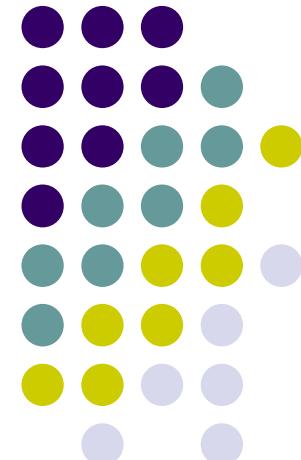


Sistema de Arquivos





Sistemas de Arquivos

Arquivos

Diretórios

Implementação do sistema de arquivos

Exemplos de sistemas de arquivos

Armazenamento da Informação a Longo Prazo



1. Deve ser possível armazenar uma quantidade muito grande de informação
2. A informação deve sobreviver ao término do processo que a usa
3. Múltiplos processos devem ser capazes de acessar a informação concorrentemente

Arquivo = abstração para dados persistentes

Diretório = abstração para agrupamento de dados



Questões sobre Sistema de Arquivos

Questões sobre as abstrações (de arqs./dirs.):

- **Como arquivos são acessados e estruturados?**
- **Como arquivos são identificados e nomeados?**
- **Quais são seus atributos?**
- **Como é definido e gerenciado o controle de acesso?**
- **Operações possíveis?**
- **Semântica de arquivos especiais**



Questões sobre Sistema de Arquivos

Questões relacionadas à implementação:

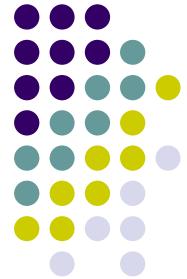
- **Como mapear blocos lógicos para setores do disco?**
- **Como garantir um compartilhamento mantendo a consistência?**
- **Como implementar diretórios?**
- **Como maximizar o desempenho no acesso a arquivos/diretórios?**
- **Manter a informação sobre espaço disponível?**



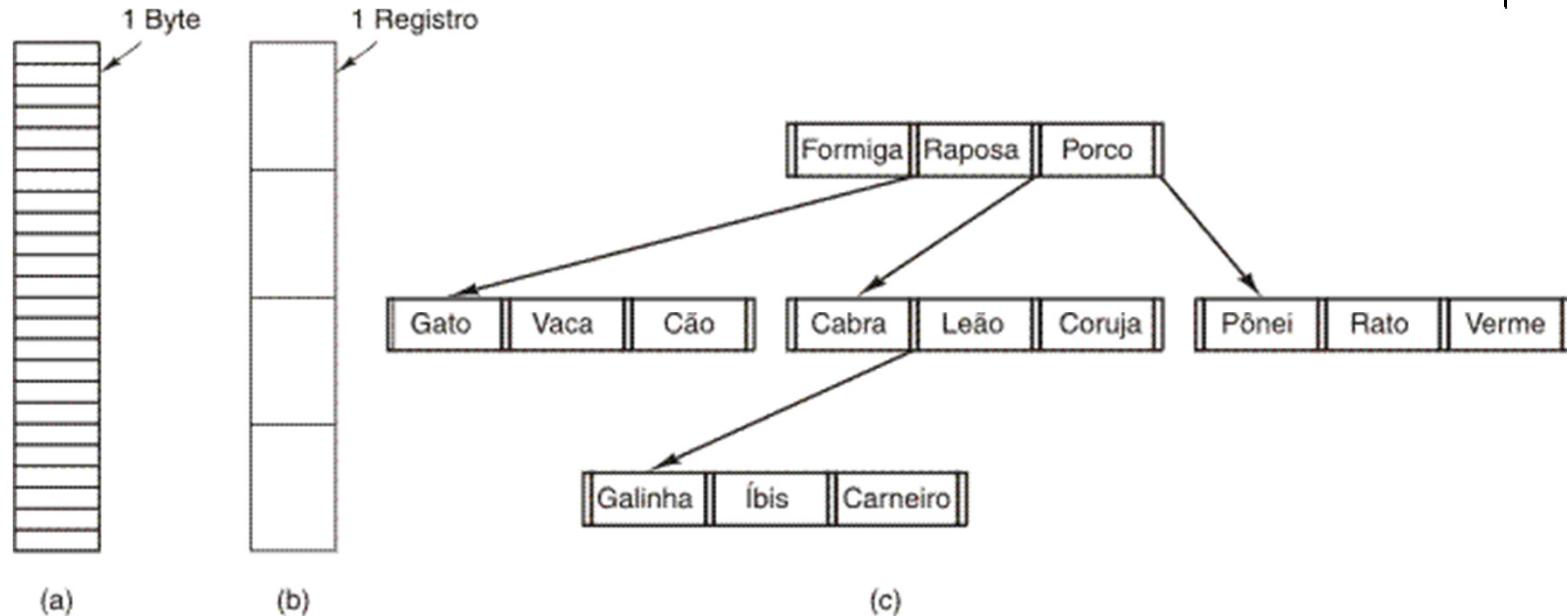
Nome, Tipo e Extensão

Extensão	Significado
file.bak	Arquivo de cópia de segurança
file.c	Programa fonte em C
file.gif	Imagen no formato de intercâmbio gráfico da Compuserve (graphical interchange format)
file.hlp	Arquivo de auxílio
file.html	Documento da World Wide Web em Linguagem de Marcação de Hipertexto (<i>hypertext markup language</i> — HTML)
file.jpg	Imagen codificada com o padrão JPEG
file.mp3	Música codificada no formato de áudio MPEG — camada 3
file.mpg	Filme codificado com o padrão MPEG
file.o	Arquivo-objeto (saída do compilador, ainda não ligado)
file.pdf	Arquivo no formato portátil de documentos (portable document format — PDF)
file.ps	Arquivo no formato PostScript
file.tex	Entrada para o programa de formatação TEX
file.txt	Arquivo de textos
file.zip	Arquivo comprimido

- **Extensão: apenas uma convenção, ou obrigatório no SA?**
- **Muitas ferramentas (make, compiladores) se baseiam nessa convenção.**
- **Quando é obrigatório, pode limitar a flexibilidade.**

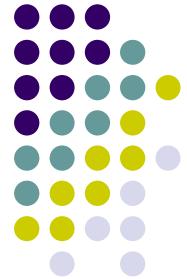


Estrutura de Arquivos



Três tipos de arquivos:

- a) **seqüência de bytes**
- b) **seqüência de registros**
- c) **Árvore (AVL, Árvores B, ...)**

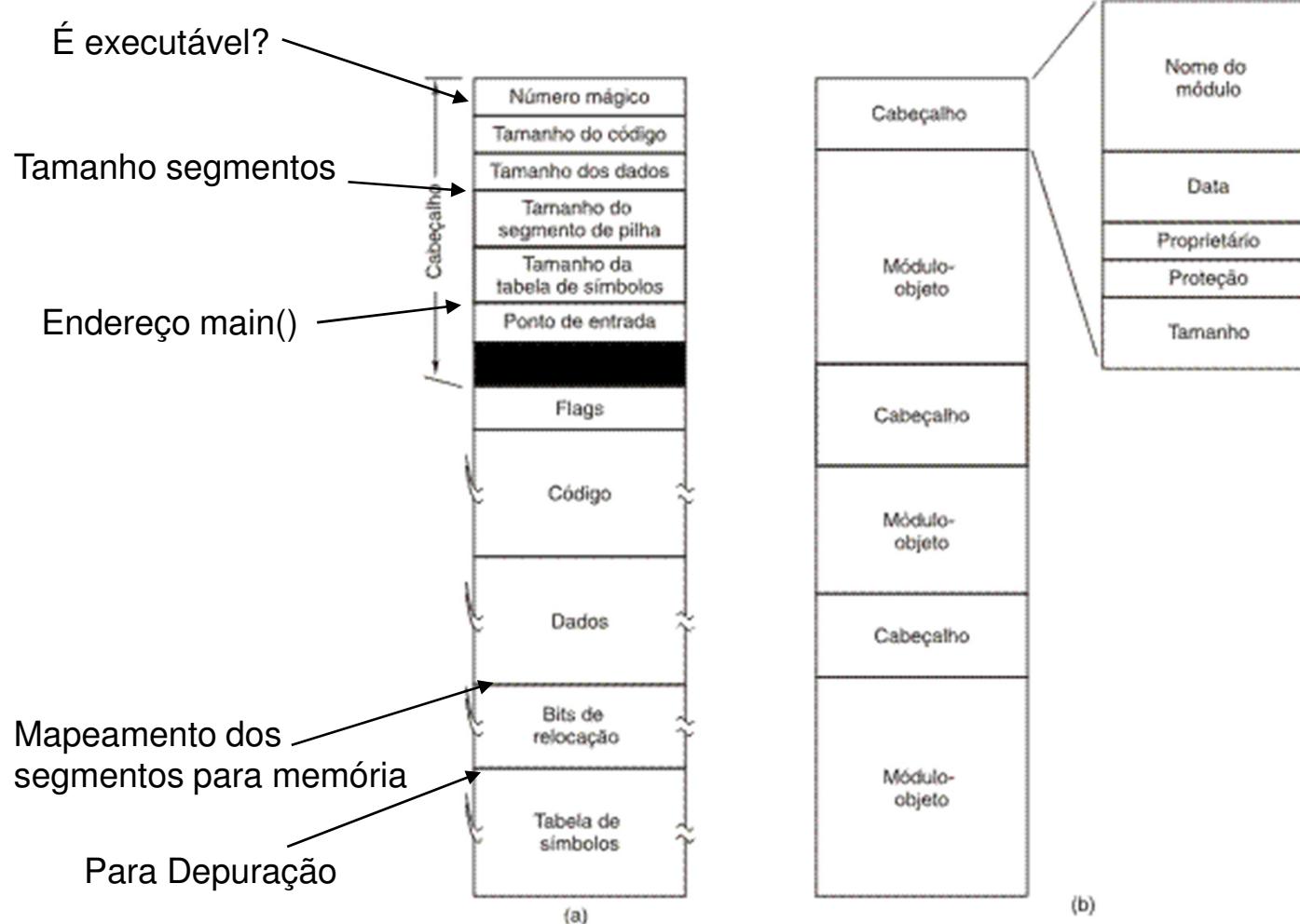


Tipos de Arquivo em UNIX

- Regular – ASCII (data/ text) ou com conteúdo binário (p.ex. executável ou dados cifrados)
- Diretório – de sistema, apenas para manter a estrutura do SA
- Arquivos especiais de caracteres - abstração de E/S serial (discos, terminais, impressoras, rede, pipes, sockets)
- Arquivos especiais de bloco – abstração de E/S de disco (HD, CD-ROM, DVD, etc.)



Tipos de Arquivos



(a) Um arquivo executável (b) Uma biblioteca estática



Acesso aos Arquivos

- **Acesso sequencial**
 - lê todos os bytes/registros a partir do início
 - não pode saltar ou ler fora de seqüência
 - conveniente quando o meio era a fita magnética
- **Acesso aleatório**
 - bytes/registros lidos em qualquer ordem
 - essencial para sistemas de bases de dados
 - ler pode ser ...
 - mover marcador de arquivo (seek), e então ler ou ...
 - ler e então mover marcador de arquivo

Obs: em SA modernos, todos os arquivos são de acesso aleatório.



Atributos de Arquivos

Atributo	Significado
Proteção	Quem pode ter acesso ao arquivo e de que maneira
Senha	Senha necessária para ter acesso ao arquivo
Criador	ID da pessoa que criou o arquivo
Proprietário	Atual proprietário
Flag de apenas para leitura	0 para leitura/escrita; 1 se apenas para leitura
Flag de oculto	0 para normal; 1 para não exibir nas listagens
Flag de sistema	0 para arquivos normais; 1 para arquivos do sistema
Flag de repositório (archive)	0 se foi feita cópia de segurança; 1 se precisar fazer cópia de segurança
Flag ASCII/binário	0 para arquivo ASCII; 1 para arquivo binário
Flag de acesso aleatório	0 se apenas para acesso seqüencial; 1 para acesso aleatório
Flag de temporário	0 para normal; 1 para remover o arquivo na saída do processo
Flag de impedimento	0 para desimpedido; diferente de zero para impedido
Tamanho do registro	Número de bytes em um registro
Posição da chave	Deslocamento da chave dentro de cada registro
Tamanho da chave	Número de bytes no campo-chave
Momento da criação	Data e horário em que o arquivo foi criado
Momento do último acesso	Data e horário do último acesso ao arquivo
Momento da última mudança	Data e horário da última mudança ocorrida no arquivo
Tamanho atual	Número de bytes no arquivo
Tamanho máximo	Número de bytes que o arquivo pode vir a ter

Atributos são Metadados contendo informações para uso e gerência dos arquivos.



Operações com Arquivos

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

Exemplo de um Programa com Chamadas ao Sistema de Arquivos

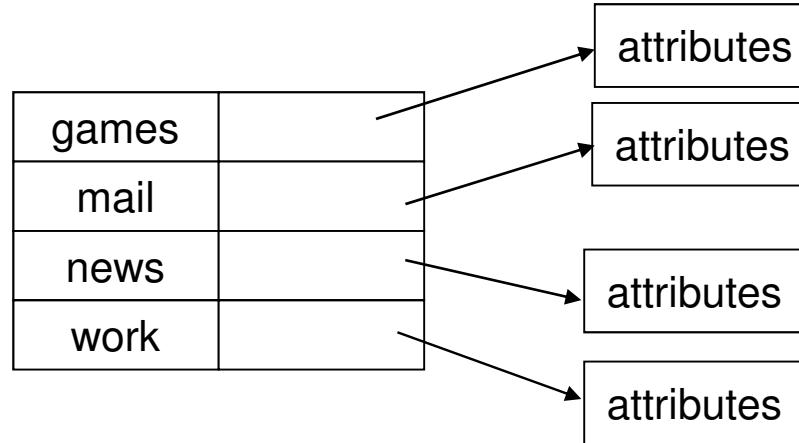


```
/* Programa que copia arquivos. Verificação e relato de erros é mínimo.*/  
  
#include <sys/types.h> /* Inclui os arquivos de cabeçalho necessários*/  
#include <fcntl.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
int main(int argc, char *argv[]); /* protótipo ANSI*/  
  
#define BUF_SIZE 4096 /* usa um tamanho de buffer de 4096 bytes*/  
#define OUTPUT_MODE 0700 /* bits de proteção para o arquivo de saída*/  
  
int main(int argc, char *argv[]){  
    int in_fd, out_fd, rd_count, wt_count;  
    char buffer[BUF_SIZE];  
  
    if (argc != 3) exit(1); /* erro de sintaxe se argc não for 3 */  
  
    /* Abre o arquivo de entrada e cria o arquivo de saída */  
    in_fd = open(argv[1], O_RDONLY); /* abre o arquivo de origem */  
    if (in_fd < 0) exit(2); /* se não puder ser aberto, saia */  
    out_fd = creat(argv[2], OUTPUT_MODE); /* cria o arquivo de destino */  
    if (out_fd < 0) exit(3); /* se não puder ser criado, saia */  
  
    /* Laço de cópia */  
    while (TRUE) {  
        rd_count = read(in_fd, buffer, BUF_SIZE); /* lê um bloco de dados */  
        if (rd_count <= 0) break; /* se fim de arquivo ou erro, sai do laço */  
        wt_count = write(out_fd, buffer, rd_count); /* escreve dados */  
        if (wt_count <= 0) exit(4); /* wt_count <= 0 é um erro */  
    }  
  
    /* Fecha os arquivos */  
    close(in_fd);  
    close(out_fd);  
    if (rd_count == 0) /* nenhum erro na última leitura */  
        exit(0);  
    else  
        exit(5); /* erro na última leitura */  
}
```



Diretórios

games	attributes
mail	attributes
news	attributes
work	attributes

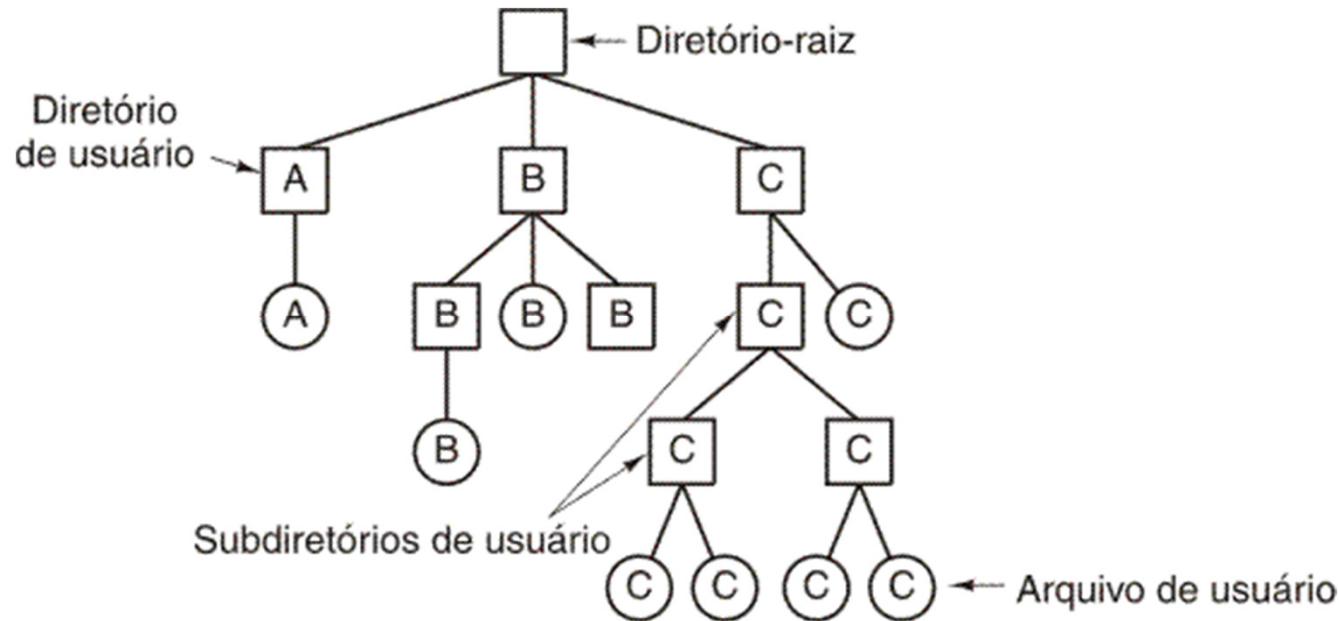


Estrutura de dados com atributos

- **Ao abrir um arquivo, procura-se o arquivo no diretório, atributos e endereço são copiados para uma Tabela de Arquivos Abertos, (Open file table).**
- **Em futuros acessos, não é necessário acessar novamente o diretório.**

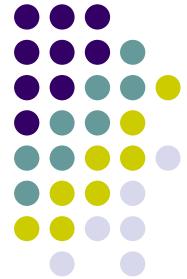


Sistemas de Diretórios Hierárquicos



- Em diretórios hierárquicos, não há limite para o no. de níveis de aninhamento, e arquivo é referenciado através de um caminho (path), e.g. /usr/ast/mailbox
- Existe o conceito de diretório corrente (process work directory - pwd)
- Arquivos podem ser referenciados usando caminhos absolutos (/usr/src/servers), ou relativos (src/servers) ao diretório corrente
- Define-se um nome especial para diretório corrente e diretório imediatamente acima. Em Unix “.” (dot) e “..” (dotdot)

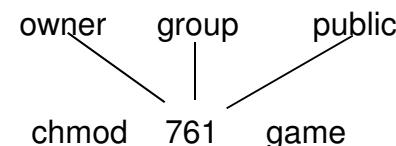
Permissões de acesso e Grupos



- Modos de acesso + comuns: read, write, execute
- Três categorias de usuário (UNIX)

			RWX
a) owner access	7	⇒	1 1 1
			RWX
b) groups access	6	⇒	1 1 0
			RWX
c) public access	1	⇒	0 0 1

- Administrador cria um novo grupo, G, e adiciona usuários ao grupo.
- Para um arquivo ou subdiretório, o dono do arquivo ∈ G pode redefinir os direitos de acesso para G
- Pode-se também modificar o grupo de arquivo ou diretório: chgrp
- Pode-se alterar os direitos: chmod





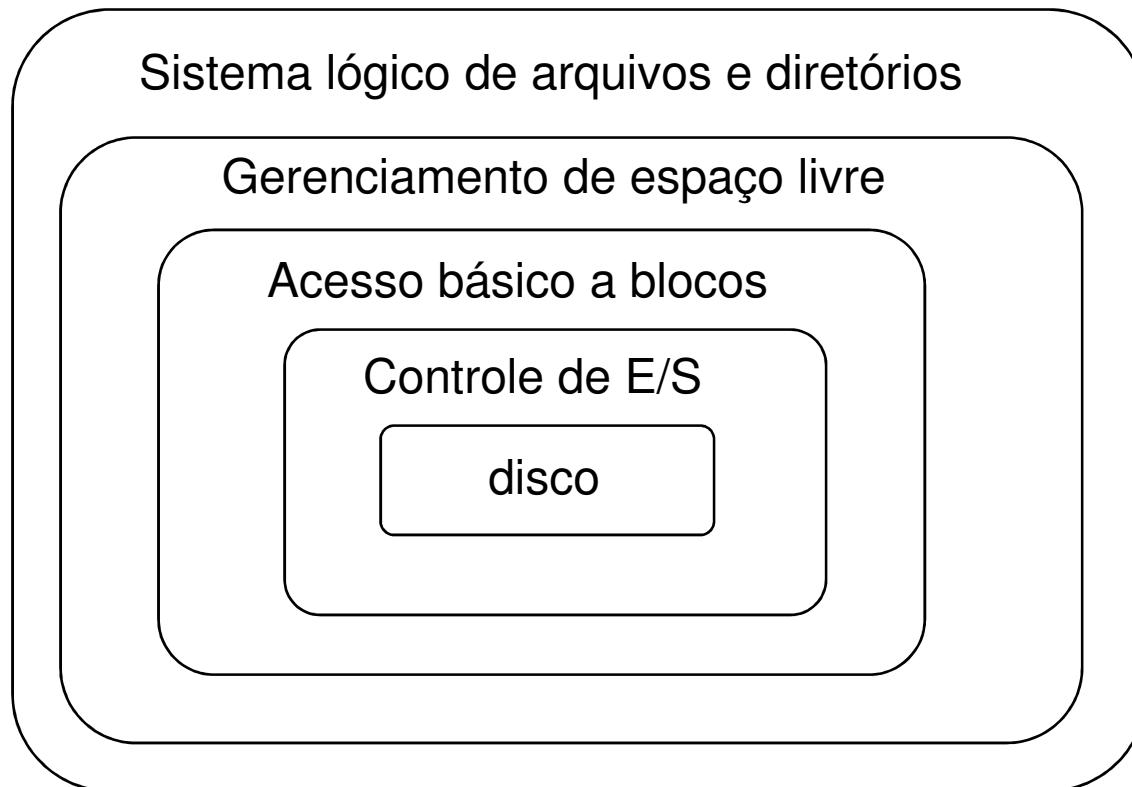
Operações com Diretórios

1. **Create**
2. **Delete**
3. **Opendir**
4. **Closedir**
5. **Readdir**
6. **Rename**
7. **Link**
8. **Unlink**

- Antes de ler ou alterar um diretório, ele precisa ser aberto. Ao final, deve ser fechado.
- Link é a forma de criar atalhos (referências adicionais) para arquivos.
- Unlink remove um arquivo de um diretório.
- Se o arquivo está só referenciado por um único diretório, unlink remove o Arquivo.
- Obs: Cópia de arquivos é feita por programa utilitário (cp do UNIX)



Camadas do sistema de arquivos



Um Sistema de Arquivos (AS) é um conjunto de tipos abstratos de dados que são implementados para o armazenamento, a organização hierárquica, a manipulação, navegação, acesso e recuperação de dados.

Implementação do Sistema de Arquivos

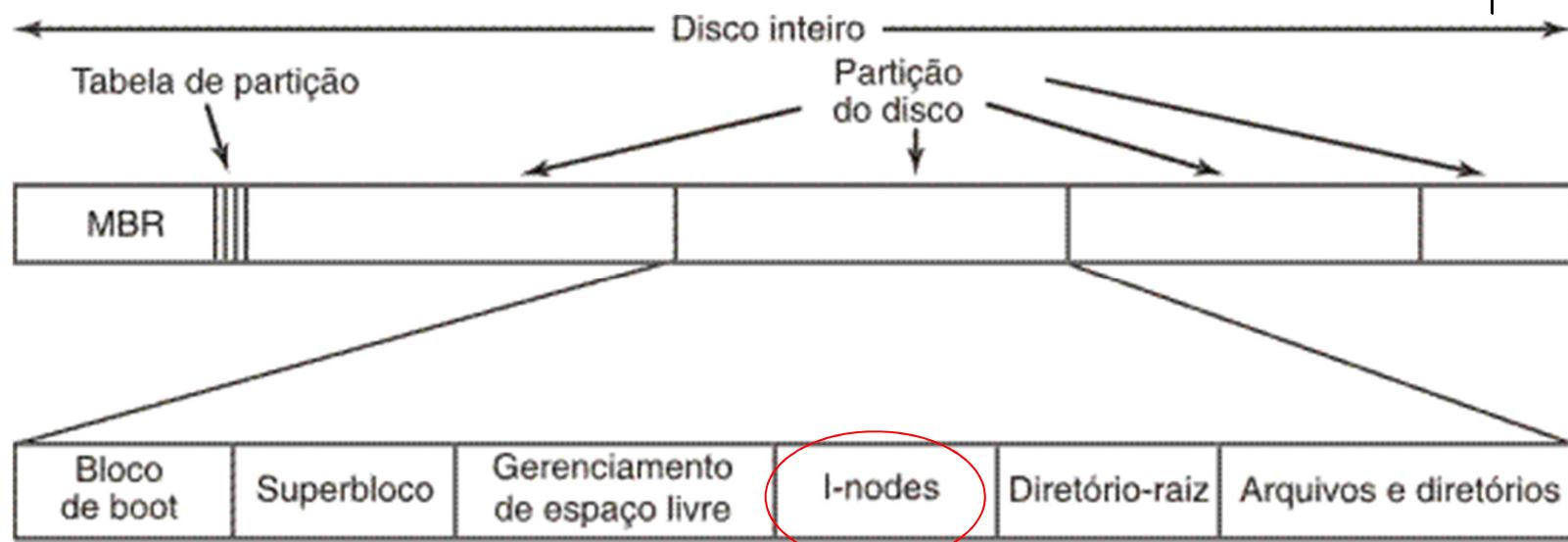


Fig.: Estrutura de cada partição no disco

Um i-node é um bloco especial (File Control Block -FCB) do Unix, que contém os atributos de um arquivo e os endereços dos blocos que constituem o arquivo.



Estrutura de uma Partição em Disco

Boot block/Partition Boot sector

- Informações necessárias para carregar o syst.operacional a partir desta partição

Super block/ Master File Table

- Quantidade e tamanho dos blocos
- Contador e ponteiro de blocos livres
- Contador e ponteiros para os FileControlBlocks (i-nodes)

Seção de i-nodes

- contém um vetor de FileControlBlocks (um para cada arquivo ou diretório)

Diretório raiz

- Nomes e atributos de sub-diretórios da raiz

Blocos com arquivos e diretórios



Estruturas de Dados que o Sistema de Arquivos mantém em Memória

Objetivo: melhorar o desempenho do acesso e gerência do sistema de arquivos

- **Tabela com informações sobre cada partição montada**
- **Estrutura de Diretórios:** diretórios acessados recentemente
- **FileTable:** para todos os arquivos abertos no sistema, contém uma cópia do FCB
- **Tabela de arquivos abertos por processo com ponteiros para a FileTable**



Questões de Implementação

Principais Questões:

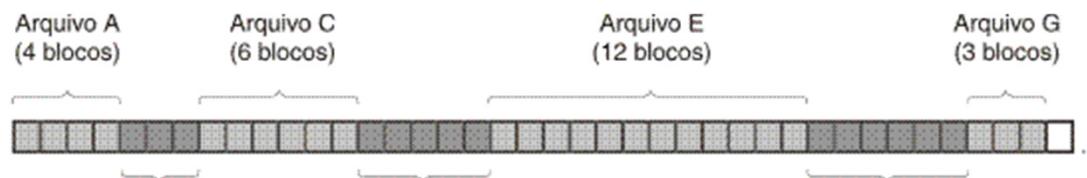
- Como são armazenados arquivos e diretórios no disco?
- Como é feito o mapeamento de arquivos para blocos de disco? Como saber em qual bloco de disco está determinado registro do arquivo?
- Como o espaço livre no disco é gerenciado?

Existem diversas formas de alocar blocos (do disco) a um arquivo (a seguir).

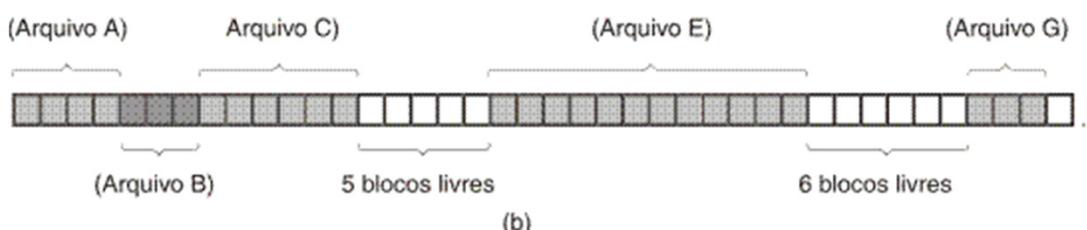


Implementação de Arquivos (1)

1. Cada arquivo é um conjunto contíguo de blocos de dados.
 - (+) Implementação simples e bom desempenho (todo arquivo é lido em uma única operação)
 - (-) O tamanho máximo do arquivo precisa ser conhecido no momento da criação.
 - (-) Há desperdício de espaço em disco já que geralmente $\text{tamanho-arquivo} < \text{tamanho-máximo}$.



(a)



(b)

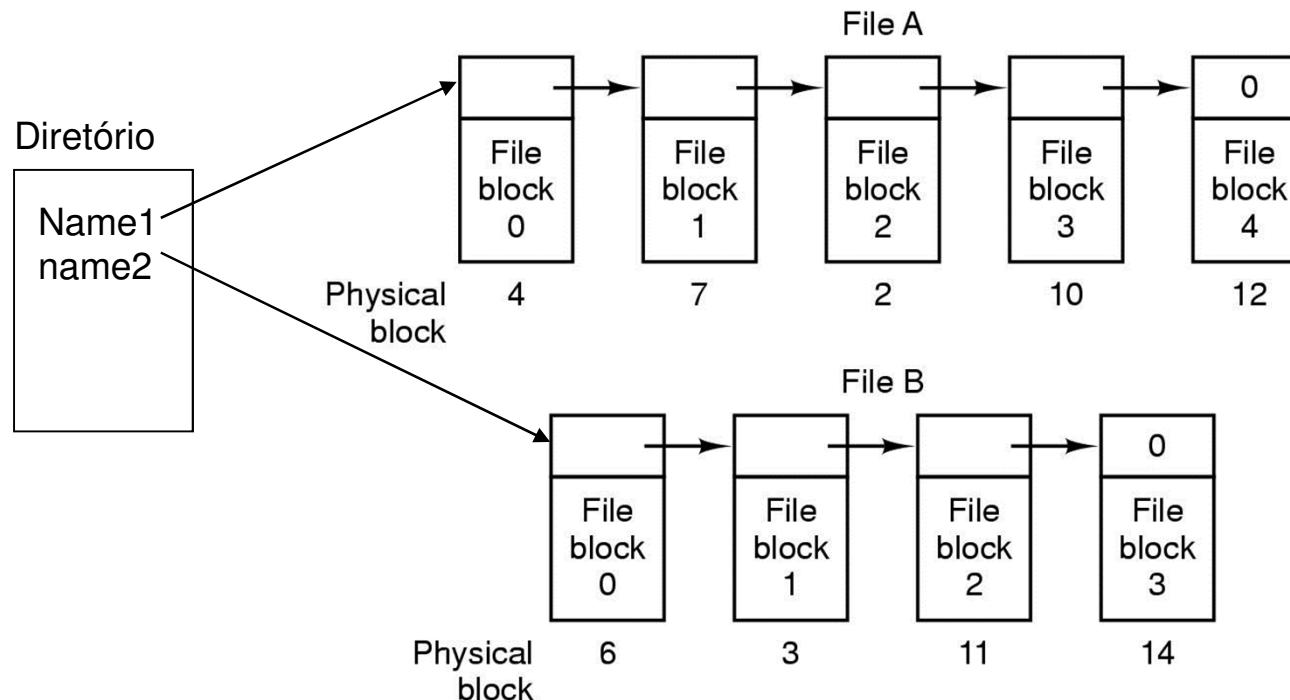
- (a) Alocação contígua do espaço em disco para 7 arquivos
- (b) Estado do disco depois dos arquivos D e E terem sido removidos

Implementação de arquivos de tamanho variável



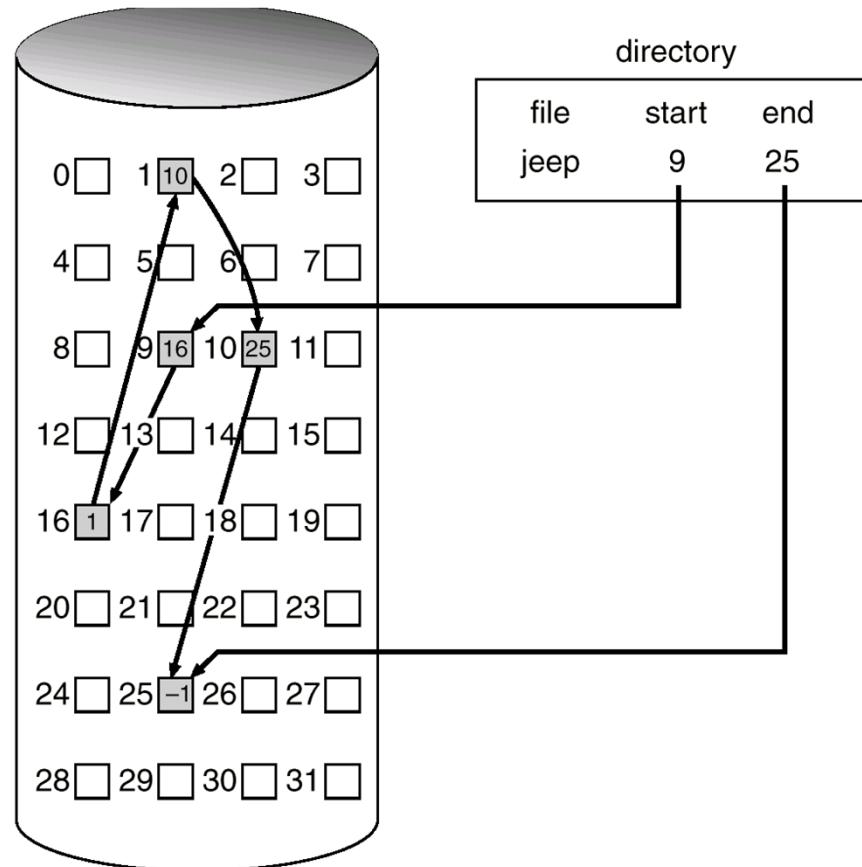
2. Lista encadeada de blocos físicos em disco

- Primeira palavra de cada bloco é um ponteiro (endereço de disco) para o próximo bloco da lista
- Todos os blocos podem ser usados (não há fragmentação externa)
- Adequado para acesso sequencial, mas pouco eficiente para acesso aleatório





Exemplo: Lista encadeada





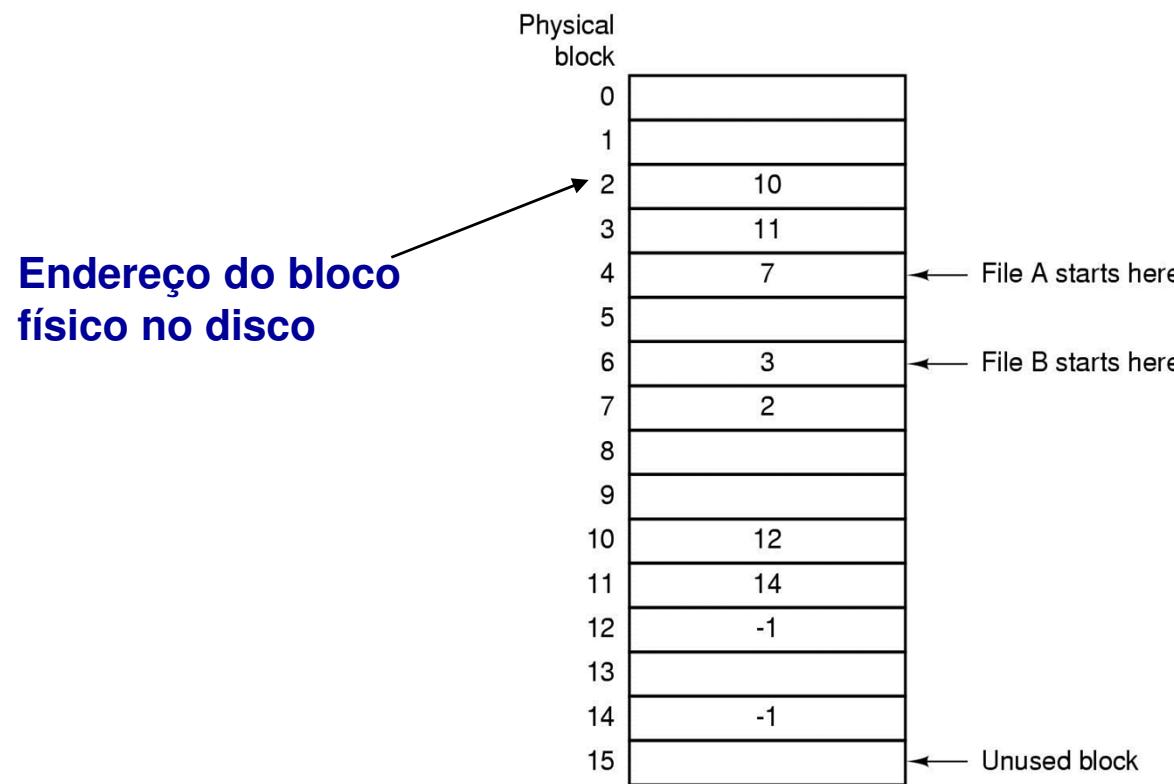
Lista Encadeada

3. **Lista encadeada usando uma tabela de alocação de arquivos em memória RAM (Exemplo: MS-DOS)**

Idéia principal: colocar os ponteiros em uma tabela de índices (em memória)

Prós: (a) O bloco físico só contém dados ; (b) acesso randômico não requer vários acessos ao disco

Contra: toda a tabela precisa estar em memória (para discos grandes isso pode ser um problema)

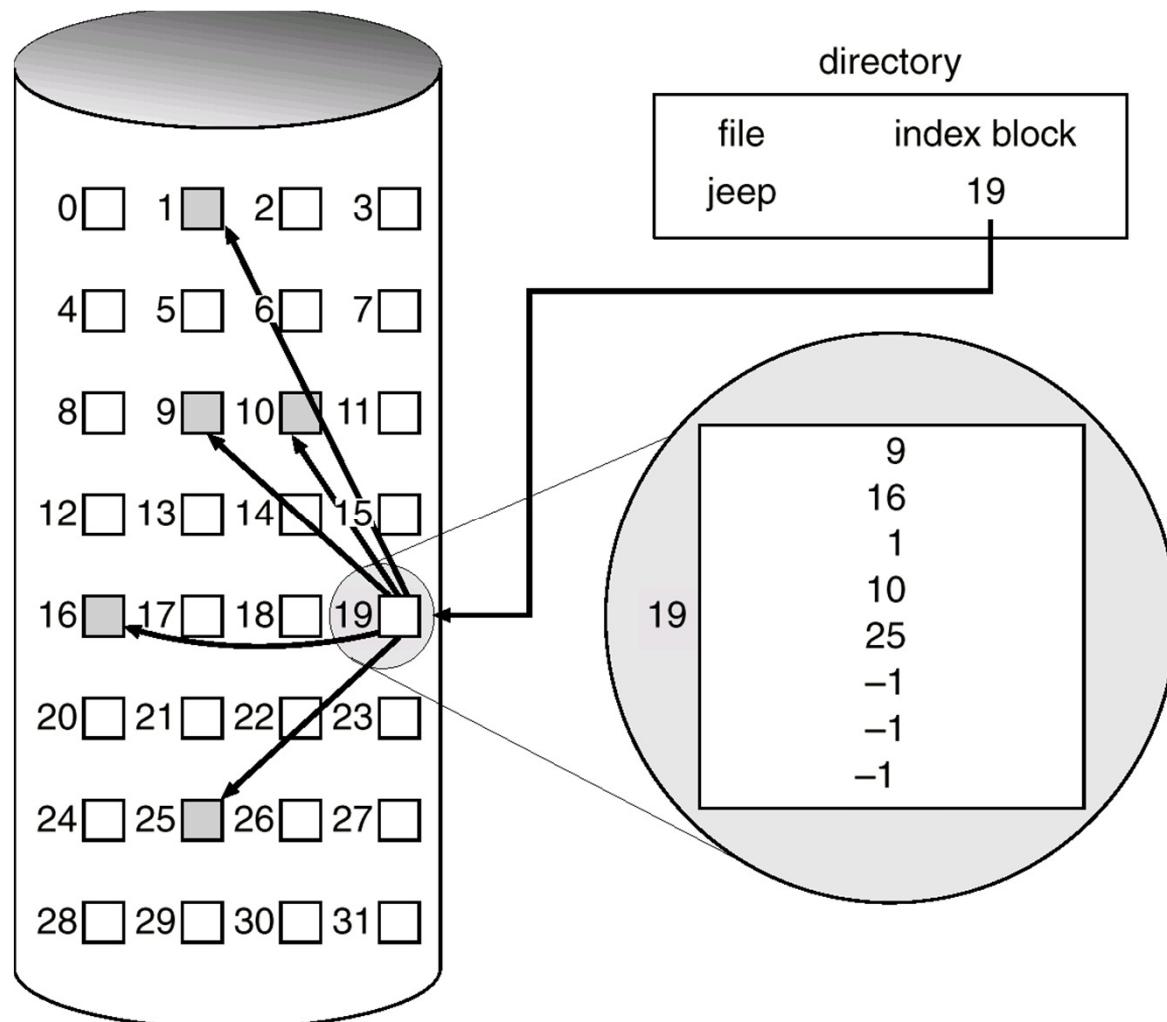


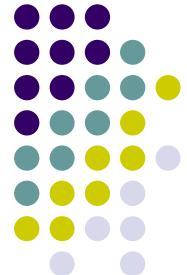
Arquivo A: usa blocos (4, 7, 2, 10, 12)



Alocação Indexada

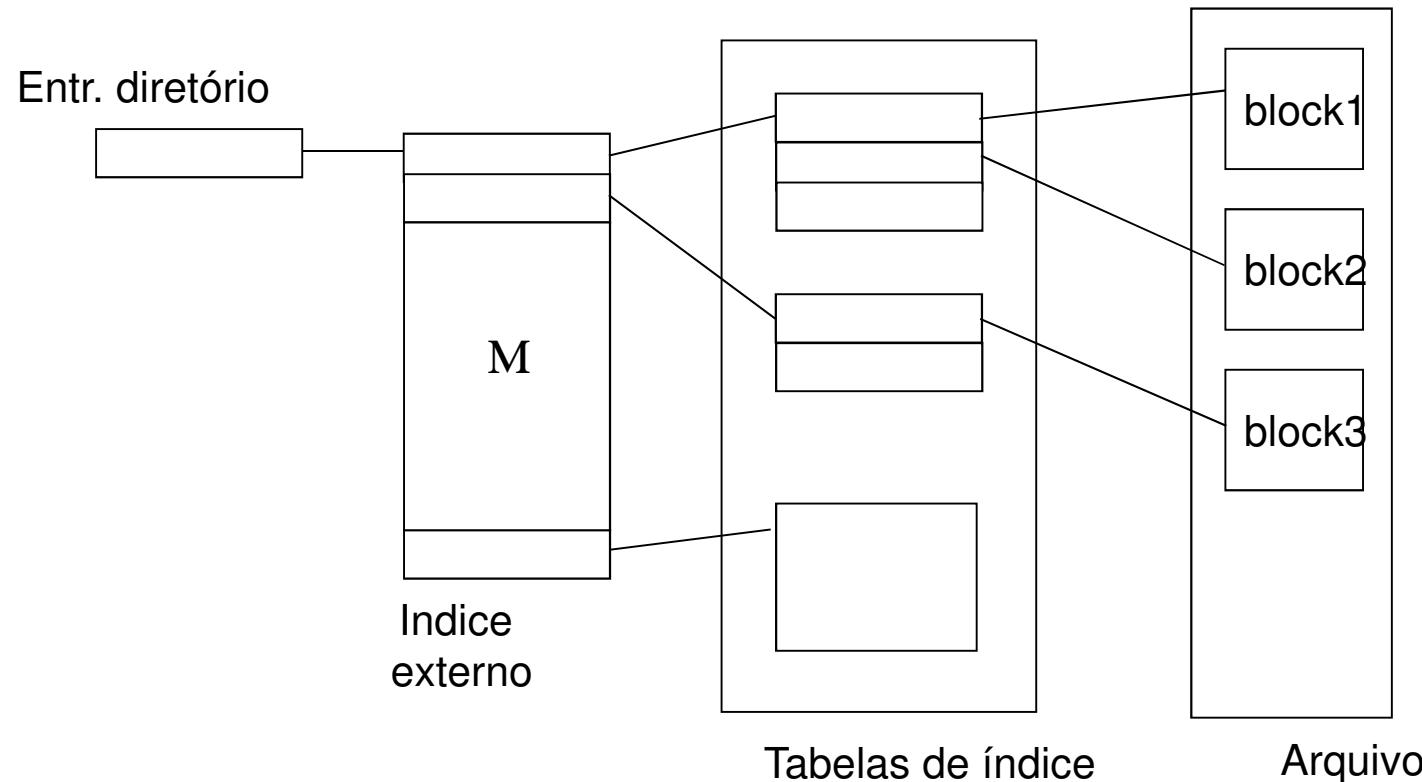
- Todos os endereços de blocos são mantidos em um bloco de índice (*index block*).





Indexação Multi-nível

- É necessária quando o tamanho do arquivo (em # de blocos) ultrapassa o número de endereços que cabem um index block
- Entrada do diretório aponta para o início do índice mais externo
- O endereço físico do registro (ou byte) no bloco precisa ser calculado





Indexação Multi-nível

Principal desvantagem: o overhead para acesso a um registro é igualmente alto tanto para arquivos pequenos, como para arquivos grandes.

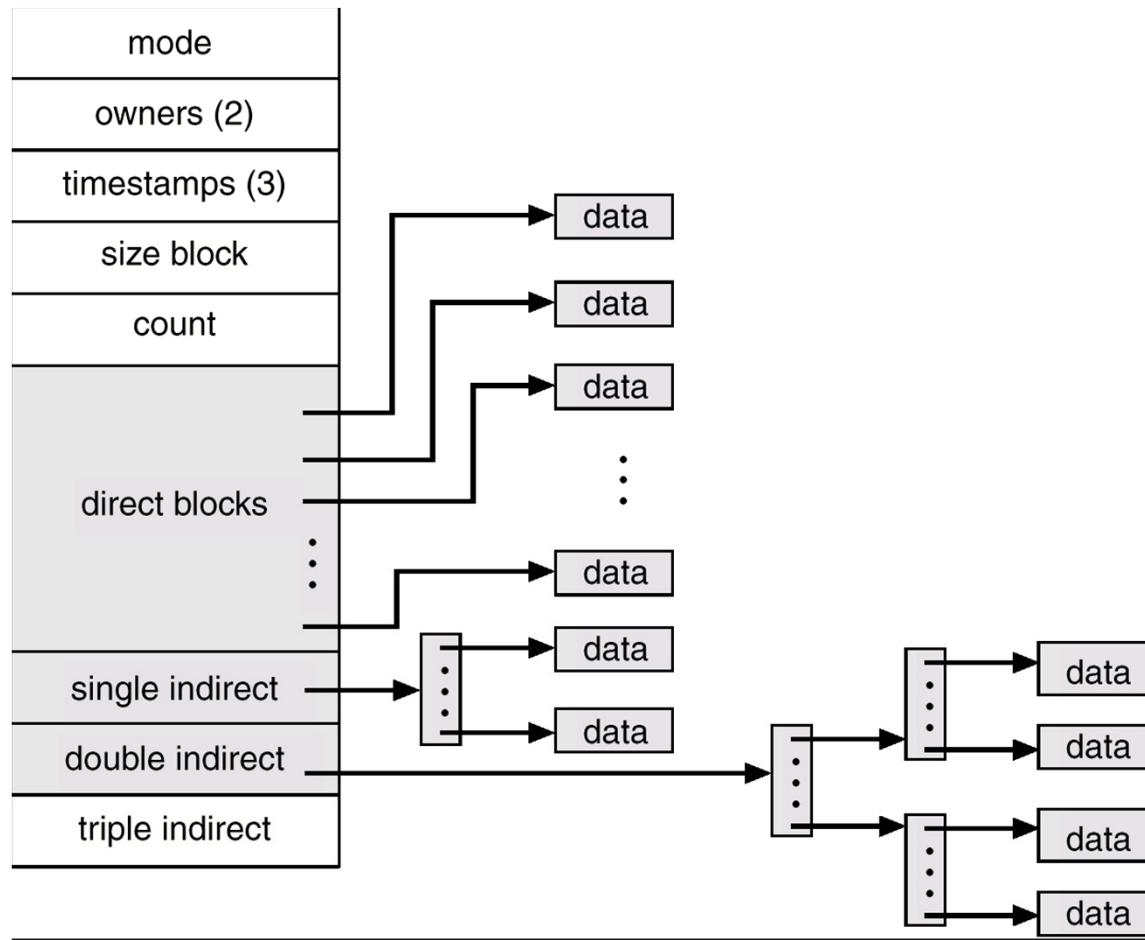
Seria bom, ter uma forma de alocação que agilizasse o acesso a arquivos pequenos (a maioria)

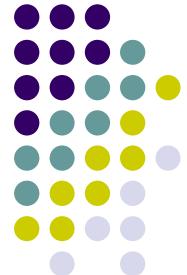
Idéia: os primeiros blocos de um arquivo podem ter uma indexação direta.



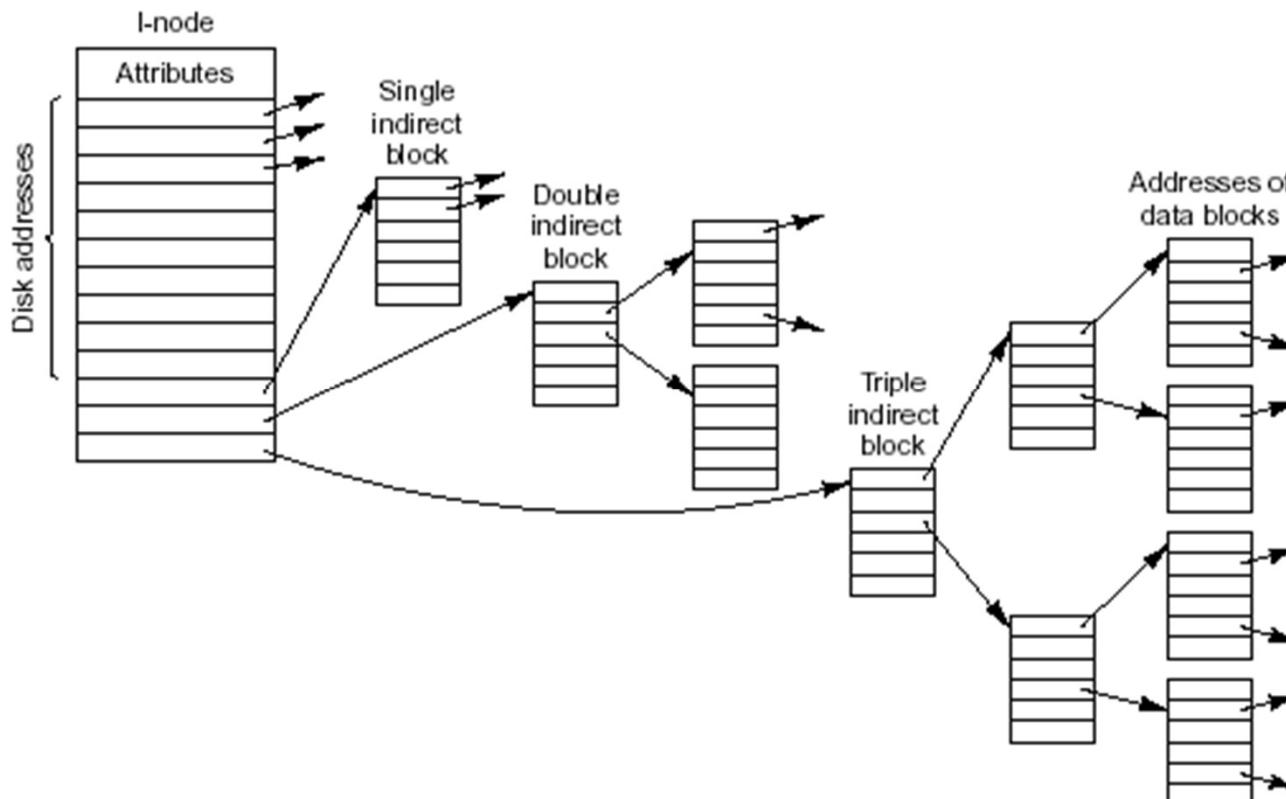
O i-node de UNIX

i-node = estrutura de dados com atributos, 12 endereços diretos de blocos e 3 endereços de blocos de índice (single, double, triple indirect)



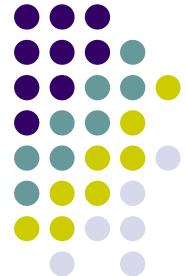


Indexação Multi-nível em Unix



Assumindo que cada bloco de índice consegue armazenar 32 endereços

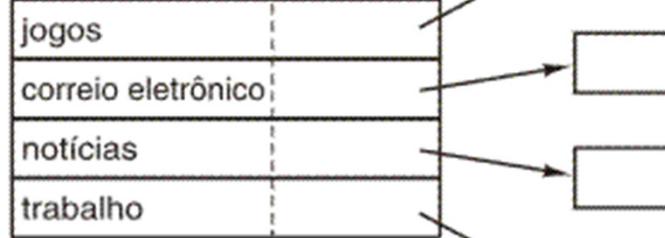
- **Endereçamento direto:** para arquivos pequenos (até 12 blocos)
- **Single indirect:** para arquivos de até $12 + 32$ blocos
- **Double indirect:** para arquivos de até $12 + 32 + 32 \times 32$ blocos
- **Triple indirect:** para arquivos de até $12 + 32 + 32 \times 32 + 32 \times 32 \times 32$ blocos



Implementação de Diretórios

jogos	atributos
correio eletrônico	atributos
notícias	atributos
trabalho	atributos

(a)



(b)

FCB ou i-node

**(a) Um diretório simples
entradas de tamanho fixo
endereços de disco e atributos na entrada de diretório**

(b) Diretório no qual cada entrada se refere apenas a um i-node



Implementação de Diretórios

Se arquivo consiste de vários blocos e não existe um FCB (ou i-node), cada entrada de diretório precisa armazenar os endereços dos blocos (exemplo de uma entrada de diretório no CP/M)

U(1)	File name (8)	Ext (3)	E(1)	Count(1)	Disk block addresses(15)
------	---------------	---------	------	----------	--------------------------

Legenda:

U = Id do usuário dono do arquivo

E = número de sequência da entrada (para grandes arquivos)

Count = contador do número de Disk block addresses ocupados nessa entrada

(n) = Quantidade de bytes usados

Note: Arquivos grandes (com > 16 blocos) precisam de mais de uma entrada de diretórios.



Implementação de Diretórios

Em MS-DOS:

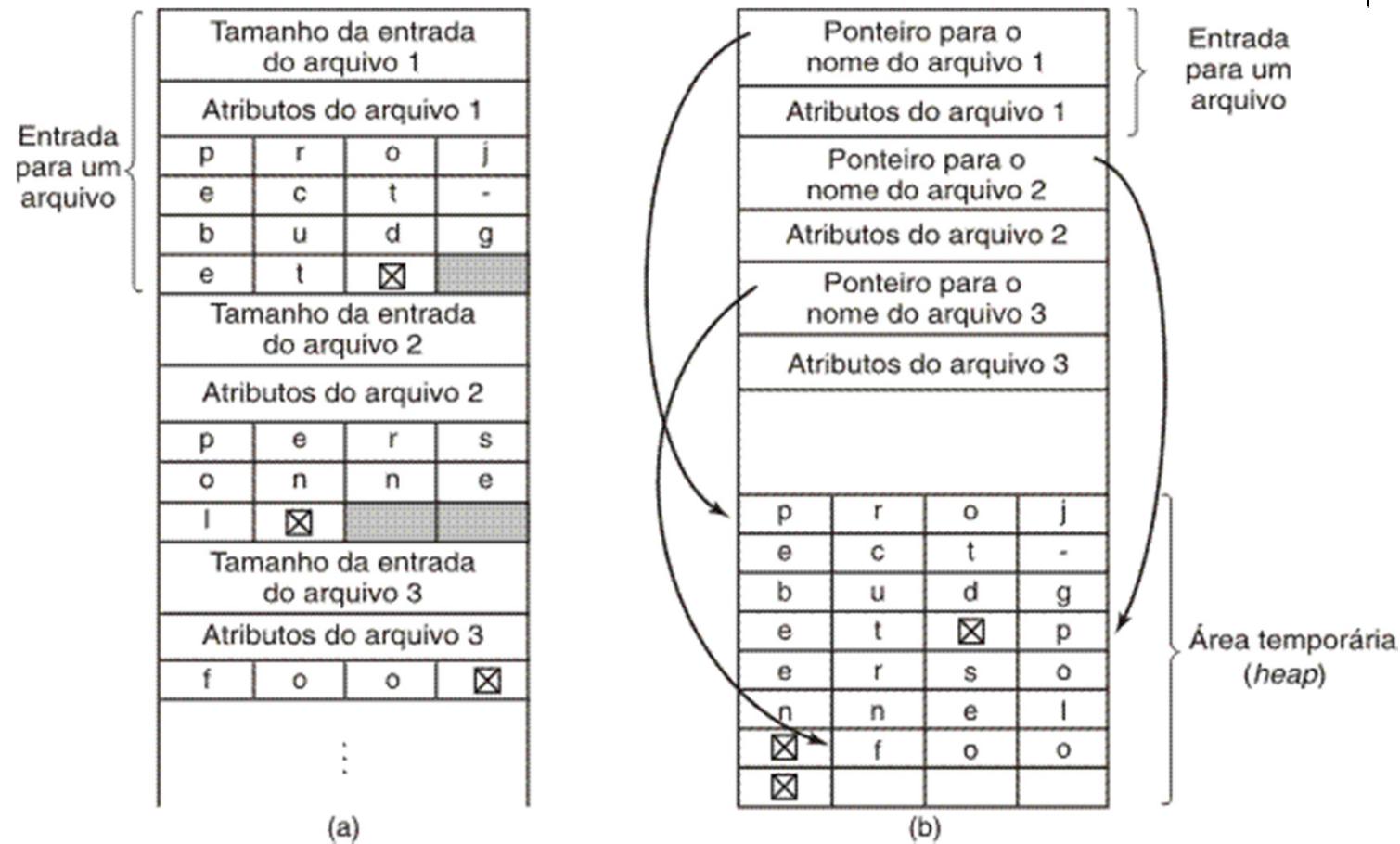
- Cada entrada contém um índice para a File Allocation Table (FAT), que corresponde ao 1º. bloco
- Nome pode ser de um arquivo ou subdiretório
- Nomes limitados a 8 caracteres e extensões a 3 caracteres
- Também contém atributos como direitos de acesso (A) para o usuário (do tipo R/W/X), data&hora (D, T) de criação/modificação e tamanho do arquivo

File name (8)	Ext (3)	A(1)	Reserved(10)	T(1)	D(1)	FAT-idx(2)	Size(4)
---------------	---------	------	--------------	------	------	------------	---------

Como armazenar nomes de arquivos/diretórios de tamanho abstritário?



Implementação de Diretórios



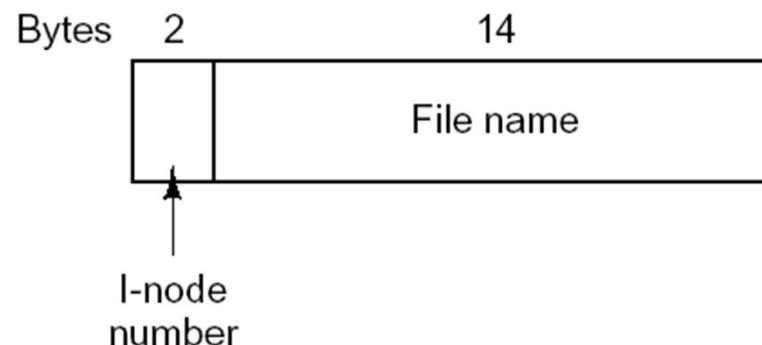
- Duas formas de armazenar nomes de arquivos (de tamanho variável) em um diretório
 - (a) Em linha
 - (b) Em uma área temporária (*heap*)



Implementação de Diretórios

Em UNIX:

- O i-node é o objeto que representa tanto um arquivo como um (sub) diretório.
- Cada entrada contém um nome (string ASCII de tamanho variável), e um número (endereço) de i-node;
- Este número é um índice para uma entrada do vetor de i-nodes (na seção da partição correspondente),
- Atributos são armazenados no i-node (juntamente com os endereços de blocos de disco)





Resolução de Nomes em UNIX

Dado um caminho (*path name*), precisa-se achar o i-node para esse nome

- Como “.” e “..” também possuem entradas no diretório, a resolução de nomes absolutos e relativos funciona exatamente da mesma forma. Para nomes relativos, simplesmente, inicia-se se a partir do diretório corrente

Exemplo: resolução de /usr/ast/mbox

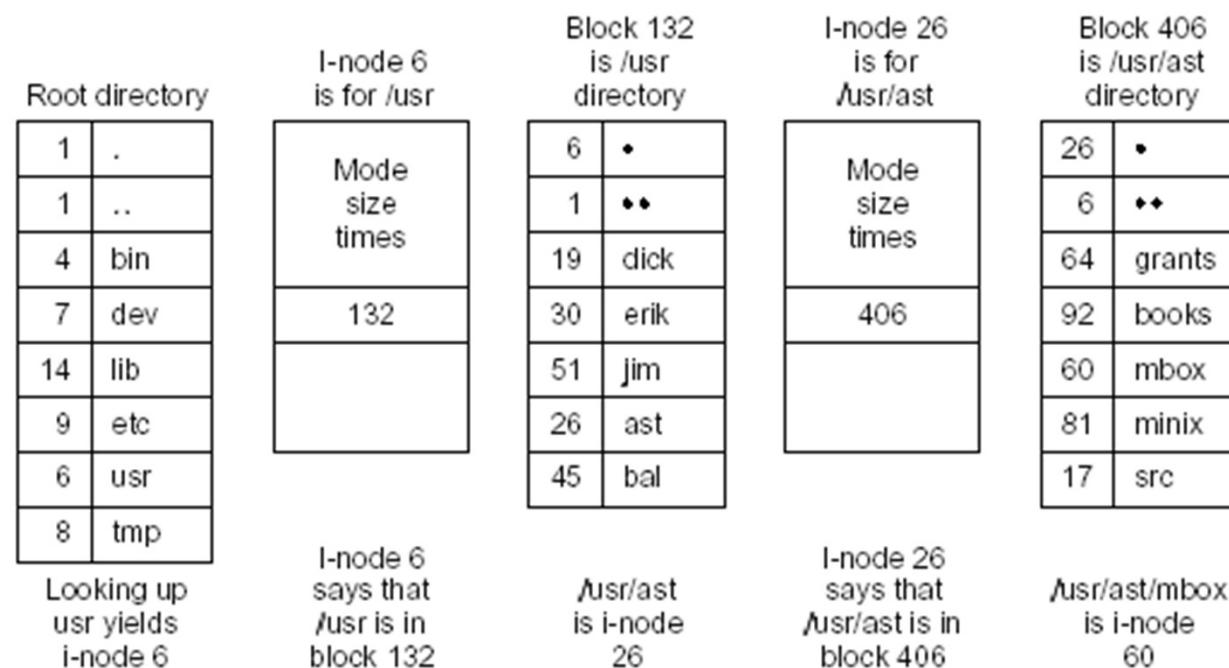
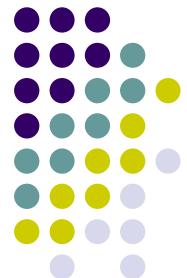
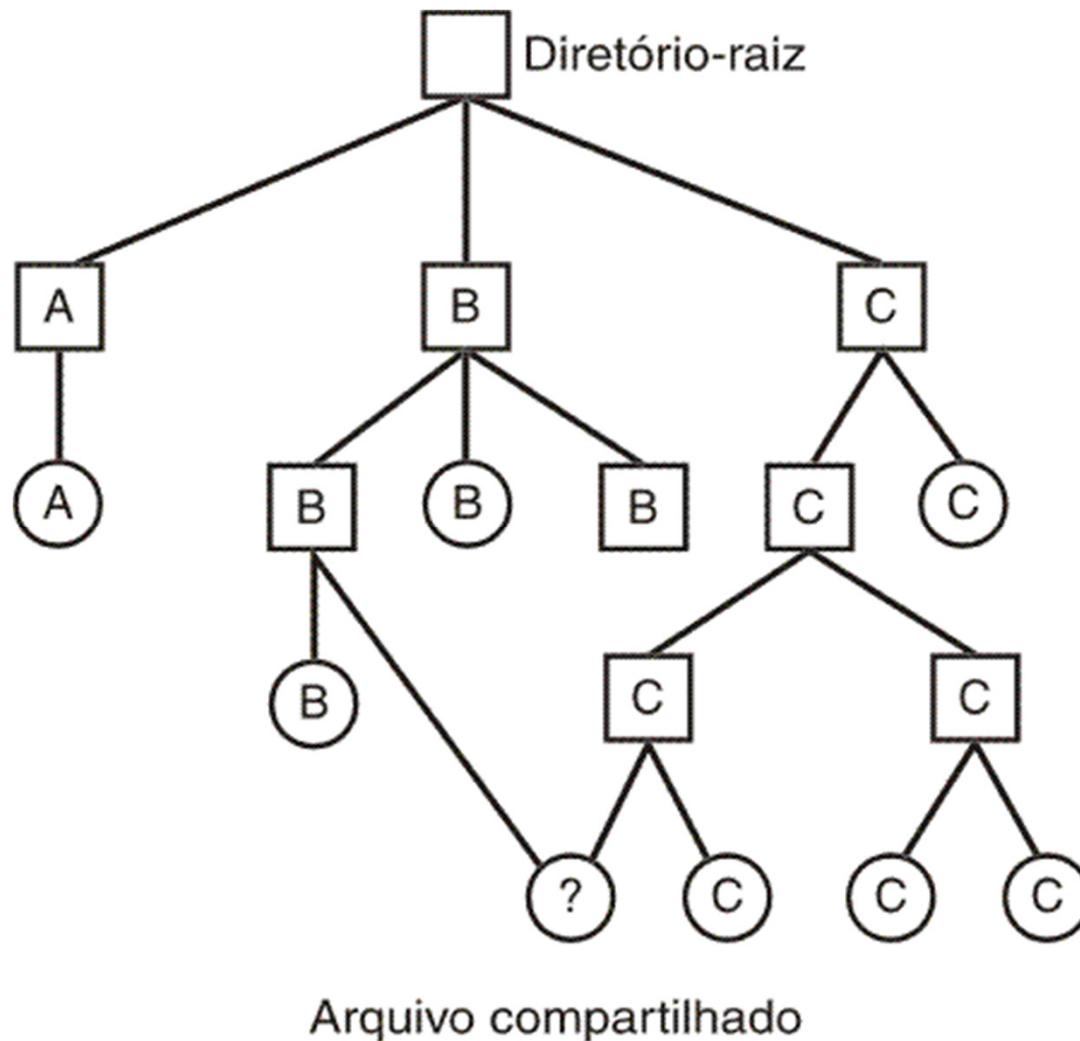


Figure 5-14. The steps in looking up /usr/ast/mbox.



Arquivos Compartilhados (1)



Sistema de arquivo contendo um arquivo compartilhado



Links Simbólicos

O i-node também mantém a informação de quantos links simbólicos estão apontando para o arquivo ou diretório.

Remoção só é efetuada quando o valor desse contador chega a 0.

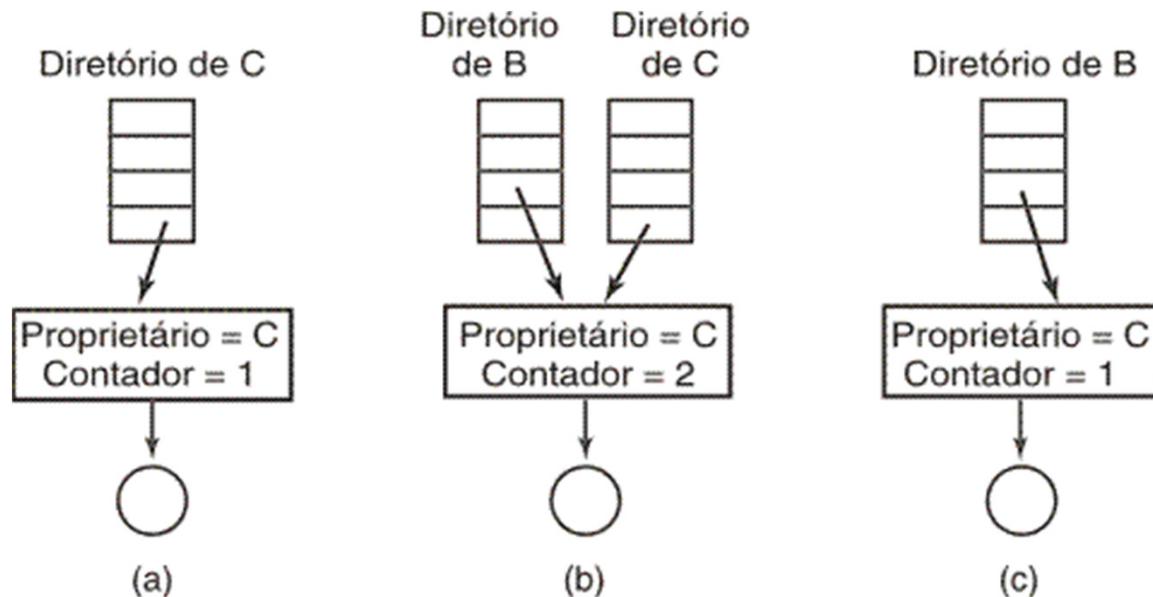


Fig.:

- (a) Situação antes da ligação
- (b) Depois de a ligação ser criada
- (c) Depois de o proprietário original remover o arquivo



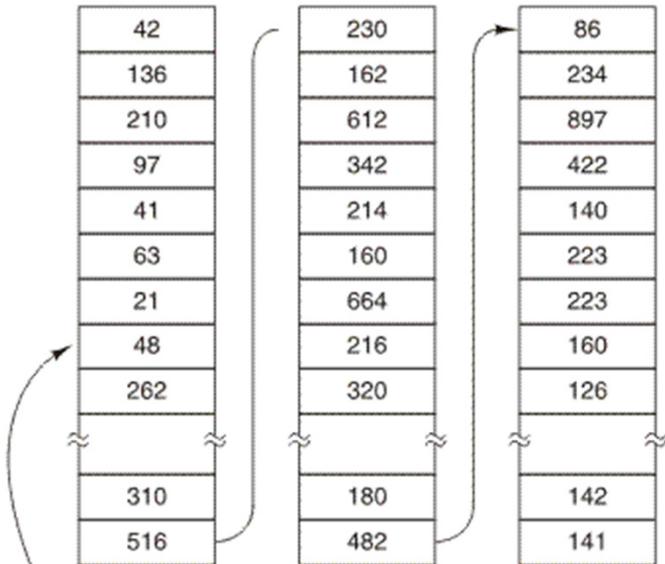
Gerenciamento de Espaço em Disco

- A grande maioria dos sistemas de arquivos dividem os arquivos em blocos de tamanho fixo que não precisam ser contíguos
- Tamanho do bloco grande → menor utilização do espaço em disco (maior desperdício de espaço)
- Tamanho de bloco pequeno → maioria dos arquivos consistirá de muitos blocos (aumenta tempo de acesso)
- Disco é organizado em setor, trilha e cilindro
- Tamanho do bloco deve ser independente do dispositivo, mas não deve ser uma fração do tamanho de um setor
- Tempo de acesso a um bloco é fortemente dependente do tempo de busca e o atraso rotacional ($\cong 10^1$ ms). Tempo de transferência é desprezível.
- Tamanhos de bloco em diversos Sistemas:
 - UNIX : 1KB
 - MS-Dos: de 512 bytes a 32 KB (dependendo do tamanho da partição, mas existe a restrição de # máximo de blocos por partição ser 2^{16})

Gerenciamento do Espaço Livre em Disco



Blocos livres de disco: 16, 17, 18



Um bloco de disco de 1 KB pode conter 256 números
de blocos de disco de 32 bits

(a)

1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
~
0111011101110111
1101111101110111

Um mapa de bits

(b)

Implementações mais comuns: Lista ligada (a) ou bit map (b)

Lista ligada: blocos livres são usados como blocos de índices com endereços de vários outros blocos livres (e.g. se tam-bloco=1KB e endereços de 32 bits, então 1 bloco pode conter 255 endereços + endereço do próximo bloco de índice)

Bit maps : devem estar em memória, mas podem também estar contidos em blocos bit-map (1 bloco pode manter 8 K de bits)



Perguntas?

