Capítulo 22

O conceito de arquivo

Desde os primórdios da computação, percebeu-se a necessidade de armazenar informações para uso posterior, como programas e dados. Hoje, parte importante do uso de um computador consiste em recuperar e apresentar informações previamente armazenadas, como documentos, fotografias, músicas e vídeos. O próprio sistema operacional também precisa manter informações armazenadas para uso posterior, como programas, bibliotecas e configurações. Para simplificar o armazenamento e busca dessas informações, surgiu o conceito de *arquivo*.

Este e os próximos capítulos apresentam os principais conceitos relacionados a arquivos, seu uso e a forma como são implementados e gerenciados pelo sistema operacional.

22.1 Elementos básicos

Um **arquivo** é essencialmente uma sequência de bytes armazenada em um dispositivo físico não volátil, como um disco rígido ou de estado sólido, que preserva seu conteúdo mesmo quando desligado. Cada arquivo possui um nome, ou outra referência, que permite sua localização e acesso. Do ponto de vista do usuário e das aplicações, o arquivo é a unidade básica de armazenamento de informação em um dispositivo não volátil, pois para eles não há forma mais simples de armazenamento persistente de dados. Arquivos são extremamente versáteis em conteúdo e capacidade: podem conter desde um texto com alguns poucos caracteres até vídeos com dezenas de gigabytes, ou ainda mais.

Como um dispositivo de armazenamento pode conter milhões de arquivos, estes são organizados em estruturas hierárquicas denominadas **diretórios**, para facilitar sua localização e acesso pelos usuários. Essa estrutura hierárquica está ilustrada no exemplo da Figura 22.1, que mostra diretórios (`Administrativo`, `Ensino`), subdiretórios (`avaliacoes`, `documentos`) e arquivos (`ci067.txt`, `programa.odt`, `cpp.png`). Os diretórios serão estudados no Capítulo 25.

A organização do conteúdo dos arquivos e diretórios dentro de um dispositivo físico é denominada **sistema de arquivos**. Um sistema de arquivos pode ser visto como uma imensa estrutura de dados armazenada de forma persistente no dispositivo físico. Existe um grande número de sistemas de arquivos, dentre os quais podem ser citados o NTFS (nos sistemas Windows), Ext2/Ext3/Ext4 (Linux), HFS (MacOS), FFS (Solaris) e FAT (usado em *pendrives* USB, câmeras fotográficas digitais e leitores MP3). A organização dos sistemas de arquivos será discutida no Capítulo 24.

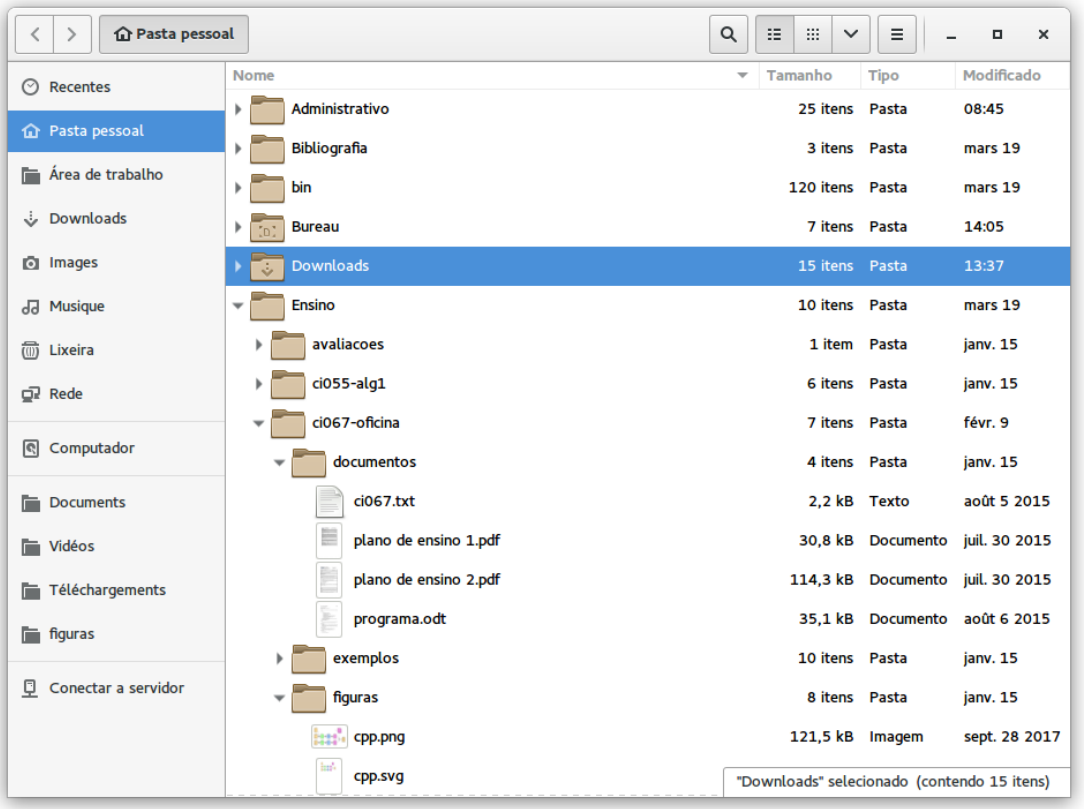

Figura 22.1: Estrutura hierárquica de arquivos e diretórios.

Finalmente, um dispositivo físico é estruturado em um ou mais **volumes** (ou partições); cada volume pode armazenar um sistema de arquivos próprio. Assim, um mesmo disco pode conter volumes com diferentes sistemas de arquivos, como FAT, NTFS ou EXT4, por exemplo.

22.2 Atributos e operações

Um arquivo é uma unidade de armazenamento de informações que podem ser documentos, imagens, código executável, etc. Além de seu conteúdo, um arquivo é caracterizado por *atributos* ou *metadados*, que são informações adicionais relativas ao conteúdo, e *operações* ou ações que podem ser realizadas sobre o conteúdo e/ou sobre os atributos.

Os **atributos** dos arquivos variam de acordo com o sistema de arquivos utilizado. Os atributos mais usuais, presentes na maioria dos sistemas, são:

Nome: *string* que identifica o arquivo para o usuário, como "`foto1.jpg`", "`documento.pdf`", "`hello.c`", etc.;

Tipo: indicação do formato dos dados contidos no arquivo, como áudio, vídeo, imagem, texto, etc. Muitos sistemas operacionais usam parte do nome do arquivo para identificar o tipo de seu conteúdo, na forma de uma extensão: "`.doc`", "`.jpg`", "`.mp3`", etc.;

Tamanho: indicação do tamanho do conteúdo do arquivo, geralmente em bytes;

Datas: para fins de gerência, é importante manter as datas mais importantes relacionadas ao arquivo, como suas datas de criação, de último acesso e de última modificação do conteúdo;

Proprietário: em sistemas multiusuários, cada arquivo tem um proprietário, que deve estar corretamente identificado;

Permissões de acesso: indicam que usuários têm acesso àquele arquivo e que formas de acesso são permitidas (leitura, escrita, remoção, etc.);

Localização: indicação do dispositivo físico onde o arquivo se encontra e da posição do arquivo dentro do mesmo;

Além destes, vários outros atributos podem ser associados a um arquivo, por exemplo para indicar se ele é um arquivo de sistema, se está visível aos usuários, se tem conteúdo binário ou textual, etc. Cada sistema de arquivos normalmente define seus próprios atributos específicos, além dos atributos usuais acima.

Nem sempre os atributos oferecidos por um sistema de arquivos são suficientes para exprimir todas as informações a respeito de um arquivo. Nesse caso, a "solução" encontrada pelos usuários é usar o nome do arquivo para registrar a informação desejada. Por exemplo, em muitos sistemas a parte final do nome do arquivo (sua extensão) é usada para identificar o formato de seu conteúdo. Outra situação frequente é usar parte do nome do arquivo para identificar diferentes versões do mesmo conteúdo[1]: `relat-v1.txt`, `relat-v2.txt`, etc.

As aplicações e o sistema operacional usam arquivos para armazenar e recuperar dados. O acesso aos arquivos é feito através de um conjunto de **operações**, geralmente implementadas sob a forma de chamadas de sistema e funções de bibliotecas. As operações básicas envolvendo arquivos são:

Criar: a criação de um novo arquivo implica em alocar espaço para ele no dispositivo de armazenamento e definir valores para seus atributos (nome, localização, proprietário, permissões de acesso, datas, etc.);

Abrir: antes que uma aplicação possa ler ou escrever dados em um arquivo, ela deve solicitar ao sistema operacional a "abertura" desse arquivo. O sistema irá então verificar se o arquivo desejado existe, verificar se as permissões associadas ao arquivo permitem aquele acesso, localizar seu conteúdo no dispositivo de armazenamento e criar uma referência para ele na memória da aplicação;

Ler: permite transferir dados presentes no arquivo para uma área de memória da aplicação;

Escrever: permite transferir dados na memória da aplicação para o arquivo no dispositivo físico; os novos dados podem ser adicionados ao final do arquivo ou sobrescrever dados já existentes;

Fechar: ao concluir o uso do arquivo, a aplicação deve informar ao sistema operacional que o mesmo não é mais necessário, a fim de liberar as estruturas de gerência do arquivo mantidas na memória do núcleo;

[1] Alguns sistemas operacionais, como o *TOPS-20* e o *OpenVMS*, possuem sistemas de arquivos com suporte automático a múltiplas versões do mesmo arquivo.

Remover: para eliminar o arquivo do dispositivo, descartando seus dados e liberando o espaço ocupado por ele.

Alterar atributos: para modificar os valores dos atributos do arquivo, como nome, proprietário, permissões, datas, etc.

Além dessas operações básicas, outras operações podem ser definidas, como truncar, copiar, mover ou renomear arquivos. Todavia, essas operações geralmente podem ser construídas usando as operações básicas acima. Por exemplo, para copiar um arquivo A em um novo arquivo B, os seguintes passos seriam necessários: abrir A; criar e abrir B; ler conteúdo de A e escrever em B; fechar A e B.

22.3 Formatos de arquivos

Arquivos permitem armazenar dados para uso posterior. A forma como esses dados são estocados dentro do arquivo é denominada *formato do arquivo*. Esta seção discute os formatos de arquivos mais usuais.

22.3.1 Sequência de bytes

Em sua forma mais simples, um arquivo contém basicamente uma sequência de bytes. Essa sequência de bytes pode ser estruturada de forma a representar diferentes tipos de informação, como imagens, música, textos ou código executável. Essa estrutura interna do arquivo pode ser entendida pelo núcleo do sistema operacional ou somente pelas aplicações que acessam esse conteúdo.

O núcleo do sistema geralmente reconhece apenas alguns poucos formatos de arquivos, como binários executáveis e bibliotecas. Os demais formatos de arquivos são vistos pelo núcleo apenas como sequências de bytes opacas, sem um significado específico, cabendo às aplicações interessadas interpretá-las.

Uma aplicação pode definir um formato próprio para armazenar seus dados, ou pode seguir formatos padronizados. Por exemplo, há um grande número de formatos padronizados para o armazenamento de imagens, como JPEG, GIF, PNG e TIFF, mas também existem formatos de arquivos proprietários, definidos por algumas aplicações específicas, como os formatos PSD e XCF (dos editores gráficos *Adobe Photoshop* e *GIMP*, respectivamente). A adoção de um formato proprietário ou exclusivo limita o uso das informações armazenadas, pois somente aplicações que reconheçam aquele formato específico conseguem ler corretamente as informações contidas no arquivo.

22.3.2 Arquivos de registros

Alguns núcleos de sistemas operacionais oferecem arquivos com estruturas internas que vão além da simples sequência de bytes. Por exemplo, o sistema *OpenVMS* [Rice, 2000] proporciona *arquivos baseados em registros*, cujo conteúdo é visto pelas aplicações como uma sequência linear de registros de tamanho fixo ou variável, e também *arquivos indexados*, nos quais podem ser armazenados pares {*chave*/*valor*}, de forma similar a um banco de dados relacional. A Figura 22.2 ilustra a estrutura interna desses dois tipos de arquivos.

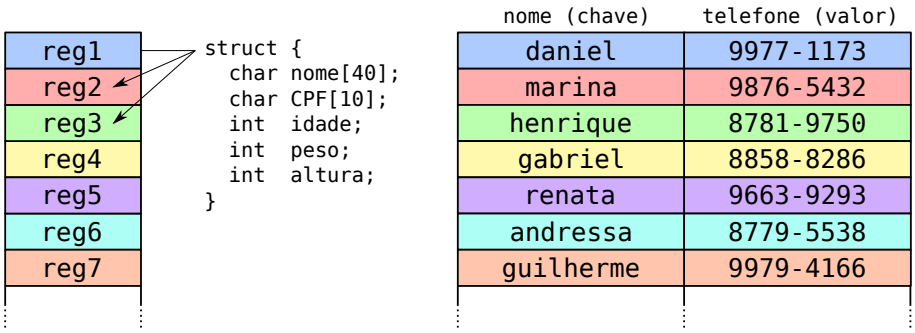

Figura 22.2: Arquivos estruturados: registros em sequência e registros indexados.

Nos sistemas operacionais cujo núcleo não suporta arquivos estruturados em registros, essa funcionalidade pode ser facilmente obtida através de bibliotecas específicas ou do suporte de execução de algumas linguagens de programação. Por exemplo, as bibliotecas *Berkeley DB* e *SQLite*, disponíveis em plataformas UNIX, oferecem suporte à indexação de registros sobre arquivos UNIX convencionais.

22.3.3 Arquivos de texto

Um tipo de arquivo de uso muito frequente é o arquivo de *texto puro* (ou *plain text*). Esse tipo de arquivo é usado para armazenar informações textuais simples, como códigos-fonte de programas, arquivos de configuração, páginas HTML, dados em XML, etc. Um arquivo de texto é formado por linhas de caracteres de tamanho variável, separadas por caracteres de controle. Nos sistemas UNIX, as linhas são separadas por um caractere *New Line* (ASCII 10 ou "\n"). Já nos sistemas DOS/Windows, as linhas de um arquivo de texto são separadas por dois caracteres: o caractere *Carriage Return* (ASCII 13 ou "\r") seguido do caractere *New Line*.

Por exemplo, considere o seguinte programa em C armazenado em um arquivo `hello.c` (os caracteres "␣" indicam espaços em branco):

```
1 int␣main()
2 {
3 ␣␣printf("Hello,␣world\n");
4 ␣␣exit(0);
5 }
```

O arquivo de texto `hello.c` seria armazenado usando a seguinte sequência de bytes[2] em um sistema UNIX:

```
1 0000  69 6e 74 20 6d 61 69 6e 28 29 0a 7b 0a 20 20 70
2        i  n  t  ␣  m  a  i  n  (  ) \n  {  \n ␣  ␣  p
3 0010  72 69 6e 74 66 28 22 48 65 6c 6c 6f 2c 20 77 6f
4        r  i  n  t  f  (  "  H  e  l  l  o  ,  ␣  w  o
5 0020  72 6c 64 5c 6e 22 29 3b 0a 20 20 65 78 69 74 28
6        r  l  d  \  n  "  )  ;  \n ␣  ␣  e  x  i  t  (
7 0030  30 29 3b 0a 7d 0a
8        0  )  ;  \n  }  \n
```

[2]Listagem obtida através do comando `hd` do Linux, que apresenta o conteúdo de um arquivo em hexadecimal e seus caracteres ASCII correspondentes, byte por byte.

Por outro lado, o mesmo arquivo `hello.c` seria armazenado usando a seguinte sequência de bytes em um sistema DOS/Windows:

```
0000  69 6e 74 20 6d 61 69 6e 28 29 0d 0a 7b 0d 0a 20
       i  n  t  ␣  m  a  i  n  (  ) \r \n  {  \r \n  ␣
0010  20 70 72 69 6e 74 66 28 22 48 65 6c 6c 6f 2c 20
       ␣  p  r  i  n  t  f  (  "  H  e  l  l  o  ,  ␣
0020  77 6f 72 6c 64 5c 6e 22 29 3b 0d 0a 20 20 65 78
       w  o  r  l  d  \  n  "  )  ;  \r \n  ␣  ␣  e  x
0030  69 74 28 30 29 3b 0d 0a 7d 0d 0a
       i  t  (  0  )  ;  \r \n  }  \r \n
```

Essa diferença na forma de representação da separação entre linhas pode provocar problemas em arquivos de texto transferidos entre sistemas Windows e UNIX, caso não seja feita a devida conversão.

22.3.4 Arquivos de código

Em um sistema operacional moderno, um arquivo de código (programa executável ou biblioteca) é dividido internamente em várias seções, para conter código, tabelas de símbolos (variáveis e funções), listas de dependências (bibliotecas necessárias) e outras informações de configuração. A organização interna de um arquivo de código depende do sistema operacional para o qual foi definido. Os formatos de código mais usuais atualmente são [Levine, 2000]:

- **ELF** (*Executable and Linking Format*): formato de arquivo usado para programas executáveis e bibliotecas na maior parte das plataformas UNIX modernas. É composto por um cabeçalho e várias seções de dados, contendo código executável, tabelas de símbolos e informações sobre relocação de código, usadas quando o código executável é carregado na memória.
- **PE** (*Portable Executable*): é o formato usado para executáveis e bibliotecas na plataforma Windows. Consiste basicamente em uma extensão do formato COFF (*Common Object File Format*), usado em plataformas UNIX mais antigas.

A Figura 22.3 ilustra de forma simplificada a estrutura interna de um arquivo de código no formato ELF, usado tipicamente para armazenar código executável ou bibliotecas em sistemas UNIX (Linux, Solaris, etc.). Esse arquivo é composto por:

- *ELF header*: descreve o conteúdo do restante do arquivo.
- *Section header table*: descreve cada uma das seções do código (nome, tipo, tamanho, etc).
- *Sections*: trechos que compõem o arquivo, como código binário, constantes, tabela de símbolos, tabela de relocações, etc. As seções são agrupadas em segmentos.
- *Program header table*: informações sobre como o código deve ser carregado na memória ao lançar o processo ou carregar a biblioteca.

- *Segments*: conteúdo que deve ser carregado em cada segmento de memória ao lançar o processo; um segmento pode conter várias seções do programa ou biblioteca, para agilizar a carga do programa na memória.

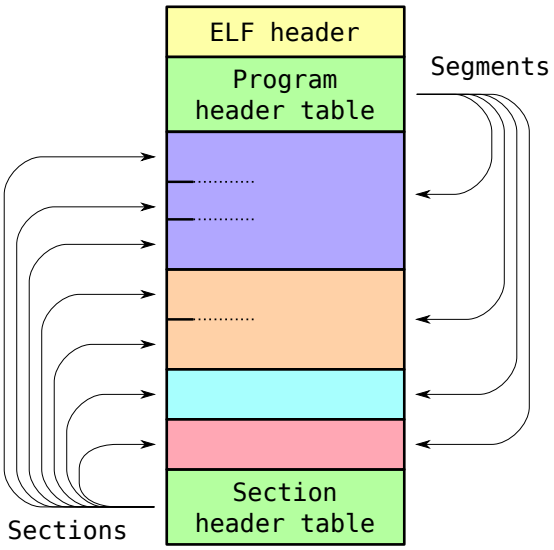

Figura 22.3: Estrutura interna de um arquivo em formato ELF [Levine, 2000].

22.3.5 Identificação de conteúdo

Um problema importante relacionado aos formatos de arquivos é a correta identificação de seu conteúdo pelos usuários e aplicações. Já que um arquivo de dados pode ser visto como uma simples sequência de bytes, como é possível reconhecer que tipo de informação essa sequência representa?

Uma solução simples para esse problema consiste em usar parte do nome do arquivo para indicar o tipo do conteúdo: assim, um arquivo "`praia.jpg`" provavelmente contém uma imagem em formato JPEG, enquanto um arquivo "`entrevista.mp3`" contém áudio em formato MP3. A estratégia de **extensão do nome**, utilizada ainda hoje na maioria dos sistemas operacionais, foi introduzida nos anos 1980 pelo sistema operacional DOS. Naquele sistema, os arquivos eram nomeados usando um padrão denominado "8.3": até 8 caracteres para o nome, seguidos de um ponto (".") e de uma extensão com até 3 caracteres, para o tipo do conteúdo.

Outra abordagem, frequentemente usada em sistemas UNIX, é usar alguns bytes no início do conteúdo do arquivo para a definição de seu tipo. Esses bytes iniciais do conteúdo são denominados **números mágicos** (*magic numbers*), e são convencionados para muitos tipos de arquivos, como mostra a Tabela 22.1.

Nos sistemas UNIX, o utilitário `file` permite identificar o tipo de arquivo através da análise de seus bytes iniciais e do restante de sua estrutura interna, sem levar em conta o nome do arquivo. Por isso, constitui uma ferramenta importante para identificar arquivos sem extensão ou com extensão errada.

Além do uso de extensões no nome do arquivo e de números mágicos, alguns sistemas operacionais definem **atributos adicionais** no sistema de arquivos para identificar o conteúdo de cada arquivo. Por exemplo, o sistema operacional MacOS 9 define um atributo com 4 bytes para identificar o tipo de cada arquivo (*file type*), e outro atributo

Tabela 22.1: Números mágicos de alguns tipos de arquivos

Tipo de arquivo	**bytes iniciais**	**Tipo de arquivo**	**bytes iniciais**
Documento PostScript	`%!`	Documento PDF	`%PDF`
Imagem GIF	`GIF89a`	Imagem JPEG	`0xFF D8 FF`
Música MIDI	`MThd`	Classes Java	`0xCA FE BA BE`
Arquivo ZIP	`0x50 4B 03 04`	Documento RTF	`{\rtf1`

com 4 bytes para indicar a aplicação que o criou (*creator application*). Os tipos de arquivos e aplicações são definidos em uma tabela mantida pelo fabricante do sistema. Assim, quando o usuário solicitar a abertura de um determinado arquivo, o sistema irá escolher a aplicação que o criou, se ela estiver presente. Caso contrário, pode indicar ao usuário uma relação de aplicações aptas a abrir aquele tipo de arquivo.

Recentemente, a necessidade de transferir arquivos através de e-mail e de páginas Web levou à definição de um padrão de tipagem de arquivos conhecido como **Tipos MIME** (da sigla *Multipurpose Internet Mail Extensions*) [Freed and Borenstein, 1996]. O padrão MIME define tipos de arquivos através de uma notação uniformizada na forma "`tipo/subtipo`". Alguns exemplos de tipos de arquivos definidos segundo o padrão MIME são apresentados na Tabela 22.2.

Tabela 22.2: Tipos MIME correspondentes a alguns formatos de arquivos

Tipo MIME	**Significado**
`application/java-archive`	Arquivo de classes Java
`application/msword`	Documento do Microsoft Word
`application/vnd.oasis.opendocument.text`	Documento do OpenOffice
`audio/midi`	Áudio em formato MIDI
`audio/mpeg`	Áudio em formato MP3
`image/jpeg`	Imagem em formato JPEG
`image/png`	Imagem em formato PNG
`text/csv`	Texto em formato CSV (*Comma-separated Values*)
`text/html`	Texto HTML
`text/plain`	Texto puro
`text/rtf`	Texto em formato RTF (*Rich Text Format*)
`text/x-csrc`	Código-fonte em C
`video/quicktime`	Vídeo no formato *Quicktime*

O padrão MIME é usado para identificar arquivos transferidos como anexos de e-mail e conteúdos obtidos de páginas Web. Alguns sistemas operacionais, como o BeOS e o MacOS X, definem atributos adicionais usando esse padrão para identificar o conteúdo de cada arquivo dentro do sistema de arquivos.

22.4 Arquivos especiais

O conceito de arquivo é ao mesmo tempo simples e poderoso, o que motivou sua utilização de forma quase universal. Além do armazenamento de dados do sistema

operacional e de aplicações, como mostrado na seção anterior, o conceito de arquivo também pode ser usado como:

Abstração de dispositivos de entrada/saída: os sistemas UNIX costumam mapear as interfaces de acesso a vários dispositivos físicos como arquivos dentro do diretório `/dev` (da palavra *devices*), como por exemplo:

- `/dev/ttyS0`: porta de comunicação serial (COM1);
- `/dev/sda1`: primeira partição do primeiro disco SATA.

Essa abstração dos dispositivos é muito conveniente, pois permite que aplicações percebam e acessem os dispositivos físicos como se fossem arquivos. Por exemplo, uma aplicação que escrever dados no arquivo `/dev/ttyS0` estará enviando esses dados para a saída serial COM1 do computador.

Abstração de interfaces do núcleo: em sistemas UNIX, os diretórios `/proc` e `/sys` permitem consultar e/ou modificar informações internas do núcleo do sistema operacional, dos processos em execução e dos *drivers* de dispositivos. Por exemplo, alguns arquivos oferecidos pelo Linux:

- `/proc/cpuinfo`: informações sobre os processadores disponíveis no sistema;
- `/proc/3754/maps`: disposição das áreas de memória alocadas para o processo cujo identificador (PID) é 3754;
- `/sys/block/sda/queue/scheduler`: definição da política de escalonamento de disco (vide Capítulo 21.1.2) a ser usada no acesso ao disco `/dev/sda`.

Canais de comunicação: na família de protocolos de rede TCP/IP, a abstração de arquivo é usada como interface para os canais de comunicação: uma conexão TCP é apresentada aos dois processos comunicantes como um arquivo, sobre o qual eles podem escrever (enviar) e ler (receber) dados entre si. Vários mecanismos de comunicação local entre processos de um sistema também são vistos como arquivos, como é o caso dos *pipes* em UNIX.

Em alguns sistemas operacionais experimentais, como o *Plan 9* [Pike et al., 1993, 1995] e o *Inferno* [Dorward et al., 1997], todos os recursos e entidades físicas e lógicas do sistema operacional são mapeadas sob a forma de arquivos: processos, *threads*, conexões de rede, usuários, sessões de usuários, janelas gráficas, áreas de memória alocadas, etc. Assim, para finalizar um determinado processo, encerrar uma conexão de rede ou desconectar um usuário, basta remover o arquivo correspondente.

Embora o foco deste texto esteja concentrado em arquivos convencionais, que visam o armazenamento de informações (bytes ou registros), muitos dos conceitos aqui expostos são igualmente aplicáveis aos arquivos não convencionais descritos neste capítulo.

Exercícios

1. Enumere os principais atributos de um arquivo.
2. Enumere as principais operações sobre arquivos.
3. Apresente e comente as principais formas de atribuição de tipos aos arquivos. Quais são as vantagens e desvantagens de cada uma?
4. Sobre as afirmações a seguir, relativas a formatos de arquivos, indique quais são incorretas, justificando sua resposta:
   (a) Um *magic number* consiste de um atributo numérico separado que identifica o tipo de arquivo.
   (b) A forma mais comum de identificação de tipo de arquivo é o uso de extensões ao seu nome.
   (c) Arquivos de texto em sistemas DOS e UNIX diferem nos caracteres de controle usados para identificar o fim de arquivo.
   (d) Para a maioria dos núcleos de sistema operacional, arquivos são quase sempre vistos como meras sequências de bytes.
   (e) ELF e PE são dois formatos típicos de arquivos de configuração.
   (f) O padrão MIME é usado no Linux para identificação de tipos de arquivos pelo sistema operacional.

Referências

S. Dorward, R. Pike, D. Presotto, D. Ritchie, H. Trickey, and P. Winterbottom. The Inferno operating system. *Bell Labs Technical Journal*, 2(1):5–18, 1997.

N. Freed and N. Borenstein. RFC 2046: Multipurpose Internet Mail Extensions (MIME) part two: Media types, Nov 1996.

J. Levine. *Linkers and Loaders*. Morgan Kaufmann, 2000.

R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom. The use of name spaces in Plan 9. *Operating Systems Review*, 27(2):72–76, April 1993.

R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Journal of Computing Systems*, 8(3):221–254, 1995.

L. Rice. *Introduction to OpenVMS*. Elsevier Science & Technology Books, 2000.