MC504 - Sistemas Operacionais

Sistemas de Arquivos

Islene Calciolari Garcia
Instituto de Computação - Unicamp
Primeiro Semestre de 2017

Sumário

Sistemas de Arquivos

- Grande quantidade de informação
- Dados persistentes (não-voláteis)
- Acesso concorrente

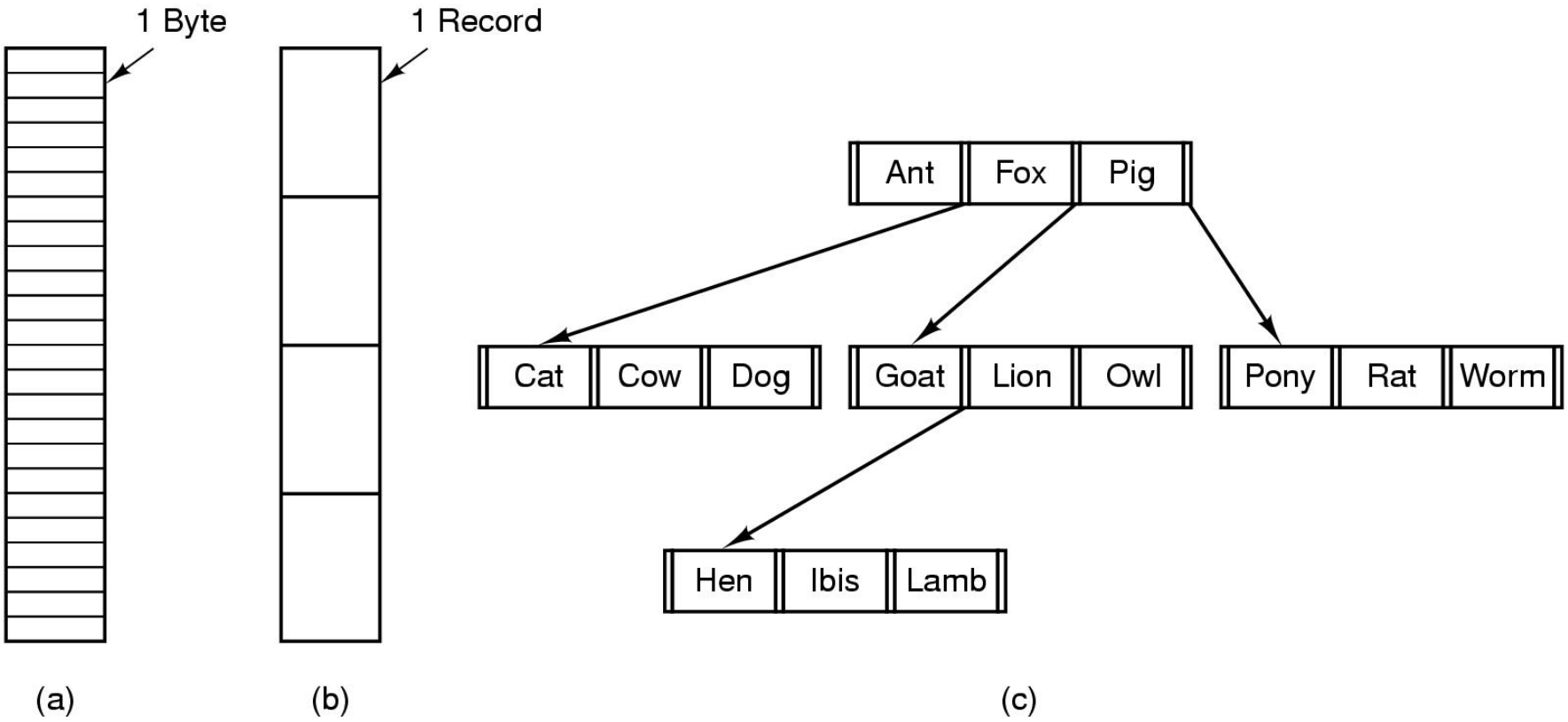
Nomes e extensões

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Tanenbaum: Figura 6.1

Arquivos podem ter mais de uma extensão: `file.ps.gz`
Comando `file` verifica o tipo dos arquivos

Estruturas de arquivos



Tanenbaum: Figura 6.2

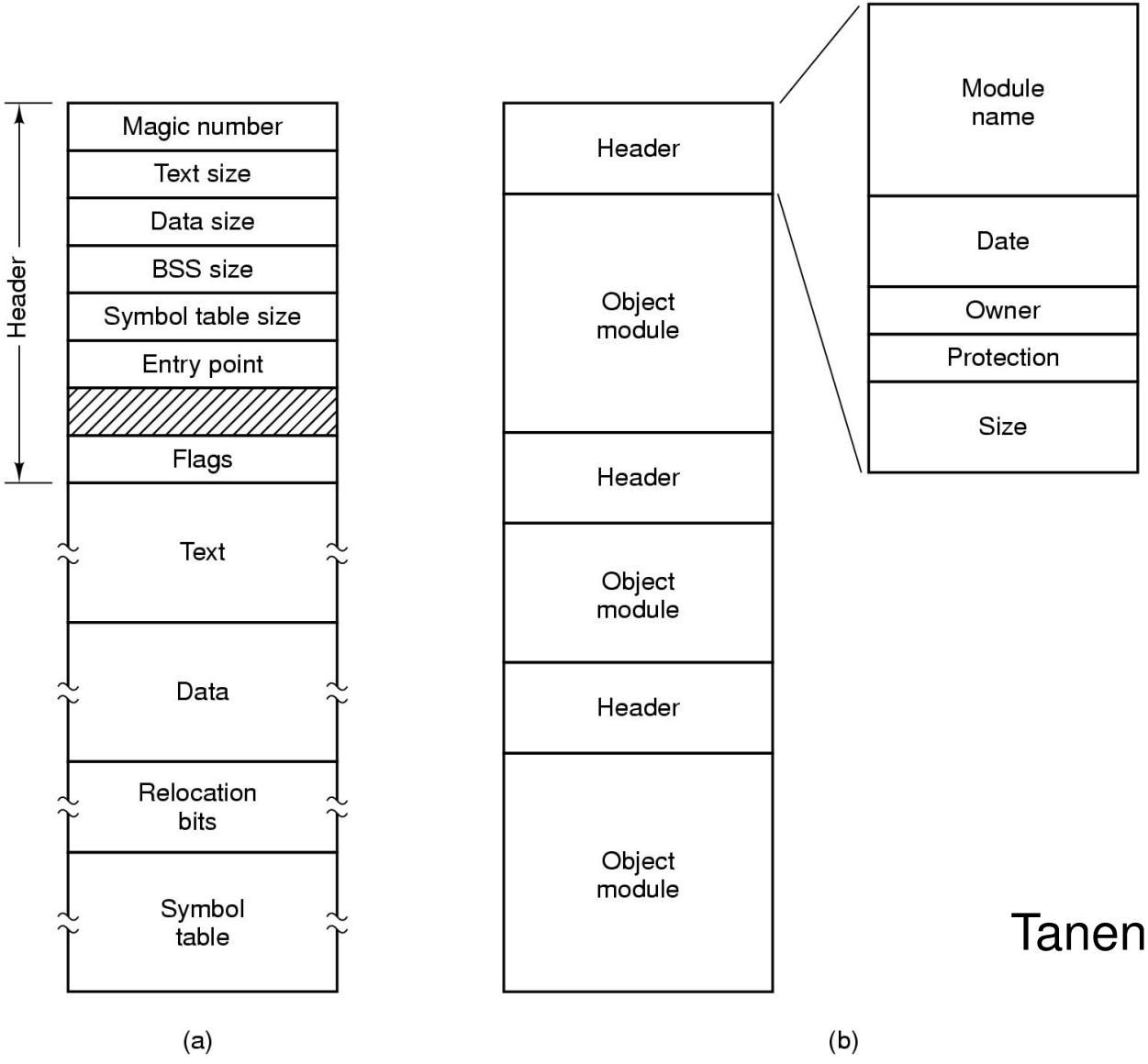
Tipos de arquivos

- regular
- diretório
- caracter
  - terminais, impressoras e rede
- bloco
  - discos

Use o comando `stat`:

```
$ stat arquivos.pdf
$ stat .
$ stat /dev/tty0
```

Exemplos: executável e archive



Tanenbaum: Figura 6.2

Acesso a arquivos

- Sequencial
  - Lê todos os bytes a partir do início
  - Fitas magnéticas
- Aleatório
  - Bytes podem ser lidos em qualquer ordem
  - Bancos de dados

Atributos de arquivos

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Tanenbaum: Figura 6.4

Veja os comandos `stat` e `make`

Operações sobre arquivos

- create
- delete
- open
- close
- read
- write
- append
- seek
- get attributes
- set attributes
- rename

Veja `struct file_operations` em
`linux-4.X.Y/include/linux/fs.h`

Programa copy

```
#define BUF_SIZE 4096
#define OUTPUT_MODE 0700

int main(int argc, char *argv[]) {
  int in_fd, out_fd, rd_count, wt_count;
  char buffer[BUF_SIZE];

  if (argc!=3) exit(1);

  in_fd = open(argv[1], O_RDONLY);
  if (in_fd < 0) exit(2);

  out_fd = creat(argv[2], OUTPUT_MODE);
  if (out_fd < 0) exit(3);
```

Programa copy

```
  while((rd_count = read(in_fd, buffer, BUF_SIZE)) > 0) {
    wt_count = write(out_fd, buffer, rd_count);
    if (wt_count <= 0) exit(4);
  }

  close(in_fd);
  close(out_fd);

  if (rd_count == 0) exit(0);
  else exit(5);
}
```

Streams and File Descriptors

File descriptors provide a primitive, low-level interface to input and output operations. [...]
The main advantage of using the stream interface is that the set of functions for performing actual input and output operations (as opposed to control operations) on streams is much richer and more powerful than the corresponding facilities for file descriptors. The file descriptor interface provides only simple functions for transferring blocks of characters, but the stream interface also provides powerful formatted input and output functions (printf and scanf) as well as functions for character- and line-oriented input and output.

Fonte: http://www.gnu.org/software/libc/manual/html_node/Streams-and-File-Descriptors.html

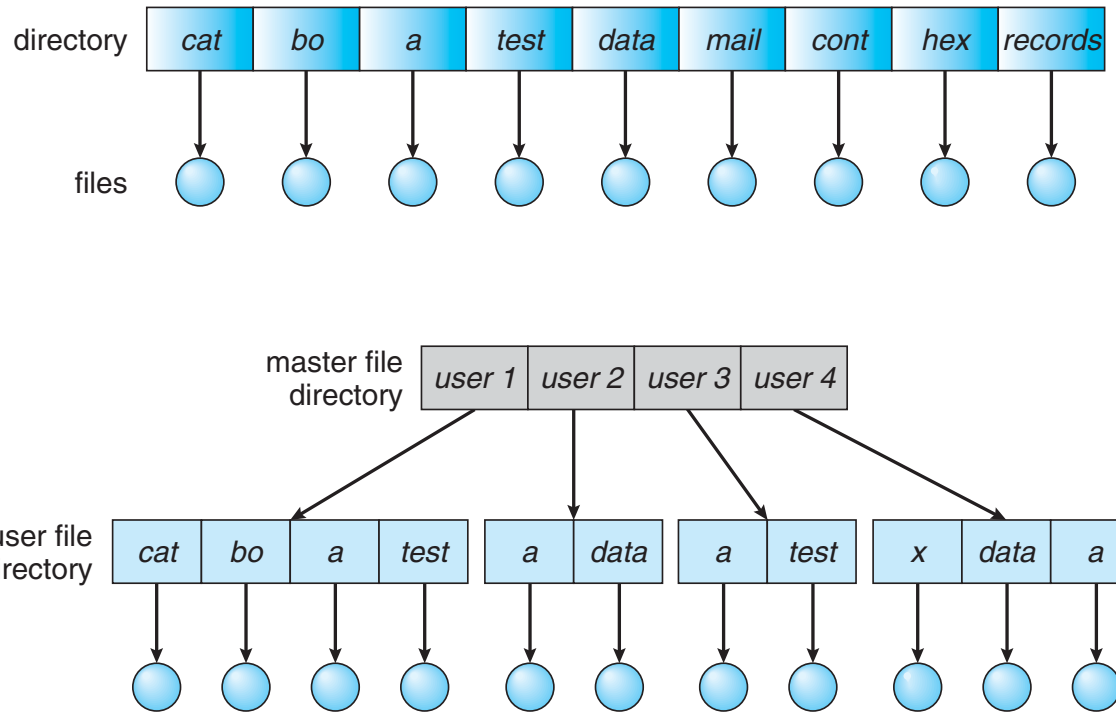
Streams

```
int fprintf(FILE *stream, const char *format, ...);

int fscanf(FILE *stream, const char *format, ...);

FILE *fopen(const char *path, const char *mode);

int fclose(FILE *stream);
```

Veja os exemplos fscanf.c e fscanf2.c

Diretórios

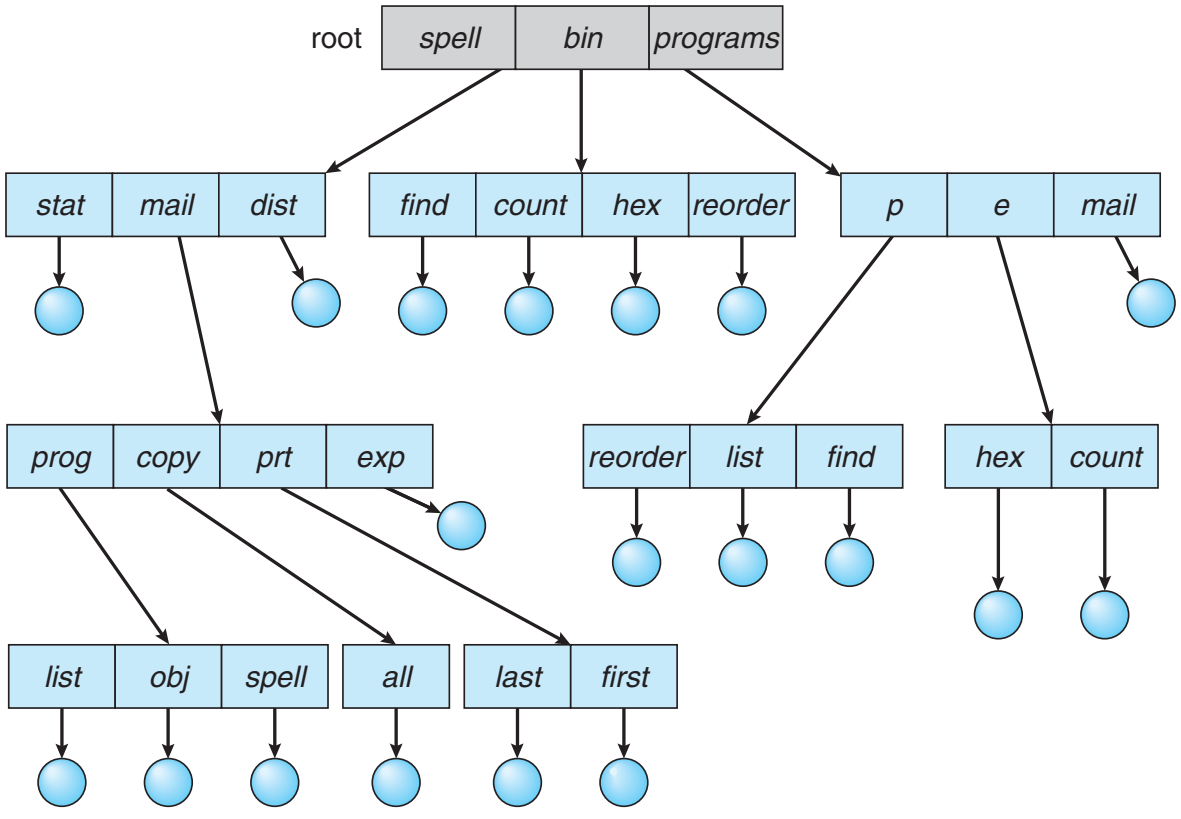
Nível único e dois níveis



Silberschatz: Figuras 11.09 e 11.10

Diretórios

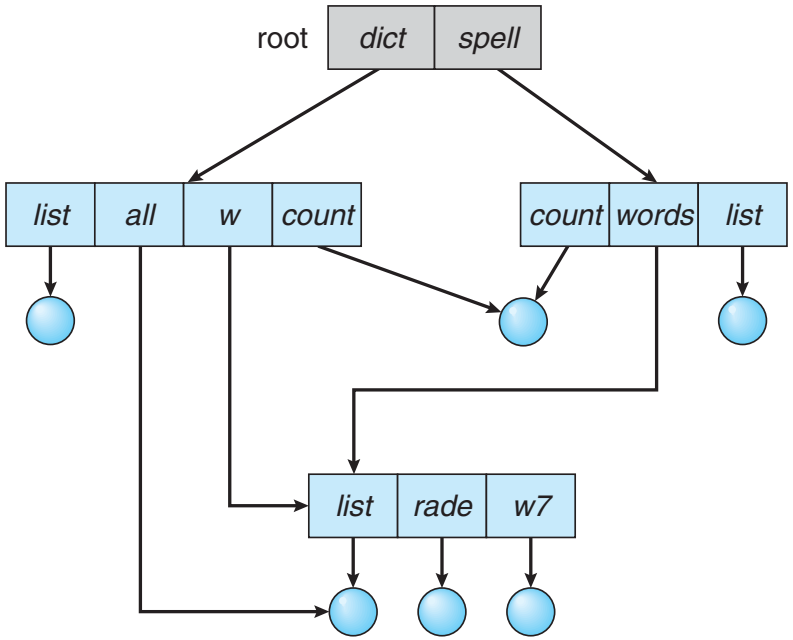
Árvore



Silberschatz: Figura 11.11

Diretórios

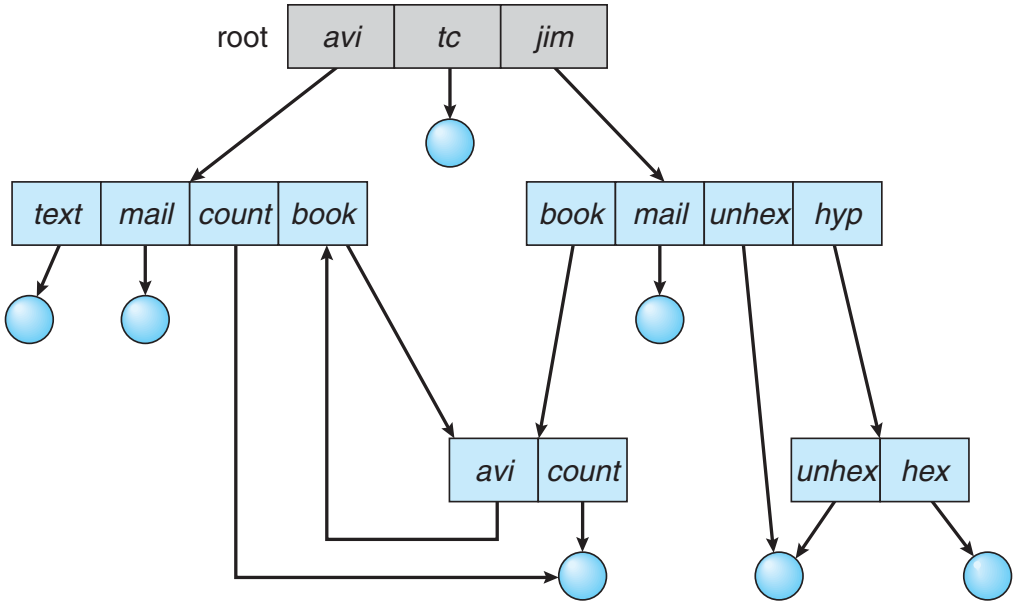
Grafo acíclico



Silberschatz: Figura 11.12

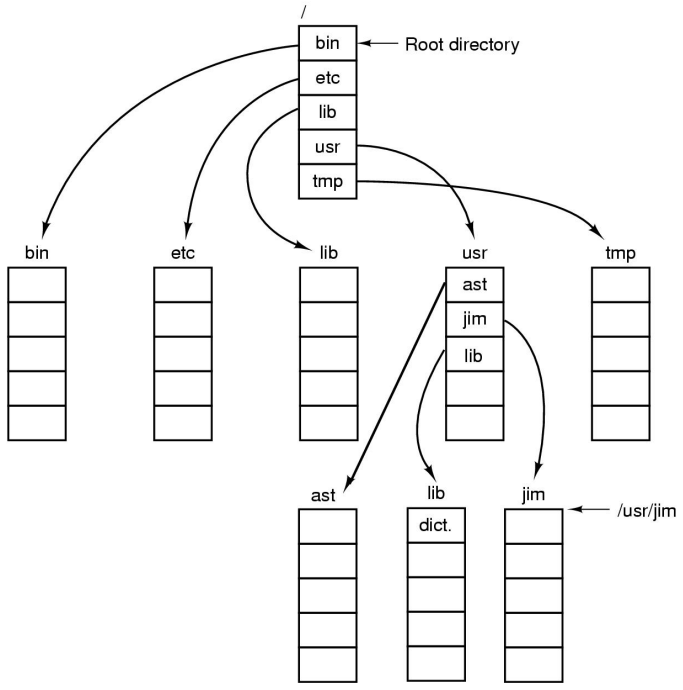
Diretórios

Grafo geral



Silberschatz: Figura 11.13

Caminhos



Tanenbaum: Figura 6.10

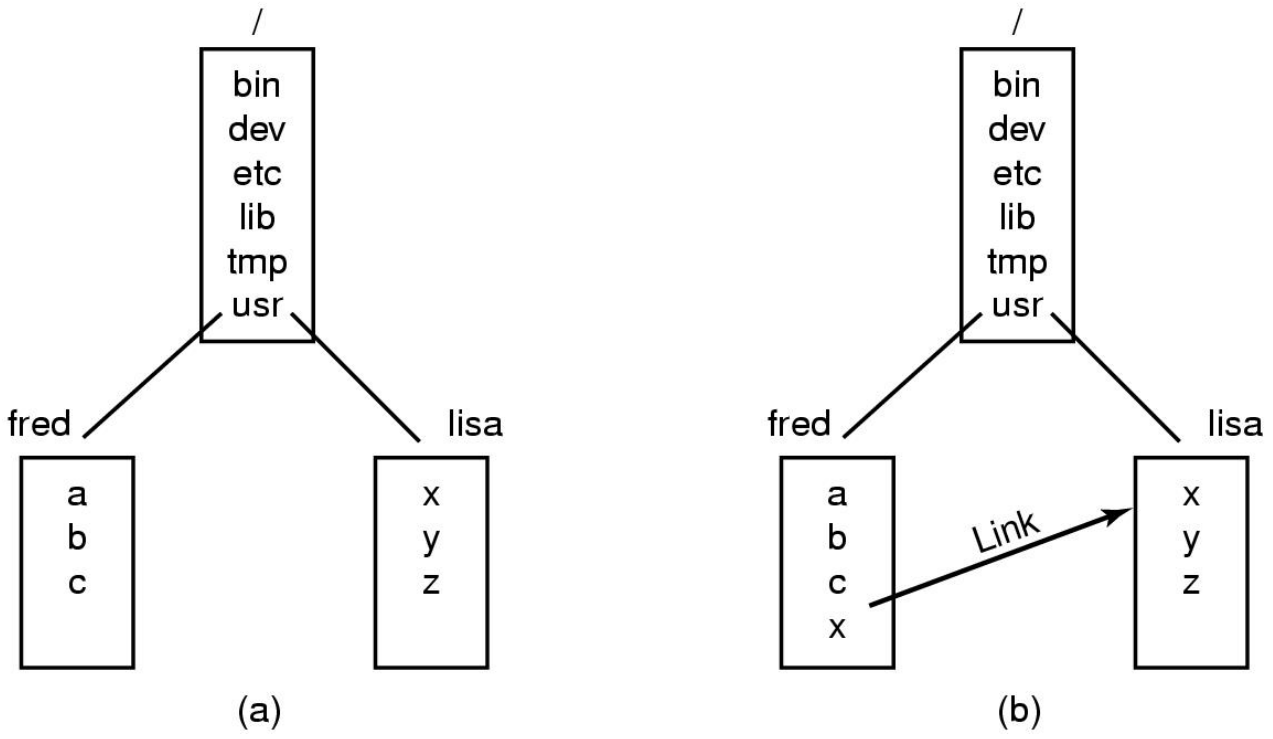
- Absolutos ou relativos?

Operações sobre diretórios

- create
- delete
- opendir
- closedir
- readdir
- rename
- link
- unlink

Veja `struct inode_operations` em `linux-4.X.Y/include/linux/fs.h` Veja o código `dir.c`

Links



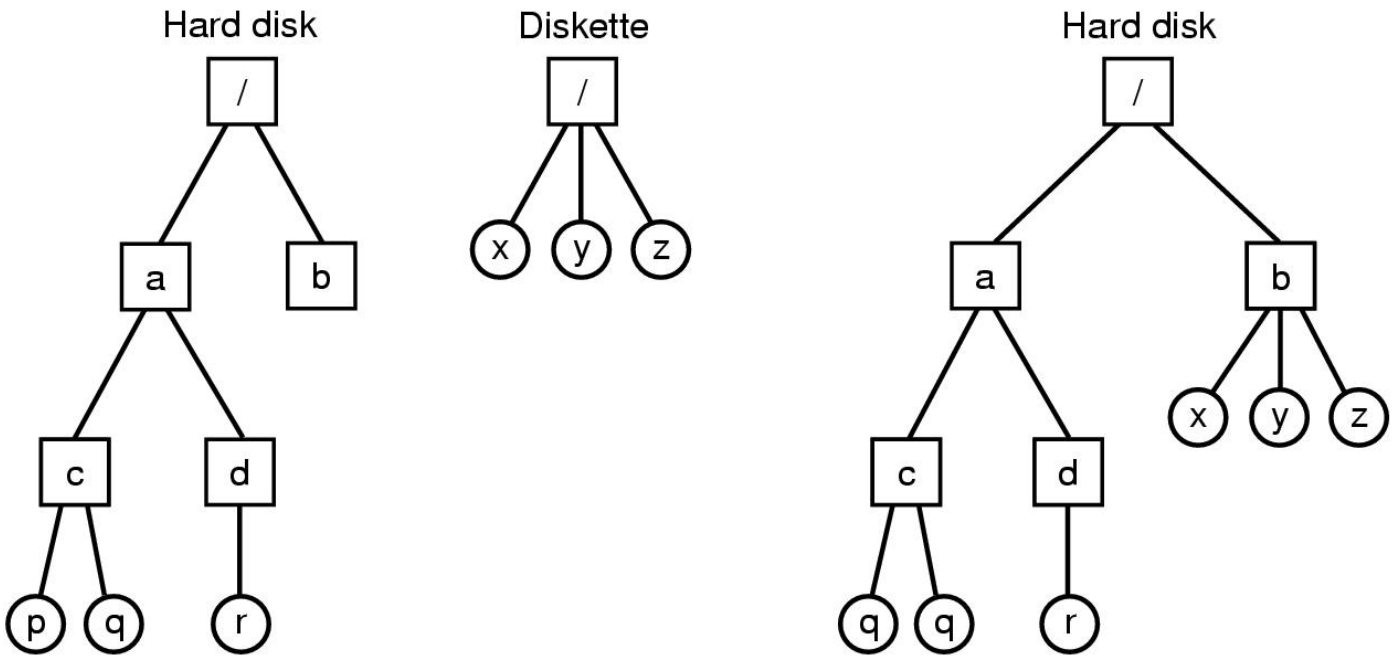
Tanenbaum: Figura 10.25

- links simbólicos ou hard links?
- caminhos absolutos ou relativos?
- como copiar?

Exercício com links e cópia

- Crie um diretório contendo
  - um arquivo regular
  - um hardlink para este arquivo
  - um link com caminho relativo para este arquivo
  - um link com caminho absoluto para este relativo
- Copie este diretório com `cp -r`
- Como ficou o resultado? Você concorda com as opções padrão do sistema?

Mount



Tanenbaum: Figura 10.26

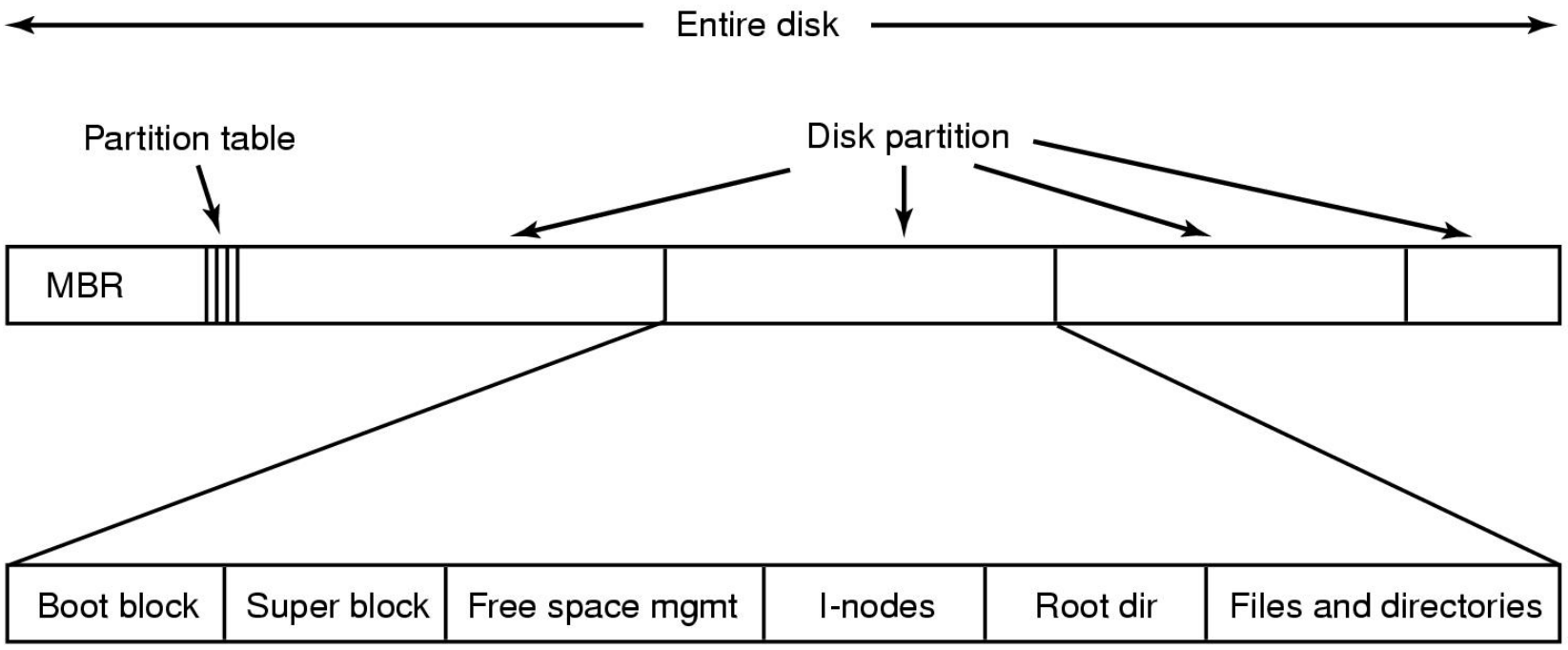
Mount

```
$ dd if=/dev/zero of=hd.dmp bs=4k count=256
$ mkfs.ext2 hd.dmp
$ mkdir -p mnt
$ sudo mount -t ext2 -o loop hd.dmp mnt
[sudo] password for islene:
```

Questões de Implementação

- Como os arquivos são armazenados
- Como o espaço livre é gerenciado
- Eficiência
- Confiabilidade

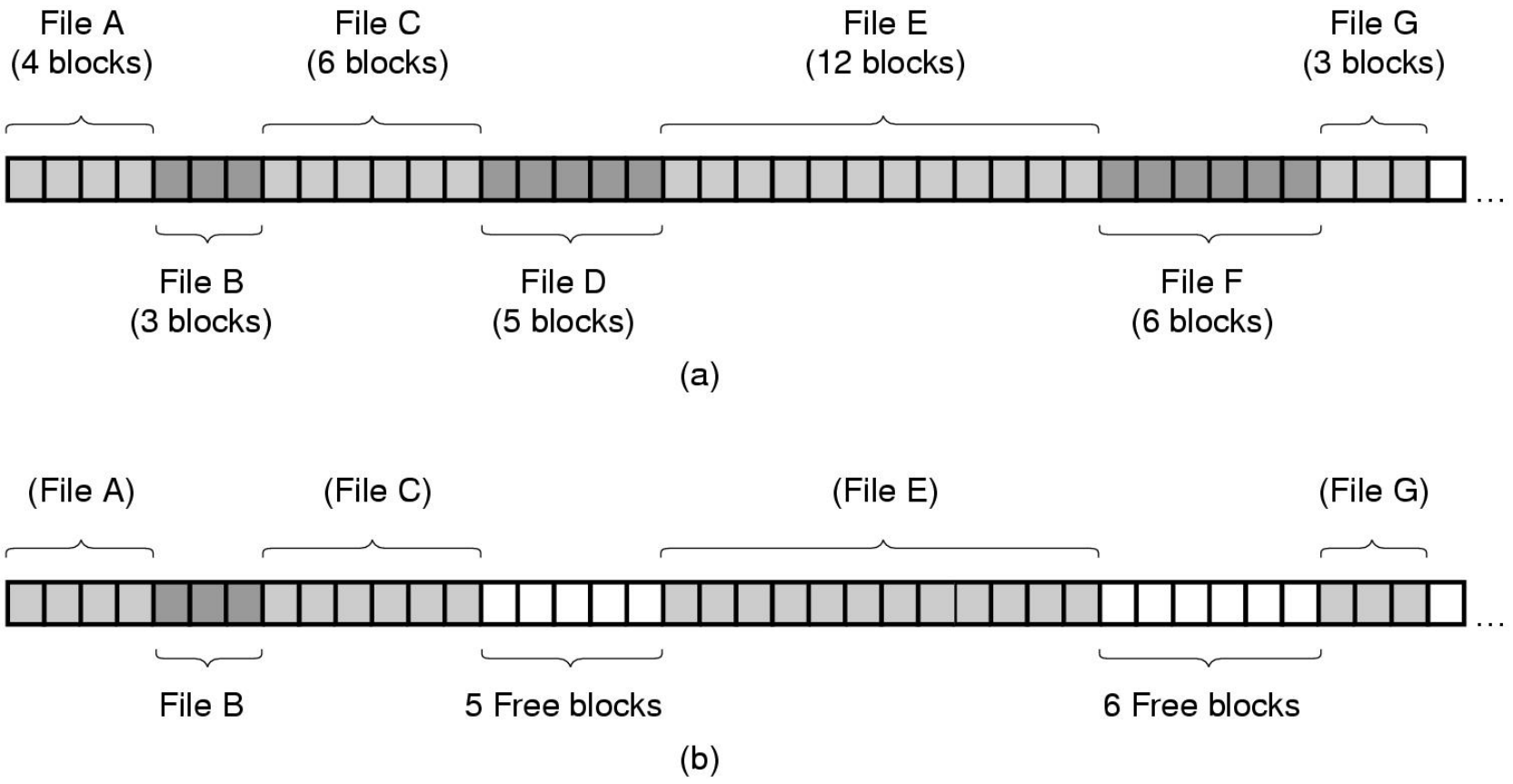
Layout



Tanenbaum: Figura 6.11

- MBR (Master Boot Record)
- Tabela de partições
- Boot block

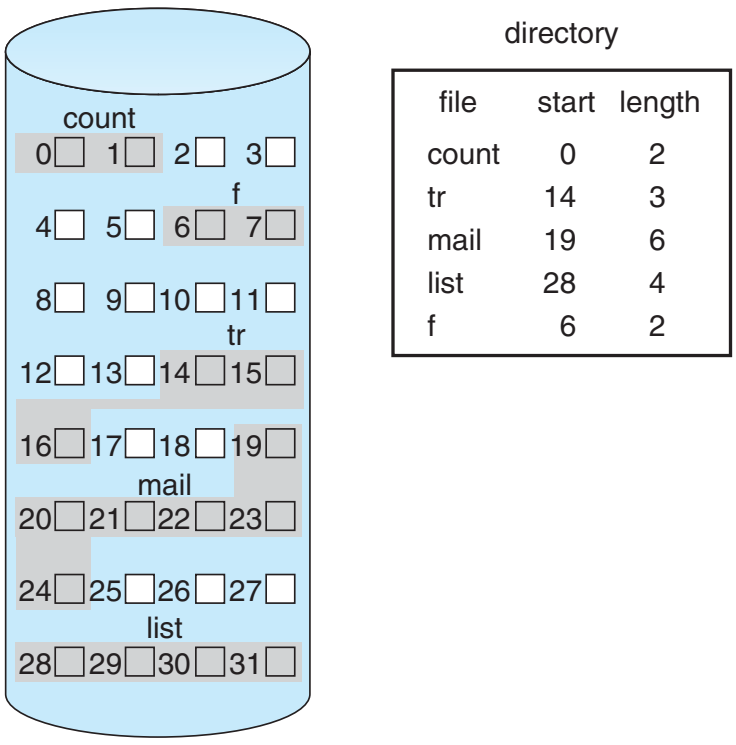
Alocação contínua



Tanenbaum: Figura 6.12

Alocação contínua

Fragmentação externa

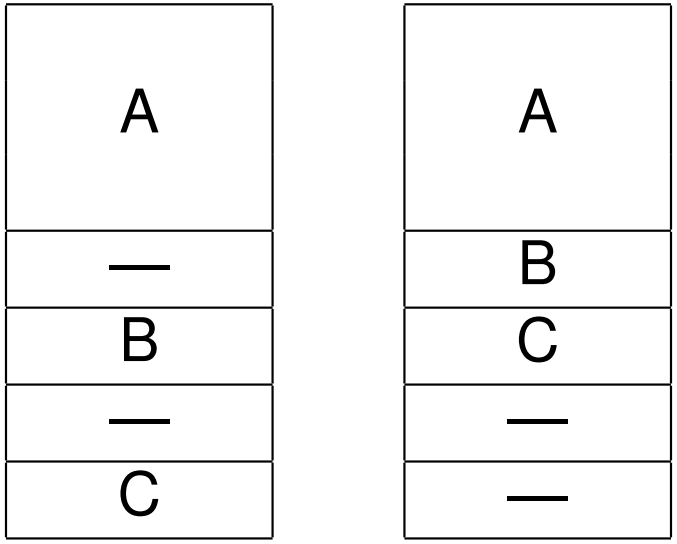


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

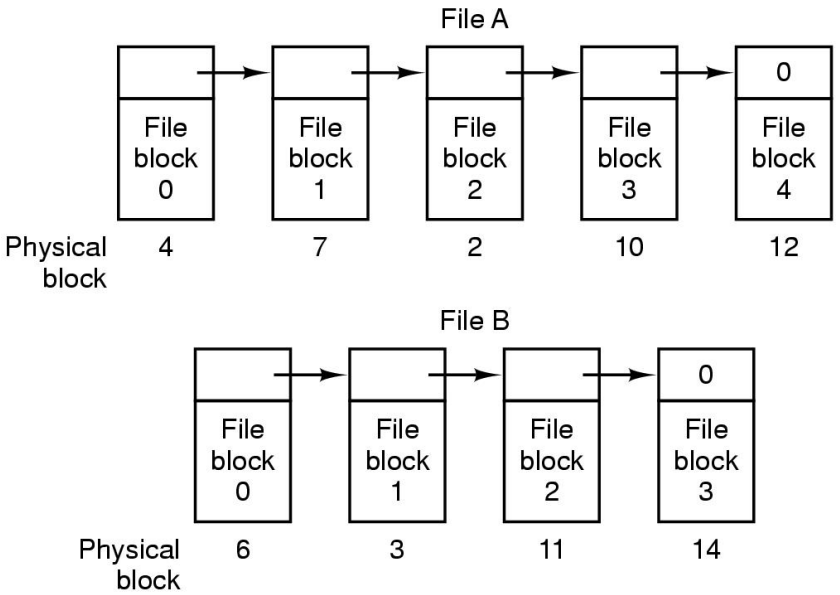
Silberschatz: Figura 12.05

Compactação do disco

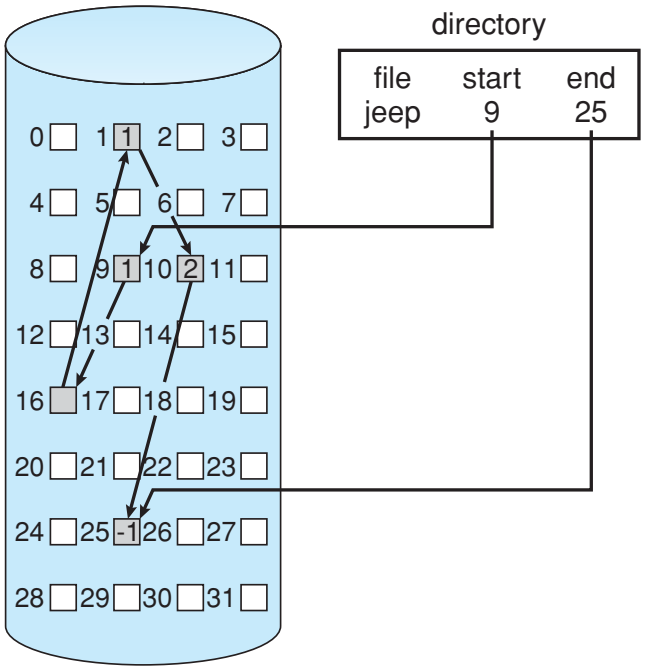


Quais são as desvantagens de se utilizar este método?

Lista ligada de blocos

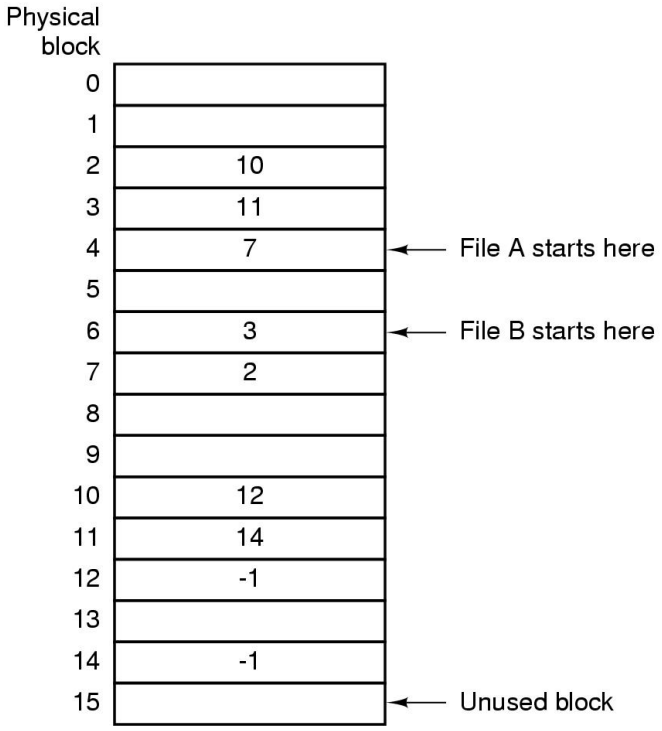


Tanenbaum: Figura 6.13

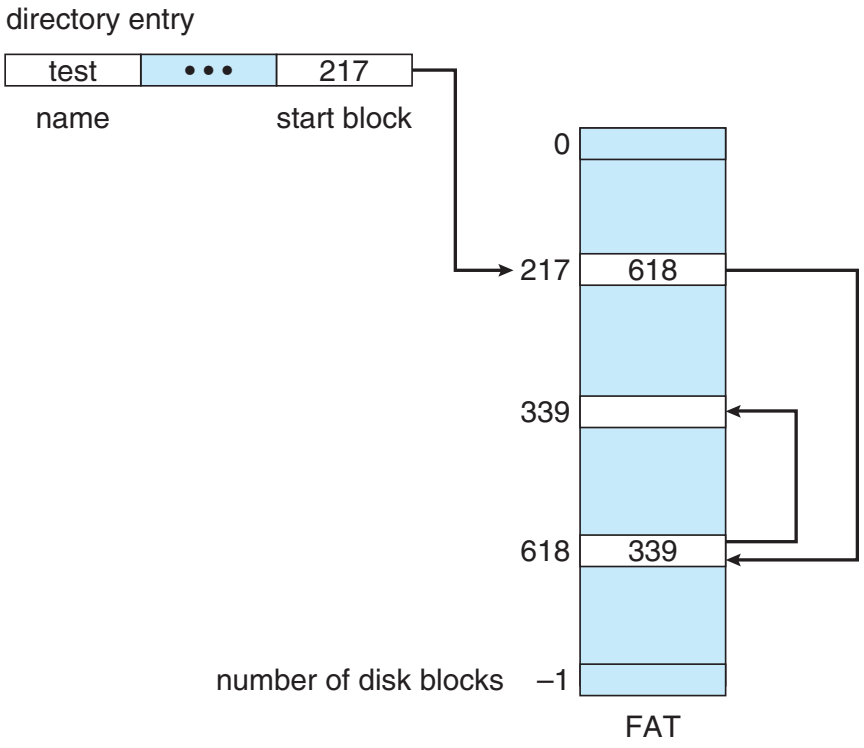


Silberschatz: Figura 12.06

File Allocation Table (FAT)

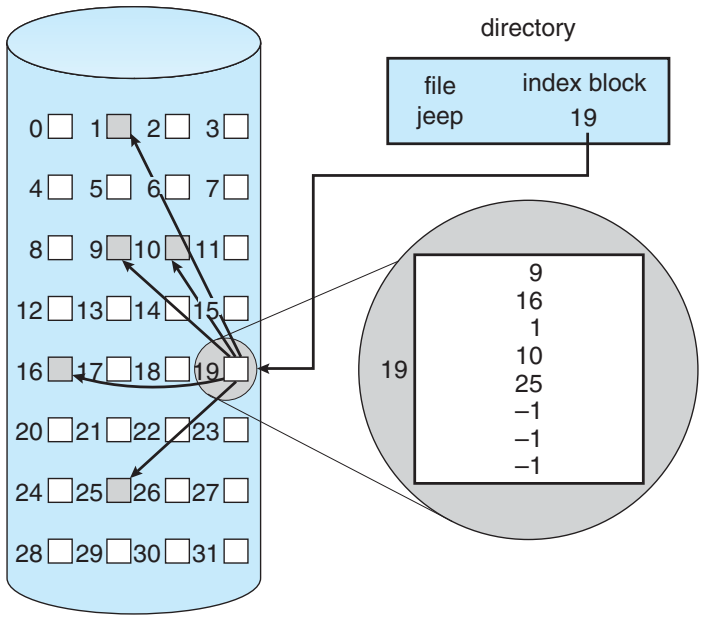


Tanenbaum: Figura 6.14



Silberschatz: Figura 12.07

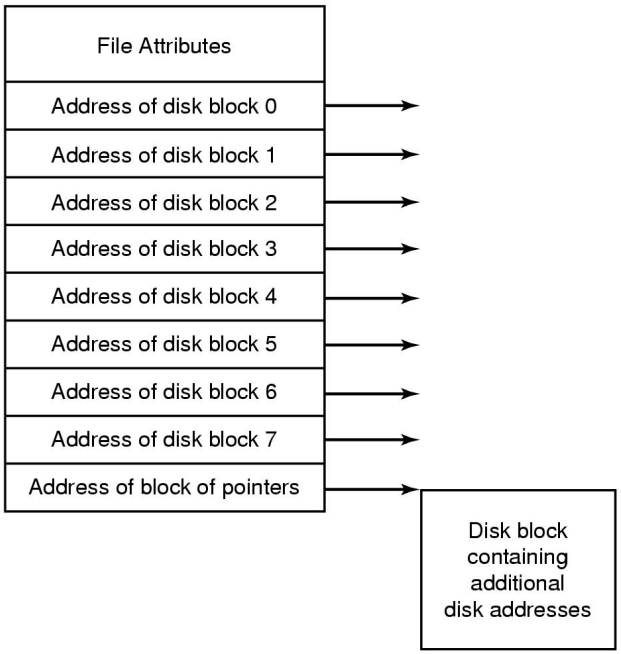
Index blocks



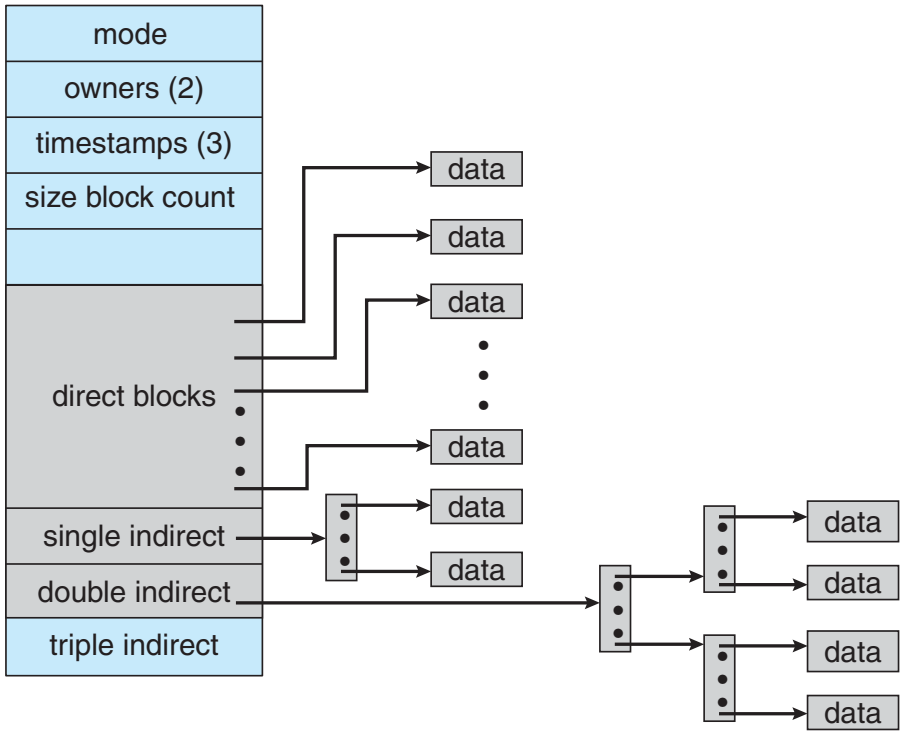
Silberschatz: Figura 12.08

Quais são as vantagens e desvantagens deste esquema em comparação à FAT?

I-node

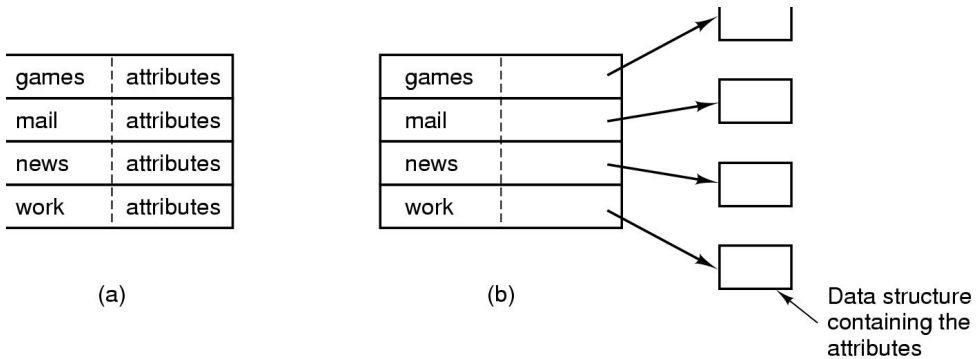


Tanenbaum: Figura 6.15



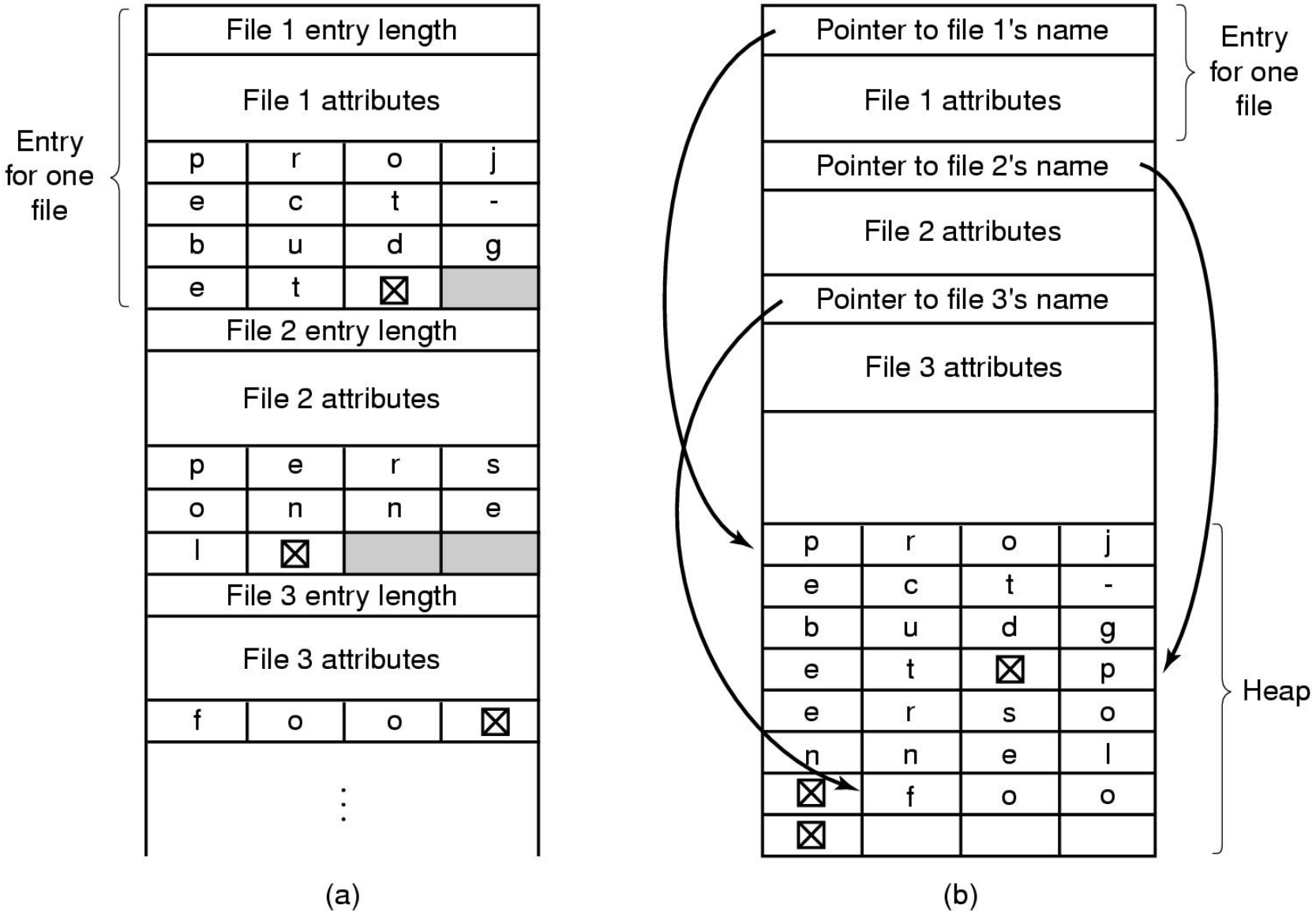
Silberschatz: Figura 12.09

Implementação de diretórios



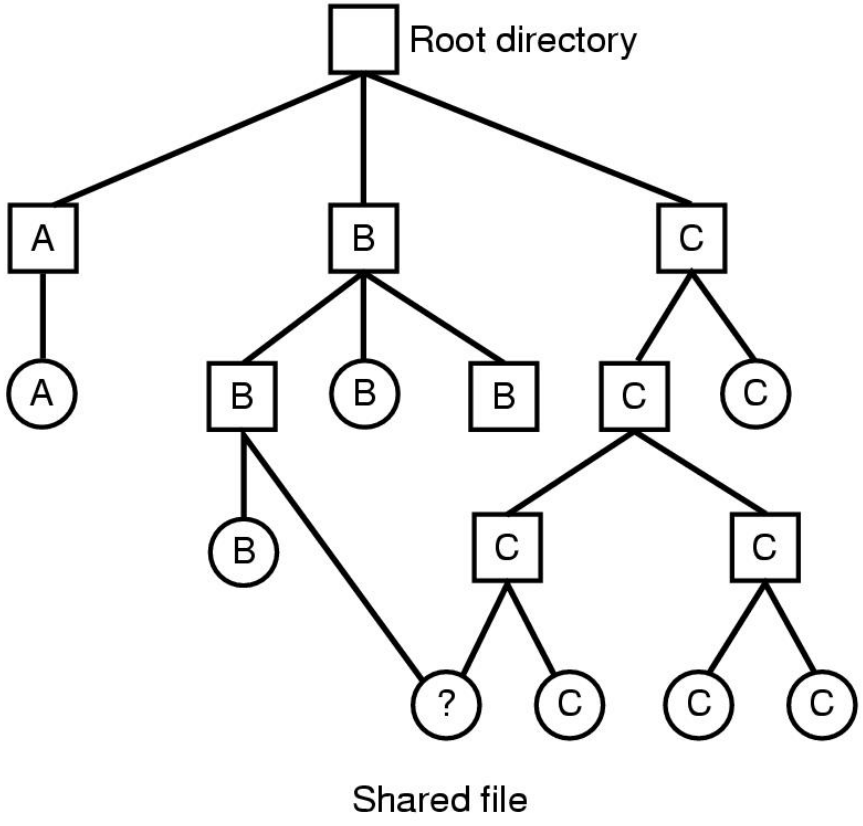
Tanenbaum: Figura 6.16

Nomes de tamanho variável



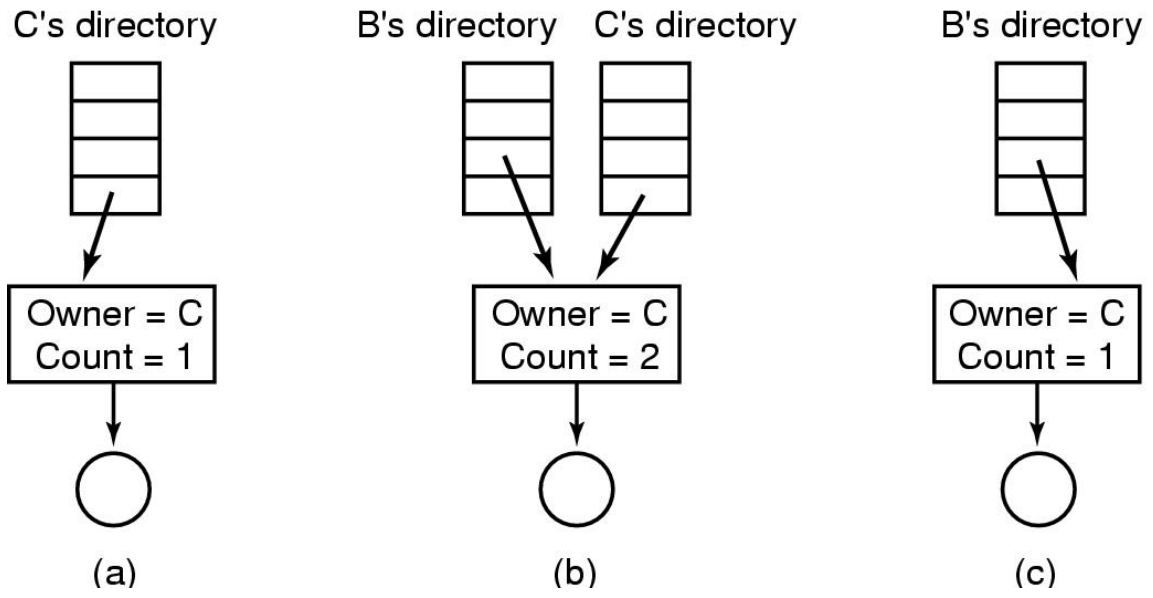
Tanenbaum: Figura 6.17

Arquivos compartilhados



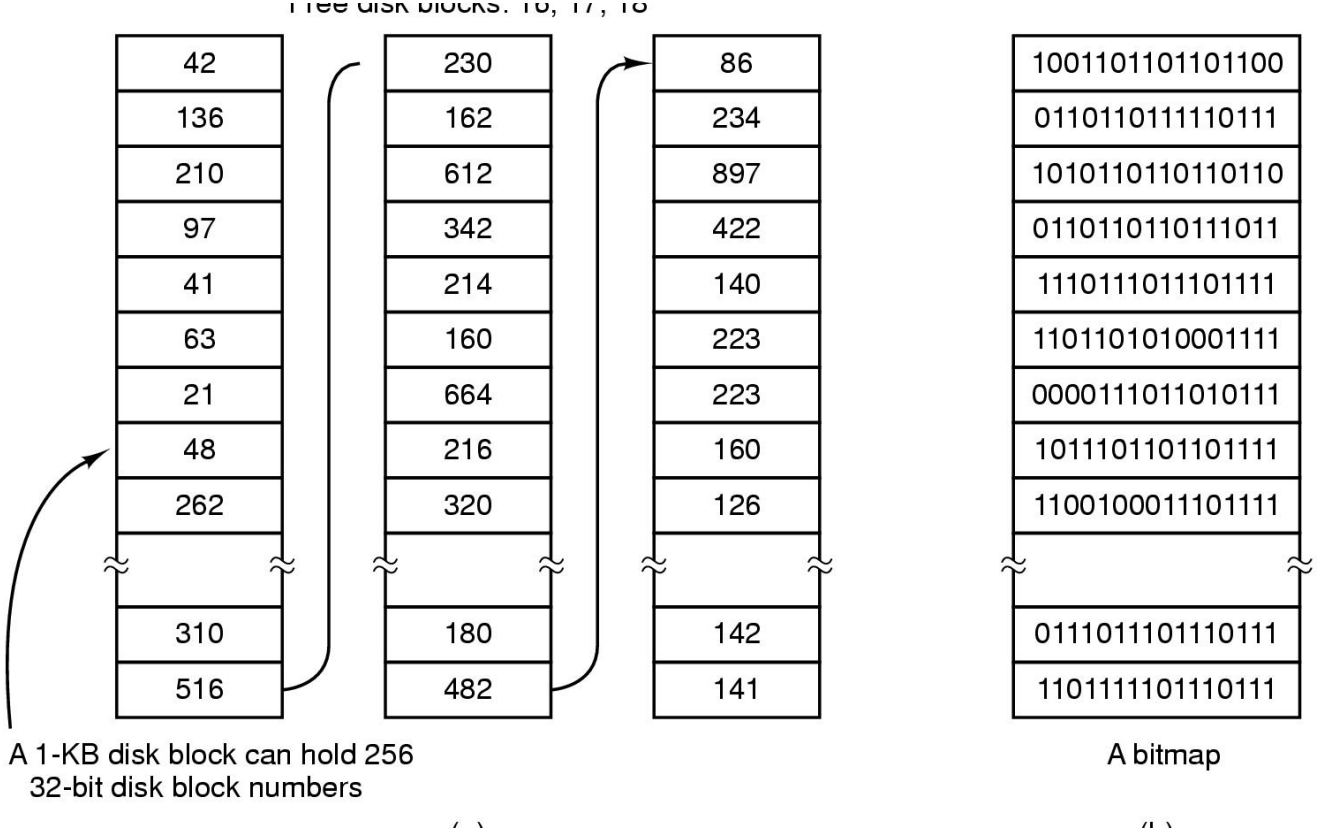
Tanenbaum: Figura 6.18

Arquivos compartilhados



Tanenbaum: Figura 6.19

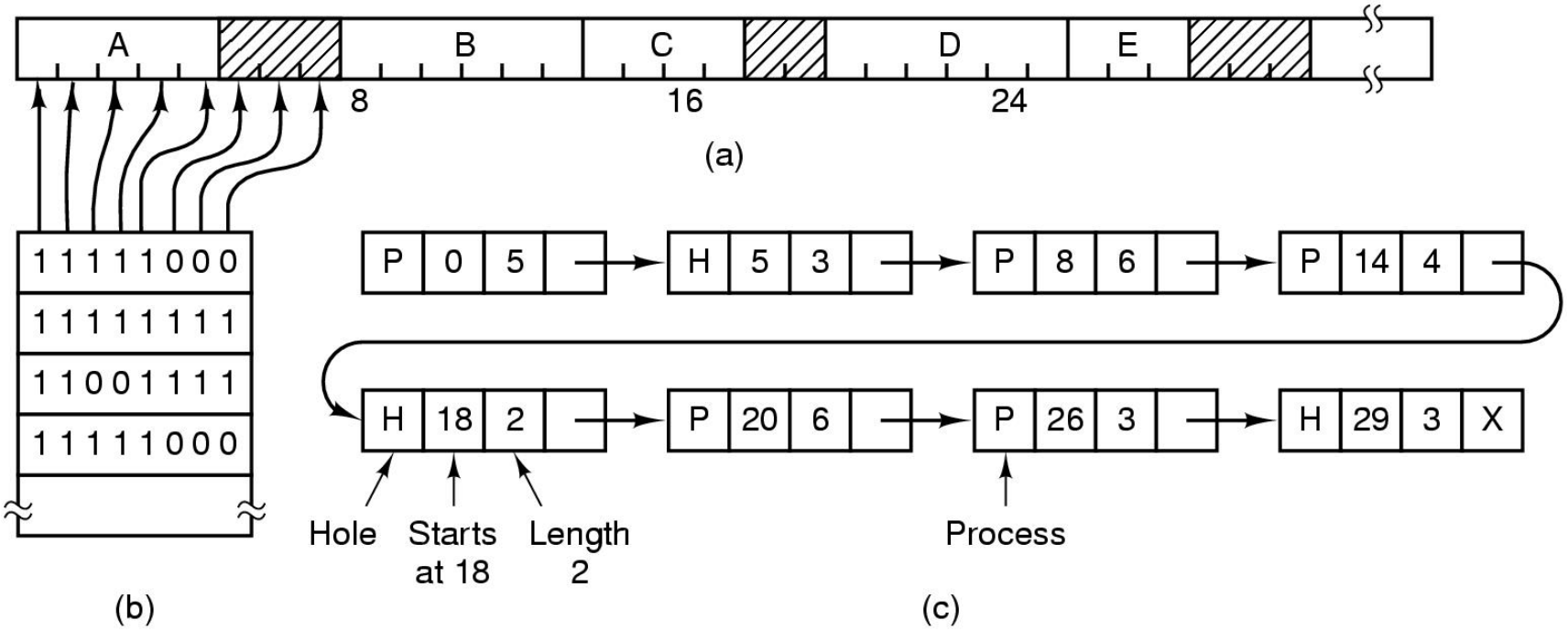
Lista de livres e bitmaps



Tanenbaum: Figura 6.21

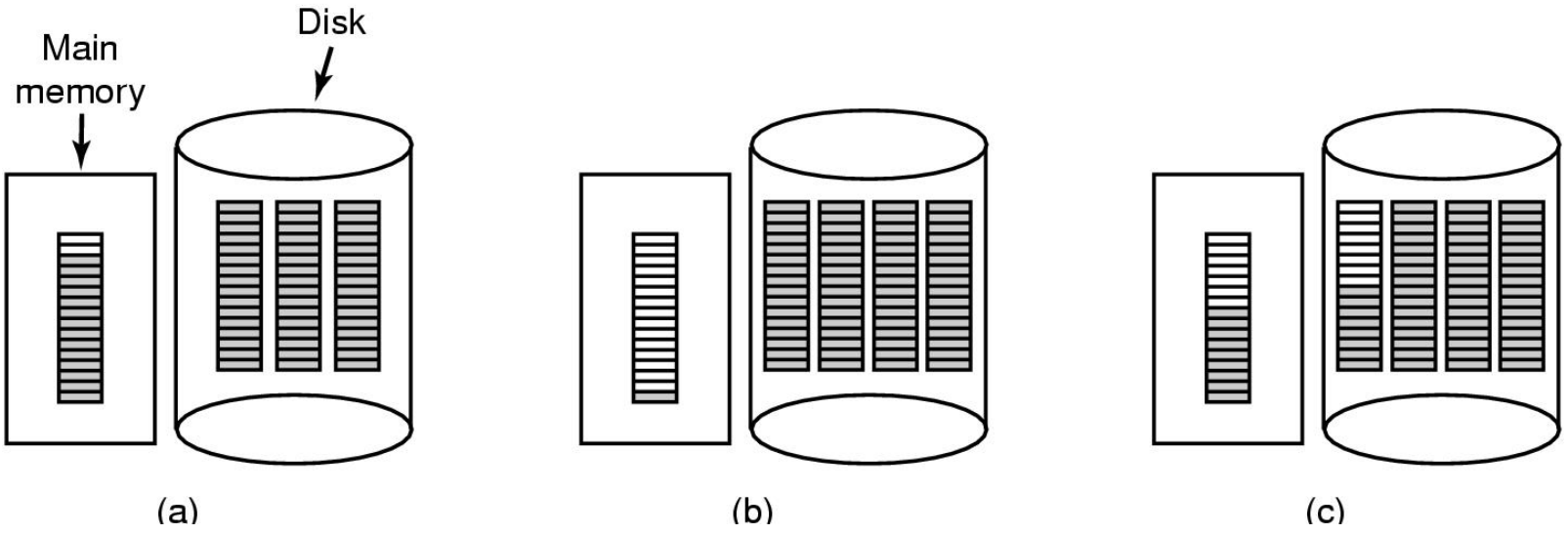
Bitmaps e lista de livres

Relembrando gerência de memória...



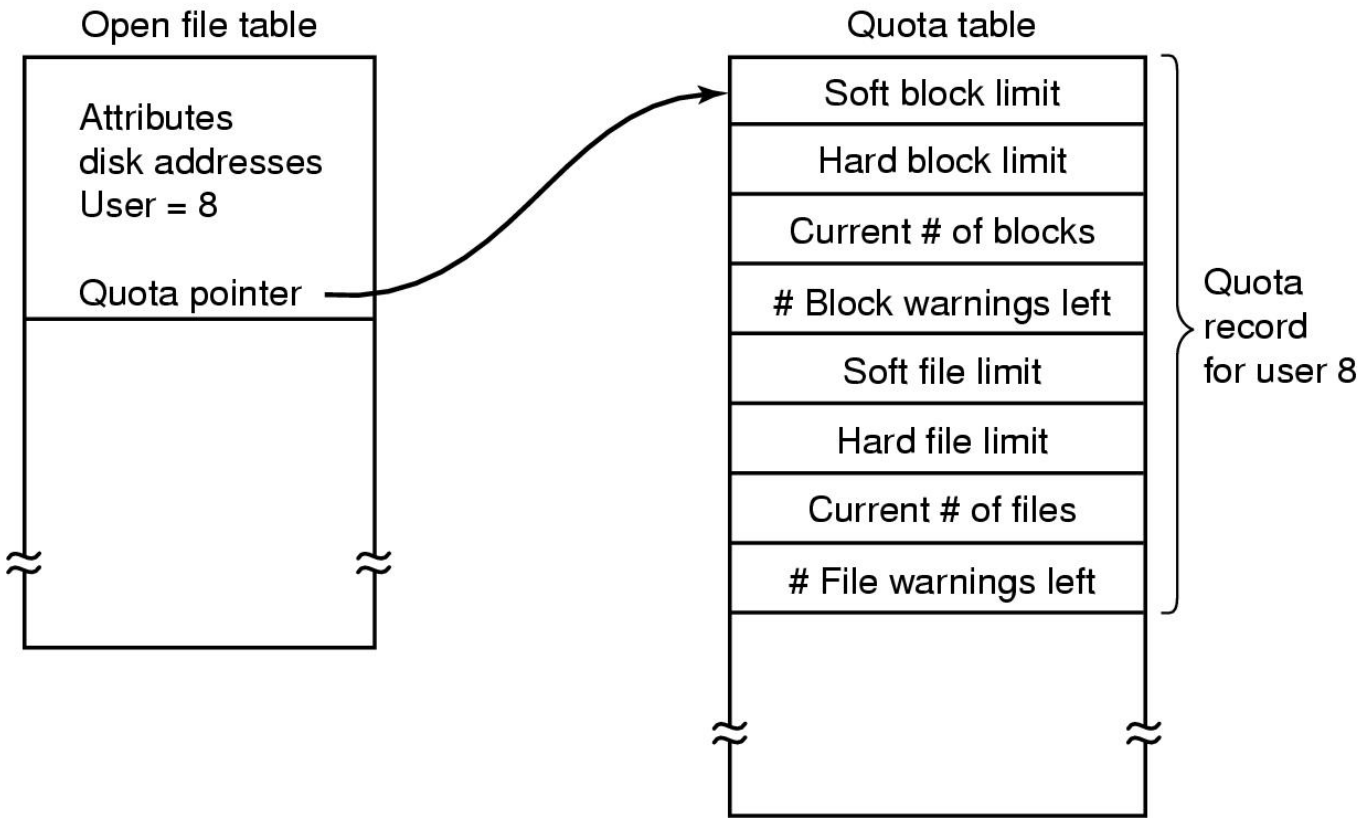
Tanenbaum: Figura 4.7

Lista de livres em memória



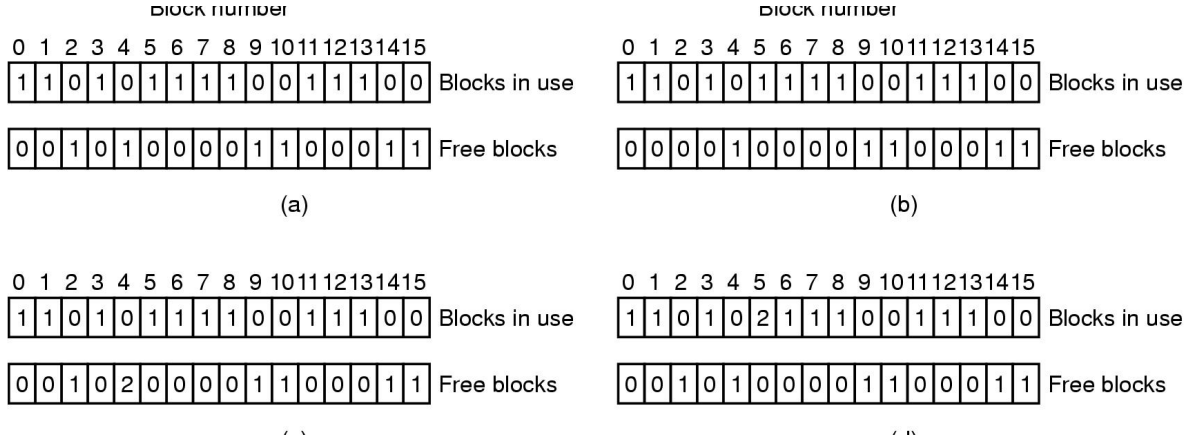
Tanenbaum: Figura 6.22

Gerência de quotas



Tanenbaum: Figura 6.23

Consistência do sistema de arquivos



Tanenbaum: Figura 6.26

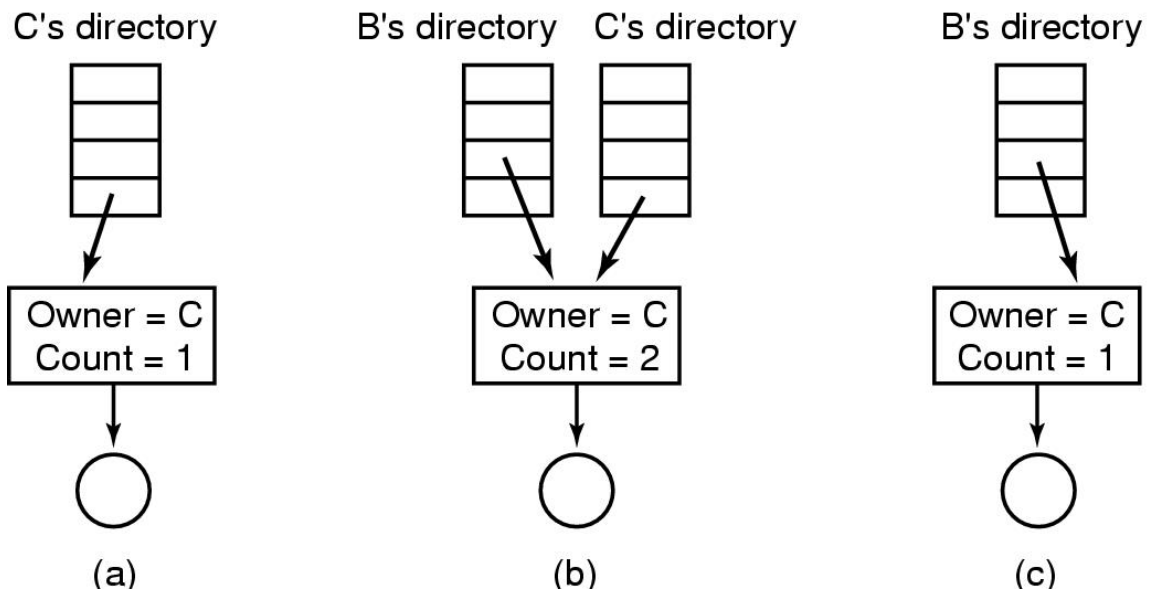
(a) consistente

(b) bloco faltando

(c) duplicação na lista de livres

(d) duplicação nos dados

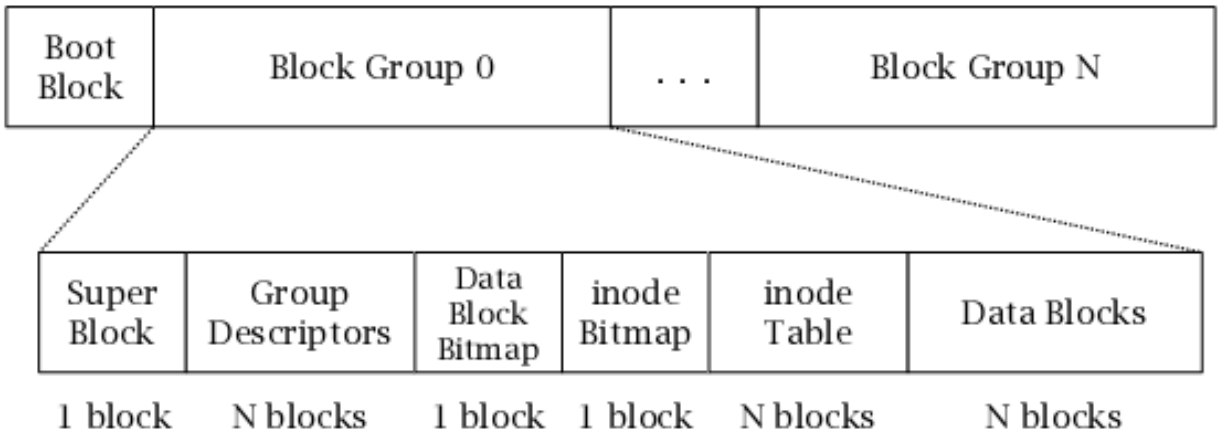
Consistência do sistema de arquivos



Tanenbaum: Figura 6.19

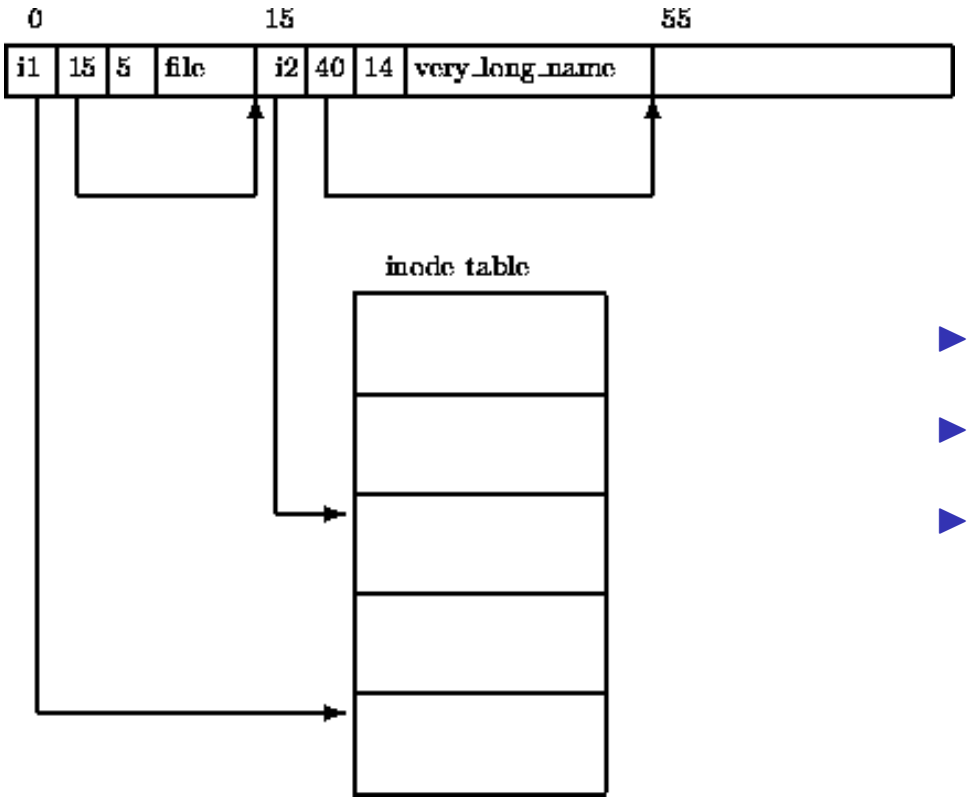
Verificar se todos os contadores estão corretos

Estrutura do Ext2



- Super block: número de blocos no disco, tamanho dos blocos, ...
- Group descriptors: localização dos bitmaps inodes/blocos e da tabela de i-nodes, ..

Diretórios no Ext2



- inode
- name length
- name

Ext2 e crashes

- Operações como criação e remoção envolvem várias escritas em disco
- Um crash pode ocorrer durante o processo e
  - um bloco foi incluído no inode, mas não está marcado como em uso no bitmap
  - um bloco foi removido do inode, mas não está marcado como livre no bitmap
  - um bloco foi reutilizado, mas o inode original ainda aponta para ele.
  - um inode foi marcado como em uso, mas não há entrada em diretório apontando para ele.
  - ...

`fsck` e recuperação

Testando com `debugfs`

- Crie um sistema de arquivos ext2 com `mkfs.ext2`
- Utilize o `debugfs` para corromper este sistema
- Verifique se o `fsck` encontra os erros

Criando e fazendo verificação inicial

```
$ dd if=/dev/zero of=ext2.dmp bs=1k count=256
$ mkfs.ext2 ext2.dmp
$ fsck ext2.dmp
e2fsck 1.42.13 (17-May-2015)
ext2.dmp: clean, 11/32 files, 23/256 blocks
$ fsck -f ext2.dmp
e2fsck 1.42.13 (17-May-2015)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
ext2.dmp: 11/32 files (0.0% non-contiguous), 23/256 blocks
```

Alterando bitmap dos inodes

```
$ debugfs -w ext2.dmp
  debugfs: write a.txt a.txt
  debugfs:  ls
  2  (12) .    2  (12) ..    11  (20) lost+found    12  (980) a.
  debugfs: freei <12>
```

Como o `fsck` consegue corrigir o erro?

lost+found

```
 debugfs: write a.txt a.txt
 debugfs: modify_inode <12>
      Link count      [1]  2
 debugfs:  ls
  2  (12) .  2  (12) ..  11  (20) lost+found  12  (980) a.txt
 debugfs: rm a.txt
 debugfs: quit
$ fsck.ext2 -f ext2.dmp
```

- Por que o inode <12> seria colocado no diretório de “achados e perdidos”?