

**MC514**  
**Sistemas Operacionais:**  
**Teoria e Prática**  
1s2006

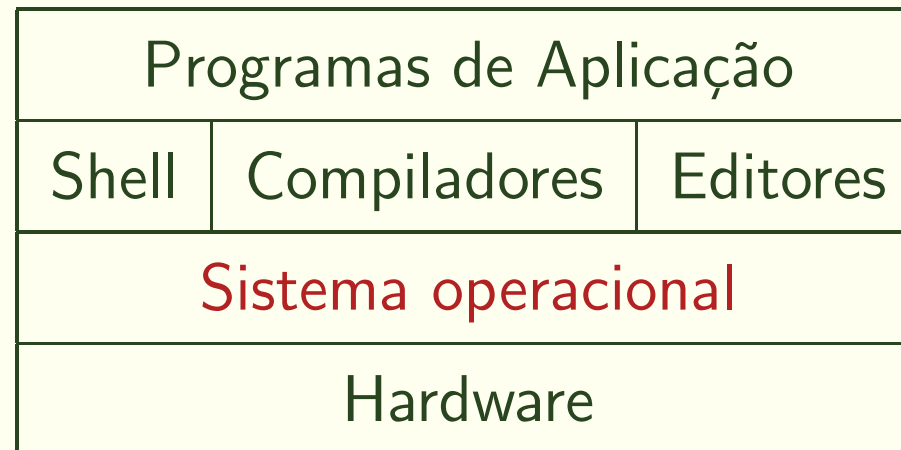
**Lista de exercícios II**

## Lista de exercícios II

- Questões básicas
- Processos e threads
- Gerência de memória
- Sistema de arquivos
- Entrada e saída

# Questões gerais

# Responsabilidades de um SO



- 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

# Modo usuário e modo kernel

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

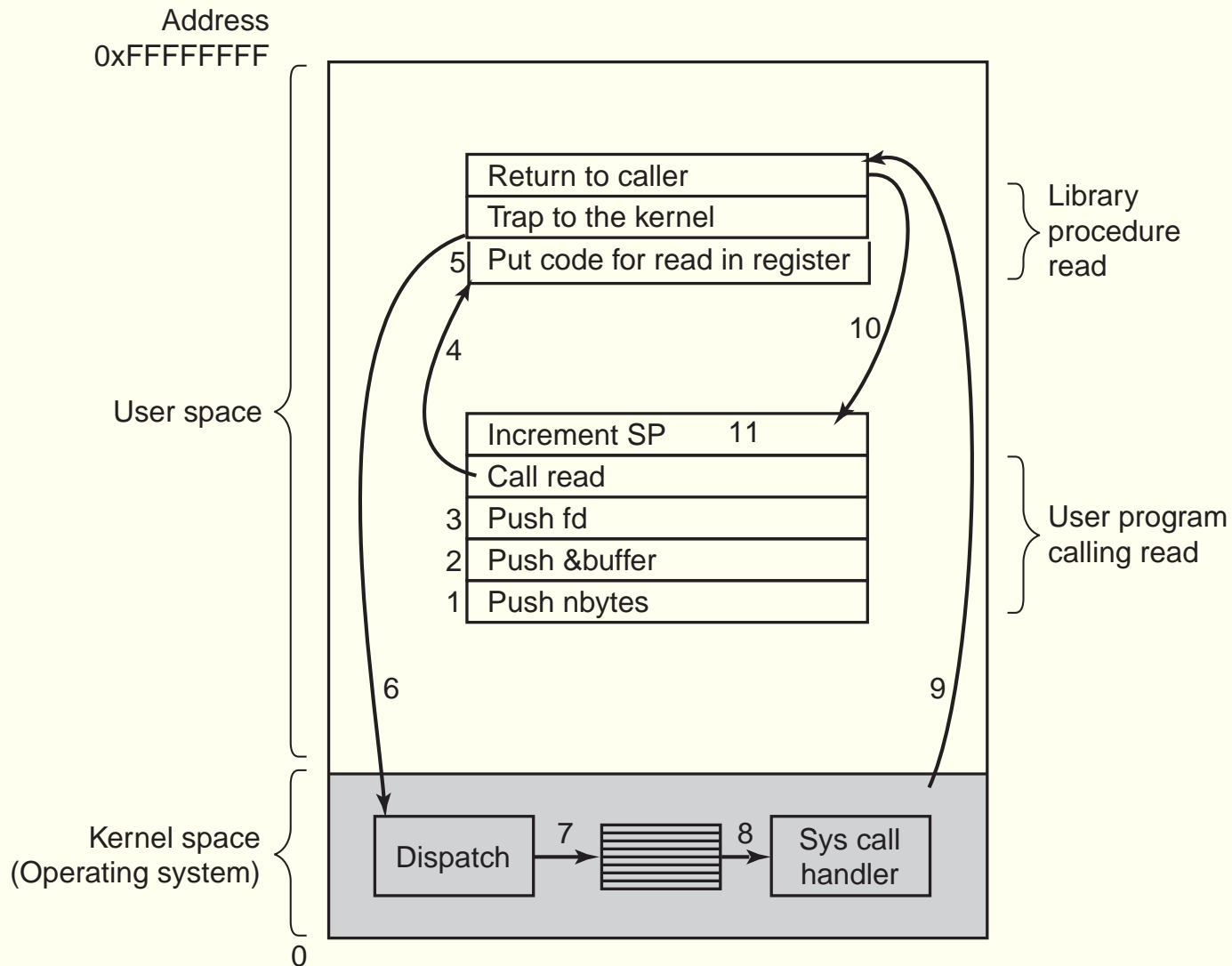
Esta distinção permite proteger os programas e o próprio SO de usos indevidos por parte de outros programas.

Um programa do usuário começa a sua execução em modo usuário, mas durante as chamadas ao sistema estará executando em modo kernel.

## Chamada ao sistema

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

# cont = read(fd, buffer, nbytes)





# Exemplos de chamadas ao sistema

## Gerência de processos

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

# Exemplos de chamadas ao 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 ao sistema

## Gerência de arquivos e diretórios

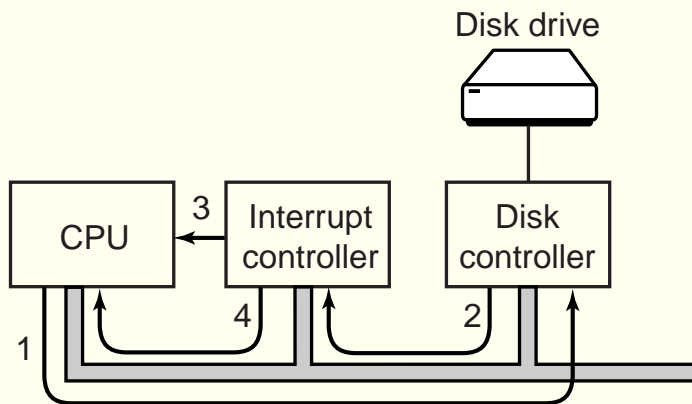
- `s = mkdir(name,mode);`
- `s = rmdir(name);`
- `s = link(name1,name2);`
- `s = unlink(name);`
- `s = mount(special,name,flag);`
- `s = umount(special);`

# Exemplos de chamadas ao sistema

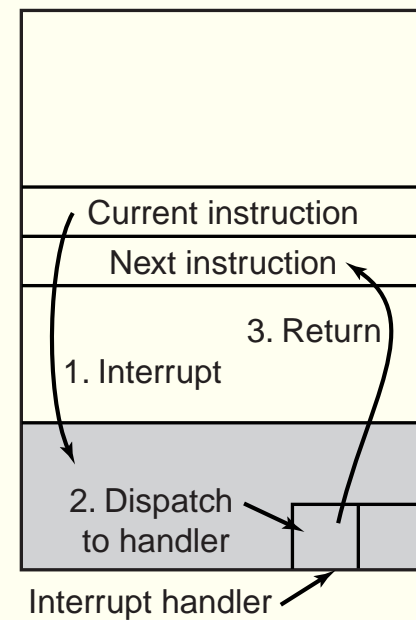
## Diversas

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

# Tratamento de interrupções



(a)



(b)

# 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

## **Sistemas monolíticos**

- Sem estrutura
- Conjunto de funções dependentes

## **Sistemas baseados em camadas**

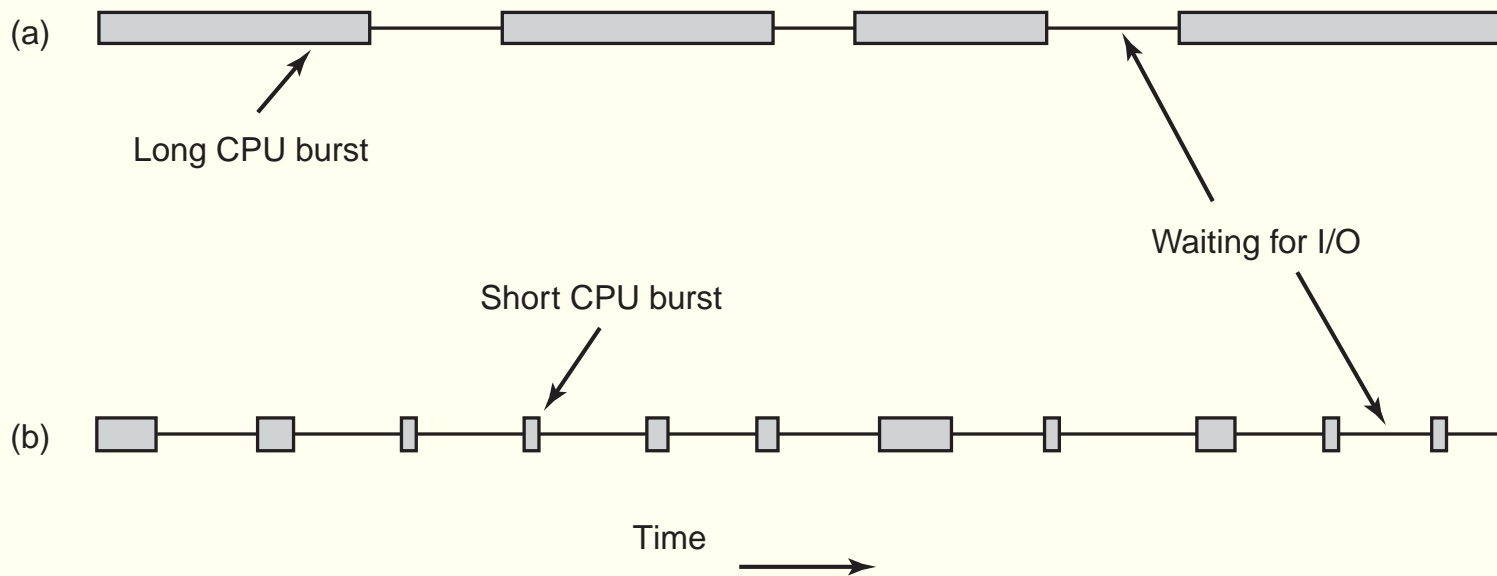
- Bem estruturado
- Nível inferior não pode invocar funções do nível superior

# Processos e threads



# Aproveitamento da CPU

## CPU-bound



## IO-bound

## 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...

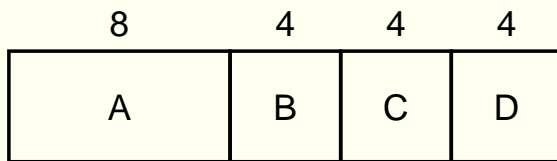
# CTSS

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

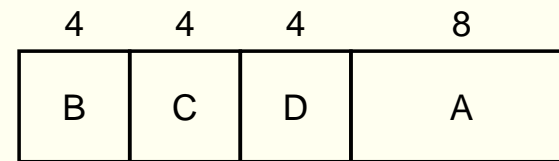
Classe 0 (1 quantum)	→	P1	P2	P5	P7
Classe 1 (2 quanta)	→	P0	P3		
Classe 2 (4 quanta)	→	P4			
Classe 3 (8 quanta)	→	P6			

# Escalonamento em sistemas batch

## Shortest Job First



(a)



(b)

- Vazão (throughput) excelente
- Turnaround time

$$(a) (8 + 12 + 16 + 20)/4 = 14$$

$$(b) (4 + 8 + 12 + 20)/4 = 11$$

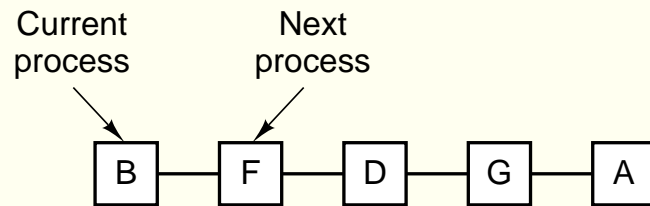
# Escalonamento em sistemas batch

## Shortest Job First

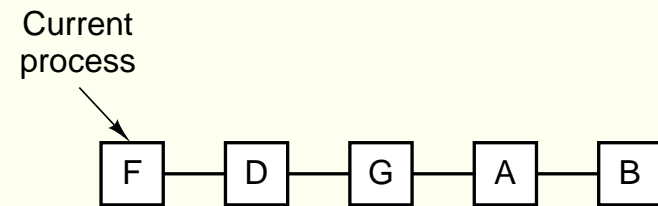
- Todos jobs precisam ser conhecidos previamente
  - Processos no tempo 0: 8 10
  - Processos no tempo 3: 4 4 8 10
- Se jobs curtos chegarem continuamente, os jobs longos nunca serão escalonados
  - Processos no tempo 100: 4 4 4 4 4 4 4 4 4 8 10



# Round-Robin



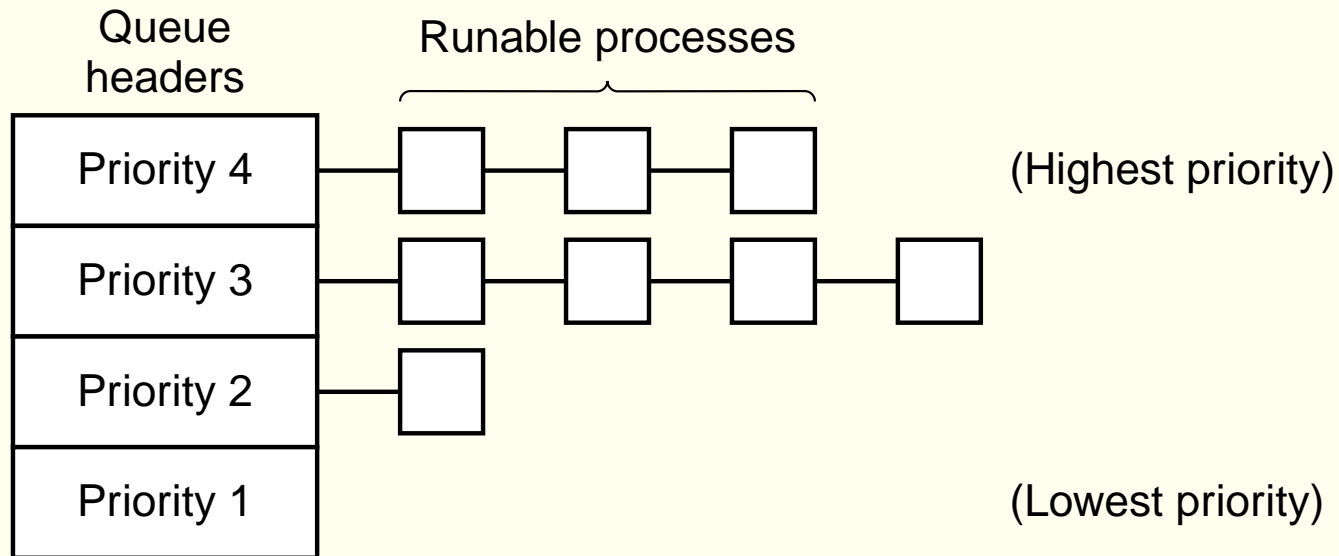
(a)



(b)

- Quantum
- Análises de custo-benefício

# Problema da prioridade invertida



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

# Podemos descartar espera ocupada?

- A nível de programas de usuário podemos usar
  - 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
```

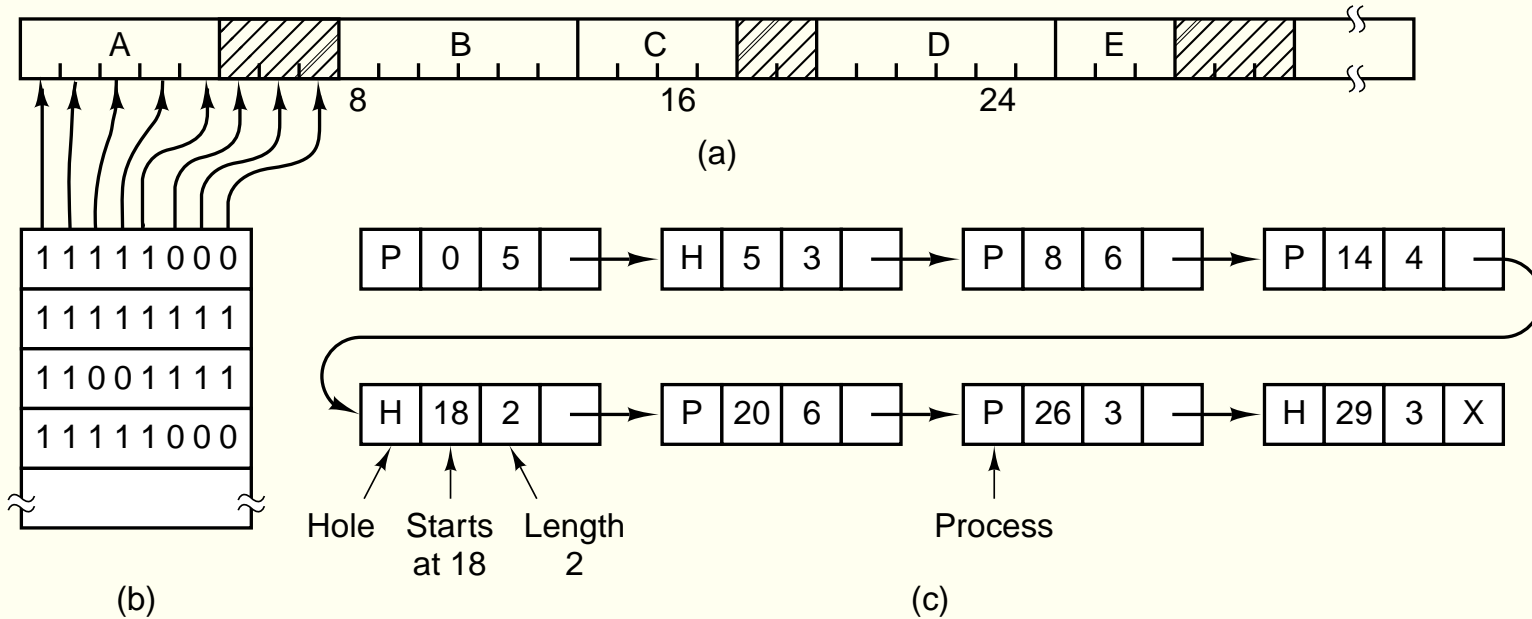
## Comando fork

```
if (pid = fork()) {  
    pai();  
    waitpid(pid, NULL, 0);  
} else  
    filho();
```

- Quando o pai morre, os filhos não morrem automaticamente
- Como ser informado da morte do filho assincronamente?

# Gerência de memória

# Bitmaps e lista de livres



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

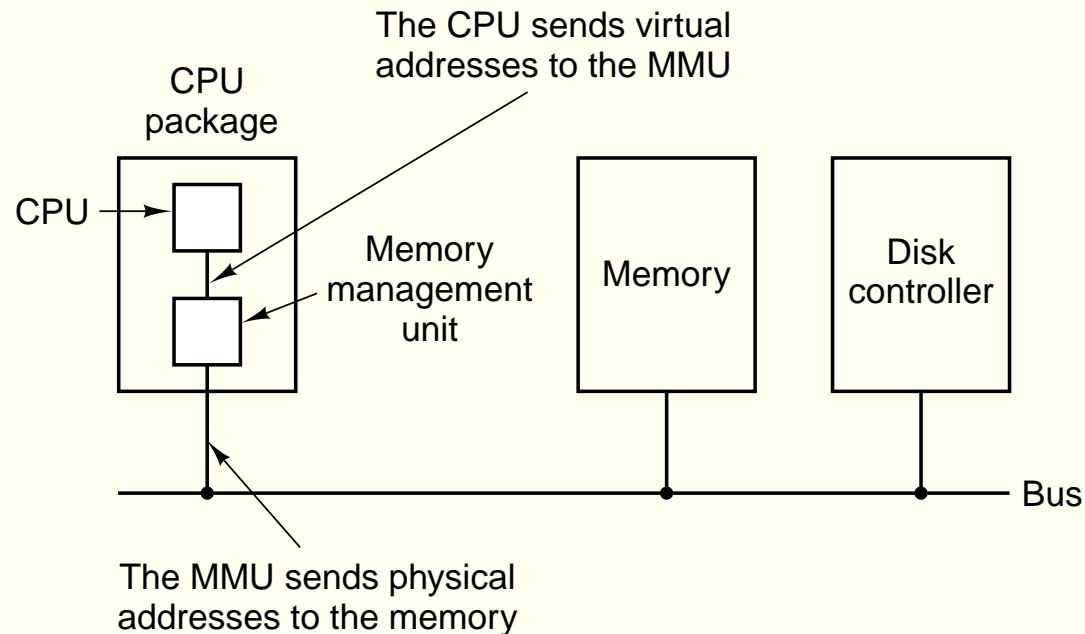
# Algoritmos para alocação de memória

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

Podem ser usados com bitmaps ou lista de livres

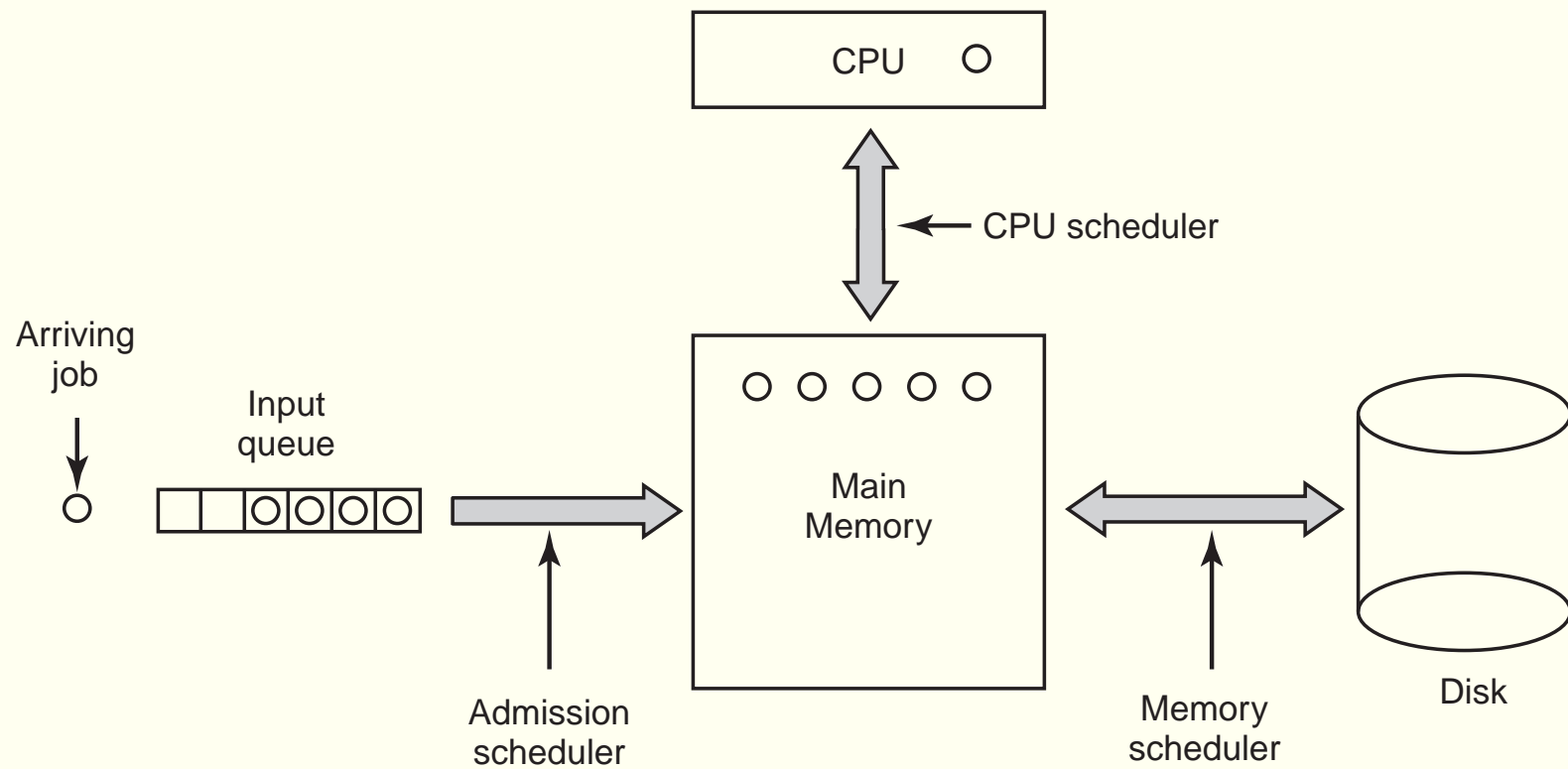


# Endereços físicos e virtuais

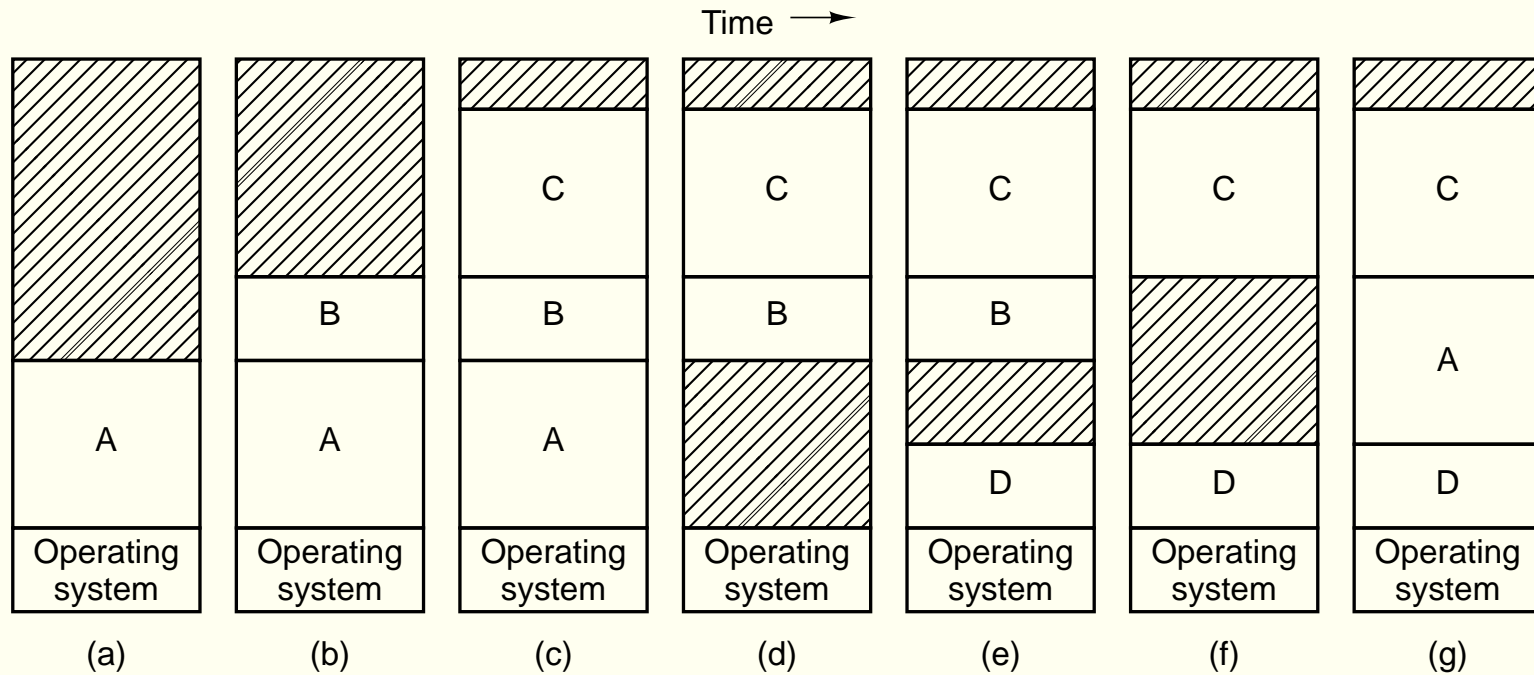


- Endereço físico: colocado diretamente no barramento
- Endereço virtual (lógico): necessita de algum mapeamento para ser colocado no barramento

# Swapping



# Swapping



Endereços não podem ser físicos!

# Overlays

```
dados d1, d2, d3, d4, d5;
```

```
f1();      g1();      h1();
```

```
f2();      g2();      h2();
```

```
f3();      g3();      h3();
```

```
main() {
```

```
    fase_1(); /* funçoes f e dados d1, d2, d3 */
```

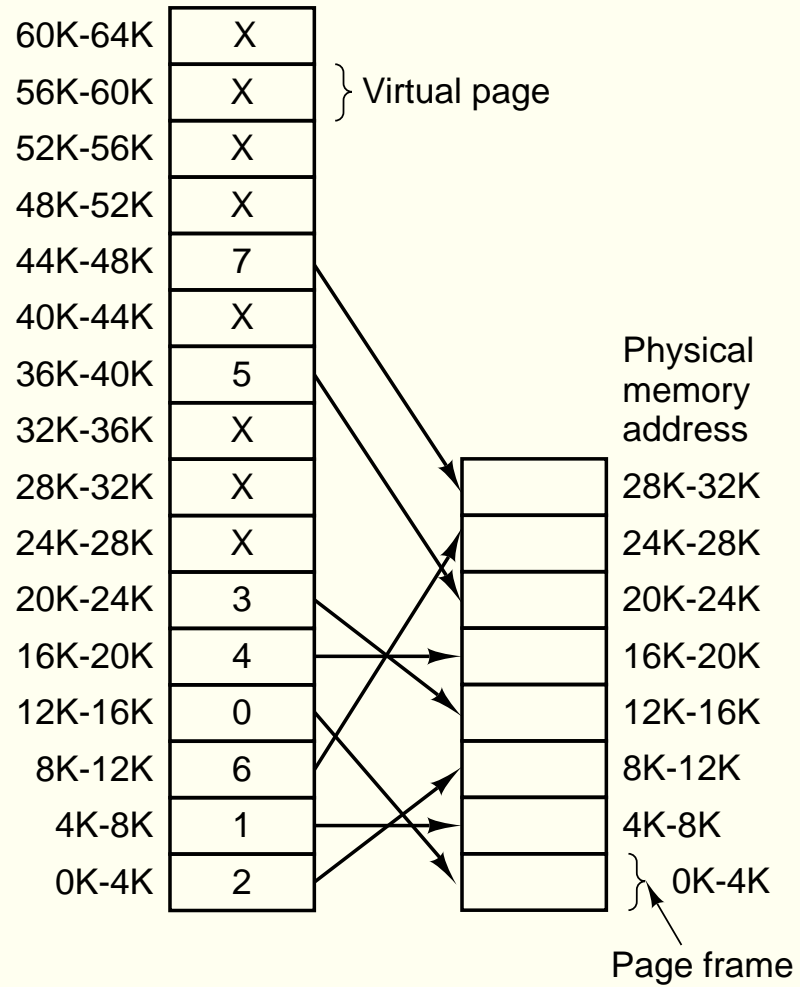
```
    fase_2(); /* funçoes g e dados d1, d2, d4 */
```

```
    fase_3(); /* funçoes h e dados d1, d2, d5 */
```

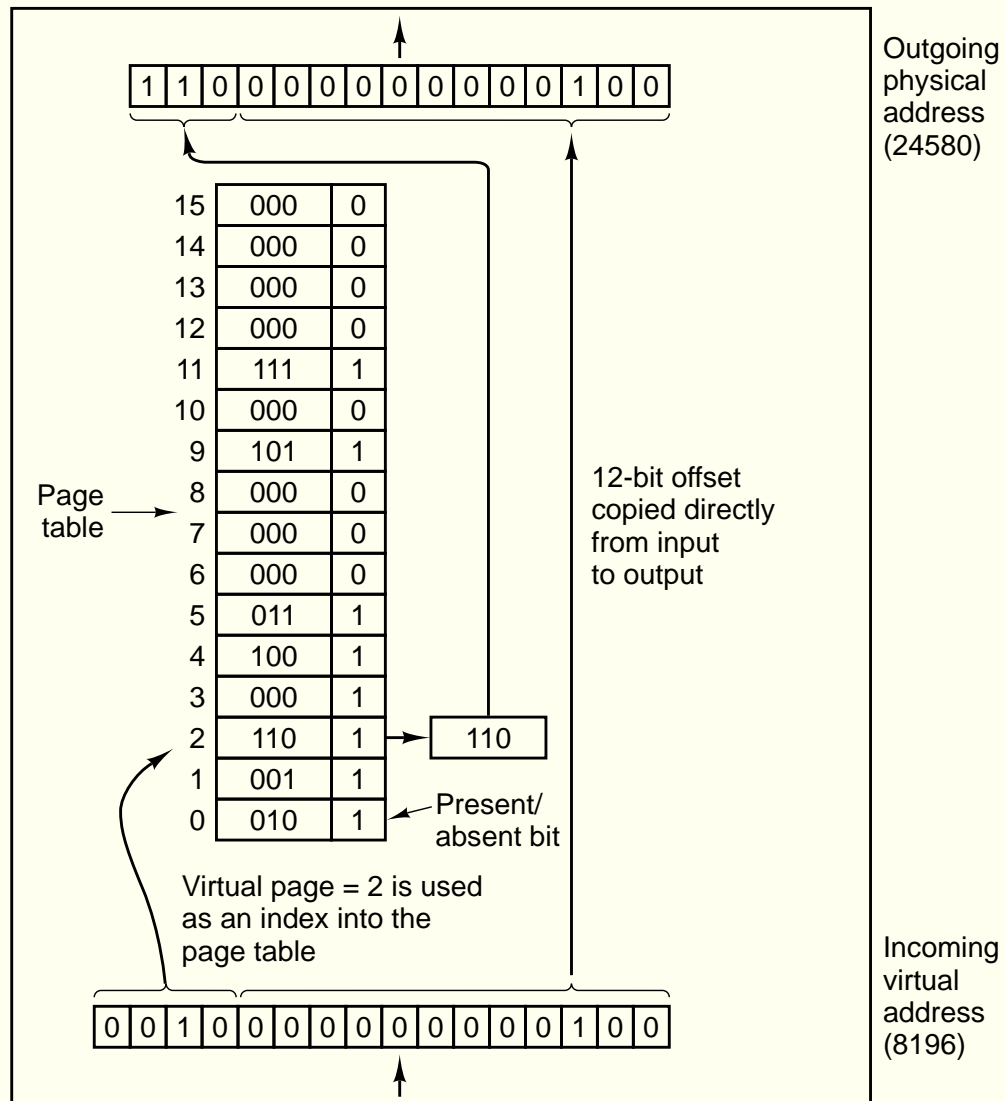
```
}
```

# Paginação

Virtual  
address  
space



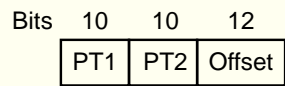
# Mapeamento dos endereços



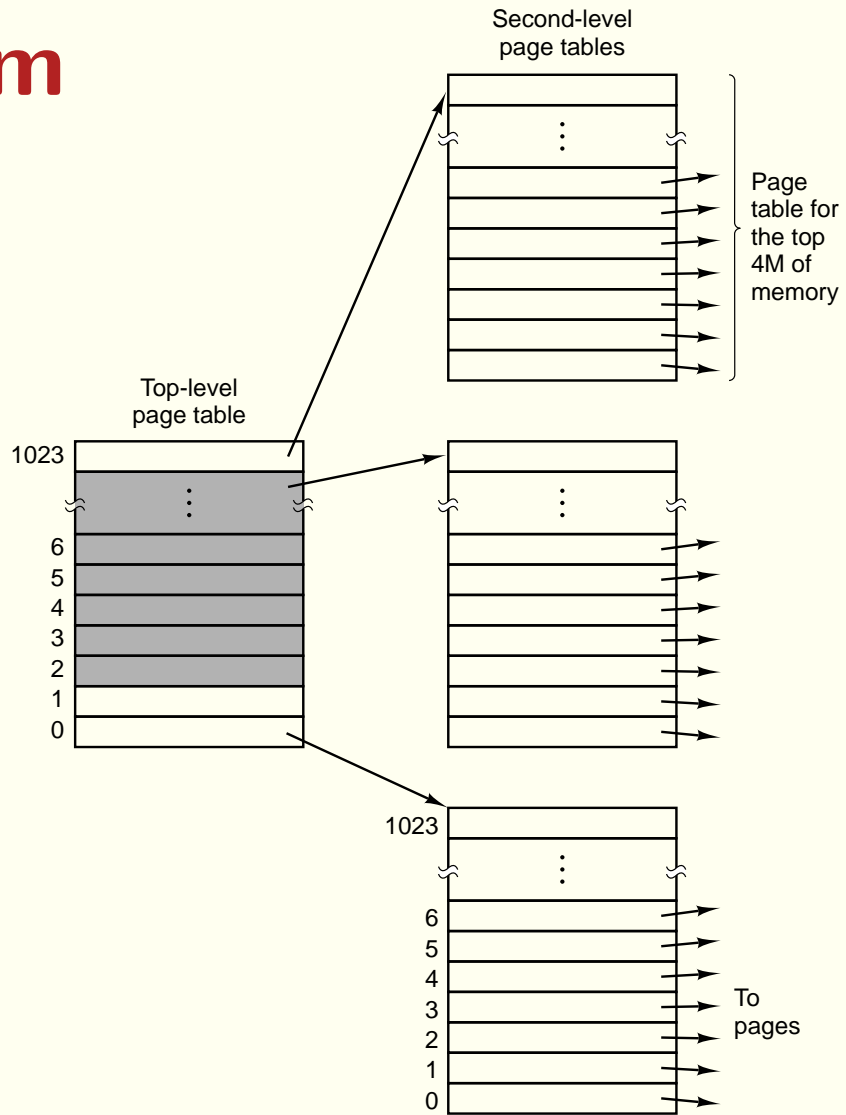
# Translation Look Aside Buffers (TLBs)

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

# Paginação em vários níveis



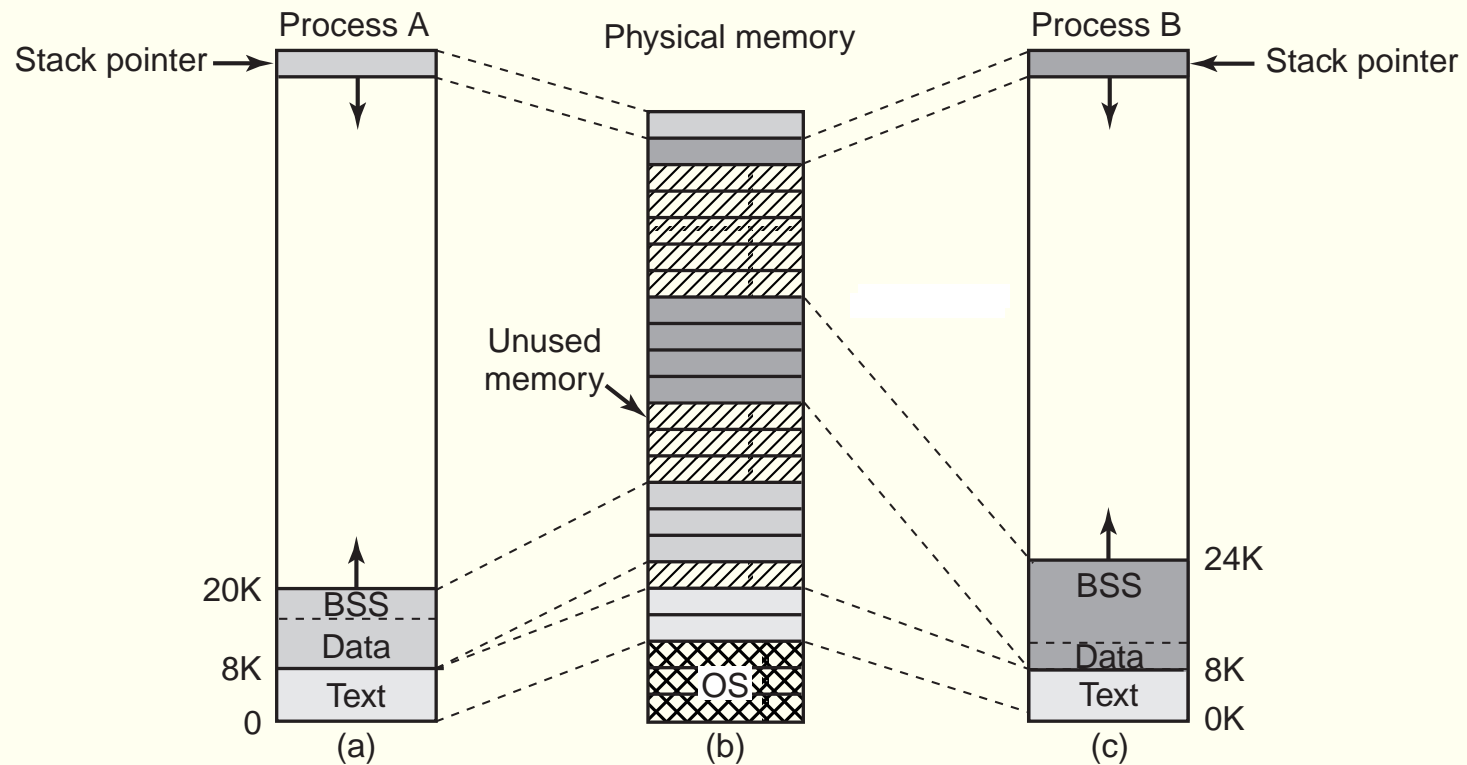
(a)



(b)



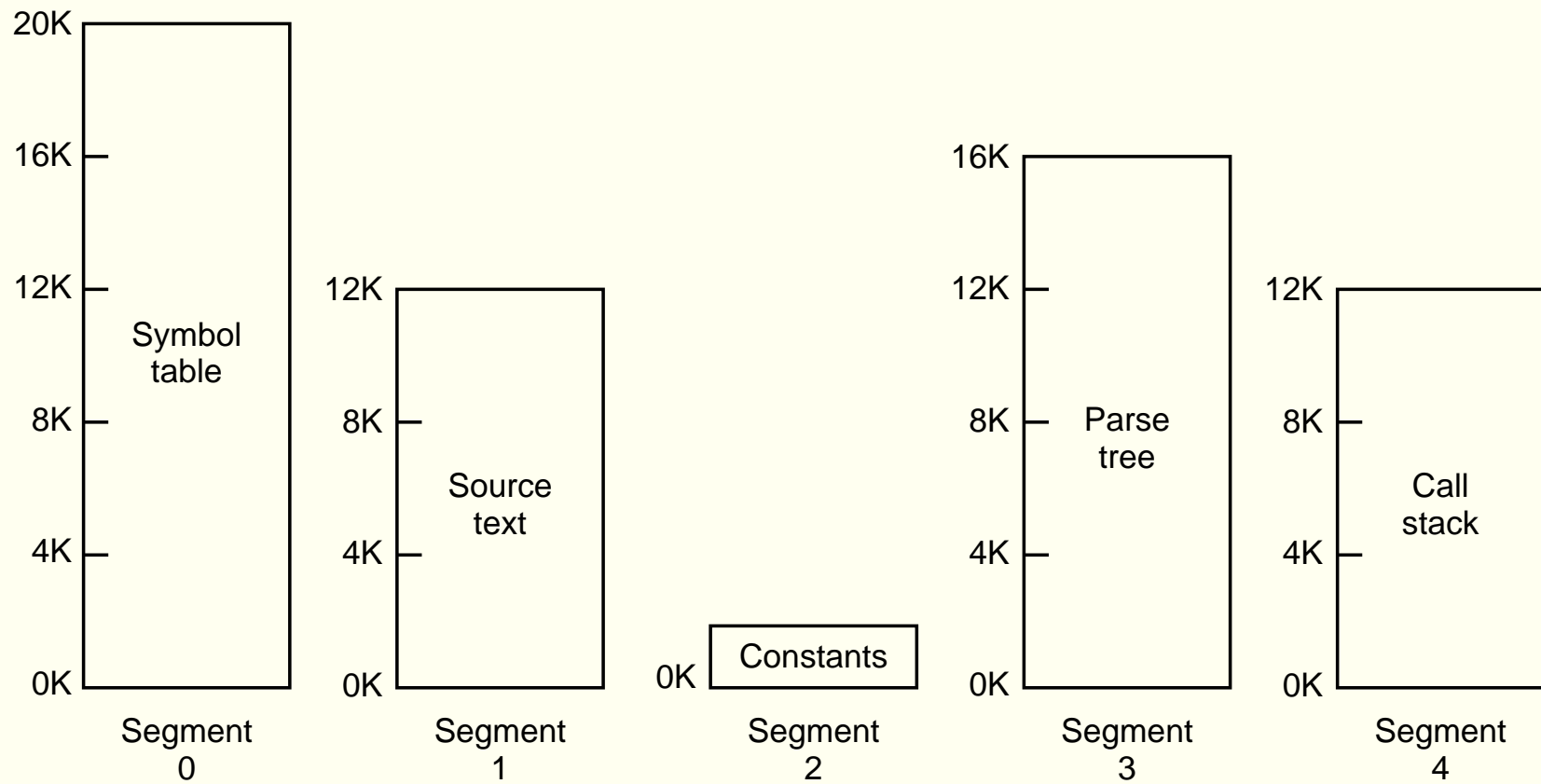
# Memória compartilhada



# 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

# Segmentação



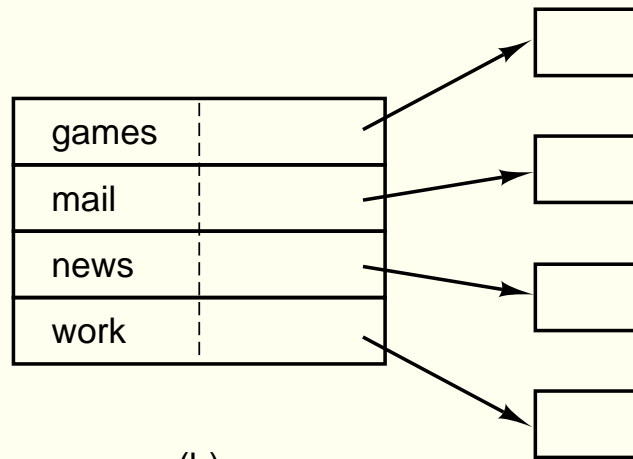
Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can 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

# Sistema de arquivos

# Atributos de arquivos

games	attributes
mail	attributes
news	attributes
work	attributes

(a)

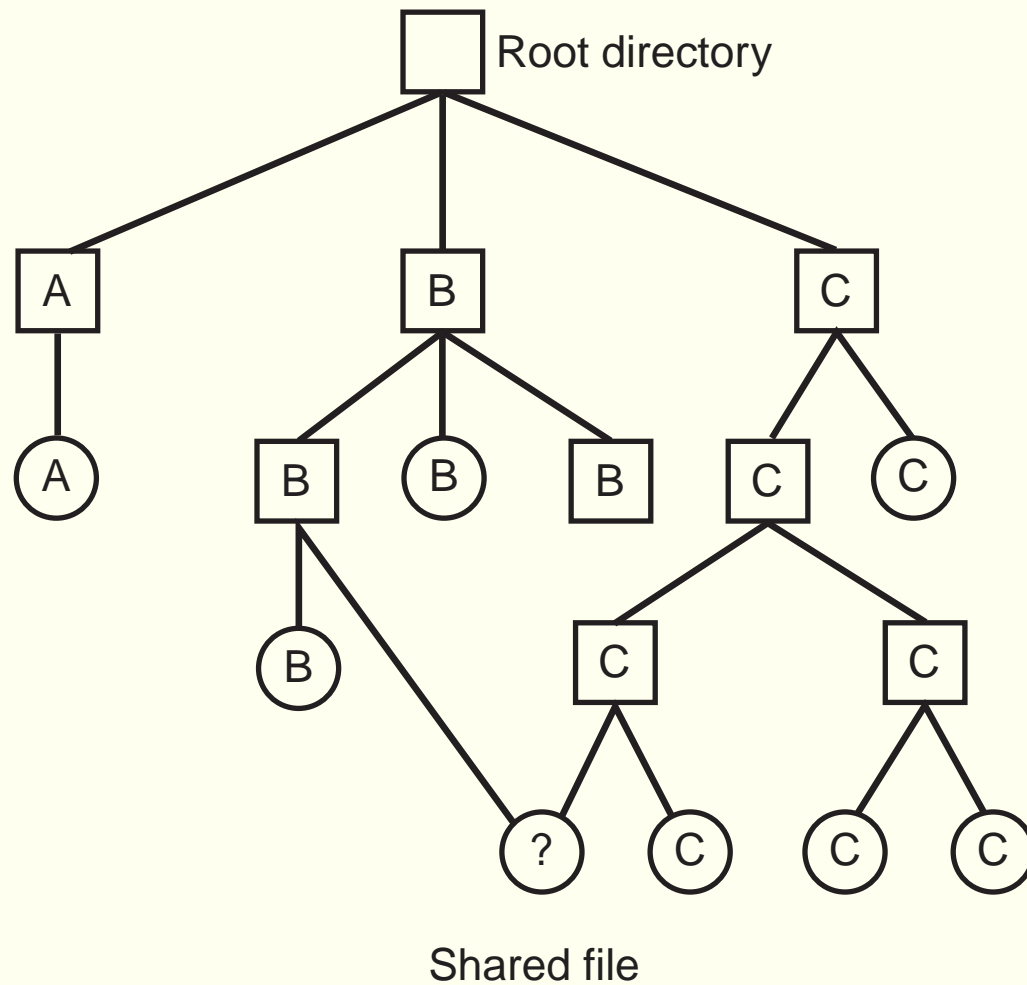


(b)

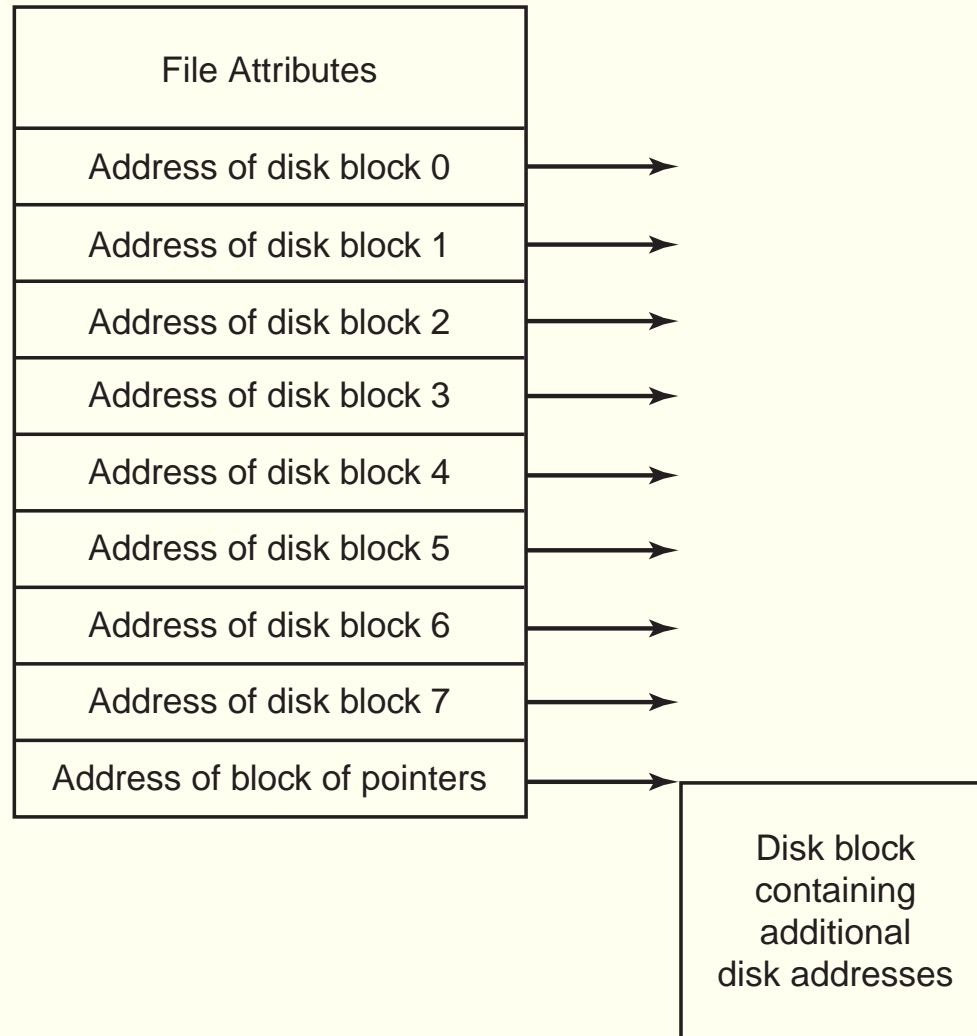
Data structure  
containing the  
attributes

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

# Arquivos compartilhados



# I-node

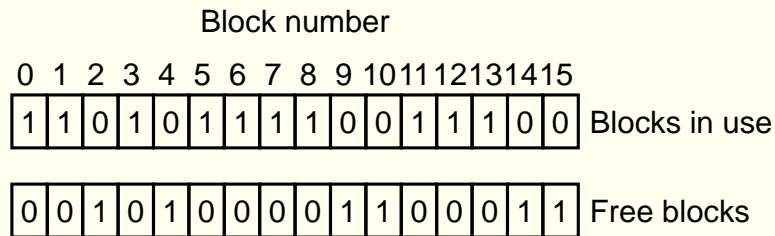




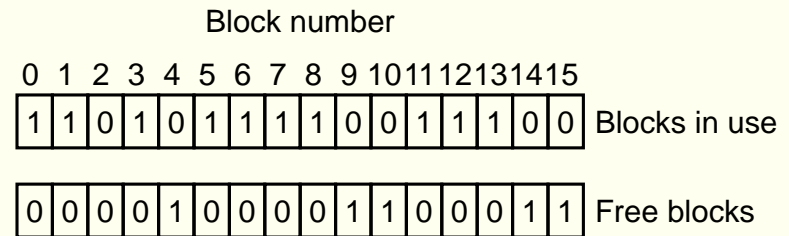
# Arquivos compartilhados

- 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

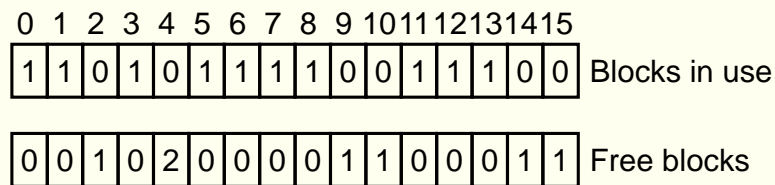
# Consistência do sistema de arquivos



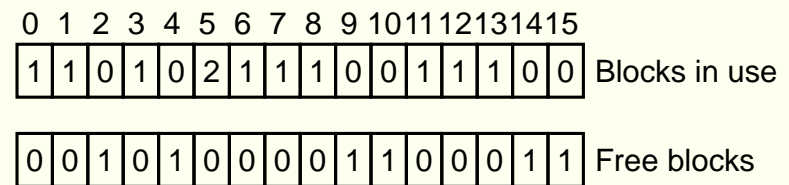
(a)



(b)



(c)



(d)

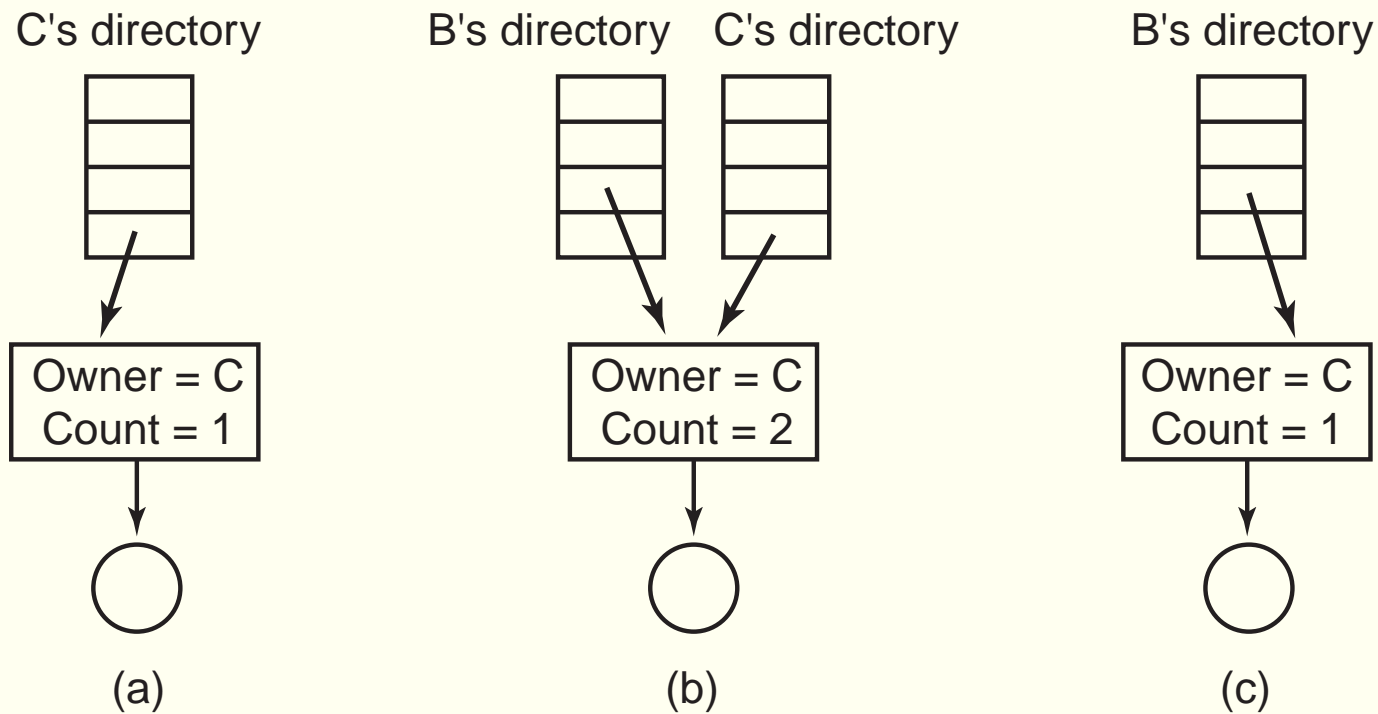
(a) consistente

(b) bloco faltando

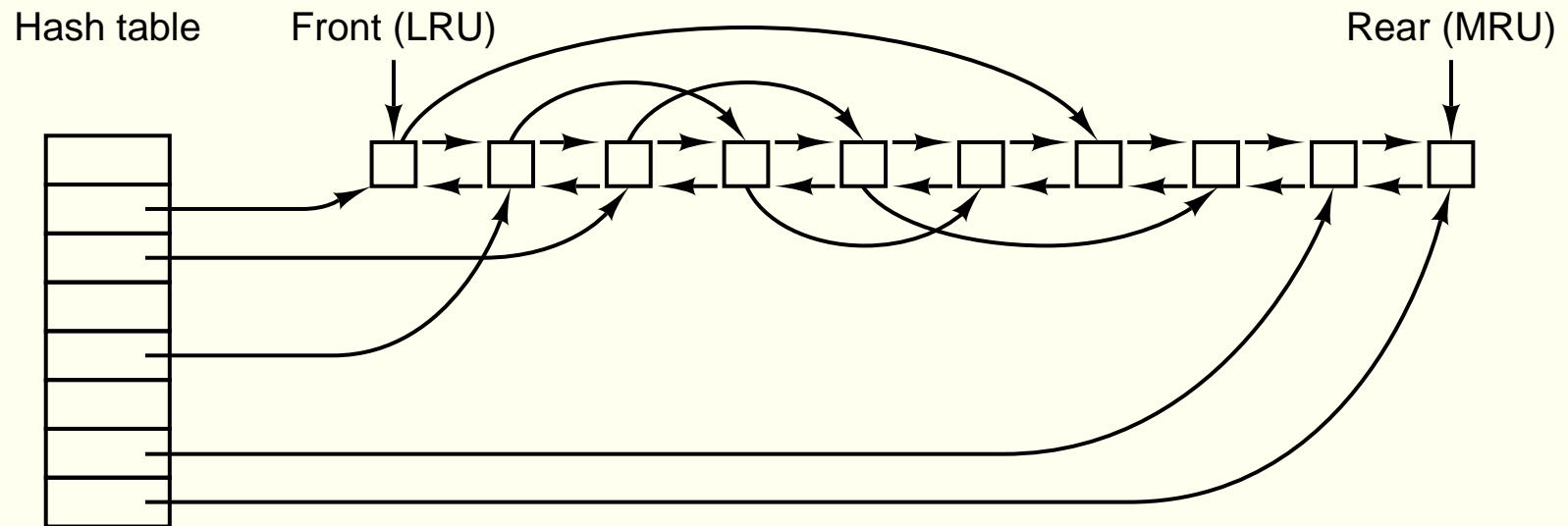
(c) duplicação na  
lista de livres

(d) duplicação nos  
dados

# Consistência do sistema de arquivos



# Caching



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

## Pipes

```
$ grep xxx log.txt > log-xxx.txt
```

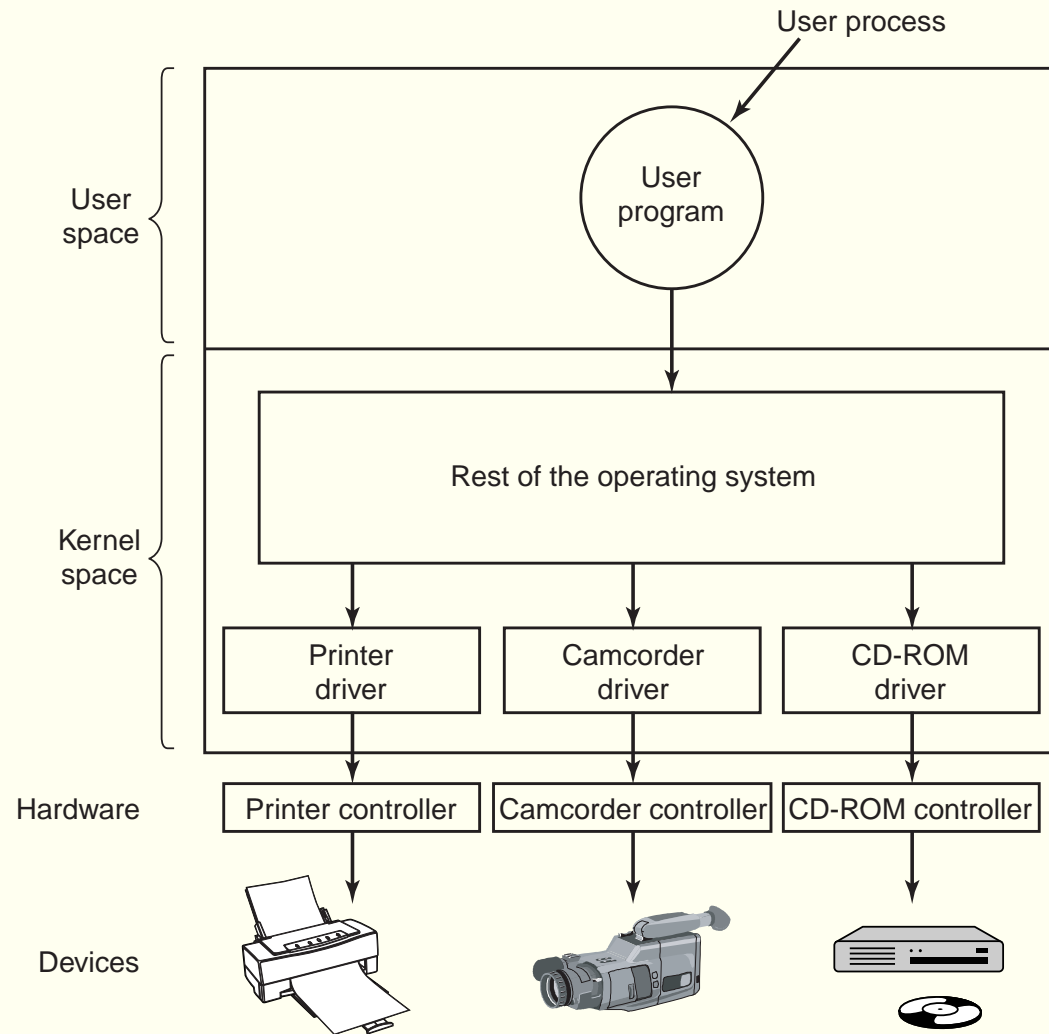
```
$ wc -l log-xxx.txt
```

```
$ rm log-xxx.txt
```

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

**Entrada e saída**

# Device drivers



# 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

## Programmed I/O

```
copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY) ;
    *printer_data_register = p[i];
}
return_to_user();
```

*/\* p is the kernel bufer \*/*  
*/\* loop on every character \*/*  
*/\* loop until ready \*/*  
*/\* output one character \*/*

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)

(a) Trecho de código do kernel

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

(b)

(b) Tratador da interrupção

# Direct Memory Access (DMA)

