MC504 - Sistemas Operacionais

Preparação para prova II

Islene Calciolari Garcia

Instituto de Computação - Unicamp

Primeiro Semestre de 2014

Preparação para prova II

- Tópicos básicos
- Processos e threads
- Gerência de memória
- Sistema de arquivos
- Entrada e saída

Responsabilidades de um SO

Programas de Aplicação		
Shell	Compiladores	Editores
Sistema operacional		
Hardware		

- Máquina estendida
- Gerenciador de recursos

Responsabilidades de um SO

- Gerência de processos
  - processos e threads
  - algoritmos de escalonamento
- Gerência de memória
  - swapping, paginação, segmentação
- Gerência de arquivos
  - armazenamento em disco, compartilhamento de arquivos, *backups*, consistência
- Gerência de I/O

Arquitetura de sistemas operacionais

Sistemas monolíticos

- Sem estrutura
- Conjunto de funções dependentes

Sistemas baseados em camadas

- Bem estruturado
- Nível inferior não pode invocar funções do nível superior

Device drivers

- Software que "conversa" com o controlador
- Os fabricantes devem fornecer device drivers para os sistemas operacionais
- Como acoplar um device driver ao kernel:
  - relink e reboot
  - entrada em um arquivo e reboot
  - on-the-fly

Como programar os dispositivos?

- Instruções especiais

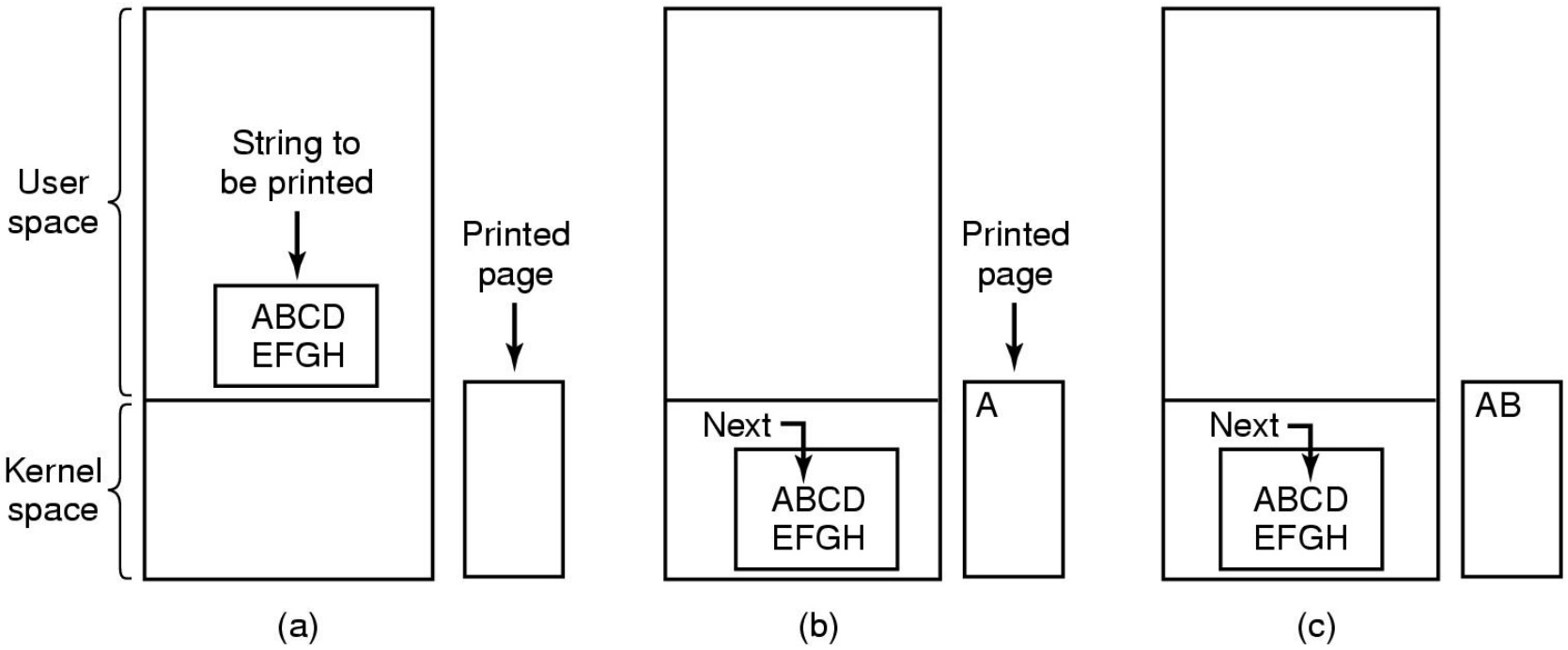
  ```
  IN REG, PORT
  OUT PORT, REG
  ```

- Memory-mapped I/O

  ```
  MOV REG, ADDR
  ```

  Conforme o valor de ADDR, a instrução MOV fará acesso a uma palavra de memória ou dispositivo

Imprimindo uma string



Tanenbaum: Figura 5.6

Imprimindo uma string

Programmed I/O

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
      while (*printer_status_reg != READY) ;  /* loop until ready */
      *printer_data_register = p[i];       /* output one character */
}
return_to_user( );
```

Tanenbaum: Figura 5.7

Trecho de código do kernel

Imprimindo uma string

Interrupt-driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

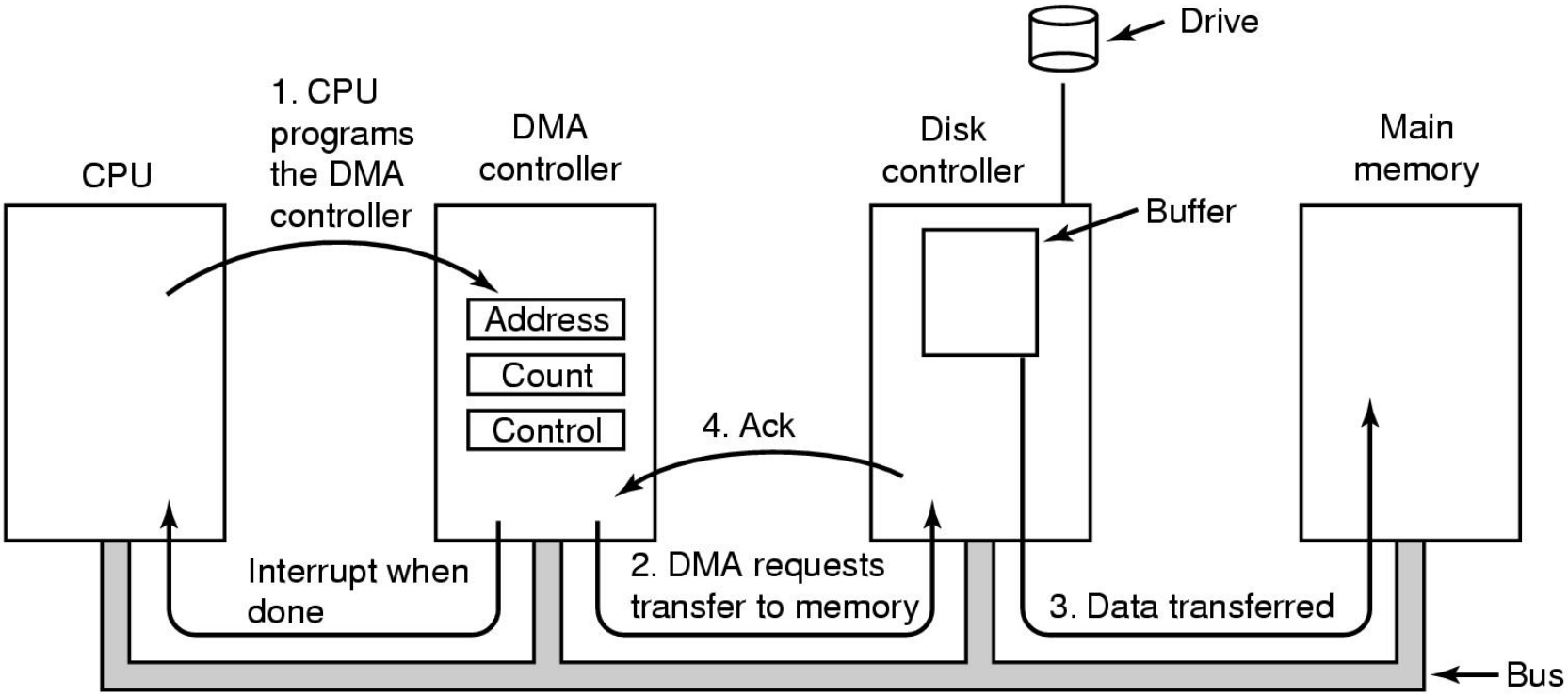
```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Tanenbaum: Figura 5.8

(a) Trecho de código do kernel
(b) Tratador da interrupção

Direct Memory Access (DMA)



Tanenbaum: Figura 5.4

Imprimindo uma string
DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Tanenbaum: Figura 5.9

(a) Chamada de sistema
(b) Tratador de interrupção

Chamada ao sistema

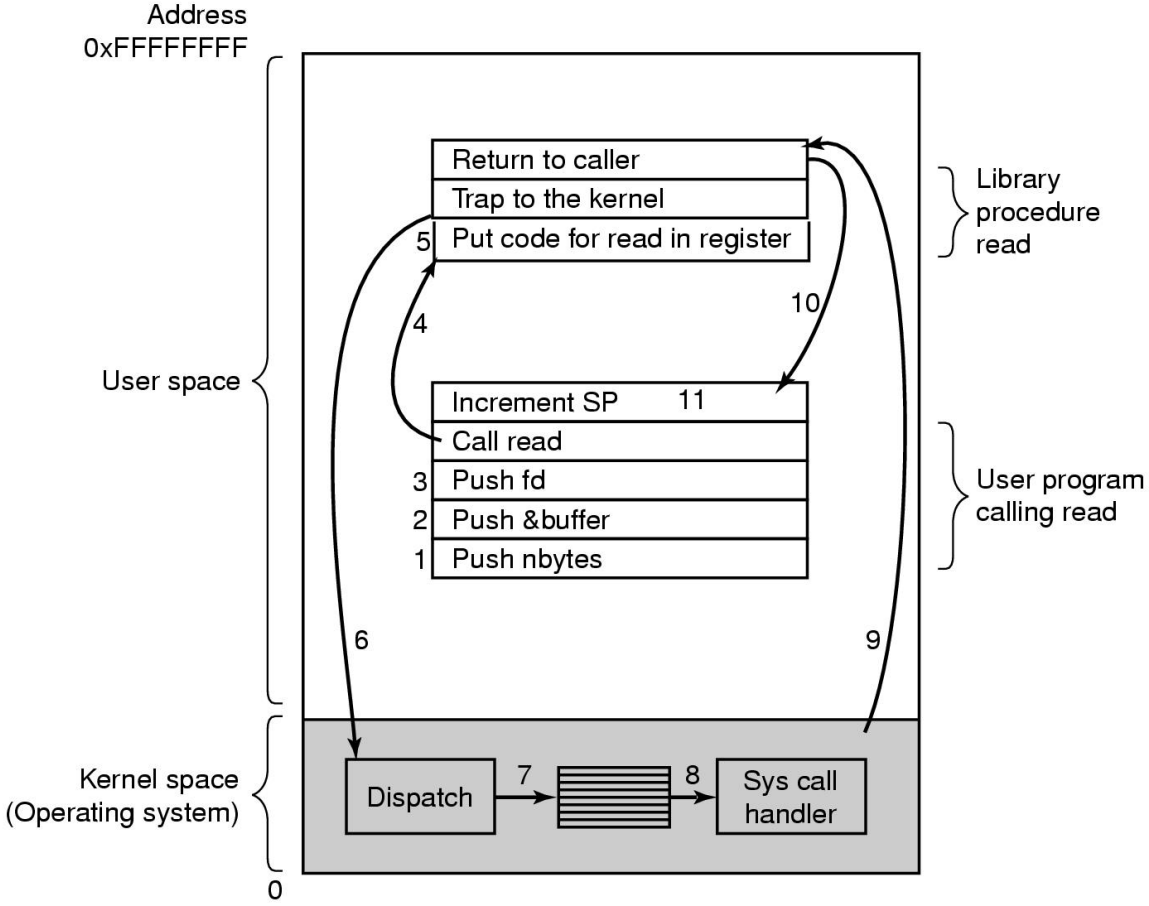
- Tipo especial de chamada de procedimento
- Passa do modo usuário para o modo kernel
- Instrução TRAP

Modo usuário e modo kernel

- Modo usuário
  - acesso restrito ao conjunto de instruções
- Modo kernel
  - acesso total ao conjunto de instruções

Esta distinção permite proteger os programas e o próprio SO de usos indevidos por parte de outros programas.
Um programa do usuário começa a sua execução em modo usuário, mas durante as chamadas de sistema estará executando em modo kernel.

cont = read(fd, buffer, nbytes)



Tanenbaum: Figura 1.17

Exemplos de chamadas de sistema

Gerência de processos

- `pid = fork();`
- `waitpid(pid, &statloc, options);`
- `s = execve(name, argv, environp);`
- `exit(status);`

Exemplos de chamadas de sistema

Gerência de arquivos

- `fd = open(file, how);`
- `s = close(fd);`
- `n = read(fd, buffer, nbytes);`
- `n = write(fd, buffer, nbytes);`
- `pos = lseek(fd, offset, whence);`
- `s = stat(name, &buf);`

Exemplos de chamadas de sistema

Gerência de arquivos e diretórios

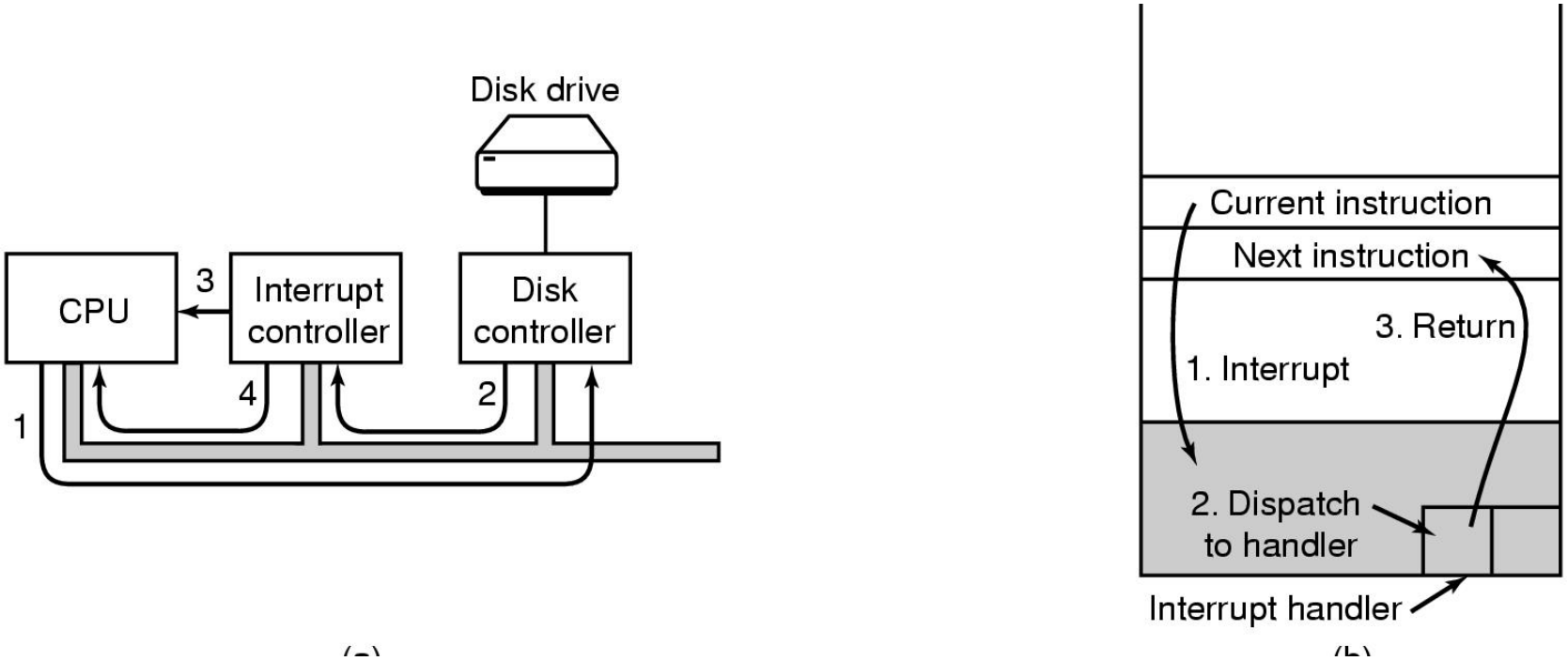
- `s = mkdir(name,mode);`
- `s = rmdir(name);`
- `s = link(name1,name2);`
- `s = unlink(name);`
- `s = mount(special,name,flag);`
- `s = umount(special);`

Exemplos de chamadas de sistema

Diversas

- `s = chdir(dirname);`
- `s = chmod(name,mode);`
- `s = kill(pid,signal);`
- `seconds = time(&seconds);`

Tratamento de interrupções

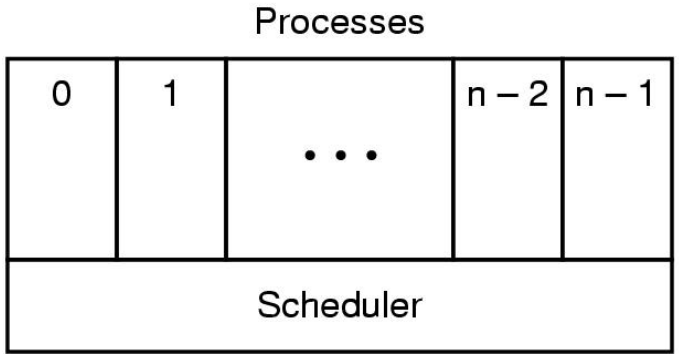


Tanenbaum: Figura 1.10

Tratamento de sinais

- Tratam a ocorrência de condições excepcionais
- Tipos de sinais
  - Divisão por zero
  - Acesso inválido à memória
  - Interrupção do programa
  - Término de um processo filho
  - Alarme

Escalonador

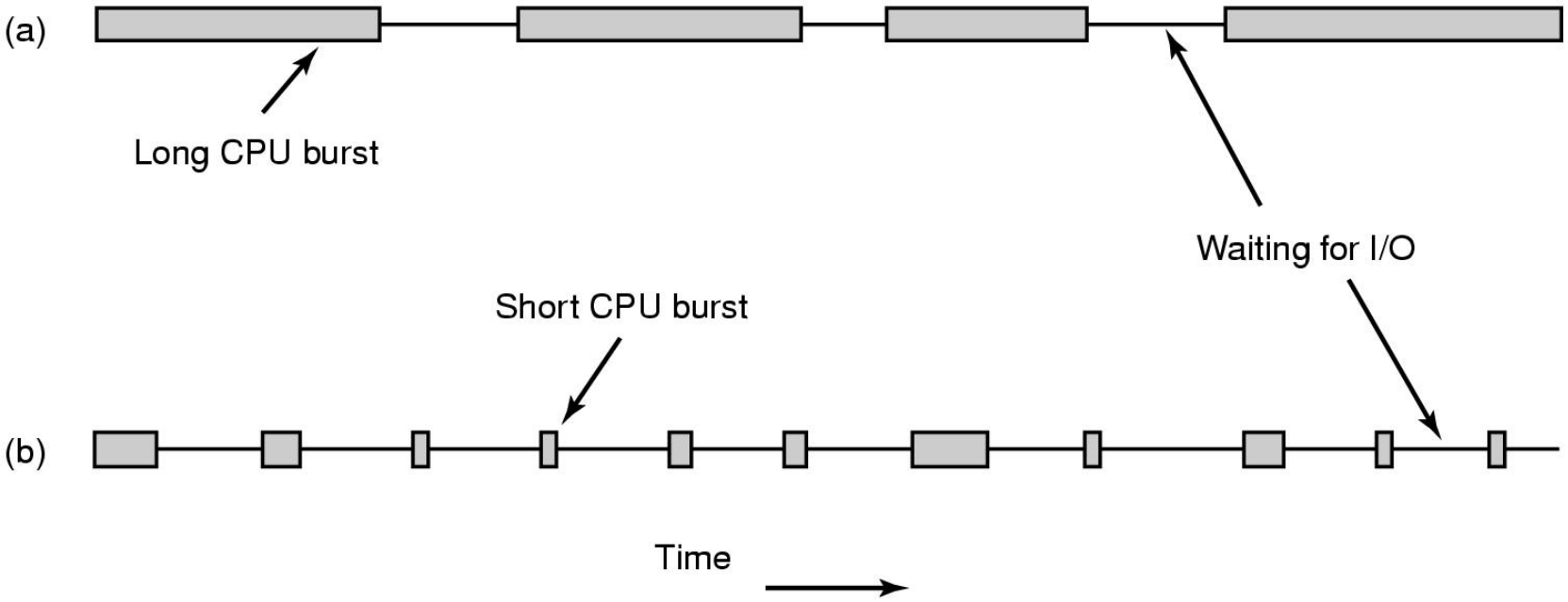


Tanenbaum: Figura 2.3

A função do escalonador é escolher qual deve ser o próximo processo a ser executado.

Aproveitamento da CPU

IO-bound



Tanenbaum: Figura 1.2-37

Processo CPU-bound

```
for (i = 0; i < 1000; i++)
    x = (x + i) * y;
```

Processo IO-bound

```
for (i = 0; i < 1000; i++)
  fprintf(arquivo, "i = %d", i);
```

Processo ?-bound

```
printf ("CPU-bound (c) ou IO-bound (i)? ");
scanf ("%c", resp);
if (resp == 'c')
  for (i = 0; i < 1000; i++)
     x = (x + i) * y;
else
  for (i = 0; i < 1000; i++)
     fprintf(arquivo, "i = %d", i);
```

Análise em tempo de execução

- Baseada no histórico do processo
- Processos IO-bound podem receber prioridade mais alta
- Nem sempre funciona...

CTSS

- É mais eficiente rodar programas CPU-bound raramente por períodos longos do que frequentemente por períodos curtos
- Como determinar a classe de um processo?

Classe 0 (1 quantum)	$\longrightarrow$	P1	P2	P5	P7
Classe 1 (2 quanta)	$\longrightarrow$	P0	P3		
Classe 2 (4 quanta)	$\longrightarrow$	P4			
Classe 3 (8 quanta)	$\longrightarrow$	P6			

Escalonamento em sistemas batch

Shortest Job First

Tanenbaum: Figura 2.39

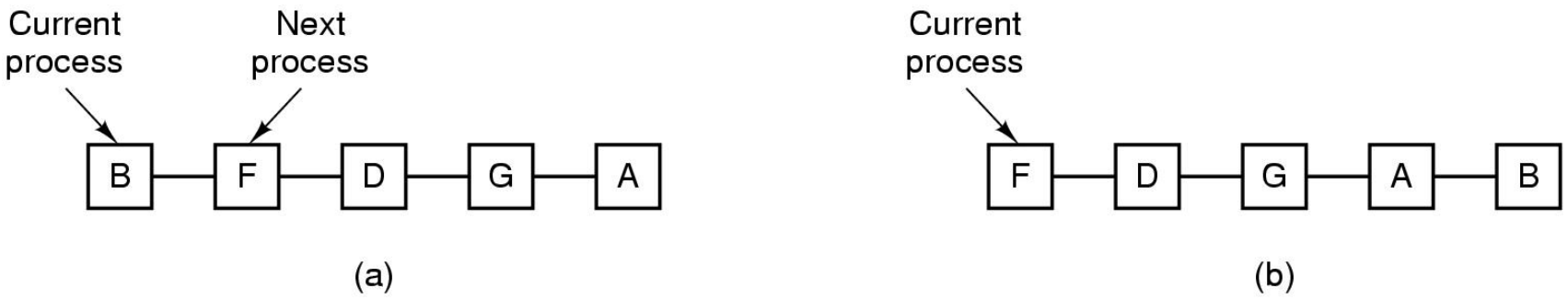
- Vazão (throughput) excelente
- Turnaround time
  - (a) $(8 + 12 + 16 + 20)/4 = 14$
  - (b) $(4 + 8 + 12 + 20)/4 = 11$

Escalonamento em sistemas batch

Shortest Job First

- Todos jobs precisam ser conhecidos previamente
  - Processos no tempo 0: 8 10
  - Processos no tempo 3: 4 4 8 10
- Se jobs curtos chegarem continuamente, os jobs longos nunca serão escalonados
  - Processos no tempo 100: 4 4 4 4 4 4 4 4 4 8 10

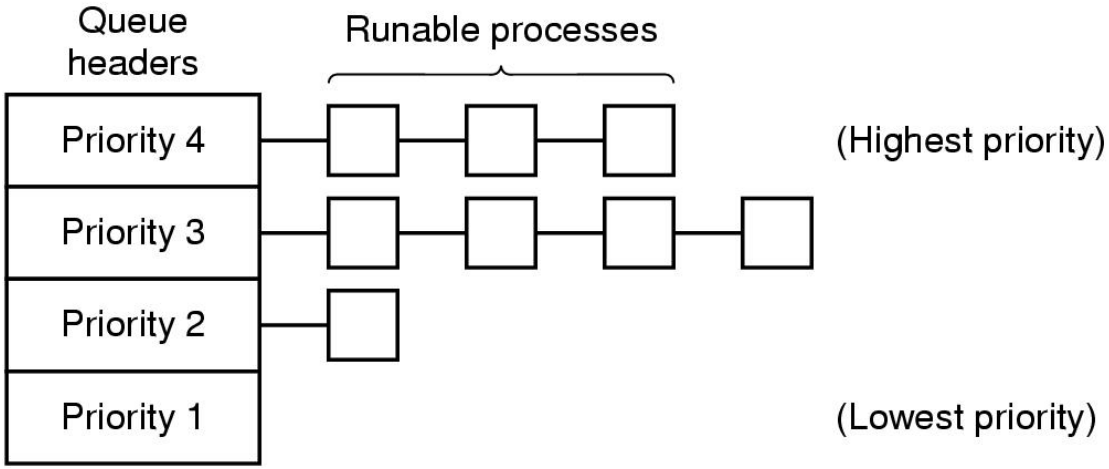
Round-Robin



Tanenbaum: Figura 2.41

- Quantum
- Análises de custo-benefício

Problema da prioridade invertida



Tanenbaum: Figura 2.42

- Processo L está na região crítica
- Processo H executa código de espera ocupada
- Processo L nunca é escalonado

Problema da prioridade invertida

Podemos descartar espera ocupada?

- Os programas de usuário podem utilizar
  - semáforos
  - locks e variáveis de condição
- Mas como estas primitivas são implementadas pelo SO?
  - interrupções podem ser desabilitadas
  - spin locks

Spin lock

```
entra_RC:
    TSL RX, lock
    CMP RX, \#0
    JNE entra_RC
    RET

deixa_RC:
    MOV lock, \#0
    RET
```

Gerenciamento de Memória

Idealmente, a memória deveria ser

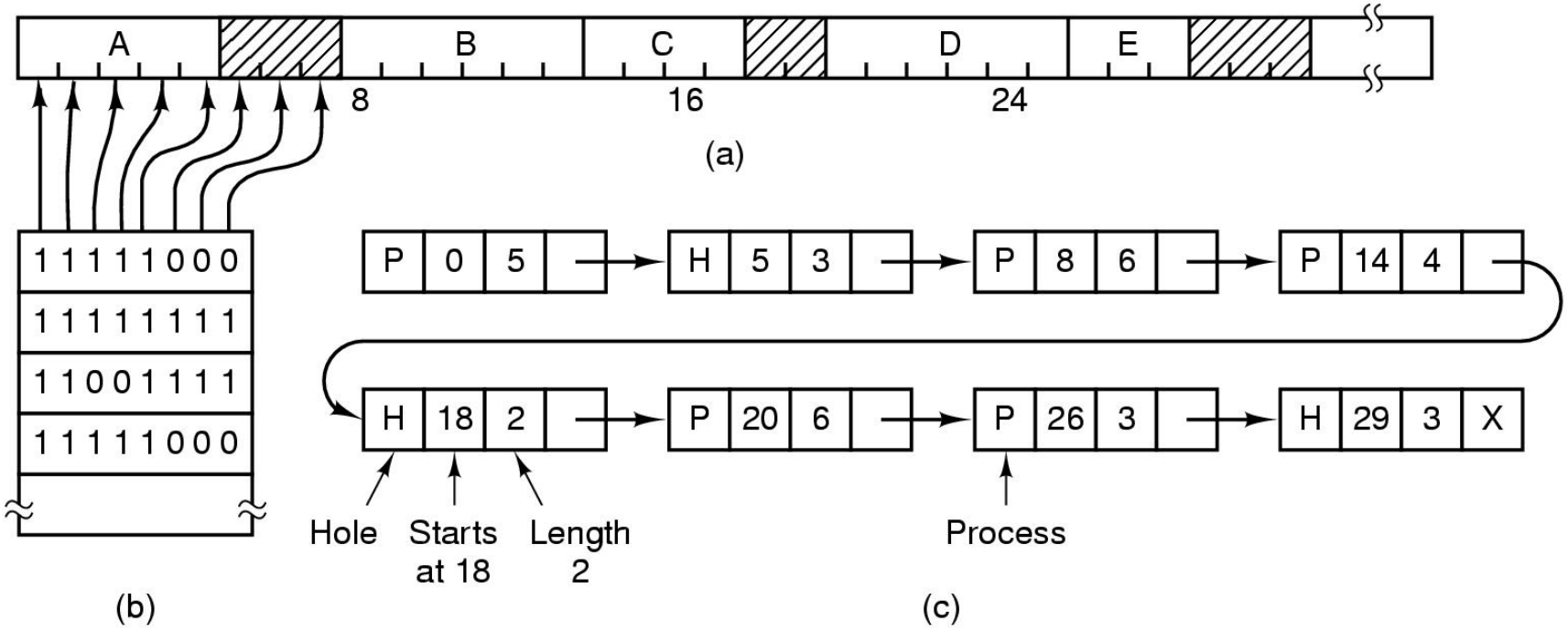
- rápida,
- de custo baixo,
- imensa e
- não volátil.

Hierarquia de memória

- pouca memória rápida e cara
- alguma memória velocidade média e preço médio
- muita memória lenta e barata

O gerenciador de memória controla a hierarquia de memória

Bitmaps e lista de livres



Tanenbaum: Figura 4.7

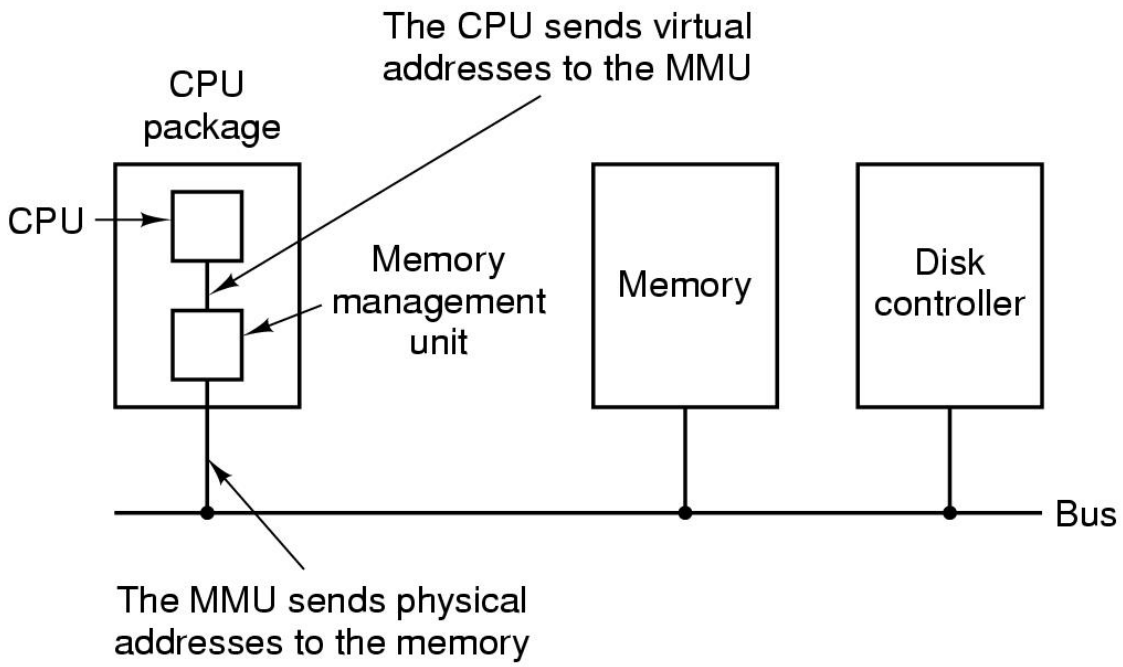
- Bitmaps: definição de unidades; busca mais lenta
- Lista de livres: atualização mais lenta

Algoritmos para alocação de memória

- *First fit*
- *Next fit*
- *Best fit*
- *Worst fit*

Podem ser usados com bitmaps ou lista de livres

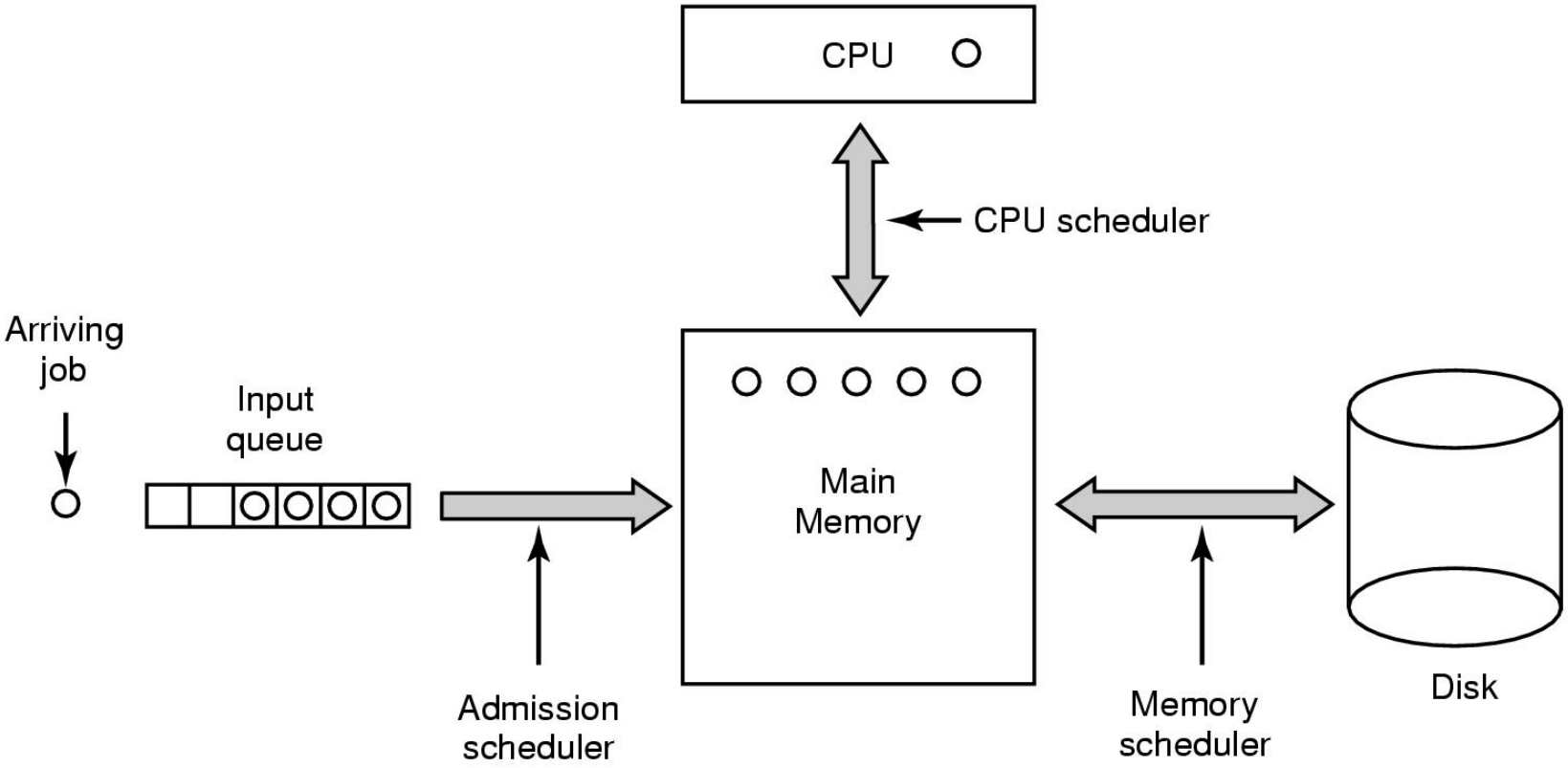
Endereços físicos e virtuais



Tanenbaum: Figura 4.9

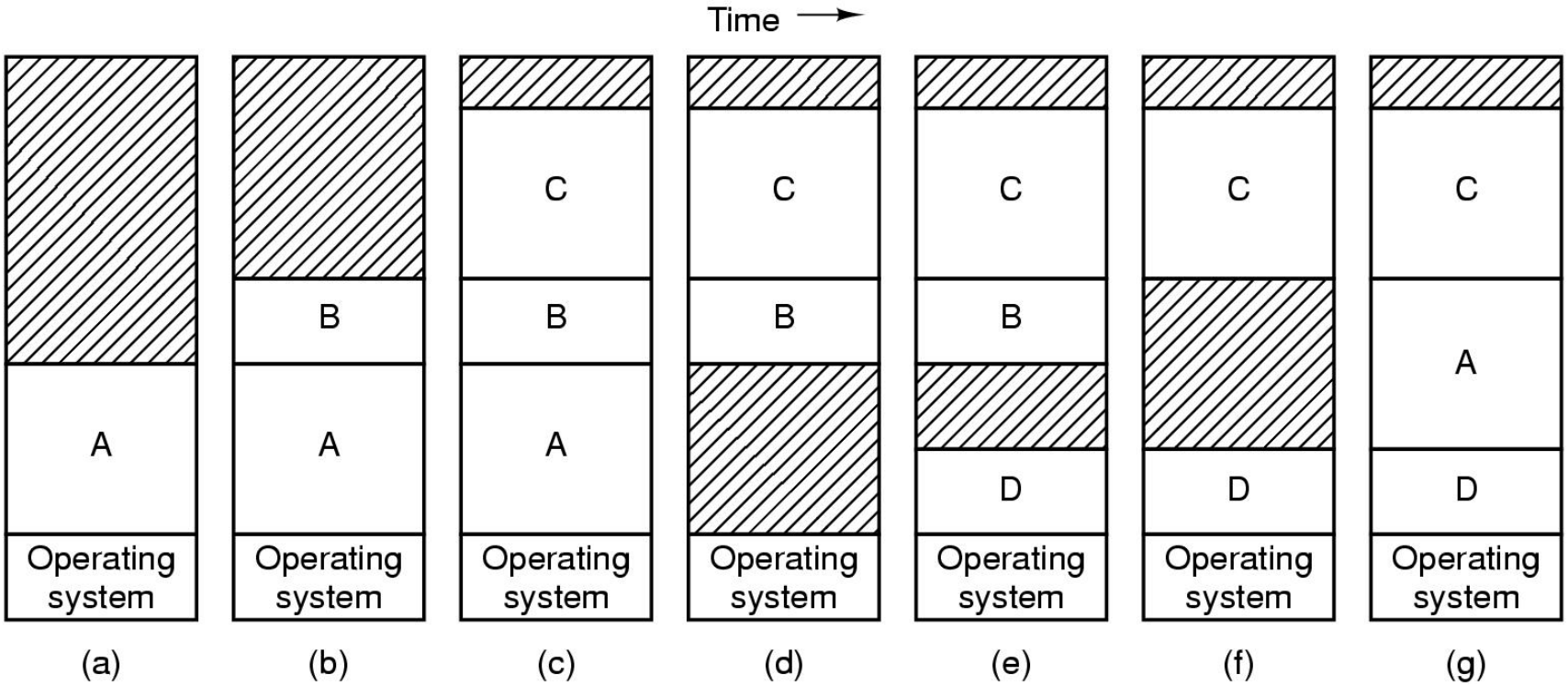
- Endereço físico: colocado diretamente no barramento
- Endereço virtual (lógico): necessita de algum mapeamento para ser colocado no barramento

Swapping



Tanenbaum: Figura 2.40

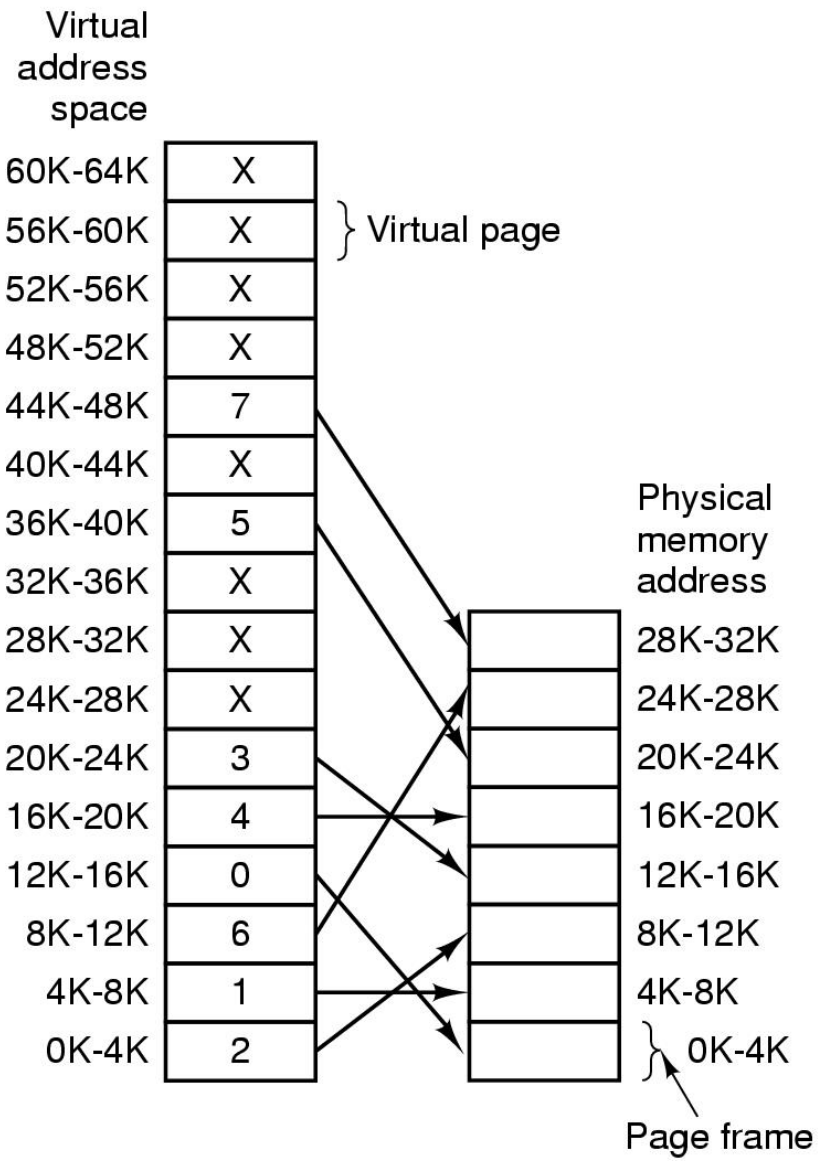
Swapping



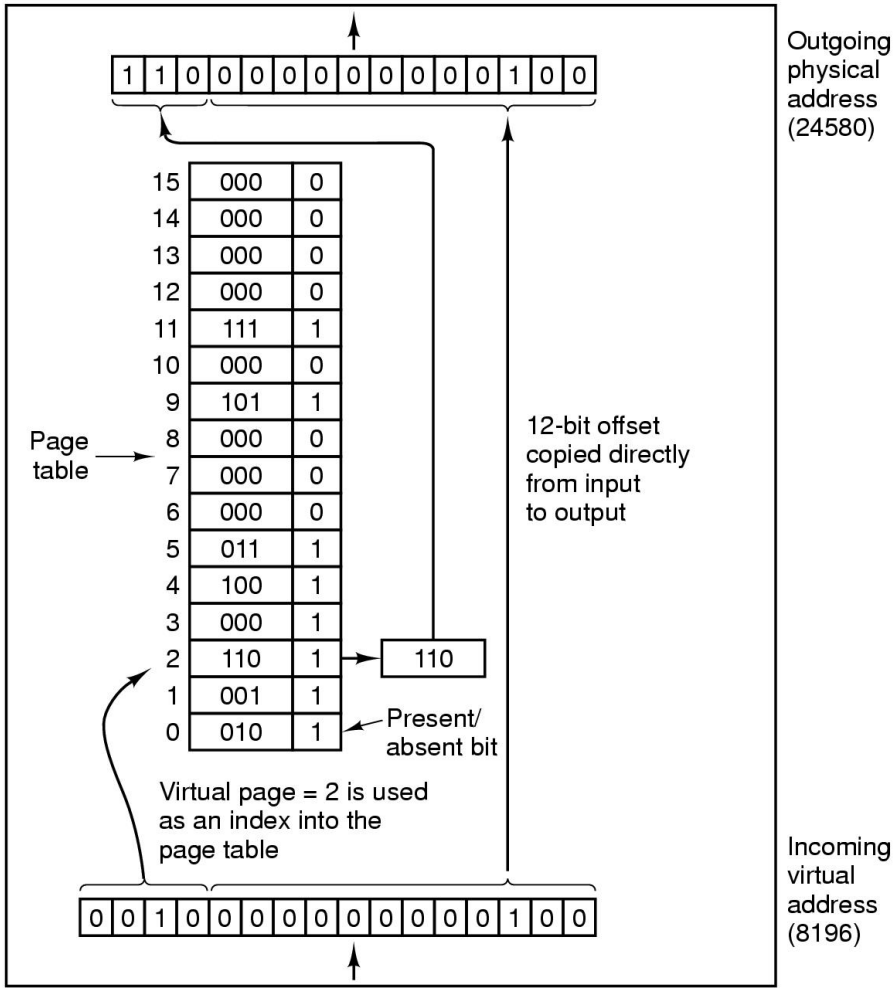
Tanenbaum: Figura 4.5

Endereços não podem ser físicos!

Paginação


Virtual address space
60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

Mapeamento dos endereços



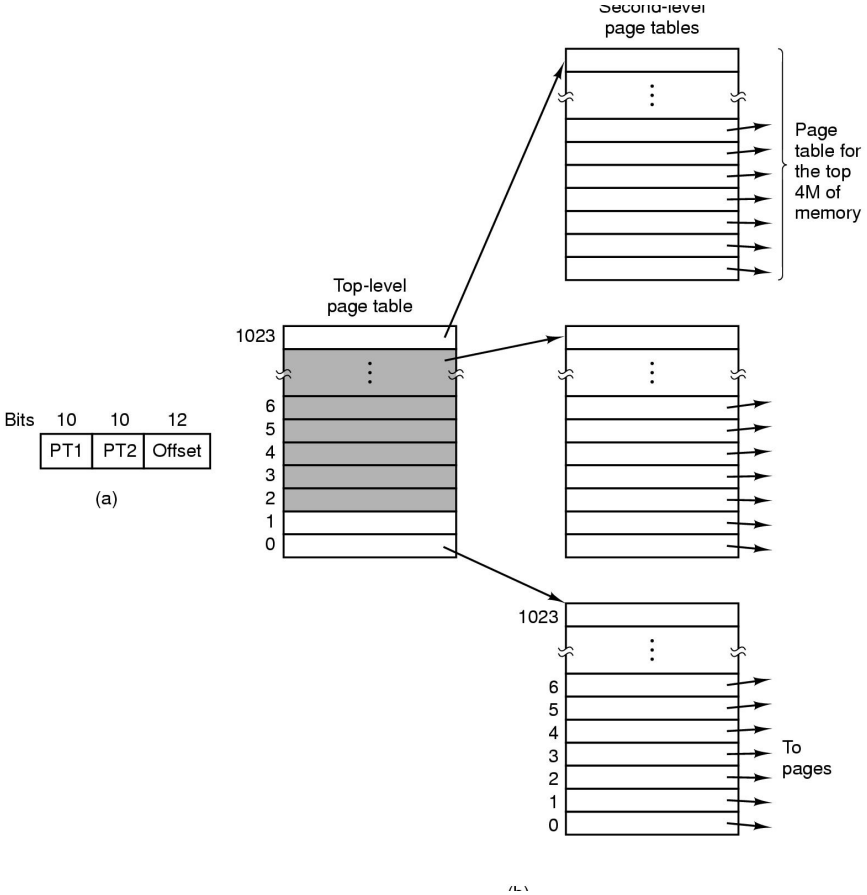
Tanenbaum: Figura 4.11

Translation Look Aside Buffers (TLBs)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

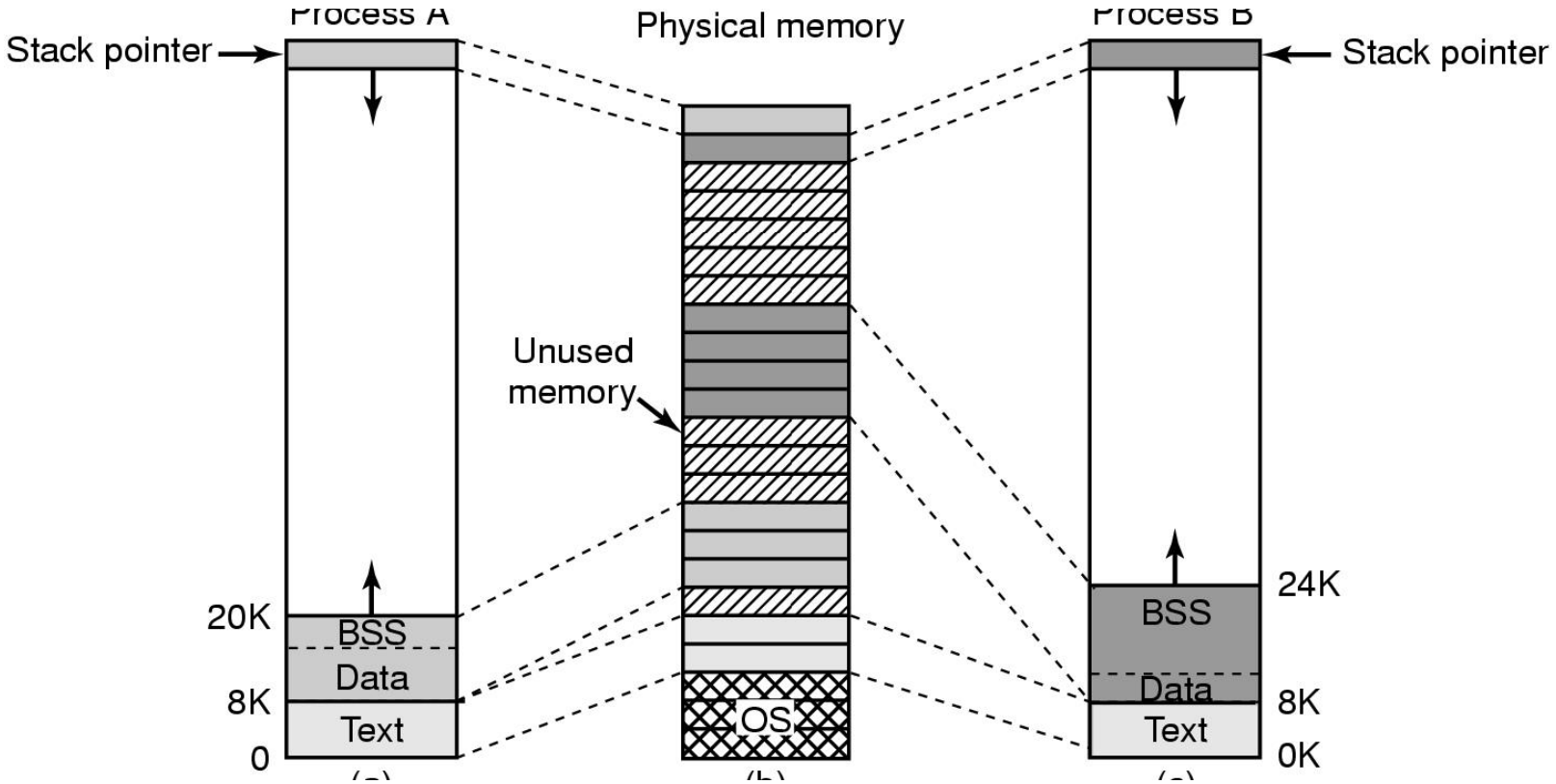
Tanenbaum: Figura 4.14

Paginação em vários níveis



Tanenbaum: Figura 4.12

Memória compartilhada



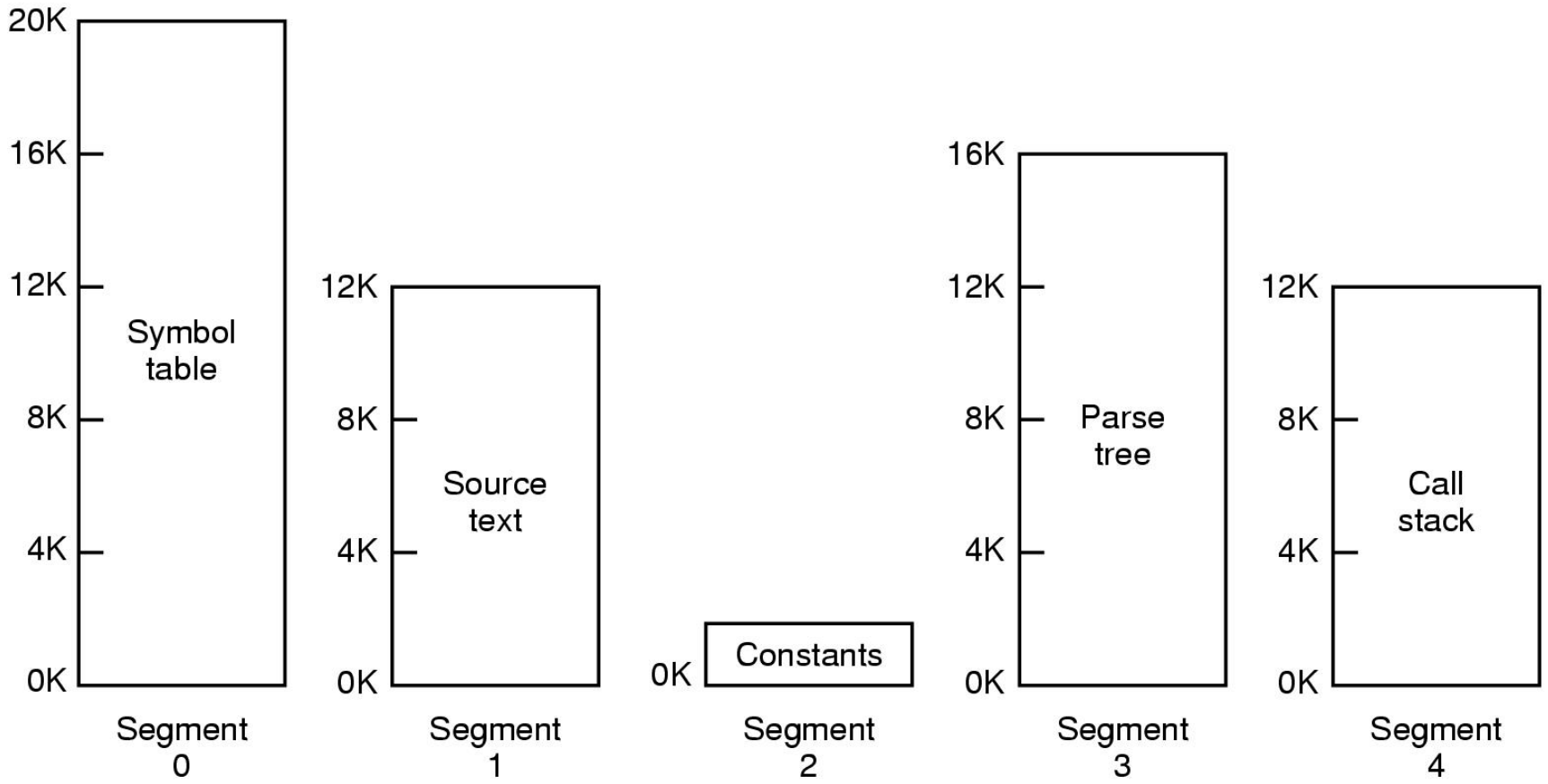
Tanenbaum: Figura 10.13

Políticas para substituição de páginas

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Tanenbaum: Figura 4.23

Segmentação



Tanenbaum: Figura 4.36

Paginação e segmentação

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Tanenbaum: Figura 4.37

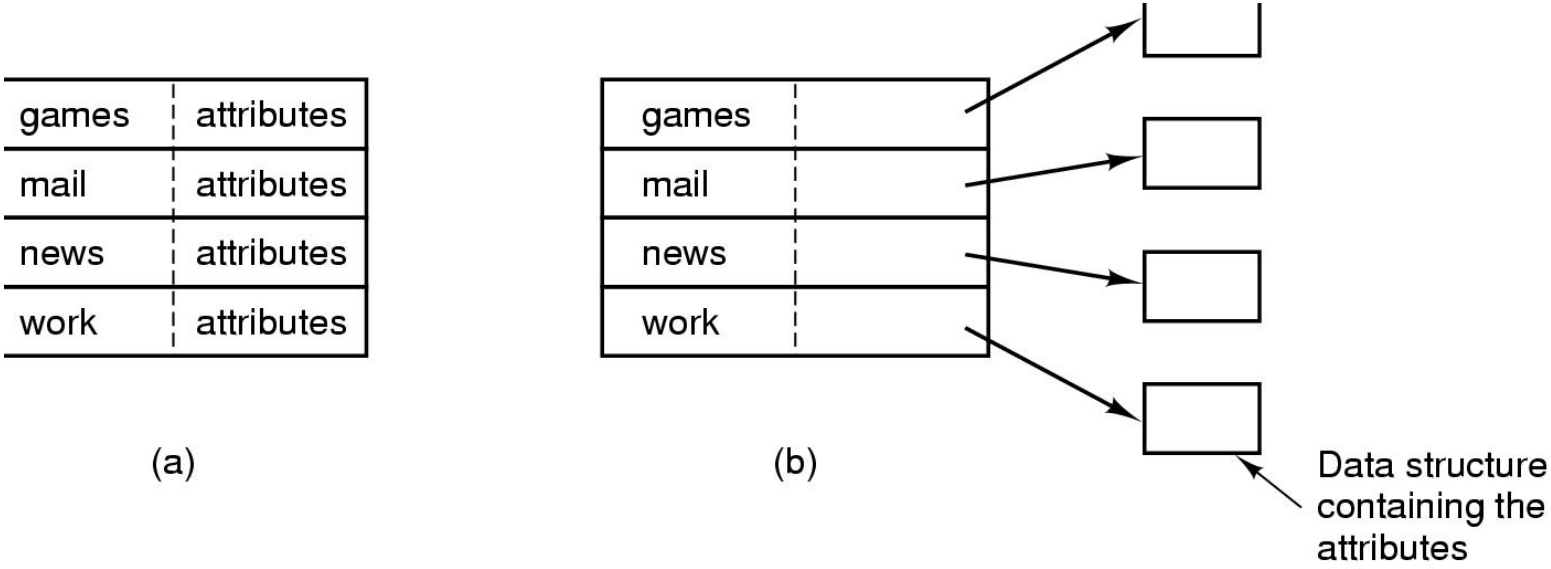
Buffer Overflow

- Ataque que insere mais dados que o espaço previamente alocado. Estes dados extras são na verdade trechos de código que serão executados pelo programa vítima.
- Defesas:
  - Proteção das páginas
  - Stack Canary

Sistemas de Arquivos

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

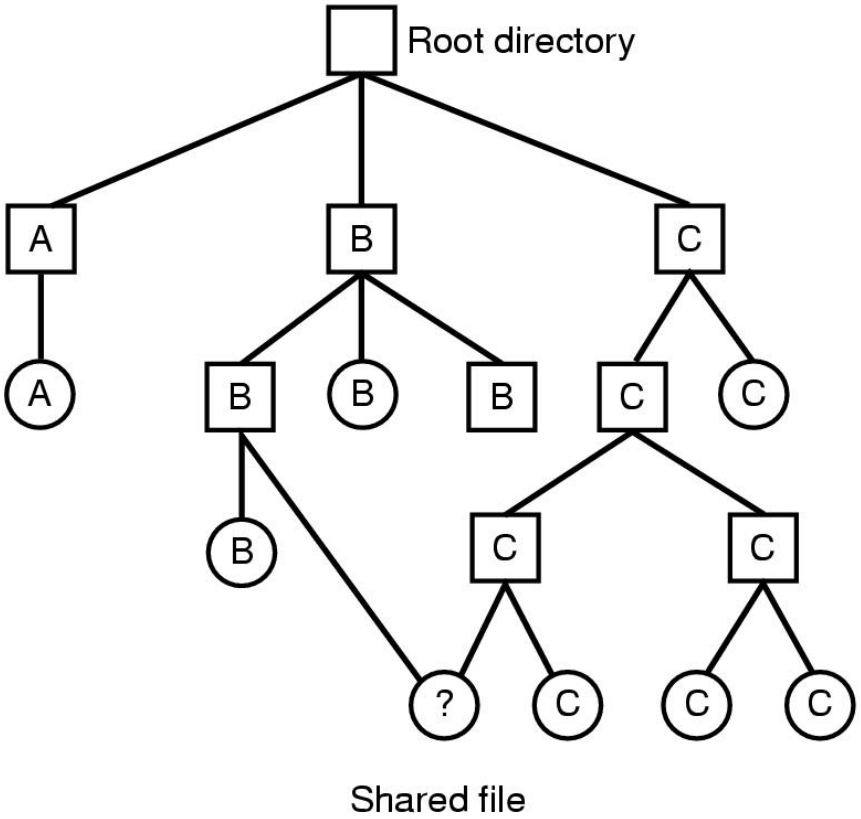
Atributos de arquivos



Tanenbaum: Figura 6.16

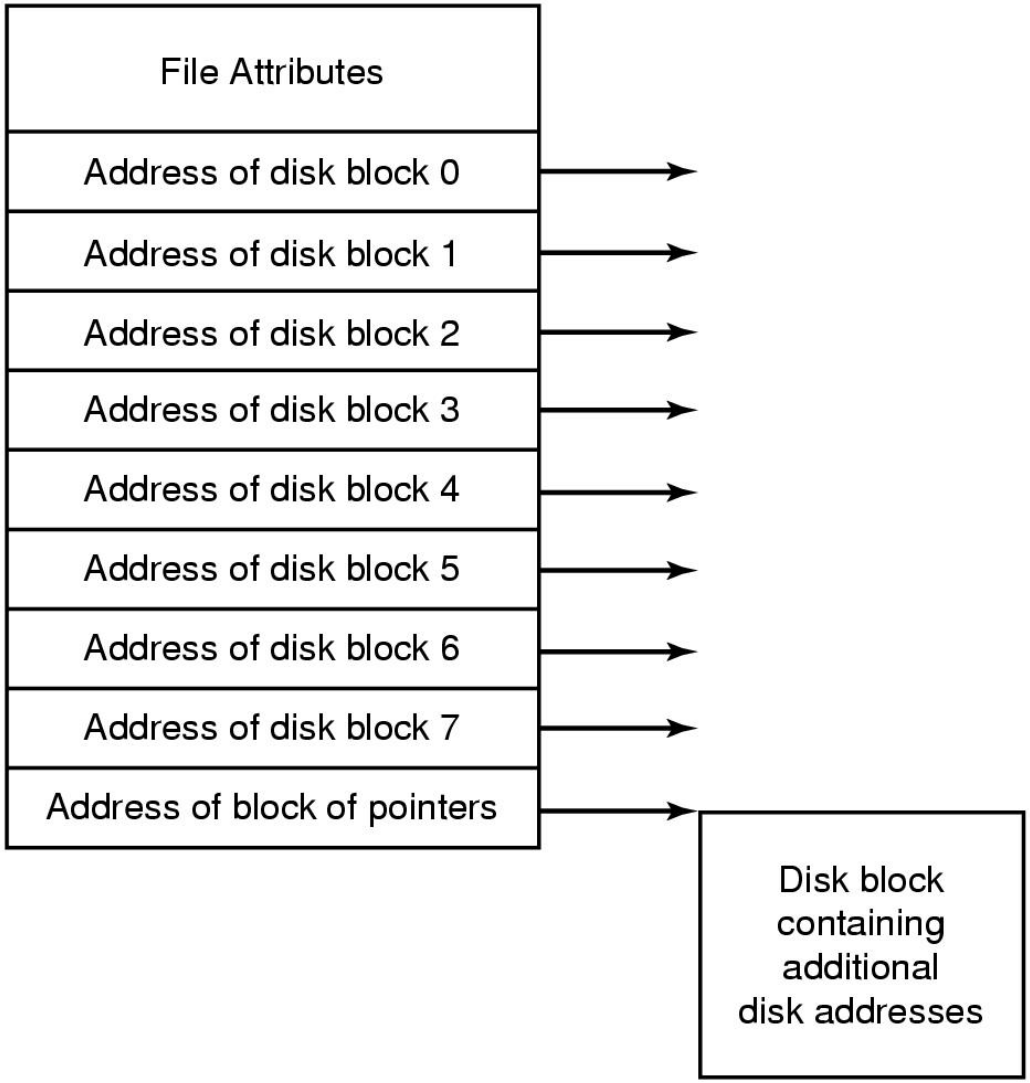
- Dificuldade de gerência de entradas grandes
- Dificuldade para o compartilhamento de arquivos

Arquivos compartilhados



Tanenbaum: Figura 6.18

I-node

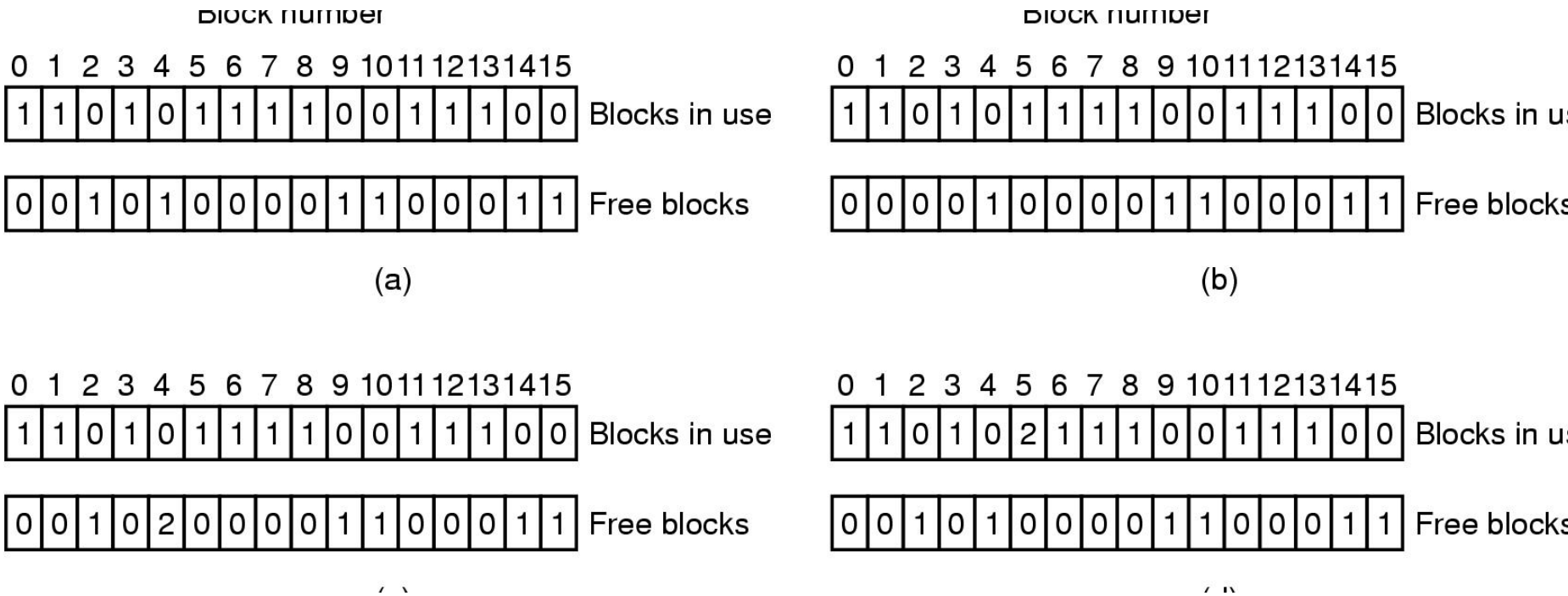


Tanenbaum: Figura 6.15

Arquivos compartilhados

- Hard links: referências para os i-nodes
- Symbolic links: apontadores (atalhos) para a outra entrada no outro diretório
  - Podem ser implementados sem a presença de i-nodes

Consistência do sistema de arquivos



Tanenbaum: Figura 6.26

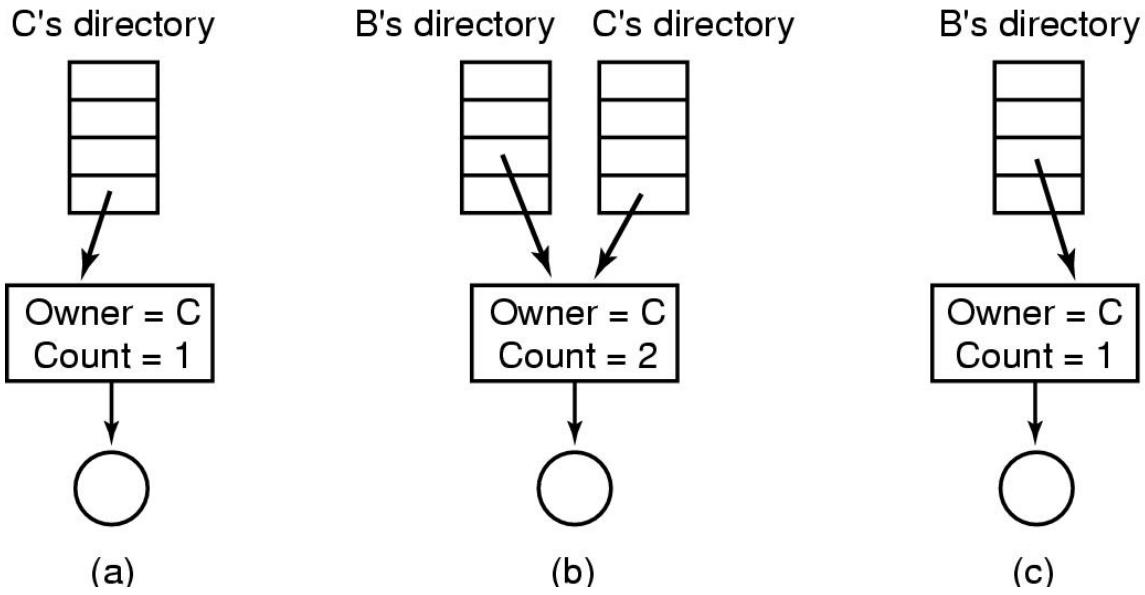
(a) consistente

(b) bloco faltando

(c) duplicação na lista de livres

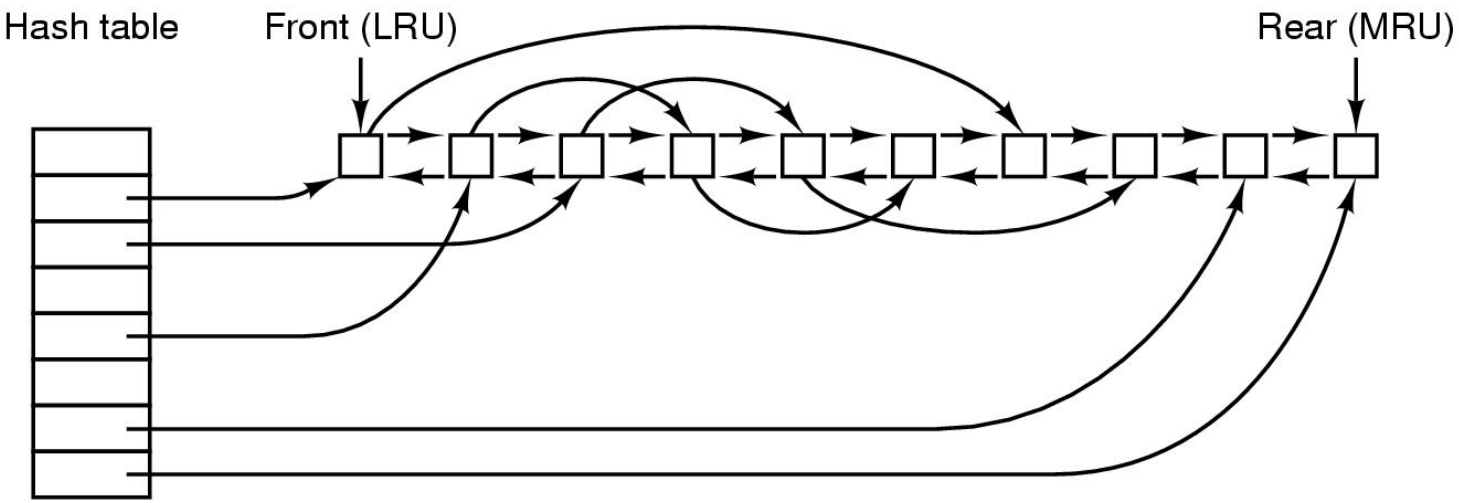
(d) duplicação nos dados

Consistência do sistema de arquivos



Tanenbaum: Figura 6.19

Caching



Tanenbaum: Figura 6.27

- Estrutura deve permitir busca rápida
- Blocos de controle devem ser tratados diferentemente

Pipes

```
$ grep xxx log.txt > log-xxx.txt
$ wc -l log-xxx.txt
$ rm log-xxx.txt

$ grep xxx log.txt | wc -l
```