

MC504/MC514 - Sistemas Operacionais

Gerência de Memória

Islene Calciolari Garcia

Instituto de Computação - Unicamp

Segundo Semestre de 2016

Sumário

- 1 Introdução
- 2 Alocação contínua
- 3 Malloc
- 4 Alocação não-contínua

Gerenciamento de Memória

Idealmente, a memória deveria ser

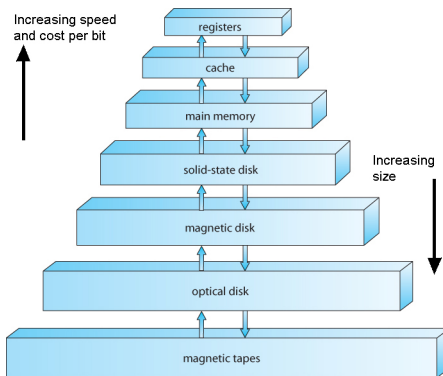
- rápida,
- de custo baixo,
- imensa e
- não volátil.

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

Hierarquia de Memória



Silberschatz et al.: Figura 1.4

Hierarquia de Memória

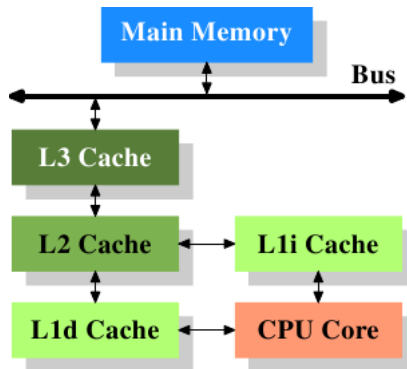
Registradores

- Internos à CPU
 - Extremamente rápidos
 - Otimizações de código podem mover temporariamente variáveis para registradores.
 - Programas podem dar palpites sobre o que deve ficar armazenado nos registradores
- ```
register int r;
```
- Veja o código `register.c` e o código assembly gerado sem otimização (`register-O0.s`) e com otimização (`register-O2.s`)

# Hierarquia de Memória

## Cache

- Internos ou muito próximos à CPU



- Cache hit e cache miss

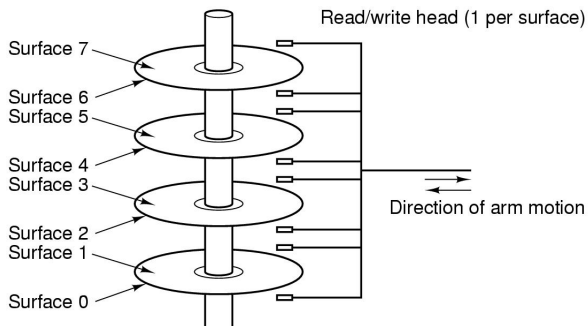
# Hierarquia de Memória

## Memória Principal

- Random Access Memory (RAM)
  - Compromisso entre preço e desempenho
  - Armazenamento volátil
- Flash memory
  - Tipo de *Solid State Disk*
  - Desempenho? Preço? Durabilidade?
  - Armazenamento não volátil

# Hierarquia de Memória

## Disco



Tanenbaum: Figura 1.8



# Hierarquia de Memória

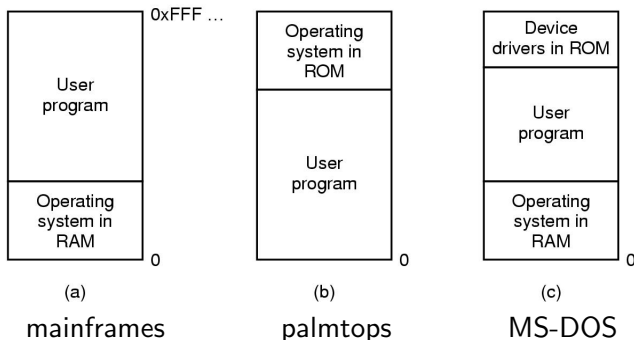
## Outros tipos de memória

- ROM (Read Only Memory)
  - rápida e barata
  - bootstrap loader está gravado em ROM
- EEPROM (Electrically Erasable ROM)
  - podem ser apagadas (erros podem ser corrigidos)
- CMOS
  - dependem de uma bateria
  - armazenam relógio e configurações
- Fitas magnéticas
  - Utilizadas (antigamente?) para backups
  - Grandes quantidades de dados, acesso sequencial

# Alocação de Espaço e Técnicas de Gerenciamento

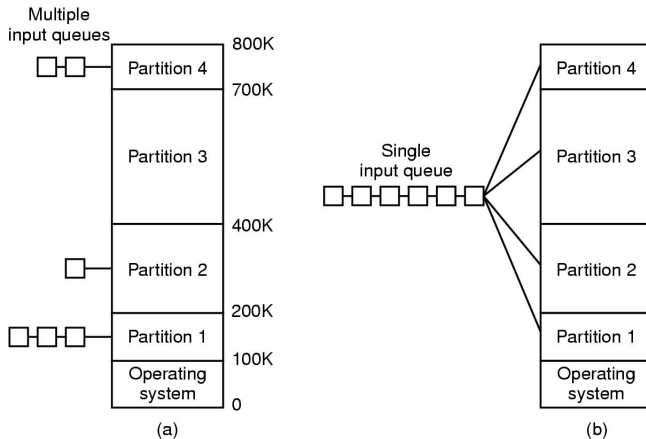
- Alocação contínua
  - máquinas antigas
  - malloc
- Alocação não-contínua
  - memória virtual

# Monoprogramação



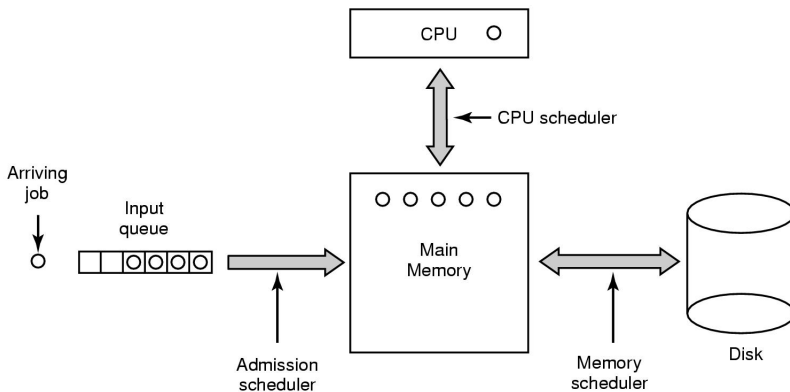
Tanenbaum: Figura 4-1

# Multiprogramação e partições fixas



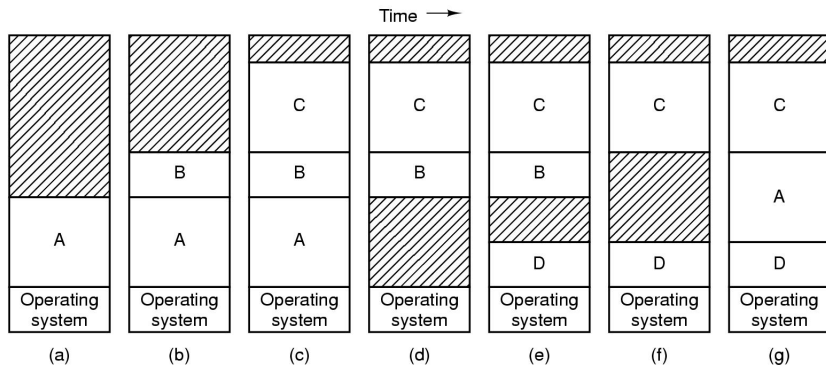
Tanenbaum: Figura 4.2

# Swapping



Tanenbaum: Figura 2-40

# Swapping



Tanenbaum: Figura 4-5

# Relocação e Proteção

- Relocação  
Um programa deve poder rodar em endereços físicos distintos.
- Proteção  
Um programa não pode fazer acesso à área de memória reservada a outro programa.
- Relocação durante a carga do programa
  - Todos os endereços precisam ser identificados e alterados
  - Não resolve o problema da proteção

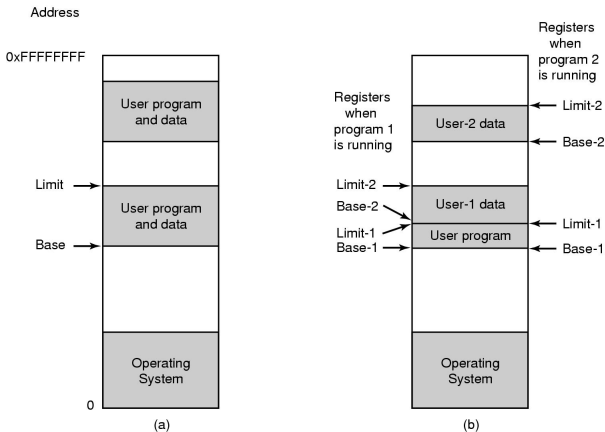
# Bits de proteção

|      |  |
|------|--|
| 1010 |  |
| 1010 |  |
| 1010 |  |
| 0011 |  |
| 0011 |  |

- Cada processo tem seus bits de proteção
- Verificação a cada acesso

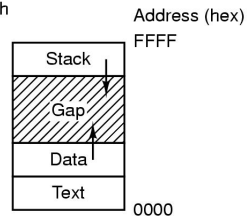
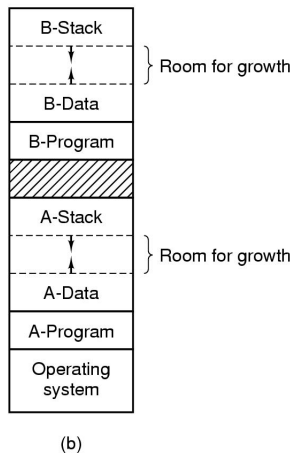
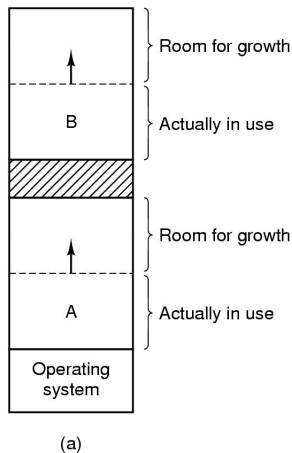


# Registradores base e limite



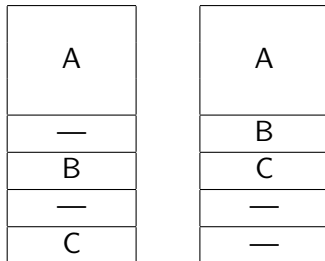
Tanenbaum: Figura 1-9

# Espaço para crescimento



Tanenbaum: Figuras 4-6 e 1-20

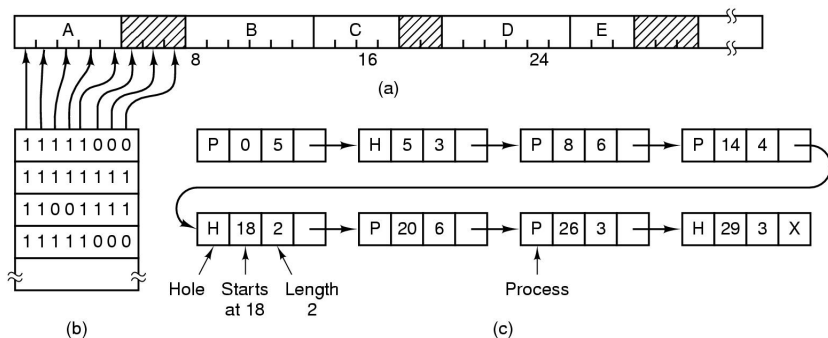
# Compactação de memória



# Malloc, free e realloc

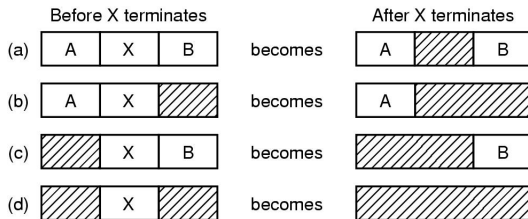
```
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

# Bitmaps e lista de livres



Tanenbaum: Figura 4.7

# Atualização da lista



Tanenbaum: Figura 4.8

Veja os códigos: `erro_malloc.c` `erro_calloc.c`

# Malloc utiliza lista de livres

## Protegida?

- Veja o código `erro_malloc.c` (o código `erro_calloc.c` zera os dados para facilitar a observação da lista de livres)
- No gdb execute comandos do tipo:

```
(gdb) p (int[10]) *s1
```

```
(gdb) p (int[10]) *(s1-2)
```

```
(gdb) set *(s1-2) = ...
```

# Algoritmos para alocação de memória

- *First fit*: primeiro bloco grande o suficiente
- *Next fit*: próximo bloco grande o suficiente
- *Best fit*: menor bloco grande o suficiente
- *Worst fit*: maior bloco grande o suficiente
- Veja o código `fit.c` e descubra a política de alocação do `malloc`
  - Blocos pequenos ( $< 64$  bytes) =
  - Blocos médios =
  - Blocos grandes ( $> 512$  bytes) =



# E o kmalloc?

```
#include <linux/slab.h>
```

```
/* ... */
```

```
int *s1 = kmalloc (size, flags);
printk("KMALLOC: %p:\n", (void *) s1);
```

# Alocação não-contínua de memória

- Diminui o problema de fragmentação
- Maior flexibilidade
  - maior número de programas na memória ao mesmo tempo
  - programas podem ser maiores do que a memória física
- Maior complexidade de gerenciamento

# Overlays

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

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

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

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

```
main() {
```

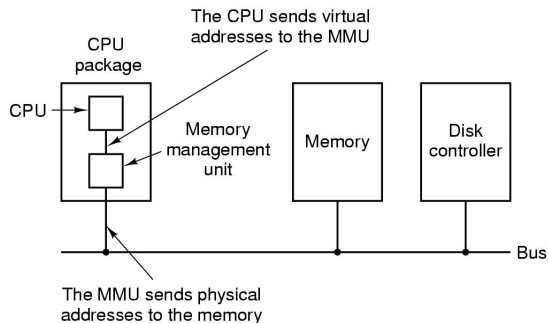
```
 fase_1(); /* funcoes f e dados d1, d2, d3 */
```

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

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

```
}
```

# Memória Virtual

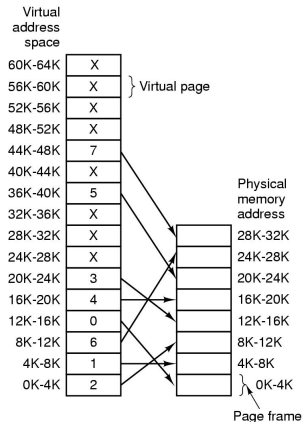


Memory Management

Unit (MMU)

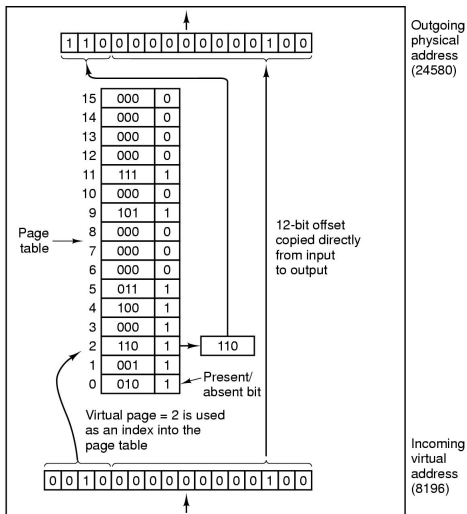
Tanenbaum: Figura 4.9

# Paginação



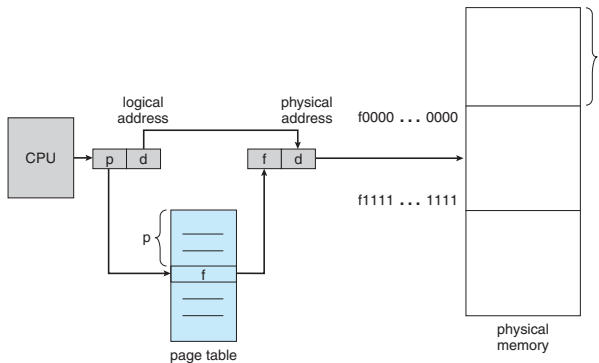
Tanenbaum: Figura 4.10

# Maapeamento dos endereços



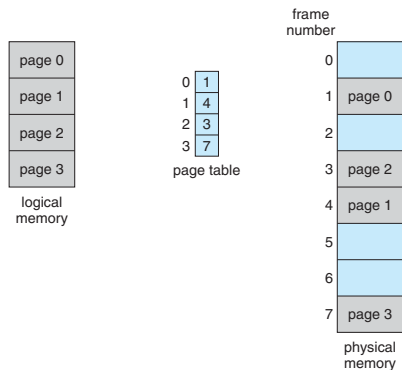
Tanenbaum: Figura 4.11

# Mapeamento de endereços



Silberschatz: Figura 8.10

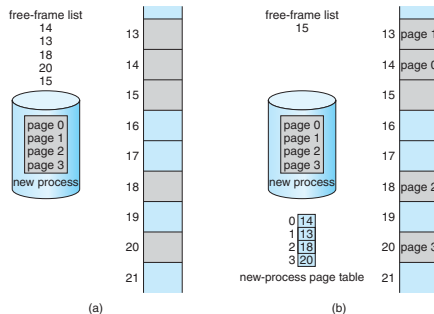
# Mapeamento de endereços



Silberschatz: Figura 8.11



# Novo processo



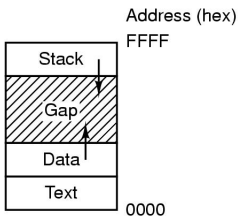
Silberschatz: Figura 8.13

# Paginação - Exemplo

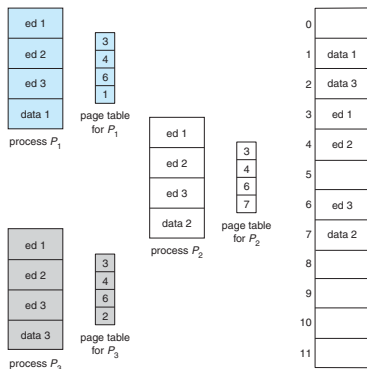
- 32 bits de endereço
- páginas de 4k
- 20 primeiros bits indicam a página
- 12 últimos bits indicam o deslocamento dentro da página
- Veja o código `pagesize.c`

# Espaço de endereçamento

- Apenas as páginas ocupadas precisam ser mapeadas
- Veja o código `sbrk.c`

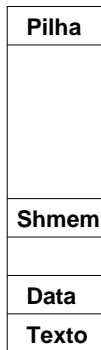
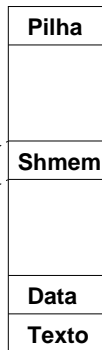


# Compartilhamento de código



Silberschatz: Figura 8.16

# Memória compartilhada

**Processo A****Processo B**

# Memória compartilhada

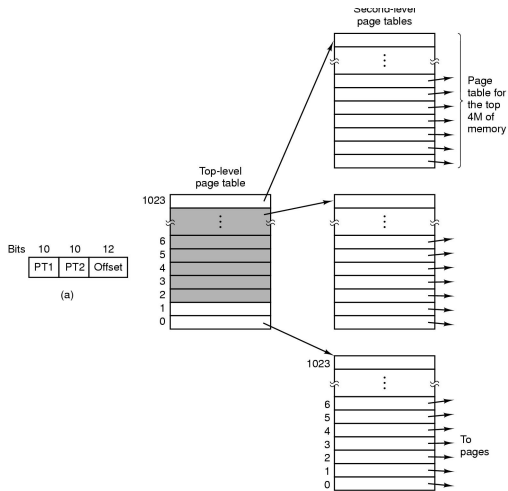
```
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shmid,
 const void *shmaddr, int shmflg);
```

- Veja os exemplos: sh1.c sh2.c sh\_fork.c sh\_server.c e sh\_client.c

# Mmap

- Um arquivo pode ser mapeado em memória
- Memória compartilhada
- Veja `map.c` `map2.c`

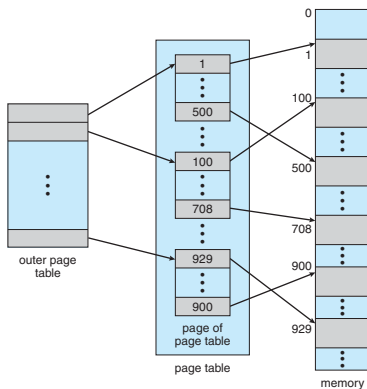
# Tabelas de mais de um nível



Tanenbaum: Figura 4.12

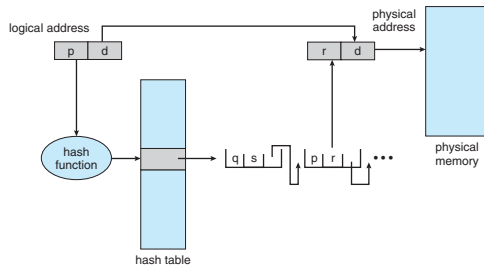


# Tabelas de mais de um nível



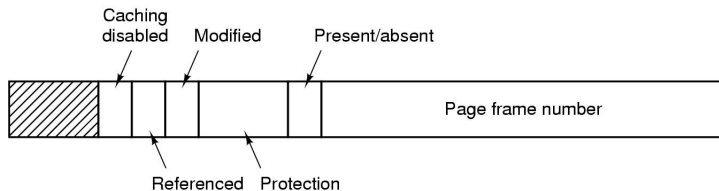
Silberschatz: Figura 8.17

# Hashed page table



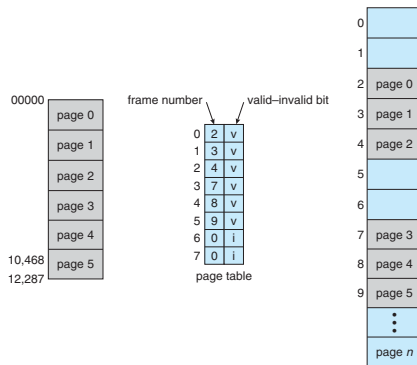
Silberschatz: Figura 8.19

# Entrada na tabela



Tanenbaum: Figura 4.13

# Bit válido e inválido



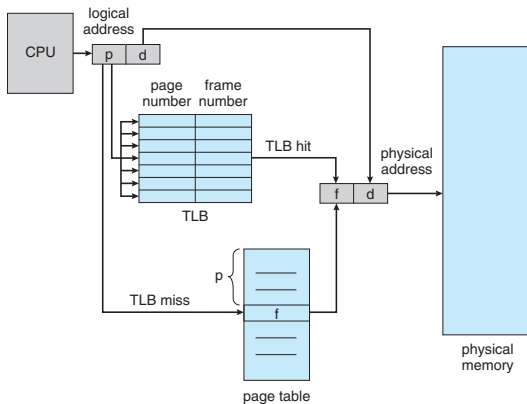
Silberschatz: Figura 8.15

# TLB: Translation Lookaside Buffer

| 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         |

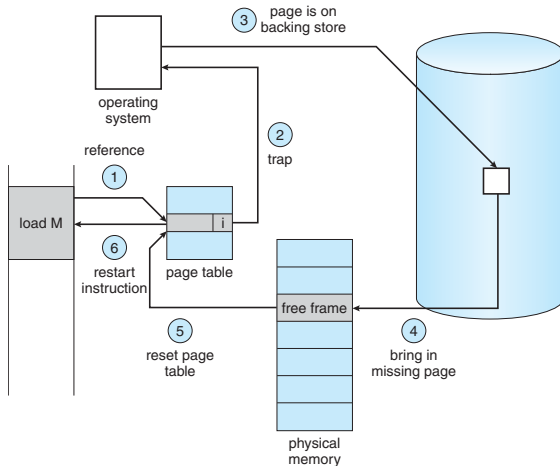
Tanenbaum: Figura 4.14

# TLB: Translation Lookaside Buffer



Silberschatz: Figura 8.14

# Page fault



Silberschatz: Figura 9.06

# Substituição de páginas

- Veja o código: pag.c

Algoritmo ótimo:

- Baseado no uso futuro de uma página
- Impossível de ser implementado
- Pode ser simulado (segunda execução do mesmo processo com a mesma entrada)
- Útil para medidas de desempenho



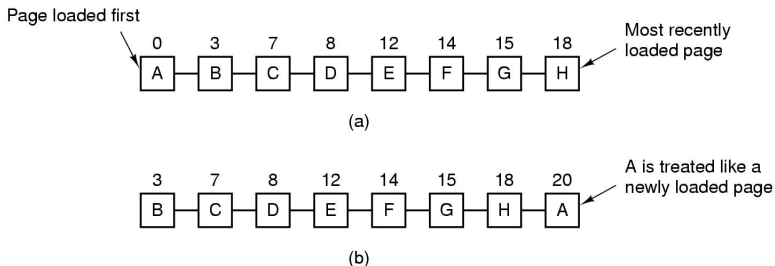
# Não usada recentemente

- Classe 0: não referenciada, não modificada
- Classe 1: não referenciada, mas modificada
- Classe 2: referenciada, mas não modificada
- Classe 3: referenciada e modificada

# First In, First Out

- Simplemente coloca as páginas em uma fila
- Pode remover páginas importantes

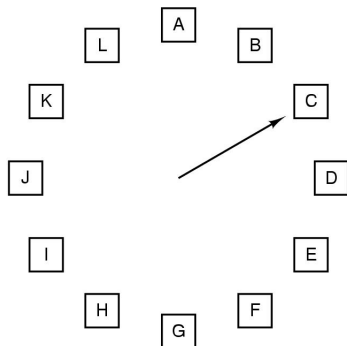
# Segunda chance



Tanenbaum: Figura 4.16

- Se o bit  $R == 0$ , a página é substituída, senão
- bit  $R$  é limpo e a página é colocada no final da fila

# Relógio



When a page fault occurs,  
the page the hand is  
pointing to is inspected.  
The action taken depends  
on the R bit:  
R = 0: Evict the page  
R = 1: Clear R and advance hand

Tanenbaum: Figura 4.17

- Implementação circular da segunda chance

# Uso menos recente

- LRU (Least Recently Used)
- Implementação utilizando lista ligada
- Implementação em hardware com contador
  - incrementado a cada instrução
  - entrada na tabela deve armazenar o contador
- Implementação com matriz  $n \times n$

# LRU em hardware

|   |      |   |   |   |
|---|------|---|---|---|
|   | Page |   |   |   |
|   | 0    | 1 | 2 | 3 |
| 0 | 0    | 1 | 1 | 1 |
| 1 | 0    | 0 | 0 | 0 |
| 2 | 0    | 0 | 0 | 0 |
| 3 | 0    | 0 | 0 | 0 |

(a)

|   |      |   |   |   |
|---|------|---|---|---|
|   | Page |   |   |   |
|   | 0    | 1 | 2 | 3 |
| 0 | 0    | 0 | 1 | 1 |
| 1 | 1    | 0 | 1 | 1 |
| 2 | 0    | 0 | 0 | 0 |
| 3 | 0    | 0 | 0 | 0 |

(b)

|   |      |   |   |   |
|---|------|---|---|---|
|   | Page |   |   |   |
|   | 0    | 1 | 2 | 3 |
| 0 | 0    | 0 | 0 | 1 |
| 1 | 1    | 0 | 0 | 1 |
| 2 | 1    | 1 | 0 | 1 |
| 3 | 0    | 0 | 0 | 0 |

(c)

|   |      |   |   |   |
|---|------|---|---|---|
|   | Page |   |   |   |
|   | 0    | 1 | 2 | 3 |
| 0 | 0    | 0 | 0 | 0 |
| 1 | 1    | 0 | 0 | 0 |
| 2 | 1    | 1 | 0 | 0 |
| 3 | 1    | 1 | 1 | 0 |

(d)

|   |      |   |   |   |
|---|------|---|---|---|
|   | Page |   |   |   |
|   | 0    | 1 | 2 | 3 |
| 0 | 0    | 0 | 0 | 0 |
| 1 | 1    | 0 | 0 | 0 |
| 2 | 1    | 1 | 0 | 1 |
| 3 | 1    | 1 | 0 | 0 |

(e)

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

(f)

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

(g)

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |

(h)

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |

(i)

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

(j)

Acessos: 0 1 2 3 2 1 0 3 2 3

Tanenbaum: Figura 4.18

# Simulando LRU em software

Página não usada frequentemente

- Contador de uso para cada página (soma o bit R a cada clock tick)
- Não esquece nada...
- Considere um compilador baseado em passos

# Não usada frequentemente

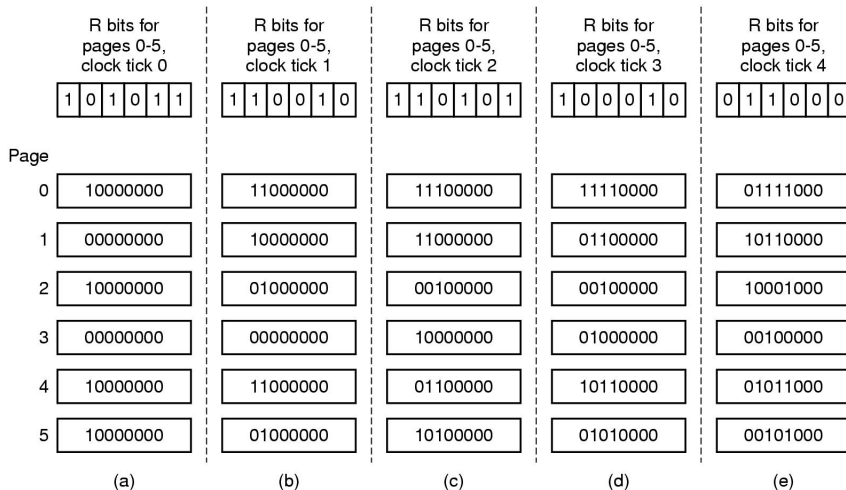
## Aging

- O contador é deslocado à direita
- Bit R é adicionado à esquerda

|      |  |             |      |  |             |
|------|--|-------------|------|--|-------------|
|      |  | 1 0 1 0 1 1 |      |  | 1 0 1 1 0 1 |
| >> 1 |  | 1 0 1 0 1   | >> 1 |  | 1 0 1 1 0   |
| + 1  |  | 1 1 0 1 0 1 | + 0  |  | 0 1 0 1 1 0 |



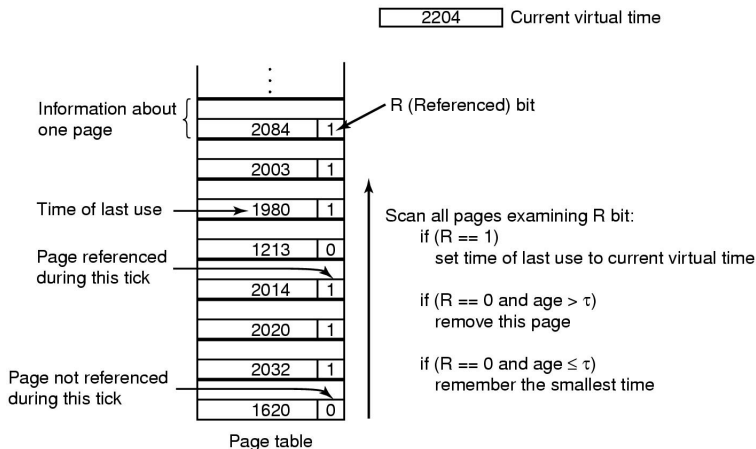
# Não usada frequentemente



# Working Set $w(k, t)$

- Conjunto de páginas utilizadas nas últimas  $k$  referências em relação ao instante  $t$ .
- Paginação sob demanda
- Prepaging
- Implementação exata é muito cara

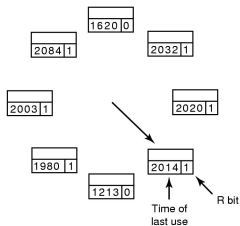
# Working Set



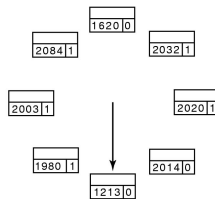
Tanenbaum: Figura 4.21

# WSClock

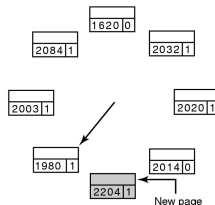
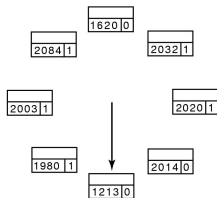
[ 2204 | Current virtual time



(a)



(b)



# Resumo

| 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