Implementação do Sistema de Arquivos

Usuários estão interessados

- Como mudar nomes
- Quais operações são permitidas
- Como é a árvore de diretórios
- Interface

Implementadores em:

- Como arquivos e diretórios são armazenados no disco
- Como o espaço em disco é gerenciado
- Como fazer tudo funcionar de forma eficiente e confiável

Implementação do Sistema de Arquivos

- Cada disco pode ser dividido em várias partições, com um sistema de arquivos ***independente*** para cada partição;
- Setor 0 do disco: *MBR (Master Boot Record)* – registro mestre de inicialização;
- No fim do MBR: *tabela de partição*, com os endereços iniciais e finais de cada partição (uma partição marcada como ativa)
- Na **inicialização** do computador:
  - ✓ BIOS lê e executa o MBR;
  - ✓ MBR localiza a partição ativa;
  - ✓ MBR lê e executa o primeiro bloco da partição (de inicialização);
  - ✓ Programa no primeiro bloco carrega o SO contido na partição;

Implementação do sistema de arquivos

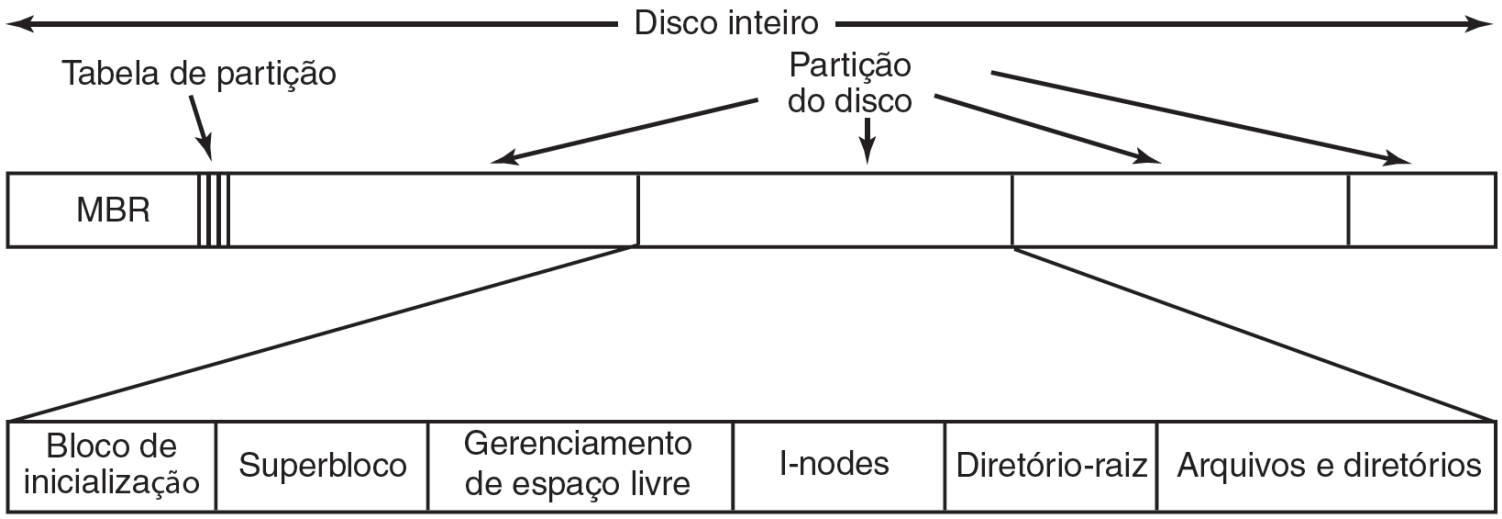

Figura 4.7 Uma organização possível para um sistema de arquivos.

Implementação do Sistema de Arquivos

Organização possível para uma partição:

Superbloco:

- ✓ contém os principais parâmetros do Sistema de Arquivos;
- ✓ lido na memória quando o computador é iniciado ou quando o Sistema de Arquivos é usado pela primeira vez;
  - ❑ Número para indicar o tipo de SA;
  - ❑ Número de blocos no SA;
  - ❑ Informações administrativas

Implementação do Sistema de Arquivos

Organização possível para uma partição (cont):

- Informações sobre blocos livres:
  - ✓ **Mapa de bits**
  - ✓ **Lista de ponteiros**
- I-nodes: arranjo de estruturas de dados com infos sobre um arquivo;
- Diretório-raiz;
- Outros arquivos e diretórios

Implementação de arquivos: alocação contígua

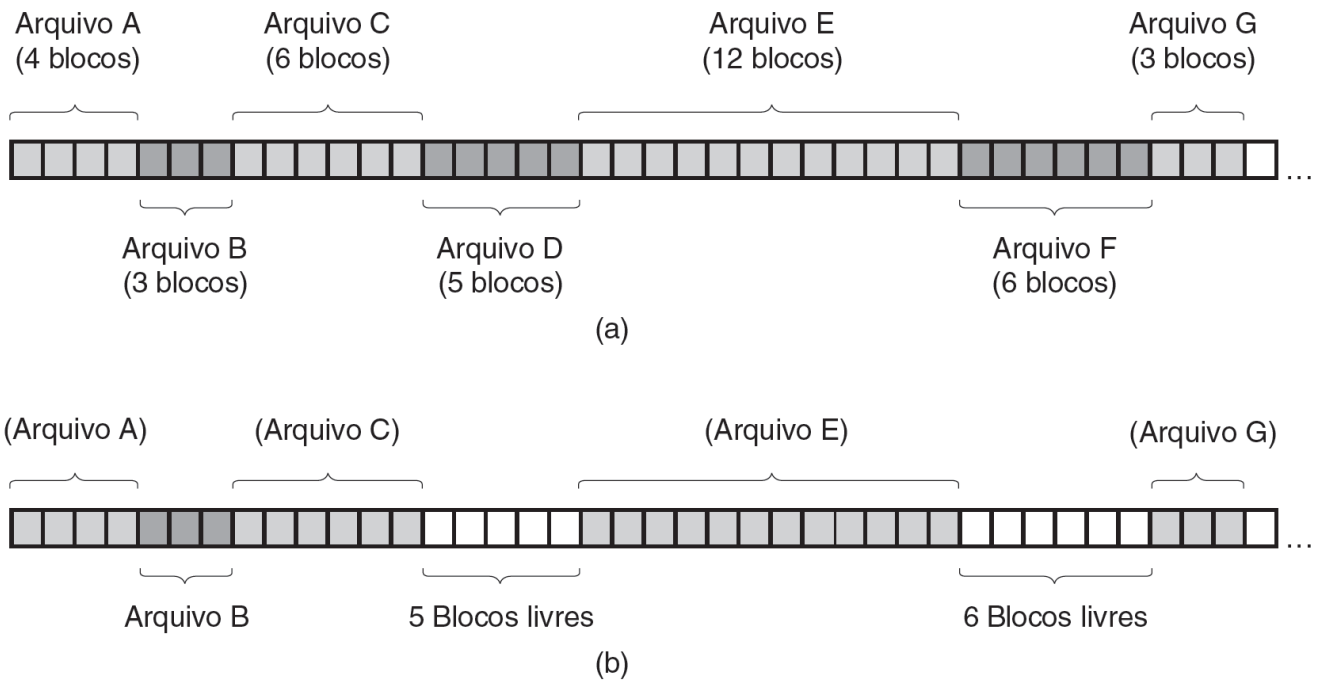

Figura 4.8 (a) A alocação contígua do espaço em disco para sete arquivos. (b) O estado do disco depois de os arquivos *D* e *F* terem sido removidos.

Alocação contígua

Vantagens

- Simples de implementar: endereço do primeiro bloco e número de blocos do arquivo;
- Desempenho excelente: basta um único ***seek;***

Desvantagem

Fragmentação do disco;

Obs: 1.compactação é custosa

2. reutilização das lacunas demandaria o gerenciamento de uma lista de lacunas

⇒ necessário saber o tamanho do arquivo novo para alocá-lo

Ex: editor de texto??

Alocação contígua

Exceção:

Ex1: Em CD-ROMs todos os tamanhos dos arquivos já são conhecidos e lista de arquivos não se altera

Ex2: Em DVDs é necessário dividir os arquivos (~4.5 GB) em partes (*extensões*)

Porque?

UDF (Universal Disk Format) - utiliza um limite de 30 bits (1 GB) para informar o tamanho do arquivo

Alocação por lista encadeada

Implementação:

A primeira palavra de cada bloco é usada como ponteiro para localizar o próximo bloco. O restante do bloco é usado para os dados.

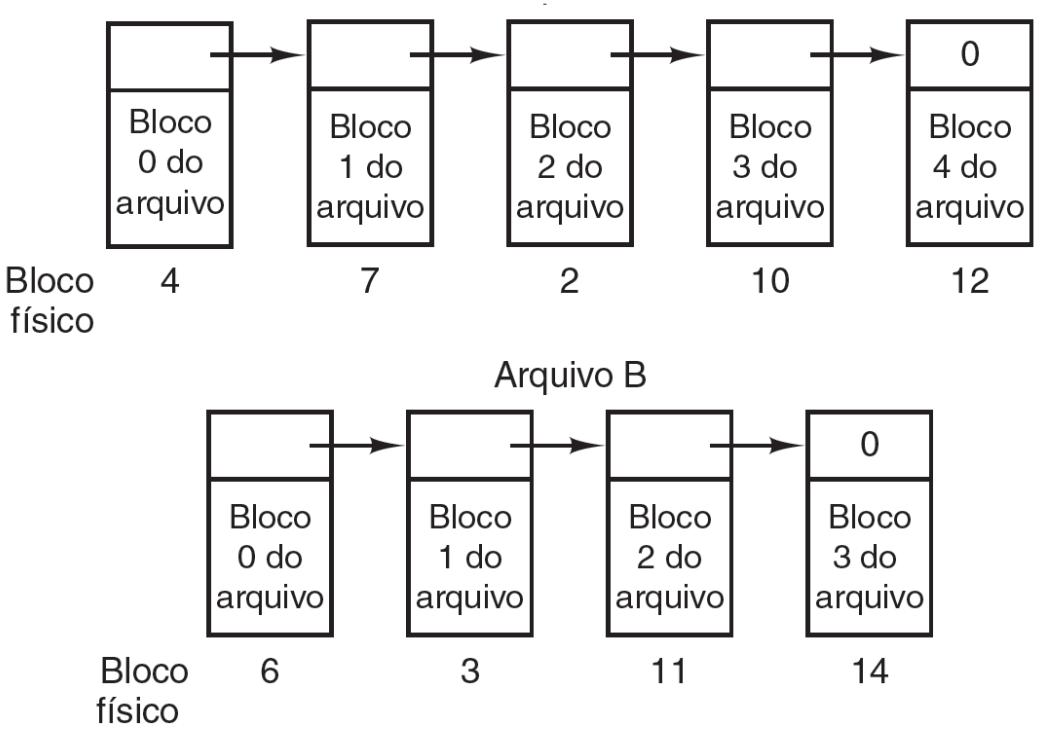

Figura 4.9 Armazenamento de um arquivo como uma lista encadeada de blocos de disco.

Alocação por lista encadeada

Vantagens:

- **Nenhum espaço é perdido na fragmentação**
- **Basta armazenar a entrada do primeiro bloco**

Desvantagens:

- **Acesso aleatório lento (1 → 2 → 3 → …… → n-1 → n → …)**
- **Quantidade de dados de um bloco não é mais uma potência de dois pois os ponteiros ocupam alguns bytes do bloco!!! Alguns programas lêem e escrevem em tamanhos de blocos**

  ⇒ concatenação de dois blocos! ⇒ sobrecarga extra

Lista encadeada usando uma tabela na memória

Implementação:

As palavras dos ponteiros de cada bloco são colocadas em uma *tabela na memória* (*FAT – file allocation table*)

Vantagens:

- ✓ Acesso aleatório mais rápido (encadeamento está todo na memória, apesar de seguir o mesmo princípio)
- ✓ Todo bloco fica disponível para dados

Desvantagem:

Toda a tabela deve estar na memória o tempo todo (tamanho pode chegar a 3 GB):

Ex: com um disco de 1 TB e blocos de 1 KB ⇒ 1 G blocos e cada entrada da tabela ocupando, no mínimo, 3 bytes

Lista encadeada usando uma tabela na memória

Arquivo A:

ocupa os blocos 4, 7 2, 10, 12

Arquivo B:

ocupa os blocos 6, 3, 11, 14

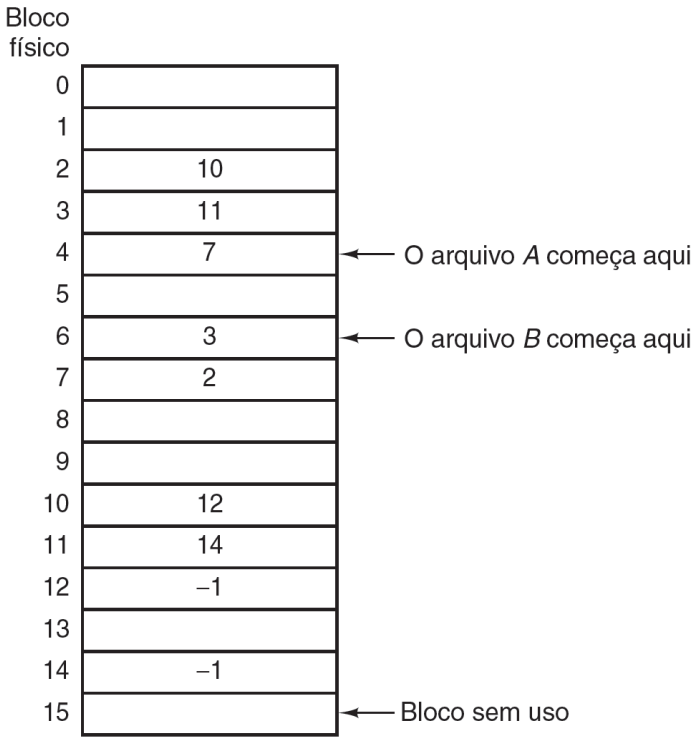

Figura 4.10 Alocação por lista encadeada usando uma tabela de alocação de arquivos na memória principal.

I-nodes

Implementação:

Associa a cada arquivo uma estrutura de dados chamada ***i-node*** (lista os atributos de cada arquivo e seus endereços em disco)

Vantagens:

- Somente precisa estar na memória quando o arquivo está aberto
- Se k arquivos podem estar abertos simultaneamente e cada i-node ocupa n bytes $\Rightarrow$ **k.n bytes ocupados** $\Rightarrow$ espaço muito menor do que o espaço ocupado pela tabela de arquivos

Problema:

Cada I-node tem espaço para um número fixo de endereços do arquivo. E se o arquivo crescer?

Solução: Reservar último endereço para um bloco extra contendo mais endereços de blocos

I-nodes

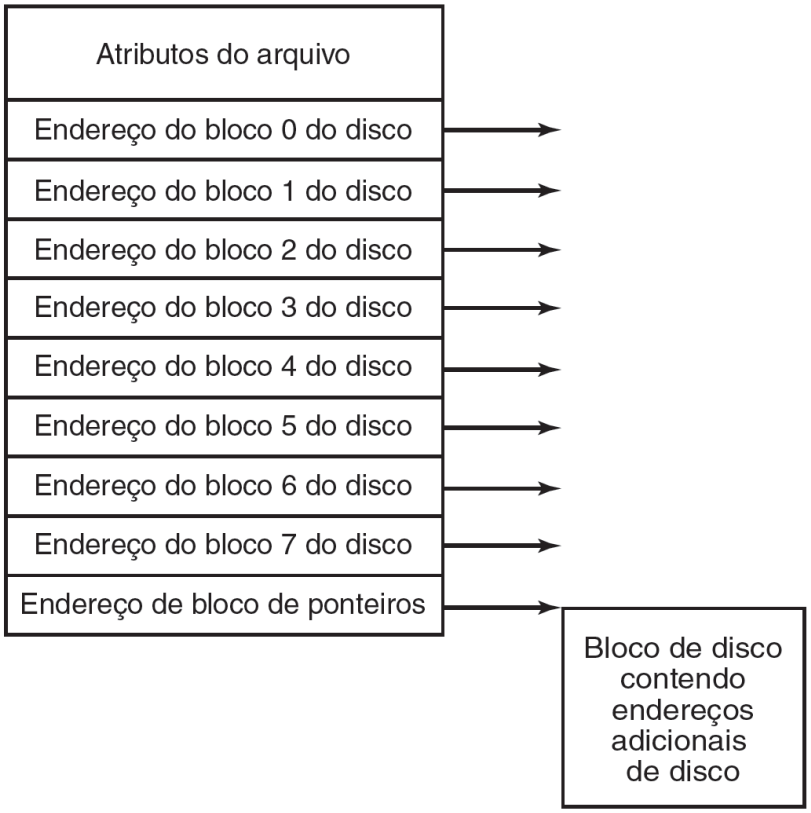

Figura 4.11 Um exemplo de i-node.

Obs: pode-se ter mais de um desses blocos ou um bloco desses apontar para outros blocos do mesmo tipo

Implementação de diretórios

Para abrir um arquivo, é necessário:

- nome do caminho, usado para localizar a entrada do diretório
- entrada do diretório fornece a informação necessária para encontrar os blocos de disco. Dependendo da implementação, pode ser:
  - ✓ ***endereço inicial (no disco) de todo o arquivo (qdo a alocação é contígua);***
  - ✓ ***o nº do primeiro bloco (listas encadeadas, sem tabela ou com tabela); ou***
  - ✓ ***o nº do I-NODE;***

Implementação de diretórios

Função principal do sistema de diretórios:

Mapear o nome ASCII do arquivo na informação necessária para localizar seus dados

Questão: onde os atributos dos arquivos (data, dono, proteção) devem ser armazenados?

Sol. 1: armazenar na entrada do diretório – Windows (Fig. 4.12a)

diretório = lista de entradas de tamanho fixo, onde cada entrada possui:

- nome
- atributos
- um ou mais endereços de disco para os blocos de cada arquivo

Implementação de diretórios

Solução 2: para sistemas que usam i-nodes

atributos dos arquivos são armazenados nos próprios i-nodes ao invés de serem armazenados na entrada do diretório

Neste caso:

Cada entrada do diretório = nome do arquivo + nº do i-node

(fig 4.12b) – solução do UNIX

Implementação de diretórios

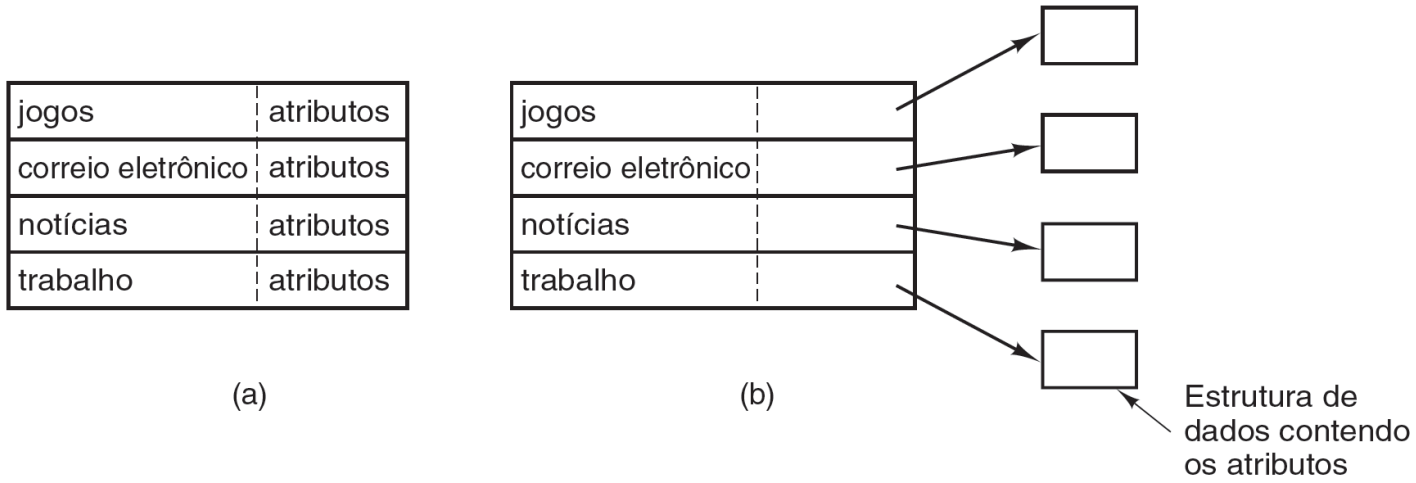

Figura 4.12 (a) Um diretório simples com entradas de tamanho fixo com os endereços de disco e atributos na entrada de diretório. (b) Um diretório no qual cada entrada se refere apenas a um i-node.

Implementação de diretórios

SO atuais aceitam nomes de arquivo com tamanhos variáveis: uma alternativa simples para implementar a solução 1:

nº fixo de 255 caracteres para o nome do arquivo:

Desvantagem:

Como cada campo reservado para o nome tem tamanho fixo, ocorre um grande desperdício de espaço no diretório, pois, normalmente, os nomes são muito menores do que 255 caracteres

Implementação de diretórios

Duas alternativas para o tamanho fixo:

tamanho da entrada + dados sobre o arquivo, num formato fixo
Fig 4.13a

Desvantagem 1:

quando o arquivo é removido fica uma lacuna de tamanho variável no diretório (próximo arquivo a entrar pode não caber nela).

Desvantagem 2:

se a entrada do diretório for grande, pode ocorrer uma falta de página durante a leitura de um nome do arquivo

Implementação de diretórios

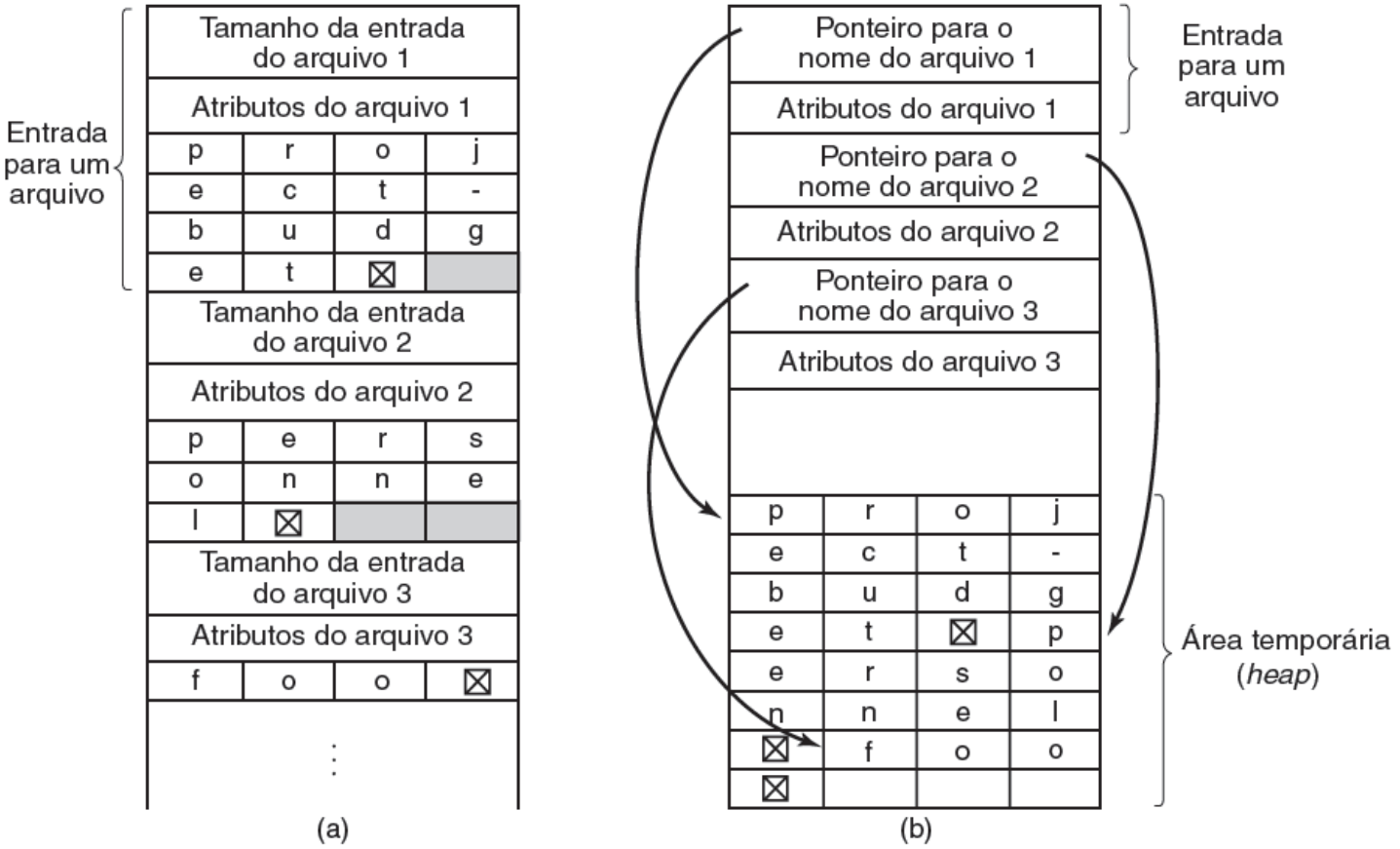

Figura 4.13 Duas maneiras de gerenciar nomes de arquivos longos em um diretório. (a) Sequencialmente. (b) Em uma área temporária.

Implementação de diretórios

Outra alternativa é ter as entradas do diretório de tamanho fixo.

Os nomes dos arquivos ficam juntos em uma área temporária separada da entrada do diretório, denominada HEAP **(fig. 4.13b)**

Vantagem 1: quando um arquivo é removido, o próximo a entrar sempre caberá, pois possui exatamente a mesma estrutura;

Vantagem 2: não há necessidade que os nomes de arquivos comecem alinhados por palavras e, portanto, não é preciso completar o nome dos arquivos com caracteres em branco **(fig. 4.13a)**

Obs: em todas as soluções apresentadas, os diretórios são pesquisados linearmente ⇒ para diretórios muito extensos, a busca por um arquivo pode ser lenta.

Implementação de diretórios

Alternativa 1 para a busca linear:

Usar uma tabela de espalhamento (tabela HASH) para cada diretório.

- Aplica-se alguma função ao nome do arquivo de tal forma que, para n entradas da tabela, ele seja mapeado entre os valores 0 e n-1.
- Se a entrada da tabela já estiver sendo usada, constrói-se uma lista encadeada a partir daquela entrada da tabela, unindo todas as entradas com mesmo valor de espalhamento.
- As entradas da tabela contêm um ponteiro para a entrada do arquivo.

  ⇒ **Busca:** nomedoarquivo → função → entrada da tabela

  → verica-se todas as entradas da lista encadeada.

  Se nome não consta da lista, arquivo não está presente

Implementação de diretórios

Alternativa 1:

Vantagem: busca mais rápida;

Desvantagem: gerenciamento mais complexo:

$\Rightarrow$ só vale a pena se número de arquivos no diretório for muito grande

Alternativa 2: colocar os resultados da busca em uma ***cache de busca*****,** que é verificada antes da procura em todo o diretório.

Obs:

cache só funciona se a maior parte das consultas envolver um número pequeno de arquivos.

Arquivos compartilhados

Às vezes é conveniente que um arquivo seja compartilhado entre vários diretórios de usuários diferentes **(fig. 4.14)**

⇒ **uso de uma ligação: LINK**

Problema quando B e C compartilham arquivo:

Dependendo da implementação do diretório, B deve ter uma cópia do endereço de disco do arquivo (hard link)

⇒ novos blocos adicionados ao arquivo por B ou C só serão visíveis no diretório que fez a adição.

1ª solução (adotada no UNIX):

Blocos de disco não são relacionados nos diretórios, mas em uma pequena estrutura de dados associada ao arquivo. Diretórios apontam para essa estrutura (I-NODE)

Arquivos compartilhados

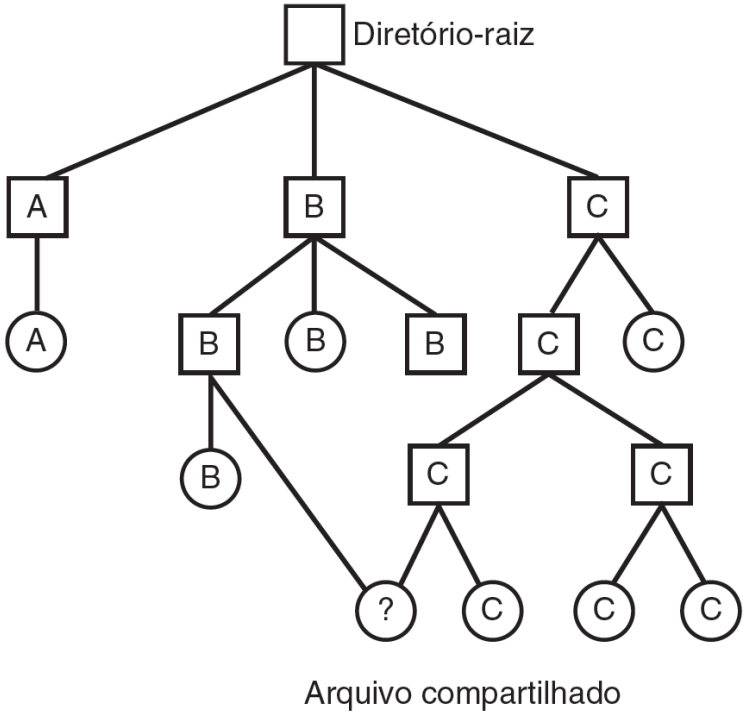

Figura 4.14 Sistema de arquivos contendo um arquivo compartilhado.

Arquivos compartilhados

Problema:

Exemplo: quando B se liga com o arquivo compartilhado:

- Proprietário: C
- Contador de ligações no i-node = 2 (B e C)
- Se C remove arquivo e limpa o i-node, B terá uma entrada apontando para i-node inválido
- Se i-node for atribuído a outro arquivo, B apontará para arquivo errado.

Solução: remover a entrada de diretório de C, deixar i-node intacto, com contador em 1 (nesse caso, C continua "pagando a conta"). Somente qdo B remover a ligação de seu diretório é que o contador irá para zero e o arquivo será removido.

Arquivos compartilhados

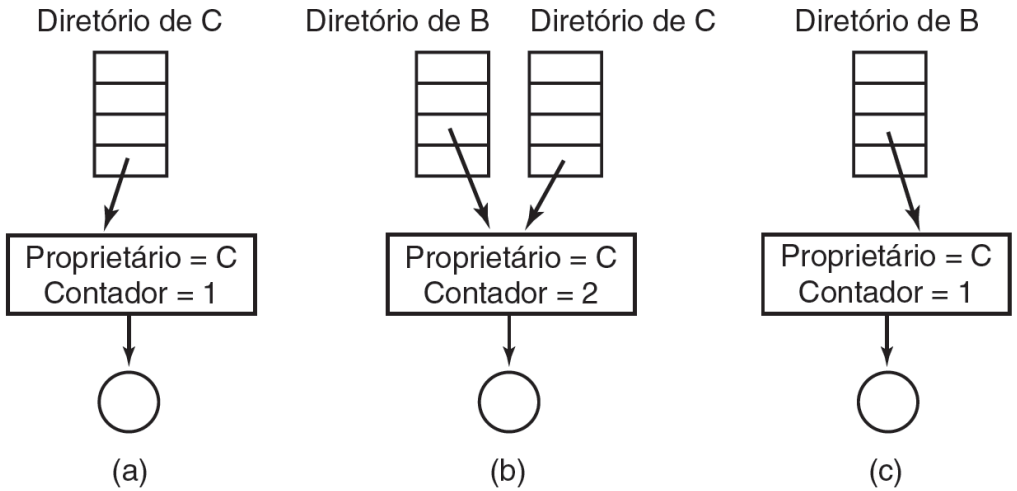

Figura 4.15 (a) Situação antes da ligação. (b) Depois da criação da ligação. (c) Depois que o proprietário original remove o arquivo.

Arquivos compartilhados

2ª solução:

- ✓ Sistema cria um novo arquivo, do tipo LINK e o insere no diretório B - contém apenas o nome do caminho do arquivo
- ✓ Quando B lê o arquivo ligado, o SO sabe que o mesmo é do tipo LINK, consulta o nome e lê o arquivo

⇒ ***ligação simbólica***

Arquivos compartilhados

Características da ligação simbólica:

- Somente o proprietário tem um ponteiro para o i-node (os usuários com LINK simbólico têm apenas nomes de caminhos)
- Quando o proprietário remove o arquivo, ele é destruído:

  ⇒ tentativas subsequentes de usar o LINK simbólico (caminho) falharão, pois o sistema será incapaz de localizar o arquivo
- Remover o link não afeta o arquivo.

Problemas da ligação simbólica:

- Sobrecarga extra necessária: ler o arquivo com caminho, analisar sintaticamente o mesmo até chegar ao i-node

  ⇒ **pode demandar acessos adicionais ao disco**

Arquivos compartilhados

- É necessário um i-node extra para cada link simbólico e um bloco de disco extra para armazenar o caminho, pois é um arquivo novo.

  Obs: se o nome do caminho for curto, ele poderá ser armazenado no próprio i-node

  Vantagem:

  Pode ligar arquivos em qualquer lugar do mundo:

  endereço da máquina + caminho do arquivo na máquina

  Problema com as ligações (simbólicas ou não):

  um programa que faz cópia de arquivos em fita pode ler os arquivos ligados várias vezes e copiar o mesmo arquivo várias vezes na fita

Exemplos de sistemas de arquivos

Sistemas de arquivos para CD-ROMs

- Projetados para escrita única;
- Não monitoram blocos livres (arquivos não podem ser liberados ou adicionados depois da fabricação)
- CD-R (*recordable*):
  - permite inclusão de arquivos ao seu final, depois da primeira gravação;
  - arquivos nunca são removidos;
  - espaço livre está localizado numa parte contígua, ao final do disco

O Sistema de Arquivos ISO 9660

Objetivo:

tornar possível a todo CD-ROM ser lido em qualquer computador, independentemente do tipo de SO

⇒ limitações ao sistema de arquivos para que SAs mais fracos, como o MS-DOS, pudessem ler os arquivos

O Sistema de Arquivos ISO 9660

Características:

- Espiral contínua com bits em uma sequência linear
- Busca pode ser transversal
- Bits são divididos em blocos/setores lógicos de 2352 bytes
- Alguns blocos especiais (preâmbulo, correção de erros, etc)
- Porção utilizável de cada bloco: 2048 bytes
- Em CDs de música pode-se converter tempo em número de blocos.
  Ex: 1s = 75 blocos

O Sistema de Arquivos ISO 9660

Formato de uma entrada de diretório

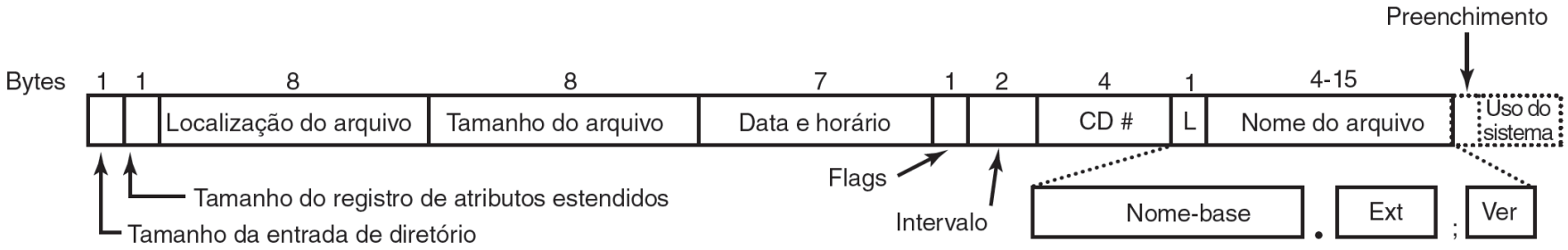

Figura 4.27 A entrada de diretório do padrão ISO 9660.

O Sistema de Arquivos ISO 9660

Formato de uma entrada de diretório

- **Tamanho da entrada do dir: número de arquivos do diretório**
- **Atributos estendidos: opcional**
- **Localização do arq: Bloco inicial do arquivo**
  - **Arq: arm. seq cont., logo: arq= bloco inicial + tamanho**
- **Ano, mês, dia, hora, min, seg, zona do fuso horário**

O Sistema de Arquivos ISO 9660

Formato de uma entrada de diretório

- **Flags diversos (ex: ocultar a entrada nas listagens)**
- **CD # : em qual CD-ROM arquivo está localizado. Entrada em um diretório de um CD-ROM pode se referir a um arquivo localizado em outro CD-ROM**
- **L – tamanho do nome do arquivo em bytes**
- **Nome do arquivo = nome base+ponto+extensão+ptvirg+versão**
- **Preenchimento: força que toda entrada de diretório seja um nº par de bytes**

O Sistema de Arquivos do MS-DOS

Características:

- Utilizado nos primeiros PCs e no Windows98;
- Apesar de antigo, tornaram-se muito utilizados em sistemas embarcados: câmeras digitais, MP3 players, iPod;
- Atualmente, existe em nº maior do que os dispositivos que usam o NTFS, mais moderno;

O Sistema de Arquivos do MS-DOS

Funcionamento:

- Para ler um arquivo, o MS-DOS faz uma chamada de sistema ***open*** para abri-lo. Esta chamada especifica um caminho até o diretório de trabalho (absoluto ou relativo);

- Caminho é procurado até o diretório final ser encontrado e carregado na memória;

- Diretório é vasculhado até o arquivo ser encontrado e, finalmente, ser aberto;

O Sistema de Arquivos do MS-DOS

Entrada de diretório tem tamanho fixo: 32 bytes

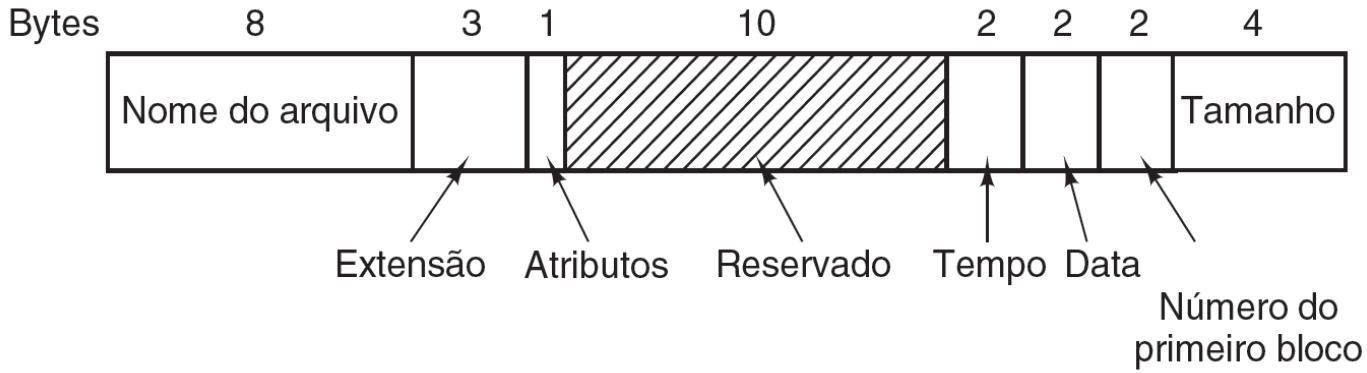

Figura 4.28 A entrada de diretório do MS-DOS.

O Sistema de Arquivos do MS-DOS

Atributos:

- ✓ indica se é somente para leitura
- ✓ se precisa ser feita cópia de segurança (bit 1 qdo modificado)
- ✓ se é um arquivo oculto
- ✓ se é um arquivo de Sistema (não podem ser removidos com um ***del***)

Campo ***tempo***: precisão de 2 seg – 2 bytes = 65536 valores (um dia = 86.400 segundos) – segundos (5 bits), minutos (6 bits) e horas (5 bits)

Campo ***data***: início em 1980 – dia (5 bits), mês (4 bits) e ano (7 bits) (no máximo até 2107)

Tamanho máximo (teórico) do arquivo (32 bits): 4 GB Real: 2 GB

O Sistema de Arquivos do MS-DOS

Monitoramento dos blocos de arquivos:

- Tabela de alocação de arquivos (FAT) na memória principal
- Entrada de diretório contém o número do primeiro bloco do arquivo
- Usado como índice em uma FAT de 64 K entradas na memória
- FAT-12, FAT-16 e FAT-32 (FAT-28): nº de bits ocupados por um endereço de disco;

Tamanho máximo de uma partição:

nº de entradas (endereços) possíveis * tamanho do bloco de dados.

Ex: FAT-16 com tam. Máximo de bloco de 32 KB ⇒ **64K end* 32 KB = 2 GB**

MS-DOS suporta, no máximo, 4 partições por disco ⇒ **8 GB de tam. do disco**

O Sistema de Arquivos do MS-DOS

Tamanho máximo de uma partição (cont):

- ✓ Aplicações comerciais: não há problemas
- ✓ Aplicações multimídia: 2 GB ≅ 9 minutos de video
- ✓ Maior vídeo capaz de ser armazenado = 36 minutos

2ª versão do Windows 95: FAT-32 – 28 bits

⇒ Limite teórico para o tam da partição: 2^28 * 2^15 (32KB) = 8 TB

Mas…

Limite é de 2 TB: internamente, o sistema monitora o tamanho da partição com blocos de 512 bytes e endereços de 32 bits:

2^32 * 2^9 = 2 TB

O Sistema de Arquivos do MS-DOS

Tamanho do bloco	FAT-12	FAT-16	FAT-32
0,5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Tabela 4.4 Tamanho máximo da partição para diferentes tamanhos de bloco. As células em branco representam combinações não permitidas.

O Sistema de Arquivos do UNIX V7

Características:

- Tem a forma de uma árvore, iniciando no diretório-raiz, adicionando ligações, formando um grafo orientado acíclico

- Nomes de arquivos de até 14 caracteres (quaisquer ASCII, com exceção do ' / ' e NULL (zero))

- Uma entrada de diretório contém uma entrada para cada arquivo naquele diretório, a saber:
  - 2 campos: **o nome do arquivo (14 bytes) e o nº do i-node para aquele arquivo (2 bytes)**

⇒ nº máximo de arquivos por SA: 2^16 inodes = 64 K arquivos

O Sistema de Arquivos do UNIX V7

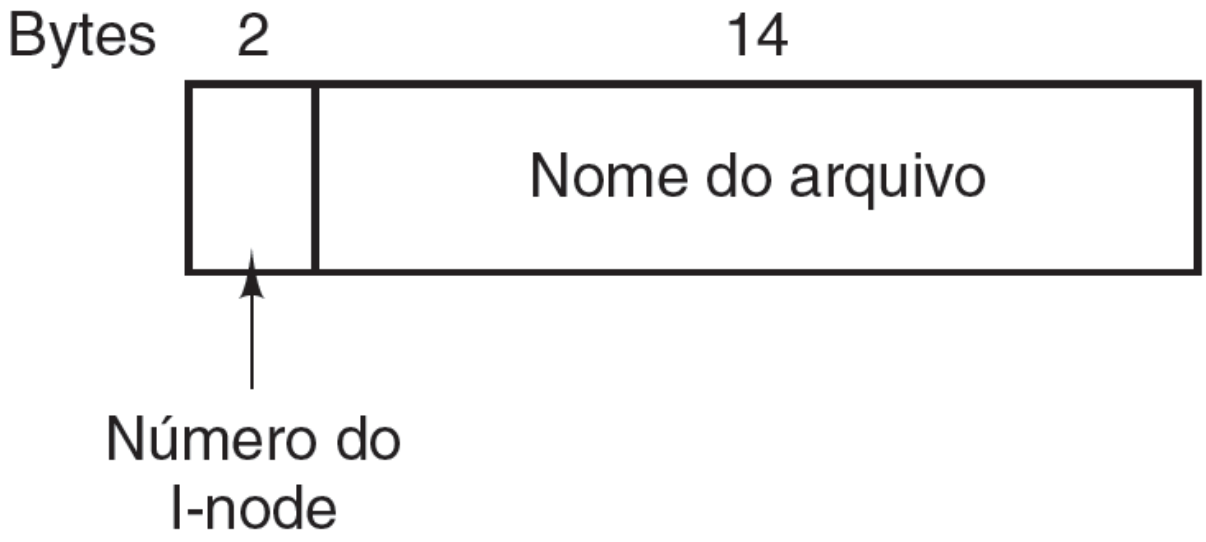

Figura 4.29 Uma entrada de diretório do UNIX V7.

O Sistema de Arquivos do UNIX V7

Características (cont.):

- atributos dos i-nodes no UNIX V7:
  - Tamanho do arquivo
  - Criação, último acesso e última alteração
  - Proprietário
  - Grupo
  - Informações de proteção
  - Contador do nº de entradas de diretórios que apontam para o i-node (a quantos diretórios o arquivo está *ligado*)

O Sistema de Arquivos do UNIX V7

Monitoramento dos blocos de arquivos:

- **Arquivos pequenos:** todos os endereços dos blocos (até 10) estão armazenados no próprio i-node (buscado do disco para a memória quando o arquivo é aberto)

- **Arquivos grandes:** um dos endereços do i-node é o endereço de um bloco chamado ***bloco indireto simples***, que contém endereços adicionais do disco

- **Para arquivos ainda maiores** pode-se ter um ***bloco indireto duplo*** (ou triplo) que aponta para um bloco que contém uma **lista de blocos** indiretos simples (ou duplos)

O Sistema de Arquivos do UNIX V7

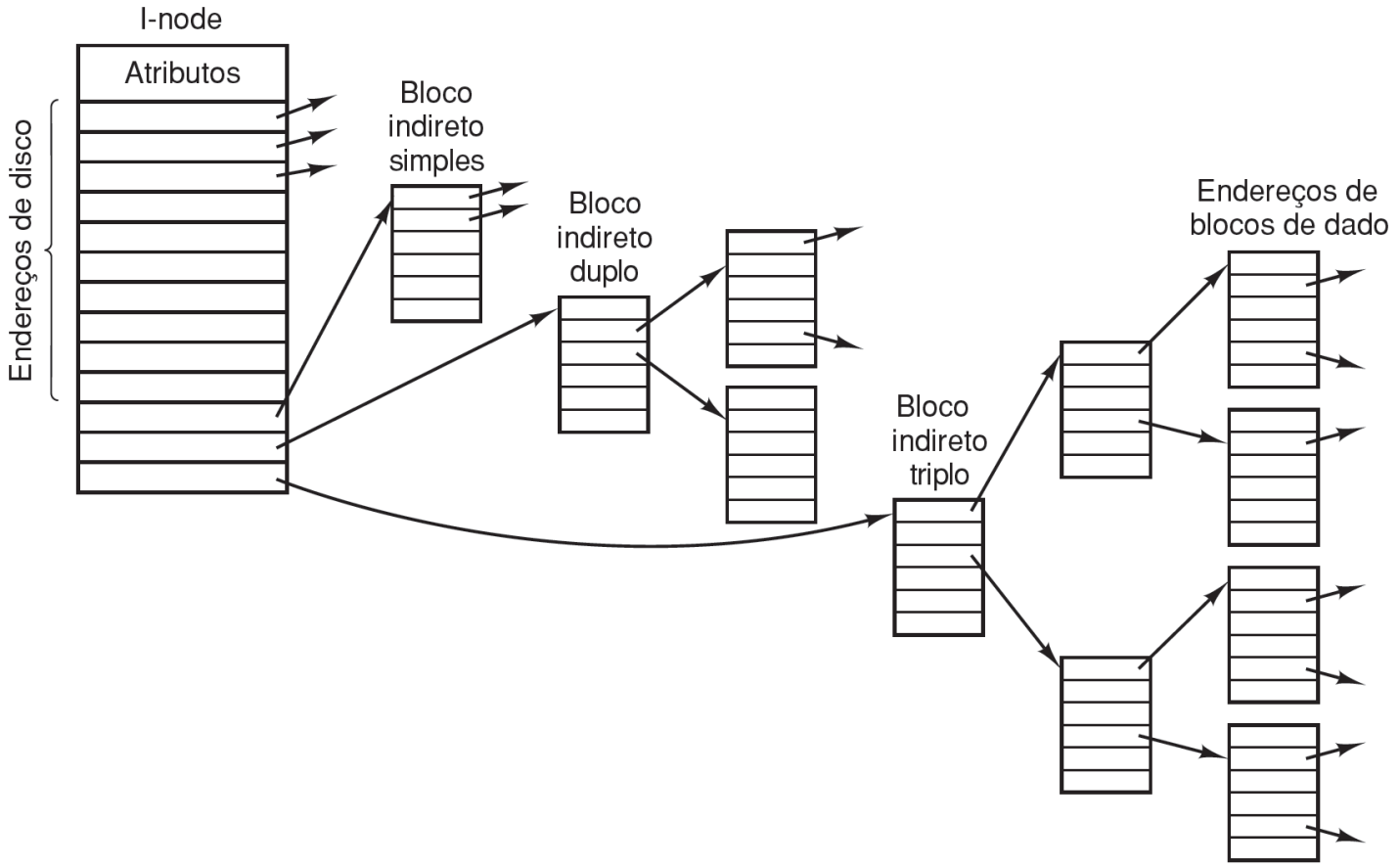

Figura 4.30 Um i-node do UNIX.

O Sistema de Arquivos do UNIX V7

Funcionamento:

Para que um arquivo seja aberto, o SA (através do nome) deve localizar seus blocos de disco

Exemplo da localização de um arquivo: /usr/ast/mbox

- SA localiza o diretório-raiz (seu i-node fica num local fixo do disco)
- lê o diretório-raiz e busca o primeiro componente do caminho **(usr)**
- encontra o número do i-node do arquivo **/usr**
- localiza o i-node de **/usr** no disco a partir de seu número
- com o i-node em mãos, localiza o diretório **/usr**

O Sistema de Arquivos do UNIX V7

Exemplo da localização de um arquivo (cont.): /usr/ast/mbox

- busca no diretório **/usr** o componente **ast**
- quando encontra a entrada para **ast**, tem-se o nº do i-node e, consequentemente, o i-node para **/usr/ast**
- a partir deste i-node o SA acessa o diretório **/usr/ast** para localizar o arquivo **mbox**
- o i-node para o arquivo **mbox** é, então, carregado na memória e mantido lá até que o arquivo seja fechado

O Sistema de Arquivos do UNIX V7

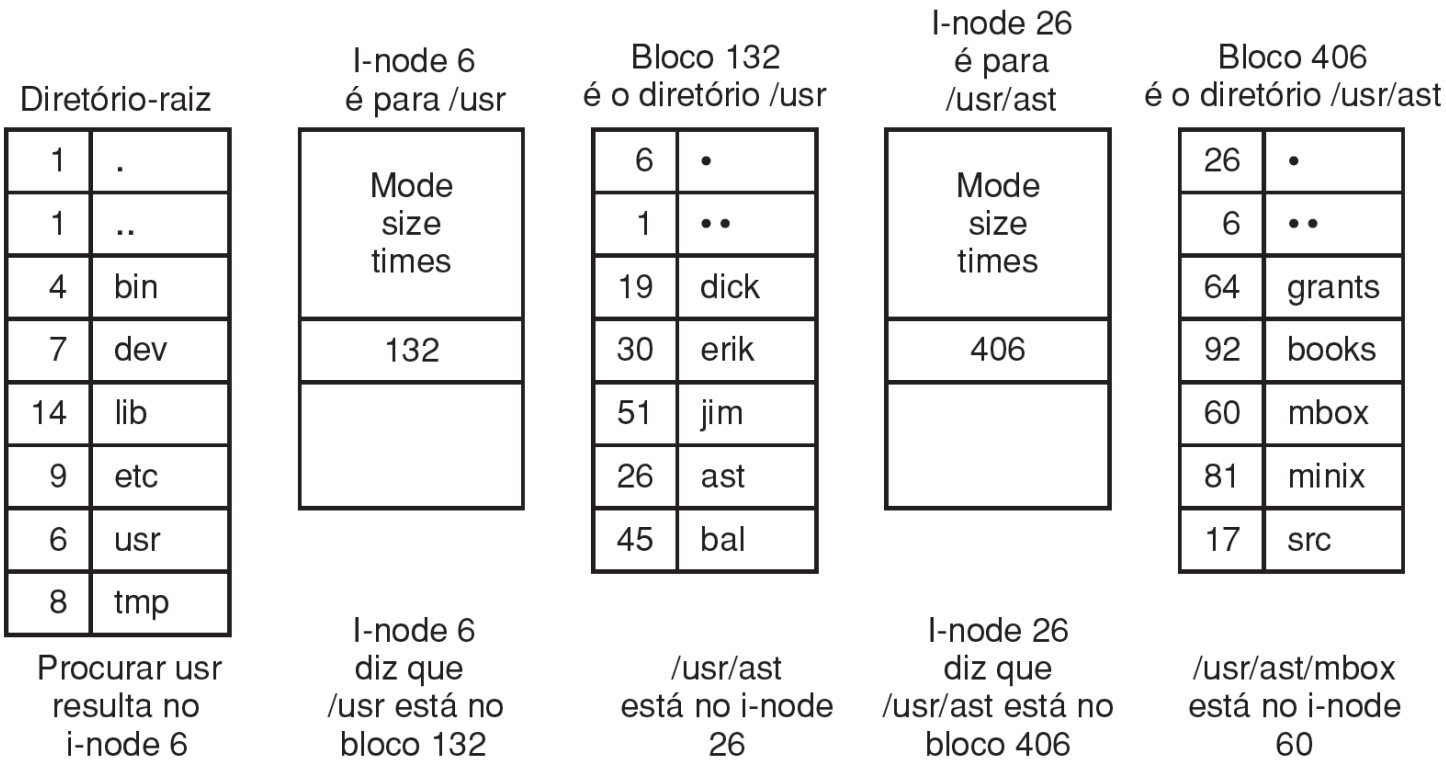

Figura 4.31 Os passos para pesquisar em */usr/ast/mbox*.

O Sistema de Arquivos do UNIX V7

Observações:

1. nomes de caminhos relativos são localizados da mesma forma (partindo do diretório de trabalho ao invés do diretório raiz)

2. Toda vez que um diretório é criado, são criadas as entradas "." e ".."

3. "." tem o nº do i-node do diretório atual

4. ".." tem o nº do i-node do diretório-pai

5. no diretório raiz, ".." aponta para si mesmo