Capítulo 25

Diretórios e atalhos

A quantidade de arquivos em um sistema atual pode ser muito grande, chegando facilmente a milhões deles em um computador *desktop* típico, e muito mais em servidores. O sistema operacional pode tratar facilmente essa imensa quantidade de arquivos, mas essa tarefa não é tão simples para os usuários. Para simplificar a gestão dos arquivos é possível organizá-los em grupos e hierarquias, usando diretórios e atalhos, cujo conceito e implementação são explicados neste capítulo.

Além de arquivos, um sistema de arquivos também precisa implementar outros elementos, como diretórios e atalhos, e precisa construir mecanismos eficientes para permitir a localização de arquivos nas árvores de diretórios, que podem ser imensas. Esta seção apresenta as linhas gerais da implementação de diretórios, de atalhos e do mecanismo de localização de arquivos nos diretórios.

25.1 Diretórios

Um diretório, também chamado de *pasta* ou *folder*, representa um contêiner de arquivos e de outros diretórios. Da mesma forma que os arquivos, os diretórios têm nome e atributos, que são usados na localização e acesso ao seu conteúdo.

Cada sistema de arquivos possui um diretório principal, denominado *diretório raiz* (*root directory*). Os primeiros sistemas de arquivos implementavam apenas o diretório raiz, que continha todos os arquivos. Posteriormente foram implementados subdiretórios, ou seja, um nível de diretórios abaixo do diretório raiz. Os sistemas de arquivos atuais oferecem uma estrutura muito mais flexível, com um número de níveis de diretórios muito mais elevado, ou mesmo ilimitado (como nos sistemas de arquivos NTFS e Ext4).

O uso de diretórios permite construir uma estrutura hierárquica (em árvore) de armazenamento dentro de um volume, sobre a qual os arquivos são organizados. A Figura 25.1 representa uma pequena parte da árvore de diretórios típica de sistemas Linux, cuja estrutura é definida nas normas *Filesystem Hierarchy Standard* (FHS) [Russell et al., 2004].

A maioria dos sistemas operacionais implementa o conceito de *diretório de trabalho* ou diretório corrente de um processo (*working directory*): ao ser criado, cada novo processo é associado a um diretório, que será usado por ele como local default para criar novos arquivos ou abrir arquivos existentes, quando não informar os respectivos caminhos de acesso. Cada processo geralmente herda o diretório de trabalho de seu

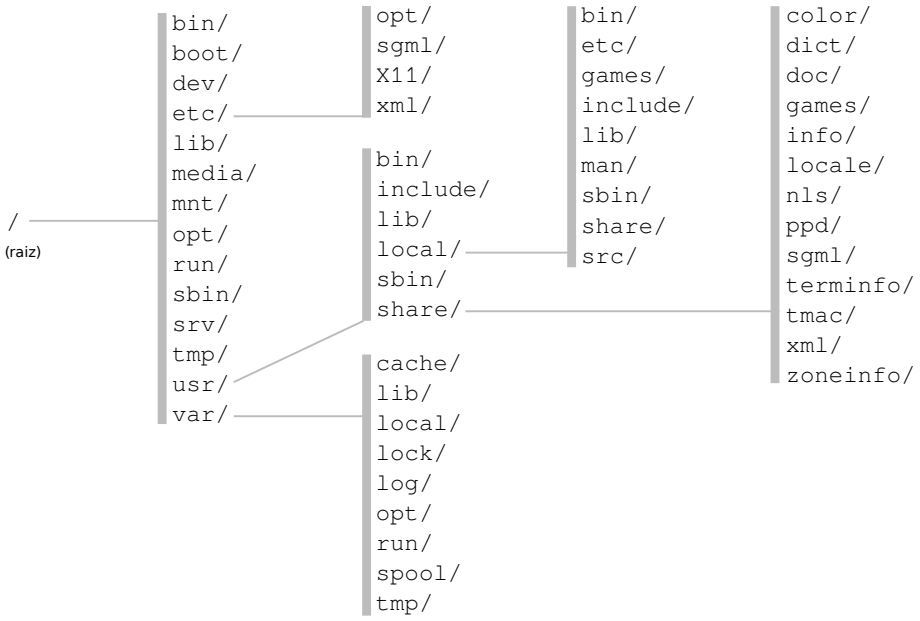

Figura 25.1: Estrutura de diretórios FHS de um sistema Linux.

pai, mas pode mudar de diretório através de chamadas de sistema (como `chdir` nos sistemas UNIX).

25.2 Caminhos de acesso

Em um sistema de arquivos, os arquivos estão dispersos ao longo da hierarquia de diretórios. Para poder abrir e acessar um arquivo, torna-se então necessário conhecer sua localização completa, ao invés de somente seu nome. A posição de um arquivo dentro do sistema de arquivos é chamada de **caminho de acesso** ao arquivo. Normalmente, o caminho de acesso a um arquivo é composto pela sequência de nomes de diretórios que levam até ele a partir da raiz, separados por um caractere específico. Cada elemento do caminho representa um nível de diretório, a partir da raiz. Por exemplo, considerando a estrutura de diretórios da figura 25.2, o arquivo `index.html` teria o seguinte caminho de acesso: `C:\Users\Maziero\public_html\index.html`.

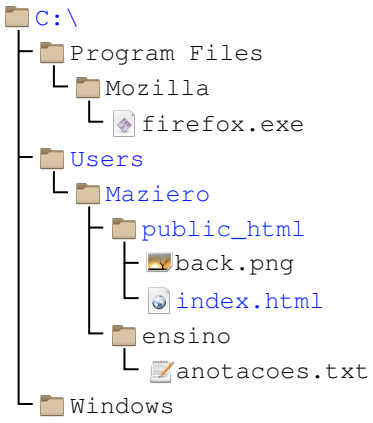

Figura 25.2: Caminho de acesso em um sistema Windows

O caractere separador de nomes no caminho depende do sistema operacional. Por exemplo, o sistema Windows usa como separador o caractere "\", enquanto sistemas UNIX usam o caractere "/"; outros sistemas podem usar caracteres como "`:`" e "`!`". Exemplos de caminhos de acesso a arquivos em sistemas UNIX seriam `/bin/bash` e `/var/log/mail.log`.

Em muitos sistemas de arquivos, um caminho de acesso pode conter elementos especiais, como "`..`" e "`.`". O elemento "`..`" indica o diretório anterior, ou seja, retorna um nível na hierarquia; por sua vez, o elemento "`.`" indica o próprio diretório atual. Esses elementos são úteis na construção de referências de acesso a arquivos fora do diretório corrente do processo.

Para acessar um arquivo, o processo precisa fornecer uma referência a ele. Existem basicamente três formas de se referenciar arquivos armazenados em um sistema de arquivos:

Referência direta: somente o nome do arquivo é informado pelo processo; neste caso, considera-se que o arquivo está (ou será criado) no diretório de trabalho do processo. Exemplos:

```
1 firefox.exe
2 back.png
3 index.html
```

Em um sistema UNIX, se o processo estiver trabalhando no diretório `/home/prof/maziero`, então o arquivo `materiais.pdf` pode ser localizado pelo sistema operacional através do caminho `/home/prof/maziero/materiais.pdf`.

Referência absoluta: o caminho de acesso ao arquivo é indicado a partir do diretório raiz do sistema de arquivos, e não depende do diretório de trabalho do processo; uma referência absoluta a um arquivo sempre inicia com o caractere separador, indicando que o nome do arquivo está referenciado a partir do diretório raiz do sistema de arquivos. Exemplos de referências absolutas:

```
C:\Users\Maziero\ensino\anotacoes.txt
/usr/local/share/fortunes/brasil.dat
C:\Program Files\Mozilla\firefox.exe
/home/maziero/bin/scripts/../../docs/proj1.pdf
```

O caminho de acesso mais curto a um arquivo a partir do diretório raiz é denominado *caminho canônico* do arquivo. Nos exemplos de referências absolutas acima, os dois primeiros são caminhos canônicos, enquanto os dois últimos não.

Referência relativa: o caminho de acesso ao arquivo tem como início o diretório de trabalho do processo, e indica subdiretórios ou diretórios anteriores, através de elementos "`..`". Eis alguns exemplos:

```
Mozilla\firefox.exe
public_html\index.html
public_html/static/fotografias/rennes.jpg
../../../share/icons/128x128/calculator.svg
```

25.3 Implementação de diretórios

A implementação de diretórios em um sistema de arquivos é relativamente simples: um diretório é implementado como um arquivo cujo conteúdo é uma relação de entradas (ou seja, uma tabela). Os tipos de entradas normalmente considerados nessa relação são arquivos normais, outros diretórios, atalhos (vide Seção 25.4) e entradas associadas a arquivos especiais, como os discutidos na Seção 22.4. Cada entrada contém ao menos o nome do arquivo (ou do diretório), seu tipo e a localização física do mesmo no volume (número do *i-node* ou número do bloco inicial). Deve ficar claro que um diretório não contém fisicamente os arquivos e subdiretórios, ele apenas os relaciona.

Em sistemas de arquivos mais antigos e simples, o diretório raiz de um volume estava definido em seus blocos de inicialização, normalmente reservados para informações de gerência. Todavia, como o número de blocos reservados era pequeno e fixo, o número de entradas no diretório raiz era limitado. Nos sistemas mais recentes, um registro específico dentro dos blocos reservados aponta para a posição do diretório raiz dentro do sistema de arquivos, permitindo que este tenha um número maior de entradas.

A Figura 25.3 apresenta a implementação de parte de uma estrutura de diretórios de um sistema UNIX. O endereço do diretório raiz (/) é indicado por uma entrada

específica no VBR – *Volume Boot Record*, dentro da área reservada do disco ou partição. Neste exemplo, as entradas em cada diretório pode ser arquivos de dados (`A`) ou diretórios (`D`). Em uma implementação real podem existir outras entradas, como atalhos (`L` - *links*) e arquivos especiais (Seção 22.4). A informação sobre o tipo de cada entrada pode estar presente na própria tabela (como no exemplo da figura), ou pode estar nos metadados do *i-node* da entrada correspondente.

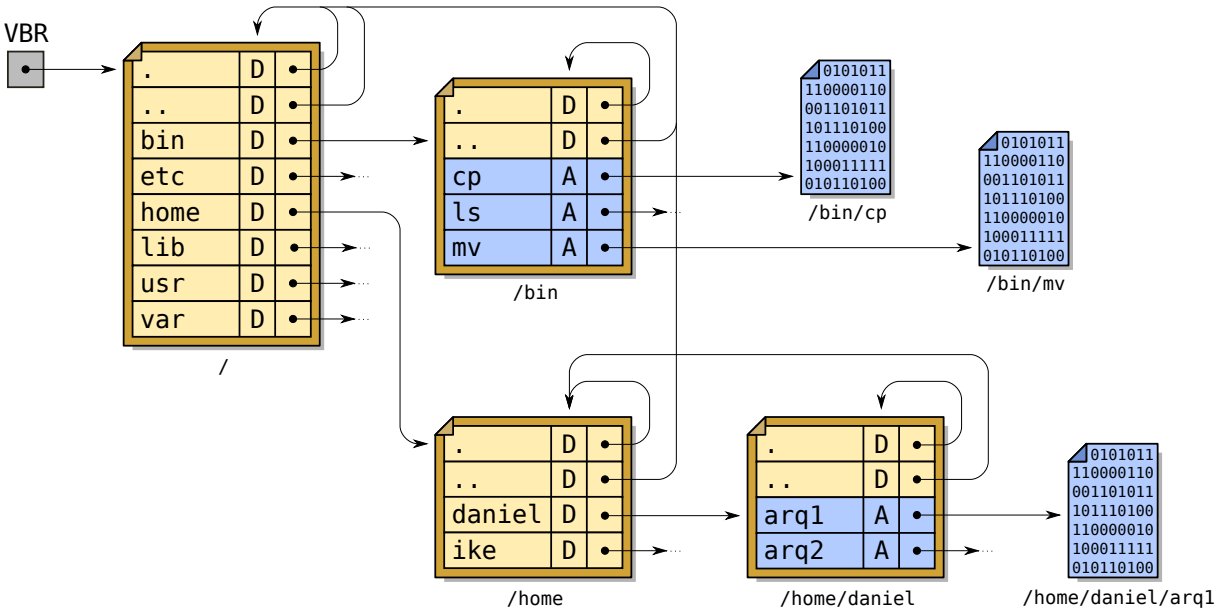

Figura 25.3: Implementação de uma estrutura de diretórios.

Na figura, podem também ser observadas duas entradas usualmente definidas em cada diretório: a entrada "`.`" (ponto), que representa o próprio diretório, e a entrada "`..`" (ponto-ponto), que representa seu diretório pai (o diretório imediatamente acima dele na hierarquia de diretórios). No caso do diretório raiz, ambas as entradas apontam para ele mesmo.

Internamente, a lista ou índice do diretório pode ser implementada como uma tabela simples, como no caso do MS-DOS e do Ext2 (Linux). Essa implementação é mais simples, mas tem baixo desempenho, sobretudo em diretórios contendo muitos ítens (a complexidade da busca em uma lista linear com n elementos é $O(n)$). Sistemas de arquivos mais sofisticados, como o Ext4, ZFS e NTFS usam estruturas de dados com maior desempenho de busca, como *hashes* e árvores.

25.4 Atalhos

Em algumas ocasiões, pode ser necessário ter um mesmo arquivo ou diretório replicado em várias posições dentro do sistema de arquivos. Isso ocorre frequentemente com arquivos de configuração de programas e arquivos de bibliotecas, por exemplo. Nestes casos, seria mais econômico (em espaço de armazenamento) armazenar apenas uma instância dos dados do arquivo no sistema de arquivos e criar referências indiretas (ponteiros) para essa instância, para representar as demais cópias do arquivo. O mesmo raciocínio pode ser aplicado a diretórios duplicados. Essas referências indiretas a arquivos ou diretórios são denominadas *atalhos* (ou *links*).

A listagem (simplificada) a seguir apresenta algumas entradas do diretório `/usr/lib/` em um sistema Linux. Nela podem ser observados alguns atalhos: as entradas `libcryptui.so` e `libcryptui.so.0` são atalhos para o arquivo `libcryptui.so.0.0.0`, enquanto a entrada `libcrypt.so` é um atalho para o arquivo `/lib/x86_64-linux-gnu/libcrypt.so.1`.

```
~> ls -l /usr/lib/

...
lrwxrwxrwx  1 root root libcrypt.so -> /lib/x86_64-linux-gnu/libcrypt.so.1
lrwxrwxrwx  1 root root libcryptui.so -> libcryptui.so.0.0.0
lrwxrwxrwx  1 root root libcryptui.so.0 -> libcryptui.so.0.0.0
-rw-r--r--  1 root root libcryptui.so.0.0.0
...
```

Assim, quando um processo solicitar acesso ao arquivo `/usr/lib/libcrypt.so`, ele na verdade estará acessando o arquivo `/lib/x86_64-linux-gnu/libcrypt.so.1`, e assim por diante. Atalhos criam múltiplos caminhos para acessar o mesmo conteúdo, o que simplifica a organização de sistemas com muitos arquivos replicados (arquivos de configuração, ícones de aplicações, bibliotecas compartilhadas, etc).

25.5 Implementação de atalhos

Como apresentado na Seção 25.4, atalhos são entradas no sistema de arquivos que apontam para outras entradas. Existem basicamente duas abordagens para a implementação de atalhos:

Atalho simbólico (*soft link*): é implementado como um pequeno arquivo de texto contendo uma *string* com o caminho até o arquivo original (pode ser usado um caminho simples, absoluto ou relativo à posição do atalho). Como o caminho é uma *string*, o "alvo" do atalho pode estar localizado em outro dispositivo físico (outro disco ou uma unidade de rede). O arquivo apontado e seus atalhos simbólicos são totalmente independentes: caso o arquivo seja movido, renomeado ou removido, os atalhos simbólicos apontarão para um local inexistente; neste caso, diz-se que aqueles atalhos estão "quebrados" (*broken links*).

Atalho físico (*hard link*): várias referências do arquivo no sistema de arquivos apontam para a mesma localização do dispositivo físico onde o conteúdo do arquivo está de fato armazenado. Normalmente é mantido um contador de referências a esse conteúdo, indicando quantos atalhos físicos apontam para o mesmo: somente quando o número de referências ao arquivo for zero, aquele conteúdo poderá ser removido do dispositivo. Como são usadas referências à posição do arquivo no dispositivo, atalhos físicos só podem ser feitos para arquivos dentro do mesmo sistema de arquivos (o mesmo volume).

A Figura 25.4 traz exemplos de implementação de atalhos simbólicos e físicos em um sistema de arquivos UNIX. As entradas de diretórios indicadas como "`L`" correspondem a atalhos simbólicos (*links*). Nessa figura, pode-se observar que a

entrada `/bin/sh` é um atalho simbólico para o arquivo `/bin/bash`. Atalhos simbólicos podem apontar para diretórios (`/lib` → `/usr/lib`) ou mesmo para outros atalhos (`/usr/bin/shell` → `/bin/sh`). Esses atalhos são transparentes para as aplicações, ou seja: uma aplicação que acessar o arquivo `/usr/bin/shell` estará na verdade acessando `/bin/bash`.

Por outro lado as entradas `/bin/cp` e `/usr/bin/copy` apontam para o mesmo conteúdo no disco, por isso são consideradas atalhos físicos a esse conteúdo. Atalhos físicos geralmente só podem ser feitos para arquivos dentro do mesmo sistema de arquivos (mesmo volume) e não são permitidos atalhos físicos para diretórios[1].

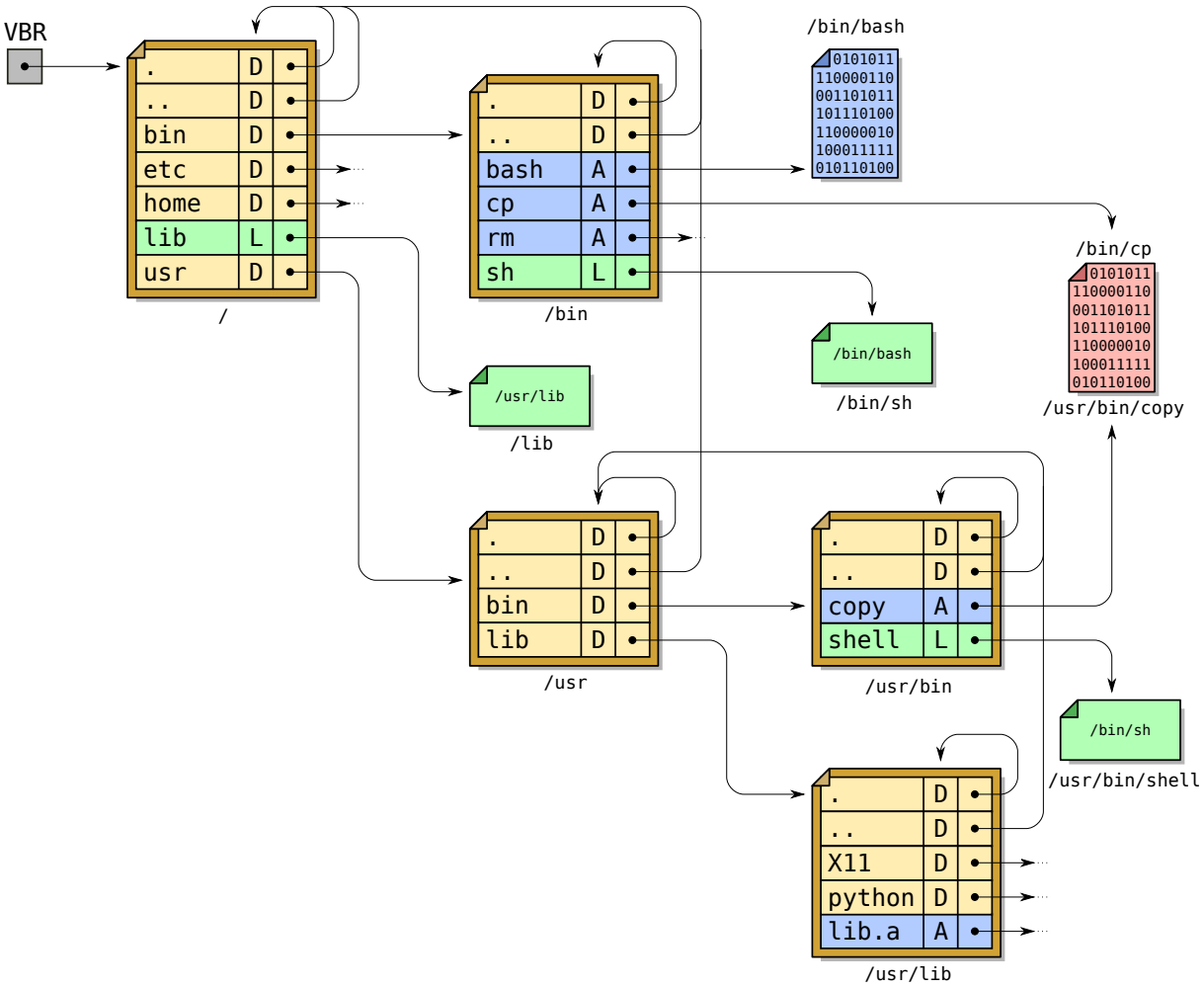

Figura 25.4: Atalhos simbólicos e físicos a arquivos em UNIX.

Em ambientes Windows, o sistema de arquivos NTFS suporta ambos os tipos de atalhos (embora atalhos simbólicos só tenham sido introduzidos no Windows Vista), com limitações similares.

[1] Atalhos físicos de diretórios transformariam a árvore de diretórios em um grafo, tornando mais complexa a implementação de rotinas que percorrem o sistema de arquivos recursivamente, como utilitários de *backup* e de gerência.

25.6 Tradução dos caminhos de acesso

A estrutura do sistema de arquivos em diretórios facilita a organização dos arquivos pelo usuário, mas complica a implementação da abertura de arquivos pelo sistema operacional. Para abrir um arquivo, o núcleo deve encontrar a localização do mesmo no dispositivo de armazenamento, a partir do nome de arquivo informado pelo processo. Para isso, é necessário percorrer o caminho do arquivo até encontrar sua localização, em um procedimento denominado *localização de arquivo* (*file lookup*).

Por exemplo, para abrir o arquivo `/home/daniel/arq1` da Figura 25.3 em um sistema UNIX (com alocação indexada) seria necessário executar os seguintes passos, ilustrados também na figura 25.5[2]:

1. Descobrir a localização do *i-node* do diretório raiz (/); essa informação pode estar no VBR (*Volume Boot Record*) do volume ou pode ser um valor fixo (por exemplo, o *i-node* 0).
2. Ler o *i-node* de / para:
   (a) Verificar se o processo tem permissão de acessar o conteúdo de /.
   (b) Descobrir em que bloco(s) está localizado o conteúdo de / (ou seja, a tabela de diretório contida em /).
3. Ler o conteúdo de / para encontrar o número do *i-node* correspondente à entrada `/home`.
4. Ler o *i-node* de `/home` para:
   (a) Verificar se o processo tem permissão de acessar o conteúdo de `/home`.
   (b) Descobrir em que bloco(s) está localizado o conteúdo de `/home`.
5. Ler o conteúdo de `/home` para encontrar o número do *i-node* correspondente à entrada `/home/daniel`.
6. Ler o *i-node* de `/home/daniel` para:
   (a) Verificar se o processo tem permissão de acessar o conteúdo de `/home/daniel`.
   (b) Descobrir em que bloco(s) está localizado o conteúdo de `/home/daniel`.
7. Ler o conteúdo de `/home/daniel` para encontrar o número do *i-node* correspondente à entrada `/home/daniel/arq1`.
8. Devolver o número do *i-node* encontrado.

O número do *i-node* devolvido no passo 8 corresponde ao arquivo desejado (`/home/daniel/arq1`), que pode então ser usado pelo restante do sistema para as operações envolvendo esse arquivo.

[2]Para simplificar, foram omitidos os tratamentos de erro.

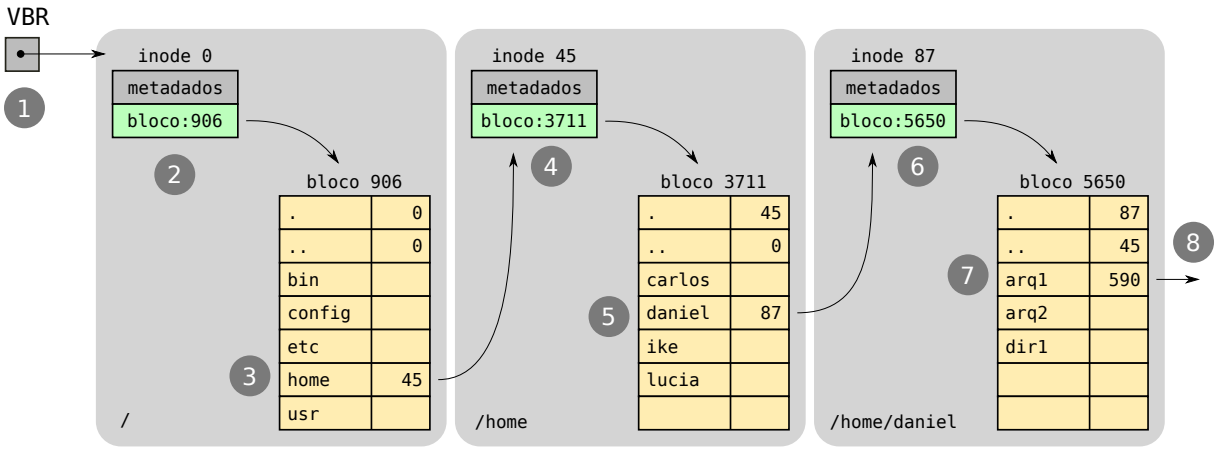

Figura 25.5: Resolução de nomes de arquivos.

Neste exemplo, foram necessárias 6 leituras no disco (passos 2-7) somente para localizar o *i-node* do arquivo `/home/daniel/arq1`, cujo caminho de acesso tem 3 níveis. Um arquivo com um caminho mais longo, como `/usr/share/texlive/texmf/tex/generic/pgf/graphdrawing/lua/pgf/gd/doc/ogdf/energybased/multilevelmixer/SolarMerger.lua` (16 níveis), apresenta um custo de localização bem mais elevado.

Para atenuar o custo de localização e melhorar o desempenho geral do acesso a arquivos, é mantido em memória um cache de entradas de diretório localizadas recentemente, denominado *cache de resolução de nomes* (*name lookup cache*). Cada entrada desse cache contém um nome absoluto de arquivo ou diretório e o número do *i-node* correspondente (ou outra informação que permita localizá-lo no dispositivo físico). A Tabela 25.1 representa as entradas do cache de nomes relativas ao exemplo apresentado nesta seção (Figura 25.5).

caminho	***i-node***
`/home/daniel/arq1`	590
`/home/daniel`	87
`/home`	45
`/`	0

Tabela 25.1: Conteúdo parcial do cache de resolução de nomes.

Esse cache geralmente é organizado na forma de uma tabela *hash* e gerenciado usando uma política LRU (*Least Recently Used*). A consulta ao cache é feita de forma iterativa, partindo do nome completo do arquivo e removendo a última parte a cada consulta, até encontrar uma localização conhecida. Considerando o conteúdo de cache da Tabela 25.1, eis alguns exemplos de consultas:

`/home/daniel/arq1`	→	*inode* 590
`/home/daniel/arq2`	→	não
`/home/daniel`	→	*inode* 87
`/home/maziero/imagens/foto.jpg`	→	não
`/home/maziero/imagens`	→	não
`/home/maziero`	→	não
`/home`	→	*inode* 45
`/usr/bin/bash`	→	não
`/usr/bin`	→	não
`/usr`	→	não
`/`	→	*inode* 0

A resolução de caminho de acesso para um atalho físico segue exatamente o mesmo procedimento, pois um atalho físico é simplesmente um caminho alternativo para um arquivo. Por outro lado, se o caminho a resolver apontar para um atalho simbólico, é necessário ler o conteúdo desse atalho e aplicar o procedimento de resolução sobre o novo caminho encontrado.

Exercícios

1. Quais as principais estruturas de diretórios empregadas em sistemas operacionais?

2. Do ponto de vista lógico, quais as principais diferenças entre a estrutura de diretórios Unix e Windows?

3. Explique os tipos de referências possíveis a arquivos em uma estrutura de diretórios.

4. Apresente e explique os dois principais tipos de atalhos em um sistema de arquivos.

5. Descreva a lista de todos os acessos a disco realizados para localizar o arquivo `/home/aluno/marcos/so/projetos/hello.c`, considerando um cache de resolução de nomes inicialmente vazio.

6. Em um sistema de arquivos, considere o arquivo F = `/bin/chrome` e o atalho L = `/usr/bin/browser`, sendo que $L \rightarrow F$. Considerando a chamada de sistema `open(L)`, descreva a sequência de acessos a disco e verificação de permissões realizada pelo sistema operacional para **localizar o conteúdo** apontado por L no disco, nas seguintes situações:

   (a) L é um **atalho físico**

(b) L é um **atalho simbólico**

7. Construa uma tabela com o conteúdo **final** do cache de resolução de caminhos de arquivos do núcleo após a abertura de todos os arquivos abaixo. Use números de *i-nodes* simbólicos (i_1, i_2, ...).

   - `/usr/bin/bash`
   - `/home/aluno/marcos/docs/hello.c`
   - `/home/aluno/maria/imagens/foto.jpg`
   - `/home/prof/joao/.bashrc`
   - `/usr/lib/lic.6.so`

Referências

R. Russell, D. Quinlan, and C. Yeoh. Filesystem Hierarchy Standard, January 2004.