock(fh)
ARG1:    BPTR     Filehandle
RES1:    BPTR     Lock associado com o filehandle, ou NULL.
RES2:    CODE     Código de erro se RES1 = NULL.
```

Esta ação solicita ao handler que retorne um lock associado com o filehandle atualmente aberto. A requisição pode falhar em função de restrições impostas pelo filesystem (quando o filehandle não estiver aberto num modo compartilhado, por exemplo). O filehandle ainda poderá ser utilizado após esta chamada, ao contrário do que acontece com o lock de ACTION_FH_FROM_LOCK.

```
Somente 2.0
ACTION_PARENT_FH     1031     ParentOfFH(fh)
ARG1:    BPTR     Filehandle é fh_Arg1.
RES1:    BPTR     Lock sobre o pai do filehandle.
RES2:    CODE     Código de erro se RES1 = NULL.
```

Esta ação obtém um lock sobre o diretório pai (ou raiz do volume, em se tratando do nível superior) de um filehandle atualmente aberto. O lock é retornado no modo compartilhado e deverá ser posteriormente liberado. Note que ao contrário do que acontece com ACTION_COPY_DIR_FH, o modo do filehandle não é importante. Para um arquivo aberto, ACTION_PARENT_FH deverá retornar um lock em qualquer circunstância.

```
Somente 2.0
ACTION_EXAMINE_ALL     1033     ExAll(lock,buf,size,type,ctl)
ARG1:    BPTR     Lock sobre o diretório a ser examinado.
ARG2:    APTR     Buffer para armazenar os resultados.
ARG3:    LONG     Tamanho (em bytes) do buffer (ARG2).
ARG4:    LONG     Tipo da requisição:
                  ED_NAME: Retorna apenas os nomes dos arquivos.
                  ED_TYPE: Retorna a informação acima mais o tipo
                    dos arquivos.
                  ED_SIZE: Retorna as informações acima mais o
                    tamanho dos arquivos.
                  ED_PROTECTION: Retorna as informações acima mais
                    a proteção dos arquivos.
                  ED_DATE: Retorna as informações acima mais 3
                    palavras longas representando a data.
                  ED_COMMENT: Retorna as informações acima mais
                    os comentários do arquivo, ou NULL.
ARG5:    BPTR     Estrutura de controle para armazenar as
                  informações referentes ao estado. Precisa ser
                  alocada com AllocDOSObject( )!
RES1:    LONG     Flag de continuação, DOSFALSE indica terminação.
RES2:    CODE     Código de erro se RES1 = DOSFALSE
```

Esta ação permite a uma aplicação obter informações sobre múltiplas entradas de diretório. Ela é particularmente útil para aplicações que necessitam obter informações sobre um grande número de arquivos e diretórios.

Esta ação preenche o buffer (ARG2) com estruturas ExAllData parciais ou completas. O tamanho da estrutura ExAllData dependerá do tipo da requisição. Se o valor no campo que identifica o tipo da requisição (ARG4) for ED_NAME, apenas o campo ed_Name será preenchido. Em vez de copiar os campos não utilizados da estrutura ExAllData no buffer, ACTION_EXAMINE_ALL trunca os campos não usados. Este efeito é cumulativo, de maneira que requisições para preencher os outros campos na estrutura ExAllData fazem com que todos os campos que aparecem na estrutura antes do campo requisitado sejam também preenchidos. Como no caso de ED_NAME mencionado acima, qualquer campo que apareça após o campo requisitado será truncado (veja a estrutura ExAllData abaixo).

Como exemplo, se o campo requisitado for definido como ED_COMMENT, ACTION_EXAMINE_ALL preencherá todos os campos da estrutura ExAllData, porque ed_Comment é o último campo. Este é o único caso em que o packet retorna uma estrutura ExAllData completa.

```
struct ExAllData {
     struct ExAllData *ed_Next;
     UBYTE *ed_Name;
     LONG   ed_Type;
     ULONG  ed_Prot;
     ULONG  ed_Days;
     ULONG  ed_Mins;
     ULONG  ed_Ticks;
     UBYTE *ed_Comment;
          /* os strings serão posicionados após o último campo usado */
};
```

Cada entrada da estrutura ExAllData possui um campo ead_Next que aponta para a próxima estrutura ExAllData. Utilizando essas ligações, um programa pode facilmente examinar as estruturas ExAllData em seqüência, sem precisar se preocupar com o tamanho real da estrutura. Não examine os campos existentes além daqueles requisitados, uma vez que estes campos certamente não terão sido inicializados (e provalvelmente irão sobrepor a próxima entrada).

A parte mais importante desta ação é a estrutura ExAllControl. Ela precisa ser alocada e liberada por meio das funções AllocDosObject( )/FreeDosObject( ). Isto permite que a estrutura seja ampliada, se necessário, em futuras revisões do sistema operacional e dos filesystems. Por ora, a estrutura ExAllControl possui quatro campos.

Entries: Este campo é mantido pelo filesystem e indica o número de entradas efetivamente presentes no buffer depois que a ação é completada. Note que um valor igual a zero é válido, pois é possível que nenhuma entrada coincida com o string de combinação de padrões.

LastKey: Este campo precisa ser inicializado pela aplicação com o valor zero antes deste packet ser usado pela primeira vez. O campo é mantido pelo filesystem como um indicador de estado da posição atual na lista de entradas a ser examinada. O filesystem pode testar este campo para determinar se esta é a primeira chamada ou uma chamada subseqüente a esta ação.

MatchString: Este campo aponta para um string de combinação de padrões que irá controlar que entradas no diretório deverão ser retornadas. Se este campo for NULL, todas as entradas serão retornadas. Caso contrário, o string será usado para selecionar as entradas que combinem com os padrões especificados antes de serem copiadas no buffer. A rotina default de comparação de padrões do AmigaDOS será utilizada, a menos que MatchFunc seja NULL (veja abaixo). Note que não é permitido à aplicação modificar este campo entre chamadas subseqüentes a esta ação para um mesmo diretório.

MatchFunc: Este campo contém um ponteiro para uma rotina de comparação de padrões alternativa para validar as entradas. Se este campo for NULL, a rotina padrão do Amiga para comparação de padrões será usada. Do contrário, MatchFunc deverá apontar para o vetor de uma função, que é chamada da seguinte maneira:

```
BOOL = MatchFunc(hookptr, data, typeptr)
                   A0      A1      A2
hookptr     Ponteiro para o vetor sendo chamado.
data        Ponteiro para estrutura ExAllData (parcialmente)
            preenchida para o item sendo examinado.
typeptr     Ponteiro para a palavra longa indicando o tipo da
            requisição ExAll (ARG4).
```

Esta função deverá retornar DOSTRUE se a entrada for aceita, ou DOSFALSE caso a mesma deva ser descartada.

Somente 2.0

```
ACTION_EXAMINE_FH     1034    ExamineFH(fh,fib)
ARG1:    BPTR    Filehandle do arquivo aberto.
ARG2:    BPTR    FileInfoBlock a ser preenchido.
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.
```

Esta função examina um filehandle e preenche a estrutura FileInfoBlock (encontrada em ARG2) com informações sobre o estado atual do arquivo. Esta rotina é análoga à ação ACTION_EXAMINE_OBJECT no que diz respeito aos locks. Como nem sempre é possível fornecer o tamanho exato do arquivo (quando os buffers não foram descarregados ou dois processos gravam dados no mesmo arquivo, por exemplo), o valor do campo fib_Size (veja o arquivo dos/dos.h) pode ser impreciso.

Somente 2.0

```
ACTION_ADD_NOTIFY     4097    StartNotify(NotifyRequest)
ARG1:    BPTR    Estrutura NotifyRequest
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.
```

Esta ação solicita a um filesystem que notifique o programa que chama quando um arquivo particular for alterado. O filesystem notifica o programa enviando uma mensagem, ou então atribuindo um sinal a uma tarefa.

```
struct NotifyRequest {
        UBYTE *nr_Name;
        UBYTE *nr_FullName; /* definido pelo DOS - não mexa */
```

```
        ULONG nr_UserData;  /* para uso da aplicação */
        ULONG nr_Flags;
        union {
          struct {
            struct MsgPort *nr_Port; /* para SEND_MESSAGE */
          } nr_Msg;
          struct {
            struct Task *nr_Task;     /* para SEND_SIGNAL */
            UBYTE nr_SignalNum;       /* para SEND_SIGNAL */
            UBYTE nr_pad[3];
          } nr_Signal;
        } nr_stuff;
        ULONG nr_Reserved[4];         /* deixe em 0  por enquanto */
                                      /* uso interno dos handlers */
        ULONG nr_MsgCount;            /* nº de msgs a resolver */
        struct MsgPort *nr_Handler;   /* handler destinatário (para
                                         EndNotify) */
};
```

Para usar este packet, uma aplicação deverá alocar e inicializar uma estrutura NotifyRequest (veja acima). Até o término da elaboração deste livro, as estruturas NotifyRequest não eram alocadas pela função AllocDosObject( ), mas isto pode mudar no futuro. O handler obtém o nome do arquivo vigiado a partir do campo nr_FullName. O filesystem atual não suporta curingas neste campo, embora nada impeça que outros handlers sejam capazes de fazê-lo.

O string em nr_FullName precisa ser um path absoluto, incluindo o nome do volume raiz (associações não são permitidas). O path absoluto é necessário porque o arquivo ou seus diretórios pais não precisam existir no momento em que a notificação é solicitada. Isto permite a execução de notificação sobre arquivos em diretórios que ainda não existem (a notificação não ocorrerá até que os diretórios e o arquivo sejam criados).

As aplicações que usam a chamada StartNotify( ) do DOS não preenchem o campo nr_FulllName da estrutura NotifyRequest, mas sim o campo nr_Name. StartNotify( ) obtém o nome no campo nr_Name e utiliza as funções GetDeviceProc( ) e NameFromLock( ) para expandir quaisquer associações (tais como ENV:), armazenando o resultado em nr_FullName. Qualquer aplicação que utilize uma interface a nível de packet em vez de StartNotify( ) precisará ter a capacidade de expandir suas próprias associações, além disso, os handlers não deverão contar com que nr_Name esteja correto.

O tipo da notificação depende do bit que estiver setado no campo nr_Flags da estrutura NotifyRequest. Se o bit NRF_SEND_MESSAGE estiver setado, a aplicação receberá a notificação das mudanças no arquivo através de uma mensagem (veja NotifyMessage no arquivo dos/notify.h). Neste caso, o campo nr_Port precisa apontar para porta de mensagens que irá receber a mensagem de notificação. Se o campo NRF_SEND_SIGNAL de nr_Flags estiver setado, o filesystem atribuirá um sinal a uma tarefa ao invés de enviar uma mensagem. Nesse caso, nr_Task aponta para a tarefa e nr_SignalNum corresponde ao número do sinal. Lembre-se que somente um desses bits deverá ser setado.

Se um programa setar o bit NRF_WAIT_REPLY, o handler precisará esperar para enviar notificações pendentes até que todas as notificações já enviadas tenham retornado. Ao receber uma requisição de notificação com o bit NRF_NOTIFY_SIGNAL setado, o handler enviará uma mensagem inicial ou atribuirá um sinal inicial se o arquivo vigiado já existir.

Os handlers deverão efetuar uma notificação somente quando o conteúdo do arquivo tiver sido efetivamente alterado. Isto compreende ações como ACTION_WRITE, ACTION_TRUNCATE, ACTION_SET_DATE, ACTION_DELETE, ACTION_RENAME, ACTION_FINDUPDATE, ACTION_FINDINPUT e ACTION_FINDOUTPUT. Outras ações como ACTION_SET, ACTION_COMMENT ou ACTION_SET_PROTECT também podem ser levadas em conta, mas isto não é obrigatório (e pode não ser esperado pela aplicação, uma vez que nesses casos não há necessidade de se ler novamente os dados).

```
Somente 2.0
ACTION_REMOVE_NOTIFY    4098    EndNotify(NotifyRequest)
ARG1:   BPTR    Ponteiro para estrutura NotifyRequest
                previamente alocada.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Esta ação cancela uma notificação (veja ACTION_ADD_NOTIFY). ARG1 é a estrutura NotifyRequest utilizada para iniciar a notificação. O handler deverá abandonar quaisquer mensagens de notificação pendentes. Note que é possível para um filesystem receber uma resposta de uma mensagem de notificação previamente enviada mesmo após a notificação ter sido terminada. Ele deve aceitar essas mensagens silenciosamente e descartá-las.

Informação/Manipulação de Volumes

As ações de Informação e Manipulação de Volumes são usadas para permitir o acesso ao volume que está sendo manipulado pelo filesystem num determinado momento.

```
ACTION_CURRENT_VOLUME    7    <somente sendpkt>
RES1:   BPTR    Ponteiro para o nó de volume do volume atual.
```

Esta ação retorna um ponteiro para o nó do volume (da lista de dispositivos do DOS) associado com o filesystem. Como o nó do volume pode ser removido da lista de dispositivos quando o filesystem monta um volume diferente (tal como quando determinado por uma ação ACTION_INHIBIT), não é possível garantir por quanto tempo este ponteiro permanecerá válido. Esta ação é geralmente usada pelo AmigaDOS para exibir a linha do volume em um requester.

```
ACTION_DISK_INFO    25    Info(...)
ARG1:   BPTR    Ponteiro para uma estrutura InfoData a ser
                preenchida.
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).

ACTION_INFO    26    <somente sendpkt>
ARG1:   LOCK    Lock
ARG2:   BPTR    Ponteiro para uma estrutura InfoData a ser
```

```
                preenchida.
RES1:    BOOL   Sucesso/falha (DOSTRUE/DOSFALSE).
```

Estas ações são usadas com o propósito de se obter informações sobre o dispositivo e o status do processo de manipulação de arquivos. A ação ACTION_DISK_INFO é usada pelo comando INFO do AmigaDOS para reportar o status do volume inserido no drive, ela preenche uma estrutura InfoData com o volume controlado pelo filesystem no momento. A ação ACTION_INFO preenche a estrutura InfoData com o volume sobre o qual o lock (ARG1) está ativo. Estas ações geralmente devem retornar DOSTRUE.

O packet ACTION_DISK_INFO tem um significado especial para os processos de manipulação de console: ao receber este packet, um handler de console deve retornar um ponteiro para a janela associada com o handle aberto.

```
ACTION_RENAME_DISK    9    Relabel(...) na versão 2.0
ARG1:    BSTR   Novo nome do disco.
RES1:    BOOL   Sucesso/falha (DOSTRUE/DOSFALSE).
```

Esta ação permite a uma aplicação mudar o nome do volume atual. Um filesystem que implemente esta função deverá também alterar o nome armazenado no nó de volume da lista de dispositivos do DOS.

```
Somente 2.0
ACTION_FORMAT    1020    Format(fs,vol,type)
ARG1:    BSTR   Nome do dispositivo (com ":" no final).
ARG2:    BSTR   Nome do volume (quando suportado).
ARG3:    LONG   Tipo da formatação (específica de filesystem).
RES1:    BOOL   Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE   Código de erro se RES1 = DOSFALSE.
```

Este packet orienta um filesystem a executar uma formatação de dispositivo (ou específica de filesystem) sobre uma mídia recém-inicializada. Ao receber esta ação, um filesystem pode assumir que a mídia já sofreu uma formatação de baixo nível, devendo proceder à gravação da estrutura de disco de alto nível necessária para criar um volume vazio.

Manutenção e Controle do Handler

Diversos packets são definidos com a finalidade de proporcionar à aplicação algum controle sobre um filesystem.

```
ACTION_DIE       5    <somente sendpkt>
RES1:    BOOL   DOSTRUE
```

Como o nome sugere, este packet solicita o término de um handler. Todos os novos handlers deverão implementar este packet. Devido aos locks pendentes e ao fato de o endereço do handler ser retornado pela rotina DeviceProc( ), é improvável que o handler possa desaparecer por completo. Mas de qualquer modo espera-se que esse handler libere tantos recursos quanto for possível, e simplesmente retorne um erro a todos os packets enviados para ele.

No futuro, o sistema pode ser capaz de determinar a existência de referências

DeviceProc( ) a serem resolvidas para um handler, e assim tornar seguro encerrá-lo completamente.

```
ACTION_FLUSH      27    <somente sendpkt>
RES1:    BOOL     DOSTRUE
```

Esta ação faz com que o filesystem descarregue todos os buffers no disco antes de retornar este packet. Quaisquer operações de gravação pendentes deverão ser processadas antes de se responder a este packet. A ação ACTION_FLUSH permite que uma aplicação se certifique de que os dados que deveriam estar no disco foram efetivamente gravados nesse disco, ao invés de se encontrarem aguardando em um buffer.

```
ACTION_MORE_CACHE     18    AddBuffers(...) na versão 2.0
ARG1:    LONG     Número de buffers.
RES1:    BOOL     DOSTRUE
RES2:    LONG     Novo número de buffers.
```

Esta ação permite a uma aplicação modificar o número de buffers internos de memória cache utilizados pelo filesystem. Note que um número positivo aumenta o número de buffers, enquanto que um número negativo diminui a quantidade de buffers disponíveis. Em ambos os casos, o número de buffers efetivamente disponíveis é retornado em RES2. Isto permite a uma aplicação inquirir o número de buffers bastando enviar um valor igual a zero (o que resultará em nenhuma mudança). Observe que o Old Filing System e o Fast Filing System na versão 1.3 do AmigaDOS não aceitam um número negativo de buffers.

```
ACTION_INHIBIT     31    Inhibit(...) na versão 2.0
ARG1:    BOOL     DOSTRUE = inibe, DOSFALSE = desinibe.
RES1:    BOOL     Sucesso/falha (DOSTRUE/DOSFALSE).
```

Esta ação é provavelmente uma das mais perigosas com a qual um filesystem necessita lidar. Quando inibido (ARG1 = DOSTRUE), o filesystem deverá não acessar nenhuma mídia subjacente, e retornar um código erro diante de qualquer tentativa de acesso a um dispositivo. Uma vez desinibido (ARG1 = DOSFALSE), o filesystem precisa assumir que a mídia foi alterada. O filesystem deverá descarregar os buffers antes de ACTION_INHIBIT, exibindo na tela uma mensagem demandando que o usuário reinsira o disco, caso necessário. O handler pode decidir rejeitar uma requisição de inibição caso algum objeto esteja aberto para gravação.

Embora isto não seja obrigatório, um handler deve ser capaz de aninhar as inibições. Antes da versão 2.0 do AmigaDOS, os handlers do sistema não mantinham uma contagem das inibições aninhadas e estavam sujeitos a condições obscuras de operação. O sistema de arquivo da ROM 2.0 introduziu a contagem do aninhamemto.

```
ACTION_WRITE_PROTECT     1023    <somente sendpkt>
ARG1:    BOOL     DOSTRUE/DOSFALSE (protege/desprotege contra
                  gravação).
ARG2:    LONG     Senha de 32 bits.
RES1:    BOOL     DOSTRUE/DOSFALSE
```

Este é um novo packet definido para o Fast Filing System. Este packet permite a uma aplicação mudar o estado do flag de proteção contra gravação de um disco (quando possível, uma aplicação não pode gravar dados em um disquete quando a

pastilha de proteção de plástico do mesmo estiver levantada). Este packet foi criado com o objetivo de permitir a proteção contra gravação em mídias não removíveis, tais como discos rígidos. O valor em ARG1 alterna o status de gravação. A senha de 32 bits permite a um programa evitar que outros programas desabilitem a proteção contra gravação no disco. Para desbloquear um disco, é necessário que o valor de ARG2 coincida com a senha do packet que bloqueou o disco, a menos que o disco tenha sido bloqueado com uma senha igual a zero. Neste caso, nenhuma senha será necessária para desbloquear o disco.

Somente 2.0

```
ACTION_IS_FILESYSTEM    1027    IsFileSystem(devname)
RES1:   BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:   CODE    Código de erro se RES1 = DOSFALSE.
```

Através desta função, um handler pode indicar se é ou não um filesystem (ou seja, se é ou não capaz de suportar o armazenamento de informações em arquivos distintos). Os programas assumirão que um handler é capaz de criar múltiplos arquivos através de chamadas a Open( ) se o handler retornar este packet com um valor DOSTRUE. Um handler não precisa suportar diretórios e subdiretórios para ser qualificado como um filesystem. Ele também não precisa suportar chamadas Examine( )/ExNext( ).

Note que a rotina IsFileSystem( ) do AmigaDOS tentará usar o lock (":",SHARED_ACCESS) caso este packet retorne ERROR_ACTION_NOT_KNOWN.

Ações Internas do Handler

Existem diversas ações que são usadas de modo genérico pelos handlers com o objetivo de permitir que mensagens que retornam de serviços solicitados (tipicamente a um dispositivo Exec) se pareçam com packets de requisição. Isto permite ao handler requisitar que uma operação assíncrona seja notificada ao ser completada. Como exemplo, um handler pode enviar para o dispositivo serial.device uma requisição de leitura, só que em vez de mandar uma requisição de I/O normal, ele envia um packet do DOS disfarçado de requisição de I/O. O dispositivo serial.device trata o packet como uma requisição de I/O normal, retornando-o para o handler assim que tiver terminado. Quando o handler recebe de volta seu packet do DOS disfarçado, ele sabe que a leitura foi completada.

```
ACTION_NIL              0     <interno>
```

Embora não se trate especificamente de uma ação, muitos retornos se assemelham a este valor porque o campo da ação não foi preenchido.

```
ACTION_READ_RETURN      1001    <interno>
```

Este retorno é geralmente usado para indicar a conclusão de uma requisição de leitura assíncrona.

```
ACTION_WRITE_RETURN     1002    <interno>
```

Este retorno é geralmente usado para indicar a conclusão de uma requisição de gravação assíncrona.

```
ACTION_TIMER            30      <interno>
```

Este retorno é usado para indicar a passagem de um intervalo de tempo. Muitos handlers têm um stream fixo de packets ACTION_TIMER de maneira a programar operações de manutenção interna e descarregar os buffers quando nenhuma atividade tiver ocorrido dentro de um determinado intervalo de tempo.

Packets Obsoletos

Existem vários tipos de packets documentados no sistema que incluem arquivos que estão obsoletos. Um filesystem não é obrigado a manipular esses packets, e qualquer programa que envie esses packets não deverá esperar que eles funcionem:

```
ACTION_DISK_CHANGE      33    <Obsoleto>
ACTION_DISK_TYPE        32    <Obsoleto>
ACTION_EVENT             6    <Obsoleto>
ACTION_GET_BLOCK         2    <Obsoleto>
ACTION_SET_MAP           4    <Obsoleto>
```

Um caso a chamar a atenção é o de ACTION_DISK_CHANGE. O comando DISKCHANGE do AmigaDOS utiliza o packet ACTION_INHIBIT para efetuar seu trabalho.

Packets Somente de Console

Os packets restantes são usados apenas pelos handlers de console, e não precisam ser implementados por um filesystem.

```
ACTION_SCREEN_MODE     994     <somente sendpkt>
ARG1:    LONG    Modo (0 ou 1)
RES1:    BOOL    Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE    Código de erro se RES1 = DOSFALSE.
```

Modifica o modo de operação do console de *raw* para *cooked* e vice-versa. ARG1 = 1 indica o modo não processado (*raw*) e ARG1 = 0 indica o modo processado (*cooked*).

```
ACTION_WAIT_CHAR     20     WaitForChar( )
ARG1:    ULONG     Tempo de espera em microssegundos.
RES1:    BOOL     Sucesso/falha (DOSTRUE/DOSFALSE).
RES2:    CODE     Código de erro se RES1 = DOSFALSE.
```

Efetua uma leitura temporizada de um caractere. A função WaitForChar( ) do AmigaDOS utiliza este packet.

Sumário dos Números de Packets Definidos

A lista a seguir apresenta todos os packets do DOS definidos pelo Commodore. Os packets 0-2049 são reservados para uso do Commodore. A menos que especificado em contrário, os packets 2050-2999 são reservados para o uso de fabricantes independentes (veja a tabela abaixo). Os packets restantes são reservados para futura expansão (Nota: os packets 2008, 2009, 4097 e 4098 são usados pelo Commodore).

Decimal	Hexa decimal	Ação
0	0x0000	ACTION_NIL
1		<Reservado pelo Commodore>
2	0x0002	ACTION_GET_BLOCK
3		<Reservado pelo Commodore>
4	0x0004	ACTION_SET_MAP
5	0x0005	ACTION_DIE
6	0x0006	ACTION_EVENT
7	0x0007	ACTION_CURRENT_VOLUME
8	0x0008	ACTION_LOCATE_OBJECT
9	0x0009	ACTION_RENAME_DISK
10-14		<Reservado pelo Commodore>
15	0x000F	ACTION_FREE_LOCK
16	0x0010	ACTION_DELETE_OBJECT
17	0x0011	ACTION_RENAME_OBJECT
18	0x0012	ACTION_MORE_CACHE
19	0x0013	ACTION_COPY_DIR
20	0x0014	ACTION_WAIT_CHAR
21	0x0015	ACTION_SET_PROTECT
22	0x0016	ACTION_CREATE_DIR
23	0x0017	ACTION_EXAMINE_OBJECT
24	0x0018	ACTION_EXAMINE_NEXT
25	0x0019	ACTION_DISK_INFO
26	0x001A	ACTION_INFO
27	0x001B	ACTION_FLUSH
28	0x001C	ACTION_SET_COMMENT
29	0x001D	ACTION_PARENT
30	0x001E	ACTION_TIMER
31	0x001F	ACTION_INHIBIT
32	0x0020	ACTION_DISK_TYPE
33	0x0021	ACTION_DISK_CHANGE
34	0x0022	ACTION_SET_DATE

35-39		<Reservado pelo Commodore>
40	0x0028	ACTION_SAME_LOCK
41-48		<Reservado pelo Commodore>
82	0x0052	ACTION_READ
83-86		<Reservado pelo Commodore>
87	0x0057	ACTION_WRITE
88-993		<Reservado pelo Commodore>
994	0x03E2	ACTION_SCREEN_MODE
995-1000		<Reservado pelo Commodore>
1001	0x03E9	ACTION_READ_RETURN
1002	0x03EA	ACTION_WRITE_RETURN
1003		<Reservado pelo Commodore>
1004	0x03EC	ACTION_FIND_UPDATE
1005	0x03ED	ACTION_FINDINPUT
1006	0x03EE	ACTION_FINDOUTPUT
1007	0x03EF	ACTION_END
1008	0x03F0	ACTION_SEEK
1009-1019		<Reservado pelo Commodore>
1020	0x03FC	ACTION_FORMAT
1021	0x03FD	ACTION_MAKE_LINK
1022	0x03FE	ACTION_SET_FILE_SIZE
1023	0x03FF	ACTION_WRITE_PROTECT
1024	0x0400	ACTION_READ_LINK
1025		<Reservado pelo Commodore>
1026	0x0402	ACTION_FH_FROM_LOCK
1027	0x0403	ACTION_IS_FILESYSTEM
1028	0x0404	ACTION_CHANGE_MODE
1029		<Reservado pelo Commodore>
1030	0x0406	ACTION_COPY_DIR_FH
1031	0x0407	ACTION_PARENT_FH
1032		<Reservado pelo Commodore>
1033	0x0409	ACTION_EXAMINE_ALL

1034	0x040A	ACTION_EXAMINE_FH
1035-2007		<Reservado pelo Commodore>
2008	0x07D8	ACTION_LOCK_RECORD
2009	0x07D9	ACTION_FREE_RECORD
2010-2049		<Reservado pelo Commodore>
2050-2999		<Reservado para Handlers de Outros Fabricantes>
4097	0x1001	ACTION_ADD_NOTIFY
4098	0x1002	ACTION_REMOVE_NOTIFY
4099-		<Reservado para Futura Expansão pelo Commodore>

Utilizando Packets Diretamente

O AmigaDOS dispõe de vários recursos que só podem ser acessados enviando-se um packet diretamente para um processo. O packet ACTION_DISK_INFO, por exemplo, pode ser utilizado para localizar o ponteiro de janela Intuition de uma janela CON: ou RAW:. Isto permite redirecionar os requesters do sistema de maneira que eles apareçam numa tela em que o usuário possa vê-los (veja o tópico "Redirecionando os Requesters do Sistema", mais acima). O ponteiro Window será retornado no campo ID_VolumeNote, e um ponteiro para o request de I/O do console será retornado no campo ID_InUse. Note que consoles auxiliares (AUX:) podem retornar um ponteiro Window NULL, além de eventualmente não possuírem uma unidade ConUnit (io_Unit) associada com seu bloco de requisição de I/O. Tenha sempre o cuidado de checar essa possibilidade quando você utilizar este packet. Se a sua aplicação roda numa janela CLI, lembre-se de que um usuário pode estar executando-a num CLI auxiliar (AUX:).

Outro exemplo é o packet ACTION_SCREENMODE_MODE, que pode ser enviado para o processo de manipulação de uma janela CON: a fim de posicionar o console no modo *raw* ou processado.

Por default, entrada de teclado proporcionada por CON: é filtrada, bufferizada e ecoada automaticamente na tela. Muitas das seqüências de escape das teclas especiais (como aquelas geradas pelas teclas de função, cursor e help) são filtradas, todos os toques no teclado são bufferizados e retidos sem visualização até que o usuário pressione a tecla Return. Os toques não filtrados (como as teclas alfanuméricas e backspace) são automaticamente ecoados na janela CON:. Este modo "processado" é perfeito para a entrada genérica de dados pelo usuário, já que oferece recursos automáticos de edição de linha (idênticos ao da linha de comandos do Shell).

Algumas vezes, entretanto, uma aplicação necessita obter de imediato as teclas individuais acionadas em uma janela CON:, ou então controlar seu ecoamento, ou ainda receber os strings de escape que o mapa do teclado gera para teclas especiais, como a tecla Help ou teclas de movimentação do cursor.

Neste caso, um packet ACTION_SCREEN_MODE com o argumento DOSTRUE

(-1L) pode ser enviado para a MsgPort de uma janela CON: para posicionar a janela CON: no modo *raw*. Neste modo, uma janela CON: atua de maneira semelhante a uma janela RAW:. A entrada através do teclado do console não é automaticamente filtrada, bufferizada ou ecoada na tela. Numa janela CON: que esteja em modo *raw*, cada toque no teclado pode ser lido imediatamente, obtendo-se seu valor ASCII ou o string para o qual a tecla é mapeada pelo mapa de configuração do teclado.

Em algumas aplicações, pode ser conveniente alternar janela CON: entre os modos processado e não processado (ou *cooked* e *raw*, em inglês), utilizando-se o modo *cooked* para processar a entrada das linhas pelo usuário, e o modo *raw* quando for necessário que os toques no teclado causem ações imediatas.

O packet ACTION_SCREEN_MODE com o argumento DOSFALSE (0L) posicionará uma janela CON: no modo *cooked*. Note que este packet pode ser também usado em consoles auxiliares (AUX:).

A porta de mensagens (MsgPort) dos processos de manipulação da maioria dos dispositivos do AmigaDOS que recebem nomes (tais como DF0:) pode ser localizada através da função DeviceProc( ). Note que DeviceProc( ) não pode ser usado para encontrar um handler CON: ou RAW: porque podem existir vários handlers para esses dispositivos. A MsgPort(ProcessID) do processo de manipulação de uma janela RAW: ou CON: está na sua estrutura FileHandle (fh_Type). A MsgPort de uma janela "*" de um processo do CLI é process->pr_ConsoleTask.

Eis como localizar a MsgPort de um handler (certifique-se sempre de que a porta não seja NULL, antes de usar estas rotinas):

1. Encontrando a MsgPort de um processo de manipulação de dispositivo com um nome exclusivo (tal como "DF0"):

```
port = (struct MsgPort *)DeviceProc("DF1:")
```

2. Encontrando a MsgPort do processo de manipulação de um arquivo aberto:

```
fh = Open("CON:0/40/640/Teste",MODE_NEWFILE);
if((fh)&&(fh->Type))
      /* Se a execução de Open for bem sucedida e fh_Type não for NULL */
{
            port = (struct MsgPort *) ((( struct
            FileHandle *)
            (fh<<2))->fh_Type);
}
```

3. Encontrando a MsgPort do handler do console de seu processo:

```
struct Task *task = FindTask(NULL);
if(task->tc_Node.ln_Type == NT_PROCESS)
{
      /* A porta pode ser NULL - verifique antes de usar! */
      port = ((struct Process *)task)->pr_ConsoleTask;
}
```

Os packets são enviados inicializando-se uma estrutura StandardPacket alinhada por palavra longa e enviando-se o packet para a porta de mensagens de um processo de manipulação de dispositivo.

A biblioteca dos.library do AmigaDOS 2.0 dispõe de novas funções simples para o envio e resposta de packets:

```
SendPkt( ) - envia de maneira assíncrona seu packet inicializado.
WaitPkt( ) - espera que um packet assíncrono seja completado.
ReplyPkt( ) - responde a um packet que foi enviado para você.
DoPkt( ) - cria e envia um packet, aguardando até que seja completado.
```

Consulte o Capítulo 6 ("Funções do AmigaDOS") para obter uma descrição completa dessas funções e de seus argumentos.

Se você precisar enviar packets de uma maneira compatível com a versão 1.3 do AmigaDOS, a função apresentada a seguir pode ser usada no lugar das funções do dos.library da versão 2.0.

```
#include <exec/types.h>
#include <exec/memory.h>
#include <libraries/DOS.h>
#include <libraries/DOSextens.h>
/*  dopkt( ) por A. Finkel, P. Lindsay, C. Scheppner
 *  Envia um packet de maneira compatível com a V1.3
 *  e aguarda até que a operação seja completada.
 *  Retorna o resultado Res1 do packet de resposta.
 */
LONG dopkt(pid,action,args,nargs)
struct MsgPort *pid;  /* identificador do processo       */
                      /* (porta de mensagens do handler) */
LONG action,          /* tipo do packet (ação desejada)  */
     args[ ],          /* ponteiro p/ uma lista de args   */
     nargs;           /* nº de argumentos na lista       */
    {
    struct MsgPort        *replyport;
    struct StandardPacket *packet;
    LONG  count, *pargs, res1;
    replyport = (struct MsgPort *) CreatePort(NULL,0);
    if(!replyport) return(NULL);
    packet = (struct StandardPacket *)
             AllocMem((long)sizeof(struct
    StandardPacket),MEMF_PUBLIC|MEMF_CLEAR);
    if(!packet)
       {
       DeletePort(replyport);
       return(NULL);
       }
    packet->sp_Msg.mn_Node.ln_Name = (char *)&(packet->sp_Pkt);
    packet->sp_Pkt.dp_Link         = &(packet->sp_Msg);
    packet->sp_Pkt.dp_Port         = replyport;
    packet->sp_Pkt.dp_Type         = action;
    /* copia os argumentos no packet */
    pargs = &(packet->sp_Pkt.dp_Arg1); /* endereço do 1º argumento */
    for(count=0;count < nargs;count ++)
       pargs[count]=args[count];
    PutMsg(pid,packet); /* envia o packet */
```

```
WaitPort(replyport);
GetMsg(replyport);
res1 = packet->sp_Pkt.dp_Res1;
FreeMem(packet,(long)sizeof(struct StandardPacket));
DeletePort(replyport);

return(res1);
}
```

Capítulo 12

Informações Adicionais para o Programador Avançado

Este capítulo descreve alguns tópicos de utilidade para o programador avançado, interessado em criar novos dispositivos para serem acrescentados ao Amiga, ou em desenvolver programas que rodem em computadores Amiga com a memória expandida além de 512K.

Os seguintes tópicos são discutidos neste capítulo:

Descrição do overlay de hunks

para os programadores que se encontram desenvolvendo programas de grande extensão.

Ligando um novo dispositivo de disco ao AmigaDOS

para o programador interessado em adicionar um disco rígido (ou qualquer dispositivo similar a um disco) como uma parte endereçável através de um nome ao sistema de arquivo.

Ligando um novo dispositivo ao AmigaDOS

para o programador acrescentar portas seriais, portas paralelas, graphic tablets, RAM disks, ou o que mais desejar ao AmigaDOS (dispositivos sem relação com o sistema de arquivo).

Usando o AmigaDOS sem o Intuition

para os programadores que quiserem instalar e utilizar seu próprio sistema de manipulação de telas, em substituição ao mecanismo oferecido pelo sistema Intuition.

Tabela de Overlay de Hunks – Visão Geral

Quando você utiliza overlays, o linker basicamente produz um arquivo de grande extensão contendo todos os módulos de objeto na forma de hunks de código relocável. A tabela de overlay de hunks contém uma estrutura de dados que descreve os hunks e seu relacionamento entre si.

Ao projetar um programa que usa overlays, você deve ter em mente como o gerenciador de overlay (também chamado de supervisor de overlay) manipula a interação entre os vários segmentos do arquivo. O que você precisa fazer, basicamente, é construir uma árvore que reflita os relacionamentos entre os vários módulos de código que são parte do programa global, e dizer ao linker como esta árvore deve ser construída.

A tabela de overlay de hunks é gerada como um conjunto de 8 palavras longas, cada qual descrevendo um nó de overlay particular que consiste numa parte do arquivo global. Cada entrada de 8 palavras longas é constituída dos seguintes dados:

Hunk de Símbolos de Overlayl–Estrutura de Entrada na Tabela

```
long seekOffset;        /* em que posição do arquivo se encontra o nó */

long dummy1;            /* um valor igual a 0 ... item para compatibilidade */

long dummy2;            /* um valor igual a 0 ... item para compatibilidade */

long level;             /* nível da árvore */

long ordinate;          /* número do item nesse nível */

long synmbolHunk;       /* número do hunk no qual está localizado o símbolo */

long symbolOffsetX;     /* (offset+4), onde offset é o offset no hunk de símbolos
                           a partir do qual a entrada do símbolo está localizada */
```

Cada um desses itens será discutido em maior profundidade nas seções a seguir.

Projetando uma Árvore de Overlay

Digamos, por exemplo, que você tenha os arquivos MAIN, A, B, C, D, E, F, G, H, I e J. Imagine que a rotina MAIN chame A, B, C e D, e que cada um desses arquivos possa chamar MAIN. Digamos ainda que a rotina E pode ser chamada a partir de A, B, C, D ou MAIN, não tendo porém qualquer relação com a rotina F. Desse modo, para que a rotina E possa ser executada, as rotinas A, B, C e D precisam estar presentes também na memória. A rotina F se parece com E – isto é, ela não precisa de E para estar presente, mas pode ser chamada a partir de A, B, C ou D. Isto significa que o gerenciador de overlay pode repartir o espaço de memória entre as rotinas E e F, uma vez que nenhuma das duas precisa sequer ser co-residente com a outra na memória para ser executada.

Se você considerar que a rotina G compartilhe o mesmo espaço que a combinação de

A, B, C e D, e as rotinas H, I e J compartilhem o mesmo espaço, você tem a base para construir a árvore de overlay para esta estrutura de programa:

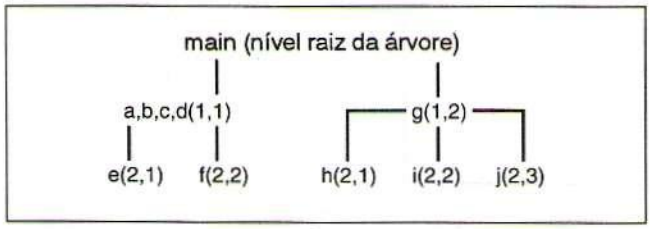


Figura 12-1

Nós não somente desenhamos a árvore, como também rotulamos suas ramificações para combinar com os números de overlay dos hunks (nível, ordenada) encontrados na tabela de overlay e que correspondem aos nós aos quais eles estão associados.

Pela descrição acima, você pode perceber que se MAIN precisar chamar qualquer rotina no segmento de programa A-D, todos esses segmentos precisarão estar residentes na memória ao mesmo tempo. Assim, eles foram todos associados a um único nó pelo linker. Enquanto A-D estiverem residentes, se você chamar rotinas em E, o linker automaticamente carregará a rotina E do disco e reinicializará o módulo (cada vez que E for novamente carregado), de maneira que suas sub-rotinas estejam disponíveis para serem executadas. Se algum segmento de A-D chamar uma rotina em F, o linker substituirá E com o conteúdo de F e o inicializará. Deste modo, A-D estão no nível 1 na árvore de overlay, e as rotinas E e F estão no nível 2, requerendo que A-D sejam carregados antes que E ou F possam ser acessados e carregados para execução.

Nota: Uma rotina só pode efetuar chamadas a rotinas de outros nós que se encontrem residentes na memória (os ancestrais do nó no qual a rotina em uso está localizada), ou então a rotinas de um nó que seja seu filho direto. Noutras palavras, MAIN não pode chamar E diretamente, mas E pode chamar rotinas em MAIN uma vez que MAIN é um ancestral.

Note também que em cada ramificação de cada sub nó, os números de ordenada começam a ser contados novamente a partir de 1 para um dado nível.

Descrevendo a Árvore

Você cria a árvore descrevendo a estrutura da mesma para o linker de overlay. Os valores numéricos, similares àqueles ilustrados na figura acima, são atribuídos seqüencialmente pelo próprio linker e aparecem na tabela de nós do hunk. Eis a seqüência de comandos de ligação de overlay que fazem com que a figura acima seja construída:

```
OVERLAY
a,b,c,d
*e
*f
g
*h
*i
*j
```

Figura 12-2

Esta descrição diz ao linker que A, B e C são parte de um único nó num determinado nível (neste caso, o nível 1). Os asteriscos antes de E e F dizem que estes estão cada qual um nível abaixo de A-D, e acessíveis apenas através de A-D ou de qualquer outro elemento mais próximo em direção à raiz da árvore. O nome G não leva asterisco, sendo considerado do mesmo nível que A-D. Isto diz ao linker que A-D ou G serão residentes na memória, mas não ambos simultaneamente. Os nomes H, I e J mostram-se relacionados com G, um nível abaixo.

Os parágrafos acima explicaram a origem do nível do nó do hunk, e a ordenada do hunk na tabela de símbolos de overlay do hunk.

Valor seekOffset

O primeiro valor para cada nó na tabela de overlay é o offset de localização (*seekOffset*). Como especificado anteriormente, o linker de overlay constrói um grande arquivo contendo todos os nós de overlay. O número do offset de localização corresponde o valor que pode ser dado à rotina Seek(*file, byte_offset*) para que a mesma aponte para o primeiro byte do cabeçalho de hunk de um nó.

firstHunk

O valor firstHunk na tabela de símbolos de overlay é usado pelo gerenciador de overlay ao remover (*unload*) um nó. Este valor especifica o hunk inicial que precisa ter sido carregado a fim de que esteja na memória o nó contendo esse símbolo. Quando uma rotina é chamada num nível e ordenada diferentes (a menos que se trate de um filho direto do nó atual e no próximo nível), é necessário liberar a memória utilizada pelos hunks sem validade, a fim de criar espaço para efetuar o overlay com o(s) hunk(s) contendo o símbolo desejado.

symbolHunk e symbolOffsetX

Estas entradas da tabela de símbolos são utilizadas pelo gerenciador de overlay para localizar o entry point, uma vez que esteja determinado que o mesmo já foi carregado. O symbolHunk mostra em qual hunk o símbolo está localizado, enquanto que symbolOffsetX-4 mostra o offset a contar do início desse hunk no qual o entry point está efetivamente localizado.

Nós de Overlay e o Linker

Durante a linkagem de um programa que utiliza overlays, o linker verifica a validade de cada referência a um símbolo. Suponha que a referência esteja no nó R de uma árvore, e o símbolo em um nó S. Esta referência será considerada legal caso uma das seguintes condições seja verdadeira:

1. R e S são o mesmo nó.
2. R é um descendente de S
3. R é o pai de S.

Referências do terceiro tipo acima são conhecidas como referências de overlay. Neste caso, o linker entra no supervisor de overlay quando o programa é executado. O supervisor de overlay verifica então se o segmento de códigos contendo o símbolo já está na memória. Caso não esteja, primeiramente o segmento de código (se existir) nesse nível e todos os seus descendentes são removidos, e em seguida o nó contendo o símbolo é carregado na memória. Um segmento de código overlay retorna diretamente para a aplicação que o chamou, e assim não é removido da memória até que outro nó seja carregado por cima dele.

Como exemplo, imagine a árvore mostrada na figura 12-3:

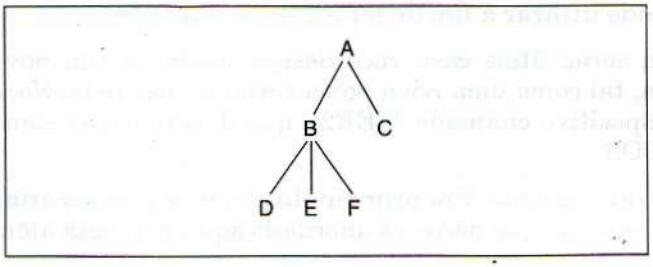


Figura 12-3. A Árvore.

Quando o linker carrega pela primeira vez o programa, somente A está na memória. Quando o linker encontra em A uma referência ao símbolo em B, ele carrega B e entra nele. Se por sua vez B chamar D, um novo nó será carregado. Quando B retornar para A, tanto B quanto D serão mantidos na memória, e o linker não precisará carregá-los novamente caso o programa venha a requisitá-los mais tarde. Agora suponha que A chame C. Primeiramente o linker remove os segmentos de código que não são necessários e podem ser sobrepostos – neste caso, B e D. Uma vez recuperada a memória, o linker poderá então carregar C.

Assim, quando o linker executa um determinado nó, todos os "ancestrais" desse nó até a raiz permanecem na memória, e possivelmente também alguns de seus descendentes.

O linker assume que todas as referências de overlay são saltos ou chamadas para sub-rotinas, e as direciona através do supervisor de overlay. Deste modo, você não deverá utilizar símbolos de overlay como labels de dados.

Procure evitar códigos impuros ao utilizar overlays, pois o linker nem sempre carrega um nó recente do arquivo de carregamento.

O linker atribui a cada símbolo que possui uma referência de overlay um número de overlay. Ele utiliza este valor, que pode ser igual a zero ou maior, para construir o label de entrada do supervisor de overlay associado com este símbolo. Este label obedece ao formato "OVLYnnnn", onde *nnnn* é o número do overlay. Você não deverá utilizar símbolos com este formato em nenhuma outra parte.

O linker une todas as seções do programa com o mesmo nome de seção. Ele faz isto de maneira a poder carregá-las continuamente na memória.

Delete todo o material de "ATOM: (Alink Temporary Object Modifier)". Isto descreve um produto que não é mais vendido e nem possui o suporte do Commodore.

Criando um Novo Dispositivo para Ser Executado Sob o AmigaDOS

Esta seção fornece informações sobre os procedimentos para o acréscimo de dispositivos que NÃO fazem parte do sistema de arquivo do DOS, tais como os handlers de portas e de console. A próxima seção contém informações sobre como acrescentar dispositivos relacionados com o sistema de arquivo (disco rígido, discos flexíveis) – isto é, dispositivos que o DOS pode utilizar a fim de ler e gravar arquivos.

As informações que se seguem serão úteis caso você deseje adicionar um novo dispositivo ao sistema do Amiga, tal como uma nova porta serial ou paralela. Você pode criar, por exemplo, um dispositivo chamado "SER2:" que deverá atuar como "SER:", no que diz respeito ao DOS.

Este procedimento compreende duas etapas. Em primeiro lugar, você precisa criar um dispositivo apropriado – um assunto que não será abordado aqui, pois está além do objetivo desta seção.

Em segundo lugar, você deve tornar esse novo dispositivo disponível como um dispositivo do AmigaDOS. Este procedimento envolve desenvolver um handler (ou seja, uma rotina de manipulação) apropriada para o dispositivo e instalá-lo nas estruturas do AmigaDOS. Para instalar um novo dispositivo e seu handler no AmigaDOS deve-se utilizar o comando MOUNT. Você precisa introduzir o dispositivo no diretório DEVS:, o handler no diretório L: e adicionar uma entrada para o dispositivo no arquivo DEVS:mountlist, que é a lista dos dispositivos montados.

O handler do dispositivo é a interface entre o dispositivo que você criou e um programa de aplicação do usuário. O núcleo do AmigaDOS tentará carregar a rotina do handler e criar um novo processo para ela, no momento em que a sua rotina for referenciada pela primeira vez. Este procedimento tem lugar automaticamente sempre que o núcleo observa que o campo Task da estrutura DevInfo é igual zero. Se o código da rotina já estiver carregado, o ponteiro de segmento de código será inserido no campo SegList. Se o valor deste campo for igual zero, o núcleo carregará o código do arquivo especificado pelo campo do nome do arquivo, e atualizará o campo SegList.

Criando Novos Dispositivos de Disco

Para criar um novo dispositivo de disco é necessário que você construa um novo nó de dispositivo, tal como descrito no tópico "Subestrutura Info" do Capítulo 11 deste livro. Você também deverá escrever um driver de dispositivo para o novo dispositivo de disco.

O driver para o novo dispositivo de disco precisa reproduzir as chamadas que são efetuadas pelo dispositivo trackdisk.device. Ele precisa também ser capaz de responder a comandos tais como Read( ), Write( ), Seek( ), e retornar informações de status, do mesmo modo como foi descrito para o driver do trackdisk.

Usando o AmigaDOS sem o Workbench/Intuition

Esta seção visa dar aos programadores algumas informações sobre como interagem o AmigaDOS e o sistema de interface Intuition, que gerencia a operação das janelas e menus. Até o término da elaboração deste livro não era possível desligar totalmente o Intuition ou o dispositivo de entrada, o que você pode fazer é instalar o seu próprio handler no stream de entrada, e dessa maneira manipular certos eventos de entrada por si mesmo, logo após o seu programa ter sido carregado e iniciado pelo AmigaDOS.

Assumindo o controle da máquina depois deste ponto, você poderá evitar que o AmigaDOS tente exibir requesters do sistema ou interagir de qualquer outra maneira com a tela, bastando modificar o DOS tal como descrito a seguir. Basicamente, você precisará dotar o seu programa de métodos alternativos para a manipulação de erros que normalmente fariam com que o DOS apresente um requester.

Outra alternativa para assumir o controle da máquina é simplesmente ignorar o sistema de arquivo por completo, utilizando o dispositivo Exec trackdisck.device para carregar seus dados e programas, e depois gerenciá-los por conta própria.

Eis os detalhes sobre o AmigaDOS e o Intuition:

Após inicializar a si próprio, o AmigaDOS abre o Intuition. Ele tenta então abrir o arquivo de configuração (criado pelo programa Preferences), passando os seus dados para o Intuition. Em seguida ele abre a janela inicial do CLI (via Intuition) e tenta executar o primeiro comando CLI. Este é normalmente um comando LOADWB (load Workbench), seguido por um ENDCLI do CLI inicial.

Observe que um programa de aplicação pode se comportar como o Workbench, visto que ele dá origem a um novo processo. O próximo comando CLI é então ENDCLI, que encerra tudo, deixando apenas o novo processo rodando (junto com o processo do filesystem). Este processo pode atribuir ao campo pr_WindowPtr o valor -1, o qual indica que o DOS deverá reportar os erros "silenciosamente", isto é, sem apresentar mensagens. Note que será INDISPENSÁVEL que a aplicação manipule todos os erros (para maiores detalhes sobre este assunto, consulte o Capítulo 11). O DOS também inicializa o campo TrapHandler da tarefa do usuário, fazendo-o apontar para a rotina do sistema que exibe um requester após um erro. Esta rotina deverá

ser substituída por uma outra, criada pelo usuário. Os procedimentos descritos até este ponto interromperão todos os usos do sistema Intuition pela tarefa do usuário, desde que não sejam detectados sérios problemas de corrupção da memória, caso no qual o DOS chamará o Alert do Exec imediatamente.

Ainda há o problema de que os processos dos filesystems podem solicitar um requester, na eventualidade da ocorrência de um erro de disco, ou ainda se a tarefa do filesystem resultar num *crash* devido à destruição de dados importantes na memória. Para evitar que isto aconteça, os campos pr_WindowPtr e tc_TrapHandler das tarefas do filesystem deverão receber o valor -1 e um Trap handler privativo deverá ser criado, da mesma maneira que foi feito para a tarefa do usuário. Não há maiores dificuldades para se fazer isso, bastando seguir o procedimento descrito abaixo.

Encontre a porta de mensagens para cada tarefa de filesystem, chamando para isso a função DevProc( ) e passando para ela cada um dos dispositivos executados pelo AmigaDOS (DF0:, DF1:, etc). Você obterá os nomes dos dispositivos percorrendo a lista de dispositivos apontada pela estrutura Info. Um erro indica que o dispositivo não está presente. A partir da porta de mensagens você pode localizar a base para cada tarefa do filesystem, e assim modificar essas duas áreas. Este procedimento deve ser repetido para cada unidade de disco.

O programa de aplicação pode então fechar o sistema Intuition. O Workbench, naturalmente, jamais deverá ser invocado. Note que até o término da elaboração deste livro, era impossível evitar que o DOS abrisse o Intuition.

Note que se a aplicação precisar usar qualquer outro dispositivo tal como SER:, o processo de manipulação deste dispositivo também precisará ser modificado, exatamente como foi feito com o processo do filesystem. A aplicação obviamente não deverá tentar abrir os dispositivos CON: ou RAW: uma vez que o Intuition se tornar inativo.

Índice Analítico

Acertando Data e Hora, 27
Alocação da Memória, 359
Apagando Arquivos, 28
Arquivos
  Anexando Comentários, 12
  File Header Block, 319
  File List Block, 320
  Nomeando Arquivos, 8
  Root Block, 311
  Selecionando Diretório Corrente, 10
  Selecionando Dispositivo Corrente, 11
  Terminologia, 323
  Usando Convenções de Diretórios e Dispositivos Lógicos, 15
  Usando Diretórios, 9
  User Directory Blocks, 316
Arquivos de Biblioteca
  Estrutura, 342
  Exemplo, 341
  hunk_index, 343
  hunk_lib, 342
Arquivos de Carregamento
  hunk_break, 338
  hunk_header, 336
  hunk_overlay, 337
Arquivos Objeto
  hunk_bss, 328
  hunk_code, 327
  hunk_data, 327
  hunk_debug, 334
  hunk_dreloc16, 330
  hunk_dreloc32, 330
  hunk_dreloc8, 331
  hunk_end, 335
  hunk_ext, 331
  hunk_name, 326
  hunk_reloc16, 330
  hunk_reloc32, 329
  hunk_reloc32short, 330
  hunk_reloc8, 330
  hunk_symbol, 333
  hunk_unit, 326
Arquivos WITH, 294
Associando Discos, 31
Automatizando a Seqüência de Boot, 31

Comando LIST, 25
Comando PROTECT, 26
Comandos de edição do Shell, 7
Consultando um Diretório, 24
Copiando Arquivos, 29
Copiando um Disco, 22
Criando um Novo Diretório, 30

Desenvolvimento de Programas
  Chamando Bibliotecas Residentes, 166
  Criando Programa Executável, 166
  Primeiros Passos, 165
Direcionando a Entrada e Saída dos Comandos, 19
Dispositivos do AmigaDOS, 304

Especificações dos Comandos
  ADDBUFFERS, 39
  ADDMONITOR, 40
  ALIAS, 41
  ASK, 42
  ASSIGN, 43
  AUTOPOINT, 46
  AVAIL, 47
  BINDDRIVERS, 47
  BINDMONITOR, 48
  BLANKER, 49
  BREAK, 49
  CALCULATOR, 50
  CD, 51
  CHANGETASKPRI, 52
  CLOCK, 52
  CMD, 53
  COLORS, 54
  COPY, 55
  CPU, 57
  DATE, 58
  DELETE, 59
  DIR, 61
  DISKCHANGE, 62
  DISKCOPY, 63
  DISKDOCTOR, 64

DISPLAY, 65
ECHO, 66
ED, 67
EDIT, 72
ELSE, 75
ENDCLI, 76
ENDIF, 76
ENDSHELL, 77
ENDSKIP, 77
EVAL, 78
EXCHANGE, 80
EXECUTE, 80
Estrutura das Bibliotecas do DOS, 354
Estrutura de Dados dos Processos, 350
Executando Arquivos de Comandos, 19
Exemplos, 338 a 341

File Handles, 360
Formatando um Disco, 23
Funções do AmigaDOS
AbortPkt, 182
AddBuffers, 182
AddDosEntry, 183
AddPart, 184
AddSegment, 185
AllocDosObject, 186
AssignAdd, 186
AssignLate, 187
AssignLock, 188
AssignPath, 188
AttemptLockDosList, 189
ChangeMode, 189
CheckSignal, 190
Cli, 191
Close, 191
CompareDates, 192
CreateDir, 192
CreateNewProc, 193
CreateProc, 194
CurrentDir, 195
DateStamp, 195
DateToStr, 196
Delay, 197
DeleteFile, 198
DeleteVar, 198
DeviceProc, 199
DoPkt, 200
DupLock, 201
DupLockFromFH, 201
EndNotify, 202
ErrorReport, 202
ExAll, 203
Examine, 206
ExamineFH, 207
Execute, 207
Exit, 209
ExNext, 209
Fault, 211
FGetC, 211
FGets, 212
FilePart, 213
FindArg, 213
FindCliProc, 214
FindDosEntry, 215
FindSegment, 215
FindVar, 216
Flush, 217
Format, 217
FPutC, 218
FPuts, 218
FRead, 219
FreeArgs, 220
FreeDeviceProc, 220
FreeDosEntry, 221
FreeDosObject, 221
FWrite, 222
GetArgStr, 222
GetConsoleTask, 223
GetCurrentDirName, 223
GetDeviceProc, 224
GetFileSysTask, 225
GetProgramDir, 225
GetProgramName, 226
GetPrompt, 226
GetVar, 227
Info, 228
Inhibit, 229
Input, 229
InternalLoadSeg, 230
InternalUnLoadSeg, 231
IoErr, 232
IsFileSystem, 232
IsInteractive, 233
LoadSeg, 233
Lock, 234
LockDosList, 234
LockRecord, 235
LockRecords, 236
MakeDosEntry, 237
MakeLink, 237
MatchEnd, 238
MatchFirst, 239
MatchNext, 240
MatchPattern, 241
MatchPatternNoCase, 241
MaxCli, 242

NameFromFH, 243
NameFromLock, 243
NewLoadSeg, 244
NextDosEntry, 245
Open, 245
OpenFromLock, 246
Output, 247
ParentDir, 247
ParentOfFH, 248
ParsePattern, 248
ParsePatternNoCase, 249
PathPart, 250
PrintFault, 250
PutStr, 251
Read, 251
ReadArgs, 252
ReadItem, 255
ReadLink, 255
Relabel, 256
RemAssignList, 257
RemDosEntry, 257
RemSegment, 258
Rename, 258
ReplyPacket, 259
RunCommand, 259
SameDevice, 260
SameLock, 261
Seek, 261
SelectInput, 262
SelectOutput, 263
SendPkt, 263
SetArgStr, 264
SetComment, 264
SetConsoleTask, 265
SetCurrentDirName, 265
SetFileDate, 266
SetFileSize, 266
SetFileSysTask, 267
SetIoErr, 268
SetMode, 268
SetProgramDir, 269
SetProgramName, 269
SetPrompt, 270
SetProtection, 270
SetVar, 271
SetVBuf, 272
SplitName, 273
StartNotify, 274
StrToDate, 274
StrToLong, 276
SystemTagList, 276
UnGetC, 277
UnLoadSeg, 278
UnLock, 279
UnLockDosList, 279
UnLockRecord, 279
UnLockRecords, 280
VFPrintf, 281
VFWritef, 281
VPrintf, 282
WaitForChar, 283
WaitPkt, 284
Write, 284
WriteChars, 285

Gabaritos, 37
Glossário, 157 a 161

Interrompendo o AmigaDOS, 20

Listando o Conteúdo de um Arquivo de Texto, 28
Listas de Segmentos, 359
Locks, 360

Mensagens de Erro, 149 a 156
Mensagens e Códigos de Erro, 300
Mudando o Diretório Corrente, 27
Mudando o Nome de um Arquivo, 28

Obtendo Informações Sobre Arquivos, 26
Os Packets, 361
Overlay, 296
Preparando um Disco CLI, 24
Preparando um Disco de Boot, 23
Programação em I/O
  Análise de Sintaxe da Linha de Comandos, 175
  I/O Bufferizado, 174
  Usando Handlers de Arquivo, 173

Redirecionando a Saída de um Comando, 27
Renomeando um Disco, 24
Resumo dos Comandos de Pontos
  FAILAT, 84
  FAULT, 85
  FILENOTE, 86
  FIXFONTS, 87
  FKEY, 87
  FONT, 87
  FORMAT, 89
  GET, 89
  GETENV, 90
  GRAPHICDUMP, 90
  ICONEDIT, 91

ICONTROL, 91
ICONX, 92
IF, 93
IHELP, 94
INFO, 95
INITPRINTER, 96
INPUT, 96
INSTALL, 97
IPREFS, 98
JOIN, 98
KEYSHOW, 99
LAB, 99
LIST, 99
LOADWB, 102
LOCK, 103
MAKEDIR, 103
MAKELINK, 104
MEMACS, 104
MORE, 110
MOUNT, 11
NEWCLI, 111
NEWSHELL, 112
NOCAPSLOCK, 113
NOFASTMEM, 113
OVERSCAN, 114
PALETTE, 114
PATH, 115
POINTER, 116
PRINTER, 117
PRINTERGFX, 117
PRINTFILES, 118
PROMPT, 118
PROTECT, 119
QUIT, 121
RELABEL, 121
REMRAD, 122
RENAME, 122
RESIDENT, 123
RUN, 125
SAY, 126
SCREENMODE, 127
SEARCH, 127
SERIAL, 129
SET, 129
SETCLOCK, 130
SETDATE, 131
SETENV, 131
SETFONT, 132
SETMAP, 133
SETPATCH, 134
SKIP, 134
SORT, 135
STACK, 136
STATUS, 137
TIME, 137
TYPE, 138
UNALIAS, 138
UNSET, 138
UNSETENV, 139
VERSION, 139
WAIT, 140
WBCONFIG, 140
WBPATTERN, 141
WHICH, 142
WHY, 143

Rodando Comandos em Segundo Plano, 18
Rodando um Programa Sob o CLI
- Ambiente Inicial em Assembler, 167
- Ambiente Inicial em C, 167
- Falhas de Execução das Rotinas, 168
- Terminando um Programa, 168

Tabela de Overlay de Hunks, 390
Tipos de Packets
- Ações Internas do Handler, 381
- Entrada/Saída Básica, 364
- Informação/Manipulação de Arquivos/Diretórios, 367
- Informação/Manipulação de Volumes, 378
- Manutenção e Controle do Handler, 379
- Obsoletos, 382
- Somente de Console, 382
- Sumário, 382

Utilização dos Packets, 385

Programação Visual, Editoração Eletrônica e Produção:

RIOTEXTO
RIOTEXTO
RIOTEXTO
riotexto
RIOTEXTO

RIOTEXTO TECNOLOGIA E PROCESSAMENTO LTDA
RUA ALMIRANTE TAMANDARÉ 66 GRUPO 707
CATETE CEP 22210 RIO DE JANEIRO RJ
TELEFONE 285-2915

Impresso por:
Portinho Cavalcanti Editora Ltda.
Rua Irineu Marinho, 52 - Parte
Telefone: 224-7732 (PABX)
Rio de Janeiro – RJ