Diretórios e atalhos

PROFESSOR: Jheymesson A. Cavalcanti

DISCIPLINA: SISTEMAS OPERACIONAIS II

1. Contexto

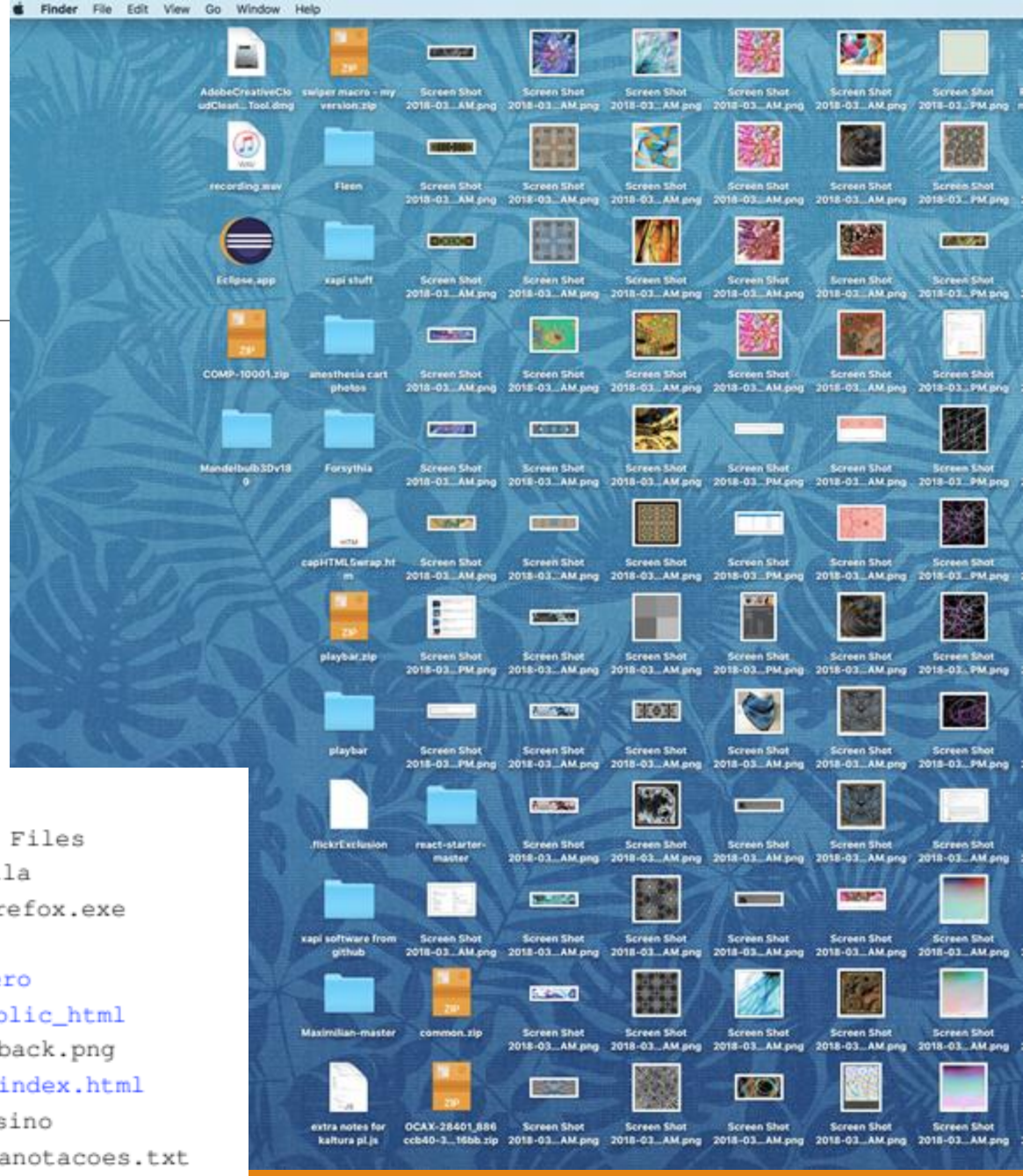
A quantidade de arquivos em sistemas atuais pode ser muito grande;

Embora o sistema operacional possa lidar facilmente com uma grande quantidade de arquivos, a tarefa não é tão simples quando falamos dos usuários;

Para simplificar a gestão dos arquivos, é possível organizá-los em grupos e hierarquias usando diretórios e atalhos;

Exemplo:

- Caminho de acesso em um sistema Windows.



1. Contexto

Por conta disso, além de arquivos, um sistema de arquivos também precisa implementar outros elementos;

Esses elementos são os próprios diretórios e atalhos;

Para isso, é necessário utilizar mecanismos eficientes para permitir a localização de arquivos nas árvores de diretórios, que podem ser imensas.

2. Diretórios

Diretório:

- Também chamado de pasta ou folder, representa um contêiner de arquivos e de outros diretórios;

Assim como os arquivos, os diretórios têm nome e atributos;

Estes também são usados na localização e acesso ao seu conteúdo.

2. Diretórios

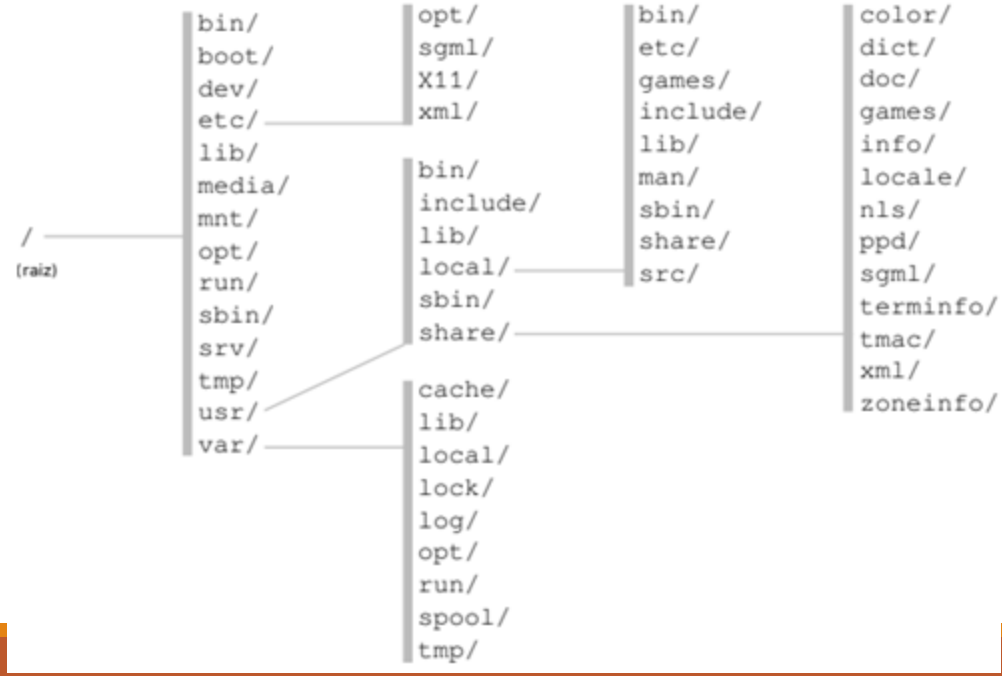
Estrutura do diretório:

Cada sistema de arquivos possui um diretório principal, denominado diretório raiz (root directory);

O uso de diretórios permite construir uma estrutura hierárquica (em árvore) de armazenamento dentro de um volume, sobre a qual os arquivos são organizados;

Exemplo:

- Árvore de diretório típica de um sistema Linux.



2. Diretórios

Estrutura do diretório:

A maioria dos sistemas operacionais implementa o conceito de diretório de trabalho ou diretório corrente de um processo (working directory);

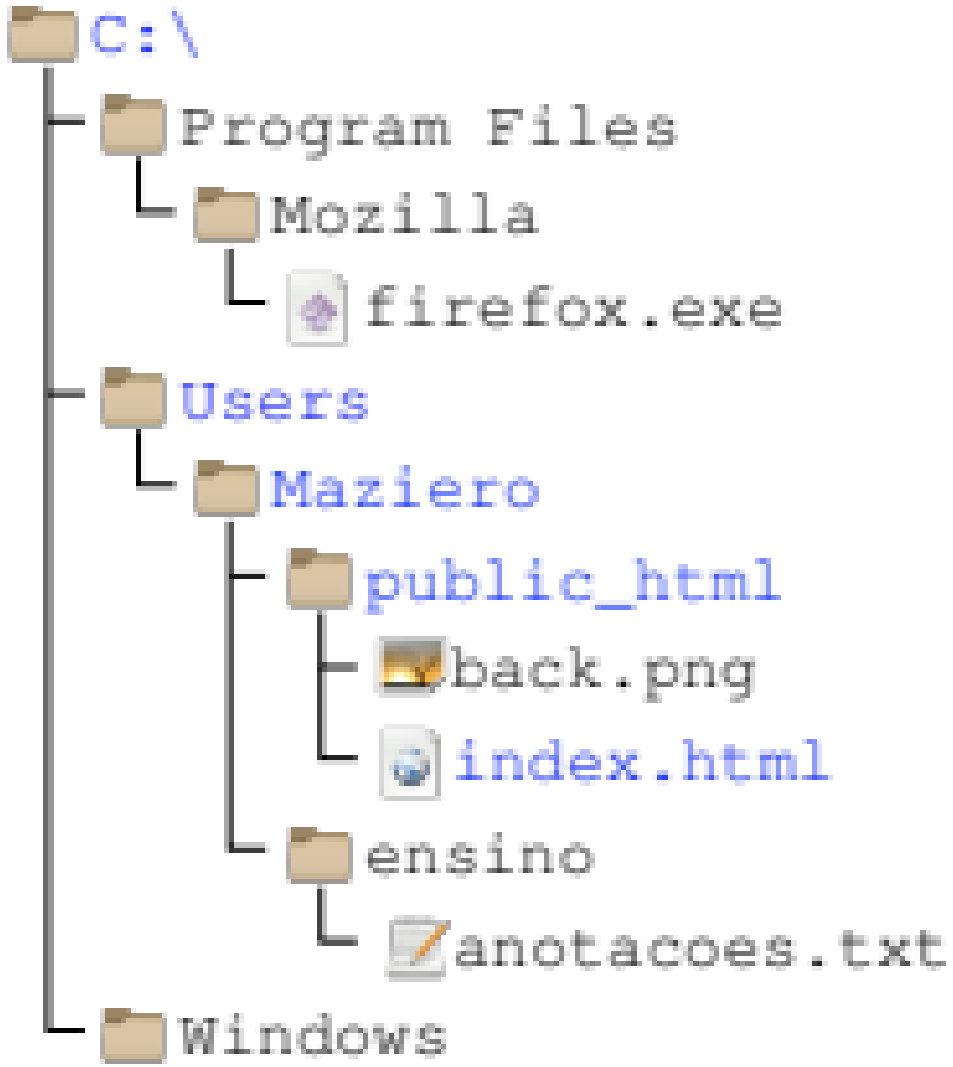
Isto significa que, ao ser criado, cada novo processo é associado a um diretório;

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

3. Caminhos de acesso

Em um sistema de arquivos, os arquivos estão dispersos ao longo da hierarquia de diretórios;



3. Caminhos de acesso

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 o levam até a raiz, separados por um caractere específico;

Cada elemento do caminho representa um nível de diretório, a partir da raiz;

Exemplo:

- Na imagem do slide anterior, o arquivo index.html teria o seguinte caminho de acesso: C:\Users\Maziero\public_html\index.html.

Obs:

- 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 “/”.

3. Caminhos de acesso

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
```

3. Caminhos de acesso

Referência absoluta:

- O caminho de acesso ao arquivo é indicado a partir do diretório raiz do sistema de arquivos;
- Em outras palavras, ele 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:

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

3. Caminhos de acesso

Referência relativa:

- O caminho de acesso ao arquivo tem como início o diretório de trabalho do processo;
- Ele indica subdiretórios ou diretórios anteriores, através de elementos “..”.
- Alguns exemplos:

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

4. Implementação de diretórios

Um diretório é implementado como um arquivo cujo conteúdo é uma relação de entradas (“tabela”);

Os tipos de entradas normalmente considerados nessa relação são:

- Arquivos normais;
- Outros diretórios;
- Atalhos;
- Entradas associadas a arquivos especiais.

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);

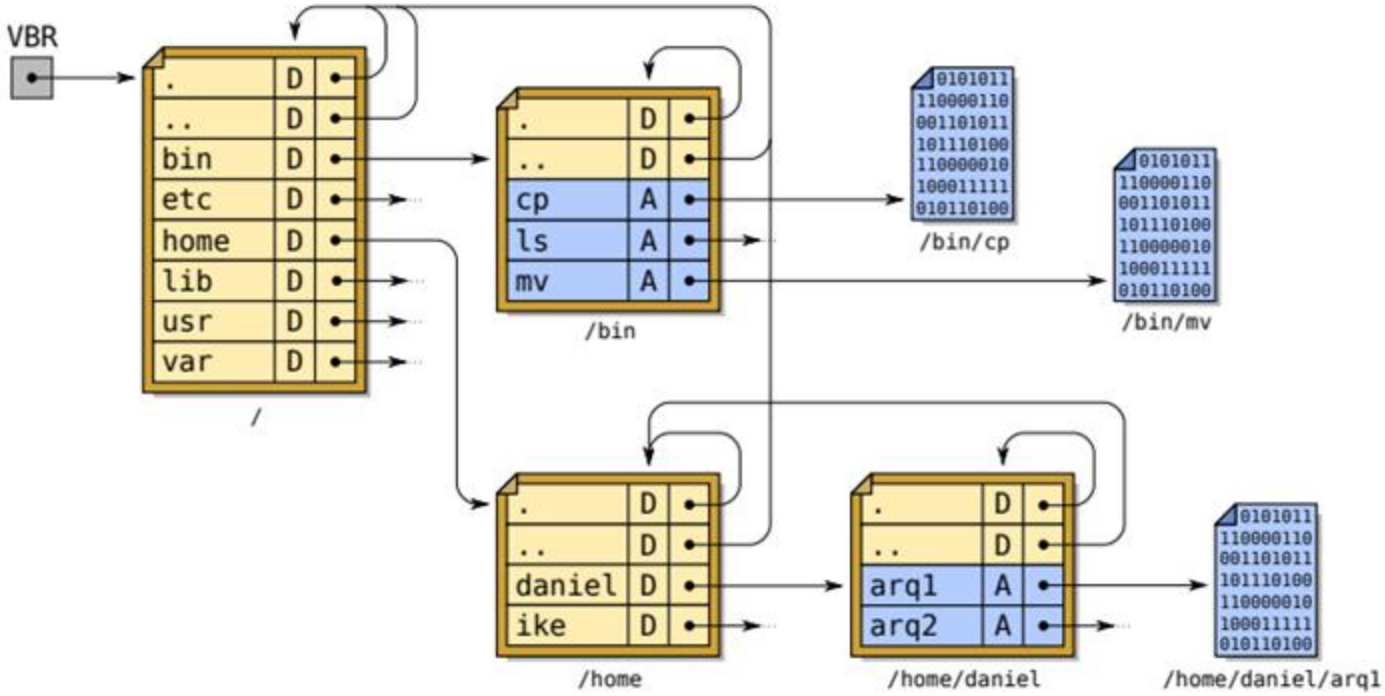
Obs:

- Deve ficar claro que um diretório não contém fisicamente os arquivos e subdiretórios, ele apenas os relaciona.

4. Implementaçã o de diretórios

Exemplo:

- Parte de uma estrutura de diretórios de um sistema UNIX;
- Aqui, o endereço do diretório raiz (/) é indicado por uma entrada específica no VBR (Volume Boot Record), dentro da área reservada do disco ou partição.



4. Implementação de diretórios

No exemplo do slide anterior, as entradas em cada diretório podem ser arquivos de dados (A) ou diretórios (D);

Em uma implementação real podem existir outras entradas, como atalhos (L - links) e arquivos especiais;

A informação sobre o tipo de cada entrada pode estar presente na própria tabela ou pode estar nos metadados do i-node da entrada correspondente;

Obs:

- Na figura, também observamos duas entradas usualmente definidas em cada diretório;
- A entrada "." (ponto), que representa o próprio diretório;
- 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.

4. Implementação de diretórios

Obs:

- 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;
- Isso ocorre porque 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.

5. Atalhos

Às vezes, é necessário que um mesmo arquivo ou diretório seja replicado em várias posições dentro do sistema de arquivos;

Nestes casos, seria mais econômico (em termos de espaço de armazenamento) armazenar apenas uma instância dos dados do arquivo no sistema de arquivos, em vez de várias;

Para isso, podemos criar referências indiretas (ponteiros) para essa instância, podendo o mesmo raciocínio ser aplicado a diretórios duplicados;

Essas referências indiretas a arquivos ou diretórios são denominadas atalhos (ou links).

5. Implementação de Atalhos

Existem basicamente duas abordagens para a implementação de atalhos:

Atalho simbólico (soft link):

É implementado como um pequeno arquivo de texto;

Este arquivo contém 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);

Obs:

- 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).

5. Implementação de Atalhos

Existem basicamente duas abordagens para a implementação de atalhos:

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;

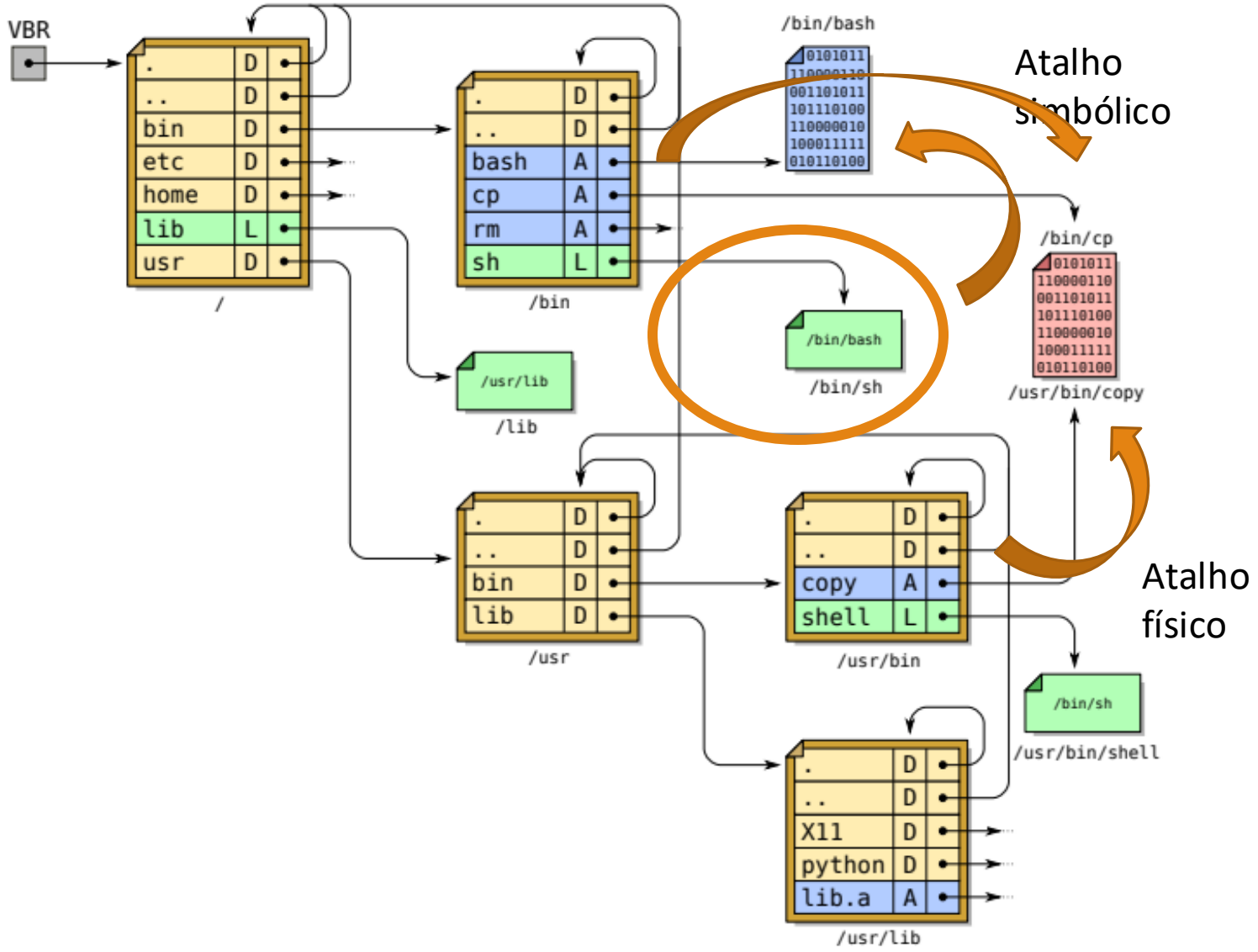
Normalmente é mantido um contador de referências a esse conteúdo, indicando quantos atalhos físicos apontam para o mesmo;

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).

5. Implementaçã o de Atalhos

Exemplos de atalhos simbólicos e físicos:



5. Implementação de Atalhos

Obs:

- Em ambientes Windows, o sistema de arquivos NTFS suporta ambos os tipos de atalhos.

6. Exemplo Linux

O Linux suporta várias dúzias de sistemas de arquivos;

MINIX
- O mais antigo e confiável;
- Bastante limitado em características (algumas datas não aparecem, máximo de 30 caracteres para nome de arquivos, etc...);
- Restrito em armazenamento (no máximo 64 Mb por sistema de arquivos);

Ext2
- O mais poderoso e popular sistema de arquivos nativo do Linux;
- Desenhado para ser facilmente compatível com os avanços das novas versões, sem a necessidade de criar novamente os sistemas de arquivos já existentes;

Ext3
- O sistema de arquivos ext3 é uma extensão de “journaling” (sistema de arquivos com diário) para o sistema de arquivos ext2 no Linux;
- O diário resulta em redução no tempo gasto recuperando um sistema de arquivos após uma queda de energia, e é portanto bastante utilizado em ambientes onde alta disponibilidade é importante;
- O ext3 é completamente compatível com os sistemas de arquivos ext2, de forma que é possível migrar um sistema de arquivos ext2 para ext3 e vice-versa;

Etc.

6. Exemplo Linux

FIGURA 10.23 Alguns diretórios importantes encontrados na maioria dos sistemas Linux.

Diretório	Conteúdos
bin	Programas binários (executáveis)
dev	Arquivos especiais para dispositivos de E/S
etc	Arquivos diversos do sistema
lib	Bibliotecas
usr	Diretórios de usuários

6. Exemplo Linux

FIGURA 10.27 Algumas chamadas de sistema relacionadas a arquivos. O código de retorno *s* é –1 se ocorrer algum erro; *fd* é um descritor de arquivo e *position* é um offset de arquivo. Os parâmetros são autoexplicativos.

Chamada de sistema	Descrição
fd = creat(nome, modo)	Uma maneira de criar um novo arquivo
fd = open(arquivo, como, ...)	Abre um arquivo para leitura, escrita ou ambos
s = close(fd);	Fecha um arquivo aberto
n = read(fd, buffer, nbytes)	Lê dados de um arquivo para um buffer
n = write(fd, buffer, nbytes)	Escreve dados de um buffer para um arquivo
posicao = lseek(fd, deslocamento, de-onde)	Move o ponteiro do arquivo
s = stat(nome, &buf)	Obtém a informação de estado do arquivo
s = fstat(fd, &buf)	Obtém a informação de estado do arquivo
s = pipe(&fd[0])	Cria um pipe
s = fcntl(fd, comando, ...)	Trava de arquivo e outras operações

6. Exemplo Linux

FIGURA 10.29 Algumas chamadas de sistema relacionadas a diretórios. O código de retorno *s* é –1 se ocorrer algum erro; *dir* identifica uma cadeia de diretórios e *entradir* é uma entrada de diretório. Os parâmetros são autoexplicativos.

Chamada de sistema	Descrição
s = mkdir(caminho, modo)	Cria um novo diretório
s = rmdir(caminho)	Remove um diretório
s = link(caminho_velho, caminho_novo)	Cria uma ligação para um arquivo existente
s = unlink(caminho)	Remove a ligação para um arquivo
s = chdir(caminho)	Troca o diretório atual
dir = opendir(caminho)	Abre um diretório para leitura
s = closedir(dir)	Fecha um diretório
entradir = readdir(dir)	Lê uma entrada do diretório
rewinddir(dir)	Rebobina um diretório de modo que ele possa ser lido novamente

6. Exemplo Linux (VFS)

O Linux adota uma abordagem usada em outros sistemas UNIX: o Sistema de Arquivos Virtual (VFS);

Este sistema habilita as aplicações a interagirem com sistemas de arquivos diferentes, implementadas em tipos diferentes de dispositivos, locais ou remotos;

O VFS define um conjunto de sistemas de arquivos básicos e as operações que são permitidas nessas abstrações;

Invocações das chamadas de sistema no VFS:

- Acessam as estruturas de dados VFS;
- Determinam o sistema de arquivos exato ao qual arquivo acessado pertence;
- E, através de ponteiros de função armazenados nas estruturas de dados VFS, invocam a operação correspondente no sistema de arquivos especificado.

6. Exemplo Linux (VFS)

As quatro estruturas de sistemas de arquivos principais suportadas pelo VFS são:

FIGURA 10.30 Abstrações de sistemas de arquivos fornecidos pelo VFS.

Objeto	Descrição	Operação
Superbloco	Sistema de arquivos específico	read_inode, sync_fs
Dentry	Entrada de diretório, componente único de um caminho	create, link
I-nodo	Arquivo específico	d_compare, d_delete
Arquivo	Arquivo aberto associado a um processo	read, write

6. Exemplo Linux (VFS)

Superbloco:

- Contém informações críticas a respeito do layout do sistema de arquivos.

I-nodo:

- Descreve um arquivo;
- No Linux, os diretórios e os dispositivos também são representados como arquivos, desse modo eles terão i-nodos correspondentes.

Dentry:

- Estrutura de dados que representa uma entrada de diretório;
- Essa estrutura de dados é criada dinamicamente pelo sistema de arquivos.

6. Exemplo Linux (VFS)

Arquivo:

- É uma representação na memória de um arquivo aberto;
- É criada em resposta à chamada de sistema open;
- Ela suporta operações como read, write, sendfile, lock e outras chamadas de sistema.

6. Exemplo Linux (VFS)

Obs:

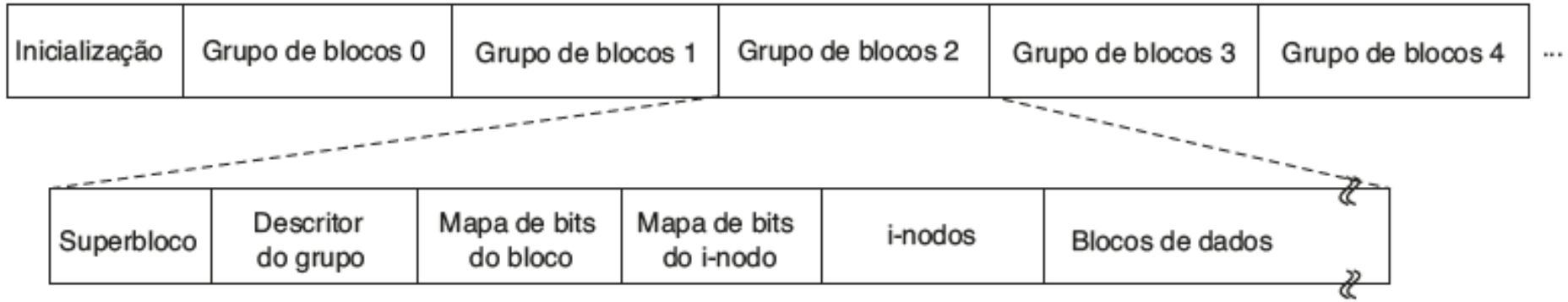
- Os sistemas de arquivos reais implementados por baixo do VFS não precisam usar exatamente as mesmas abstrações e operações internamente;
- Eles devem, no entanto, implementar operações de sistemas de arquivos semanticamente equivalentes àquelas especificadas com os objetos VFS.

7. Sistema de arquivos Ext2 do Linux

É um dos sistemas de arquivos em disco mais populares usados no Linux;

Uma partição de disco Linux Ext2 contém um sistema de arquivos com o layout mostrado na Figura abaixo:

FIGURA 10.31 Layout de disco do sistema de arquivos ext2 do Linux.



7. Sistema de arquivos Ext2 do Linux

O bloco 0 não é usado pelo Linux e contém código para inicialização do computador;

Seguindo o bloco 0, a partição do disco é dividida em grupos de blocos, desconsiderando onde vão cair os limites do cilindro;

Cada grupo é organizado da seguinte forma:

O primeiro bloco é o **superbloco**;

Ele contém informações sobre o layout do sistema de arquivos, incluindo:

- O número de i-nodos;
- O número de blocos de disco;
- O começo da lista de blocos de disco livres (em geral algumas centenas de entradas).

7. Sistema de arquivos Ext2 do Linux

Em seguida, vem o **descritor** do grupo;

Ele contém informações sobre:

- A localização dos mapas de bits;
- O número de blocos livres;
- i-nodos no grupo;
- O número de diretórios no grupo.

Essa informação é importante, tendo em vista que Ext2 tenta disseminar os diretórios uniformemente através do disco.

7. Sistema de arquivos Ext2 do Linux

Dois **mapas de bits** são usados para controlar os blocos livres e i-nodos livres, respectivamente;

Cada mapa tem um bloco de comprimento;

Em seguida, aparecem os próprios **i-nodos**;

Eles são numerados de 1 até algum máximo;

Cada i-nodo tem 128 bytes de comprimento e descreve exatamente um arquivo;

Um i-nodo contém informações suficientes para localizar todos os blocos de disco que contêm os dados do arquivo.

7. Sistema de arquivos Ext2 do Linux

Seguindo os **i-nodos** aparecem os blocos de dados;

Todos os arquivos e diretórios estão armazenados nos blocos de dados;

Se um arquivo ou diretório consiste em mais do que um bloco, os blocos não precisam ser contíguos no disco.