MC504 - Sistemas Operacionais

Gerência de Memória I

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

Sumário

Gerenciamento de Memória

Idealmente, a memória deveria ser

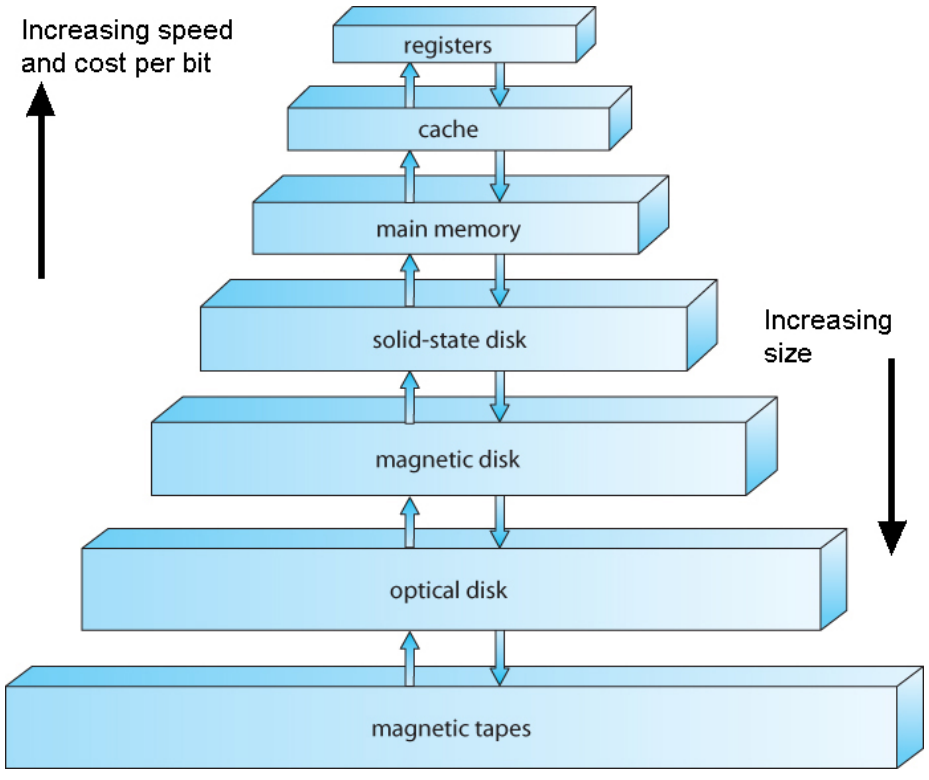
- rápida,
- de custo baixo,
- imensa e
- não volátil.

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

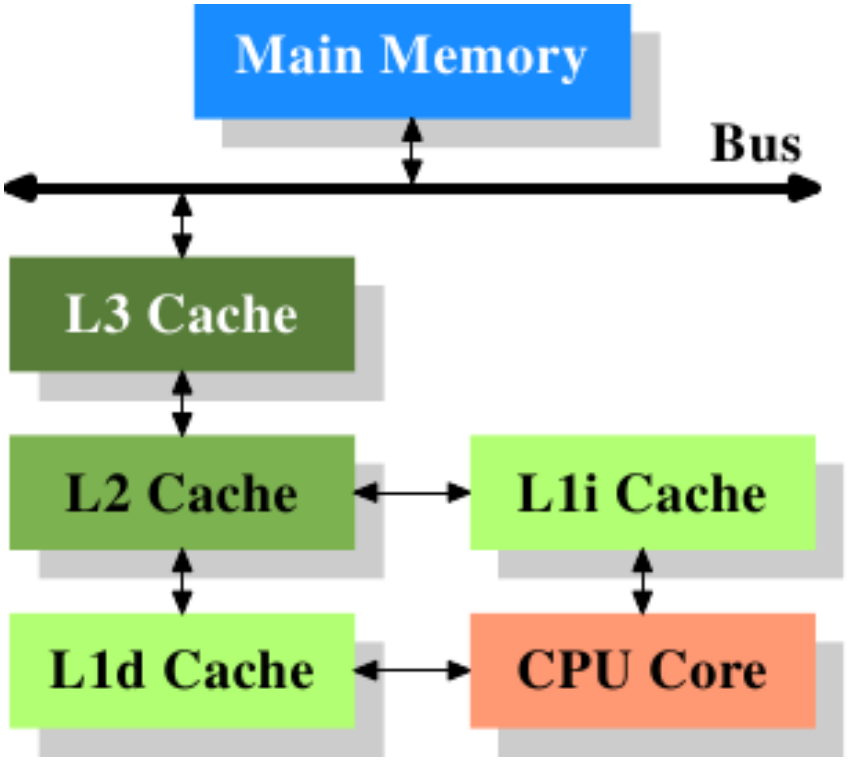
```
register int r;
```

- 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

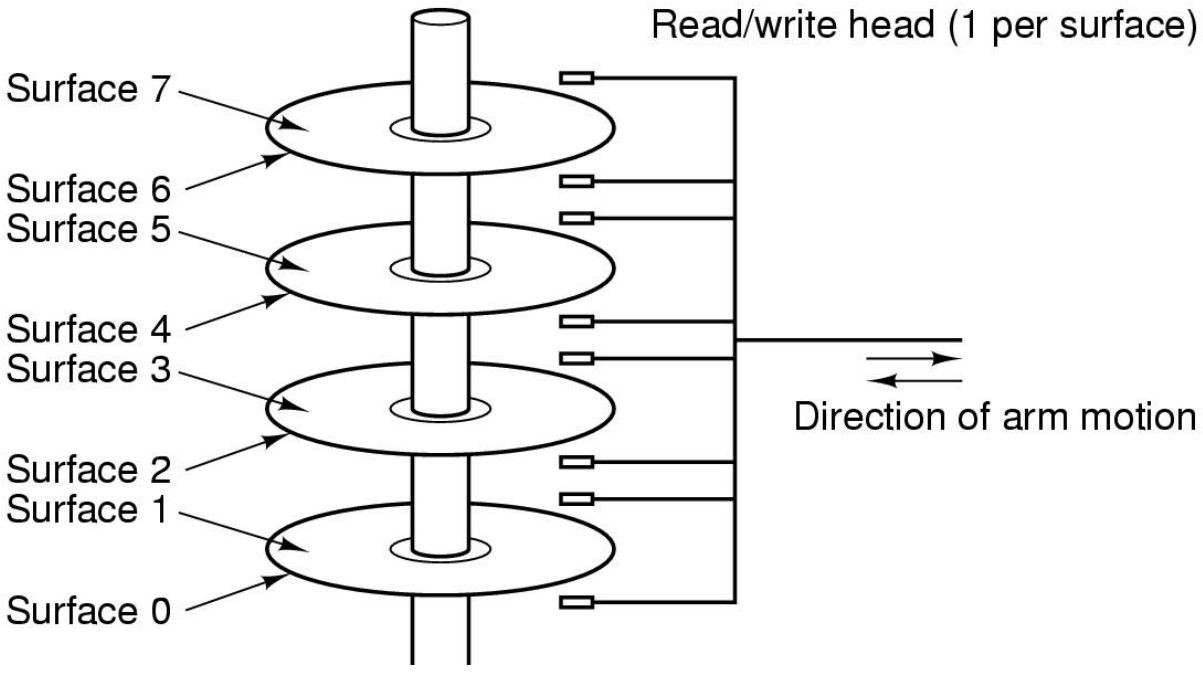
Hierarquia de Memória

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

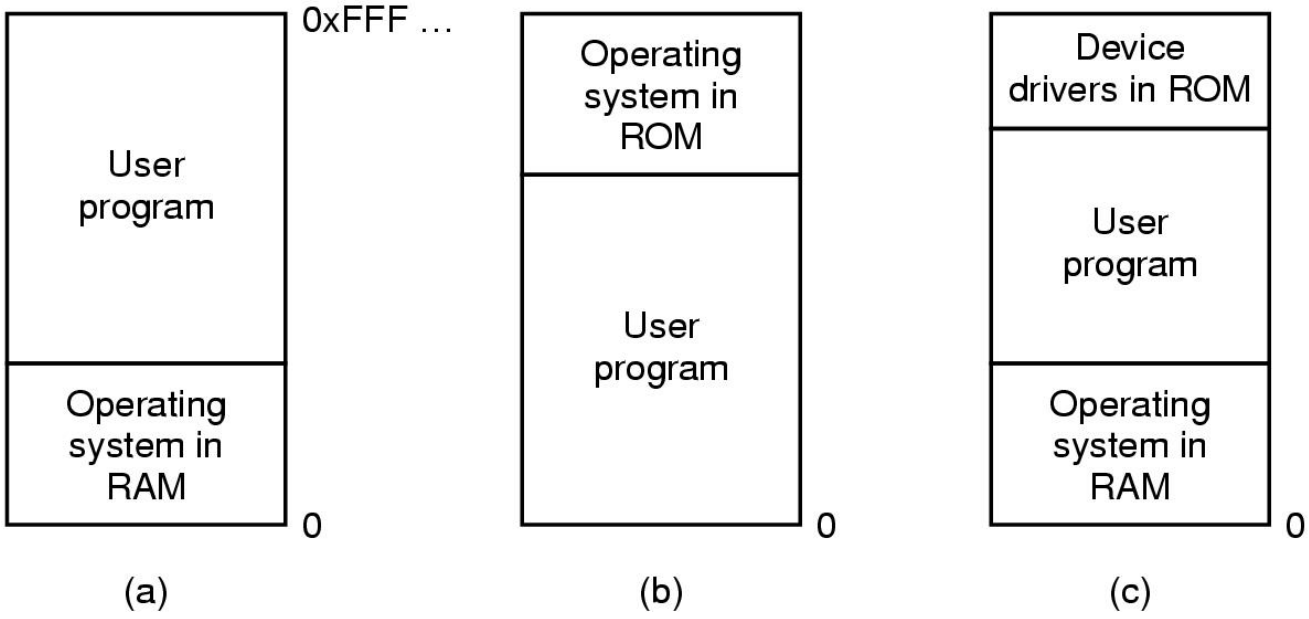
Outros tipos de memória

- 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

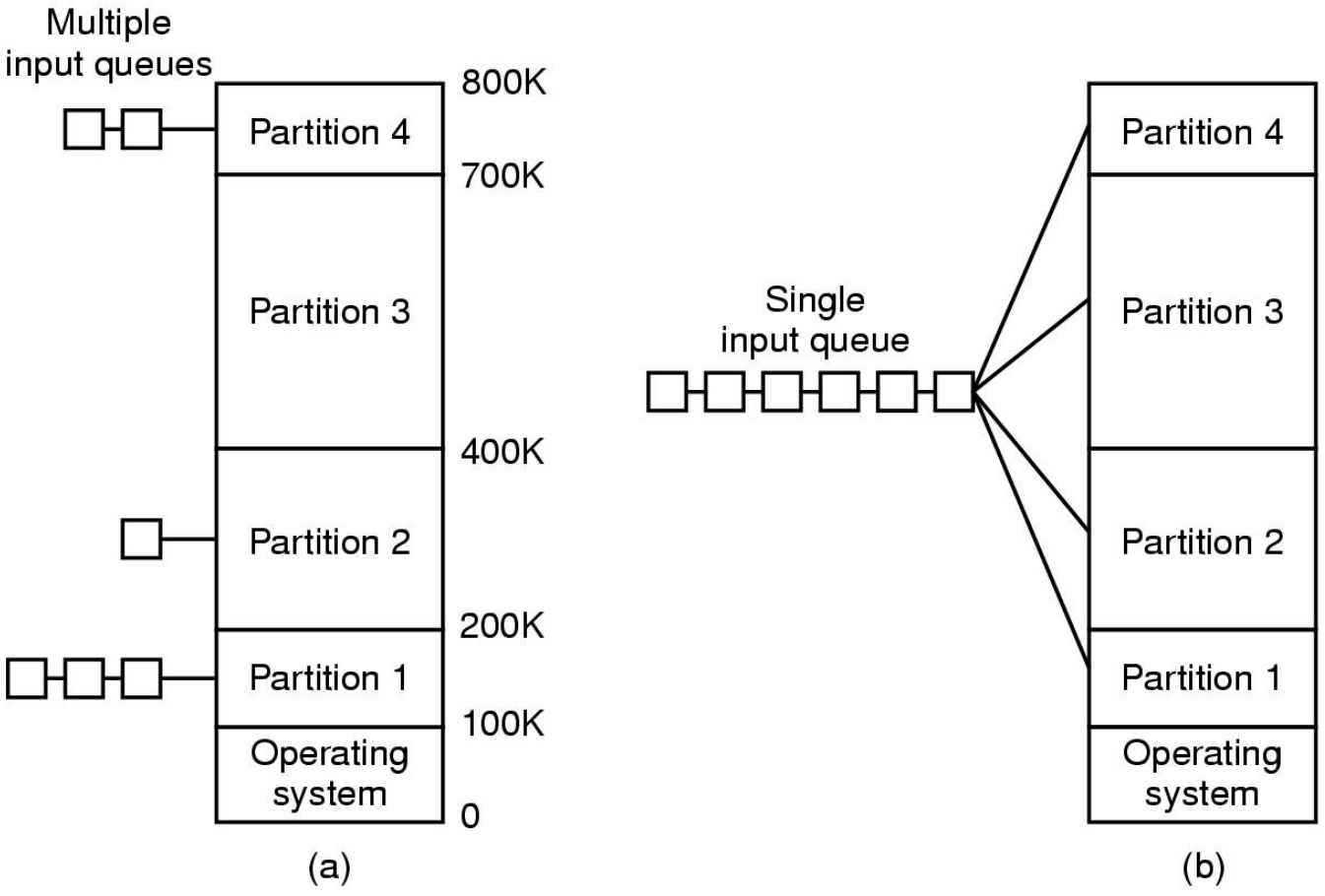
Monoprogramação



mainframes palmtops MS-DOS

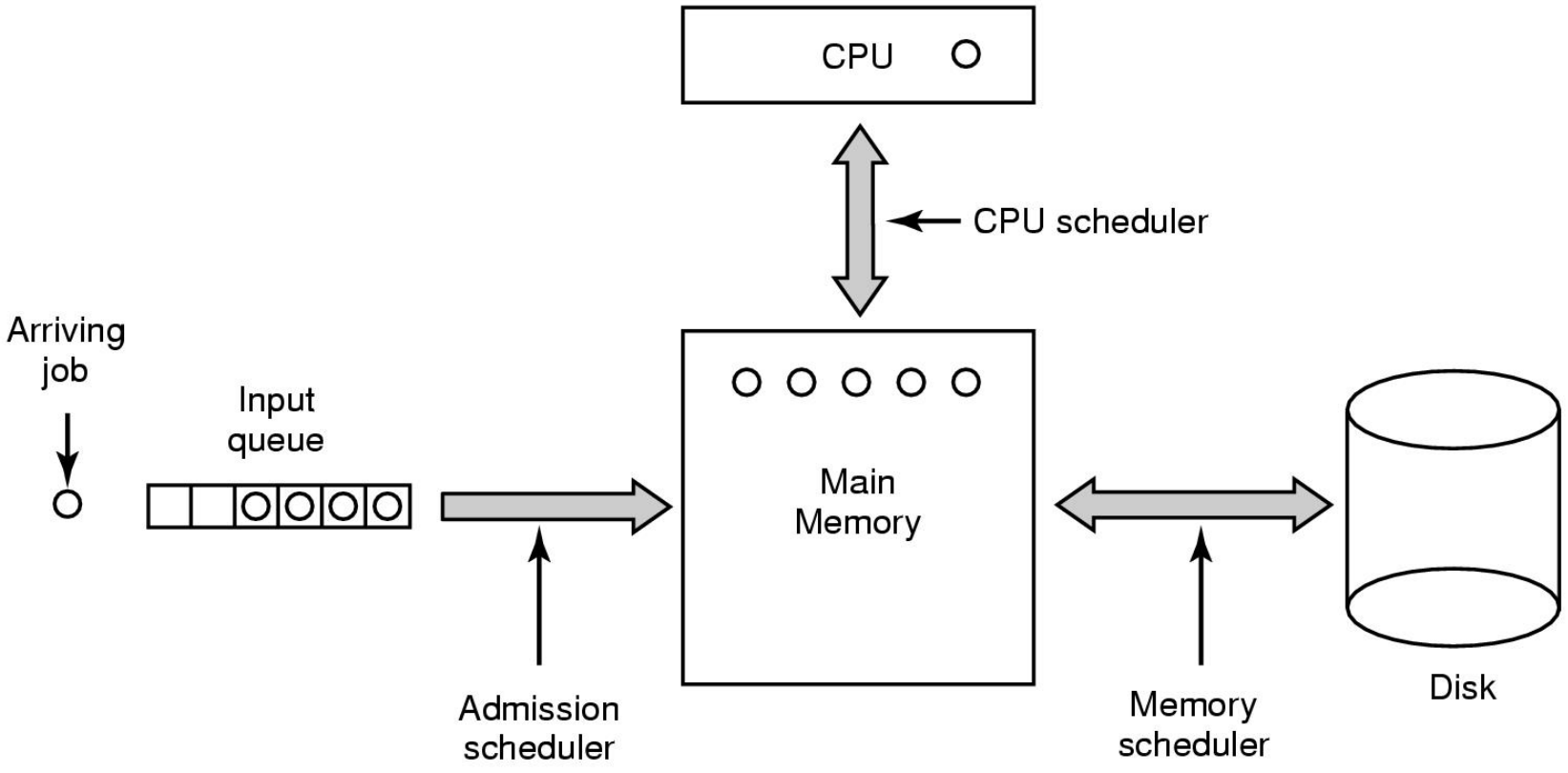
Tanenbaum: Figura 4-1

Multiprogramação e partições fixas



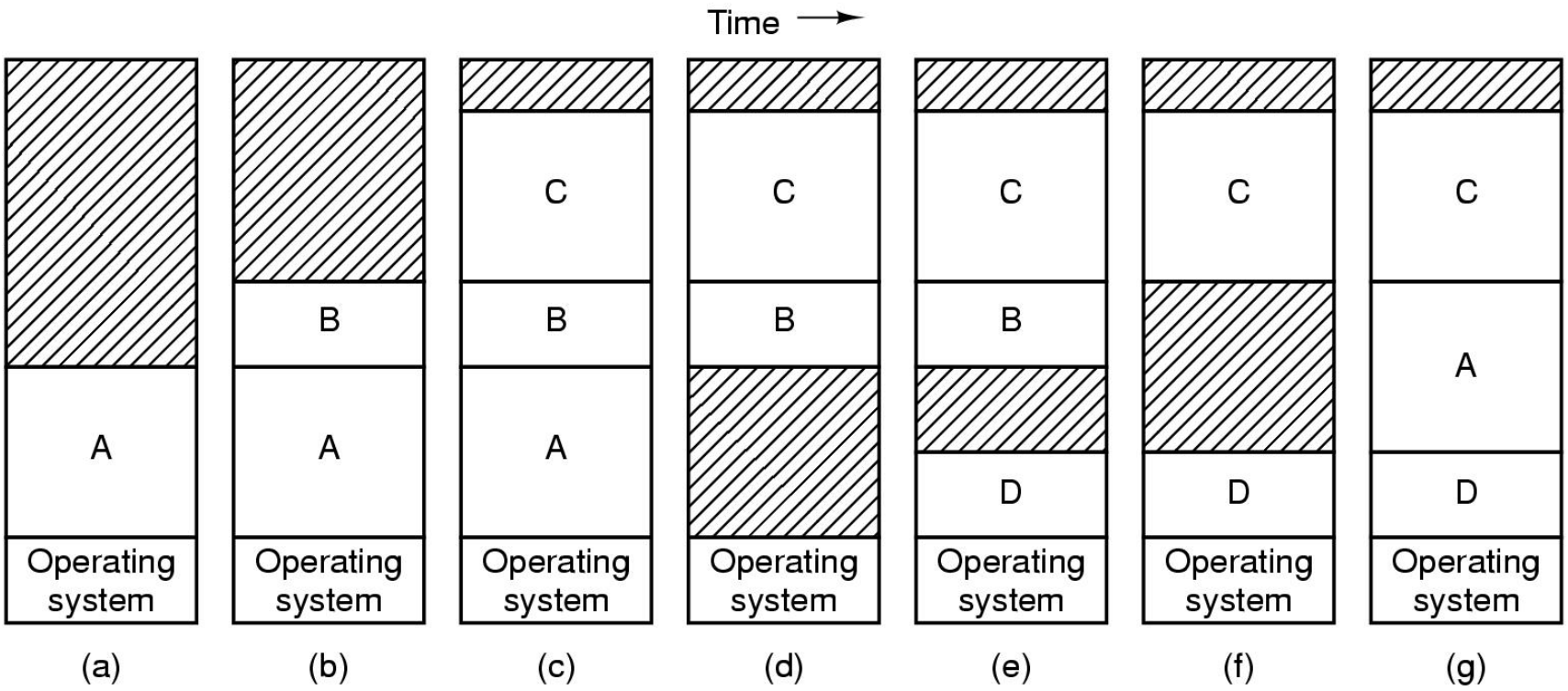
Tanenbaum: Figura 4.2

Swapping



Tanenbaum: Figura 2-40

Swapping

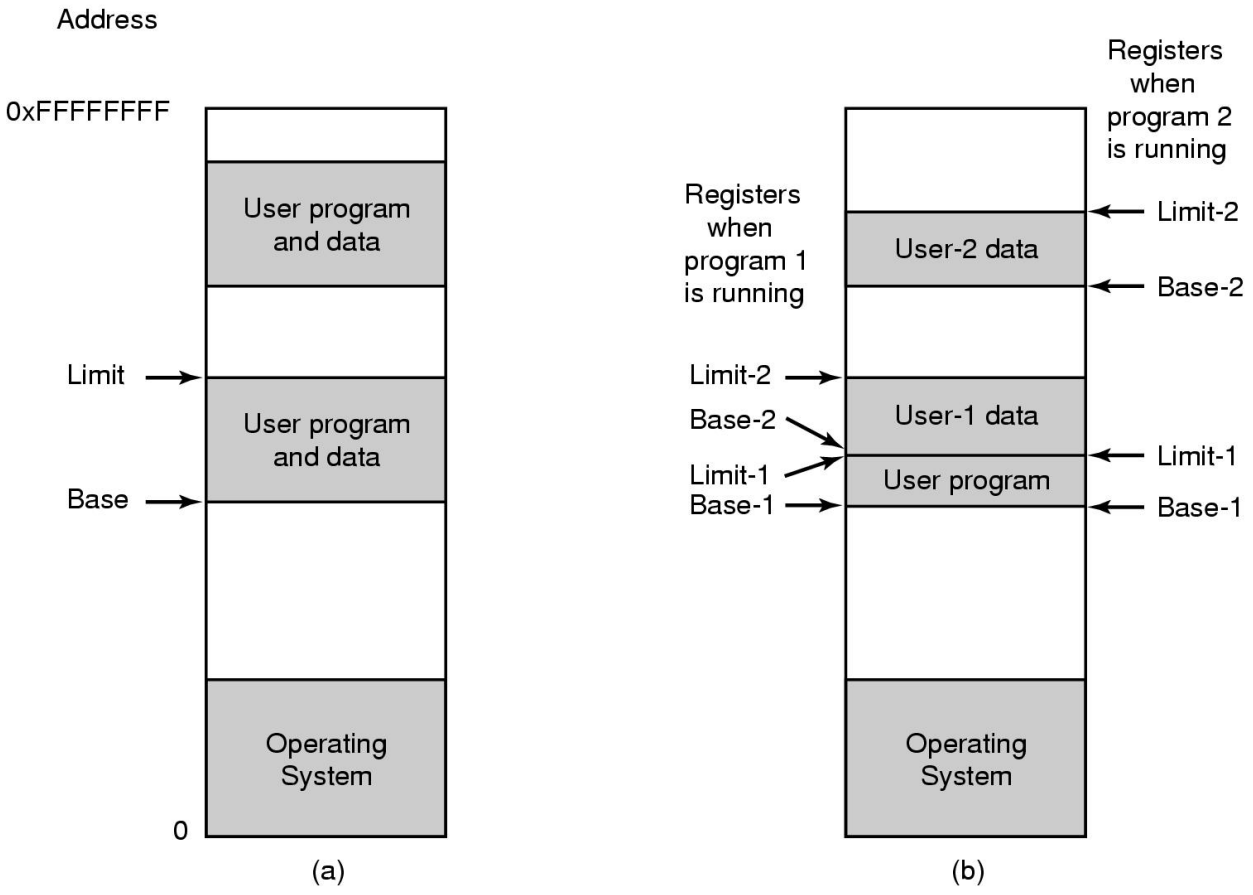


Tanenbaum: Figura 4-5

Relocação e Proteção

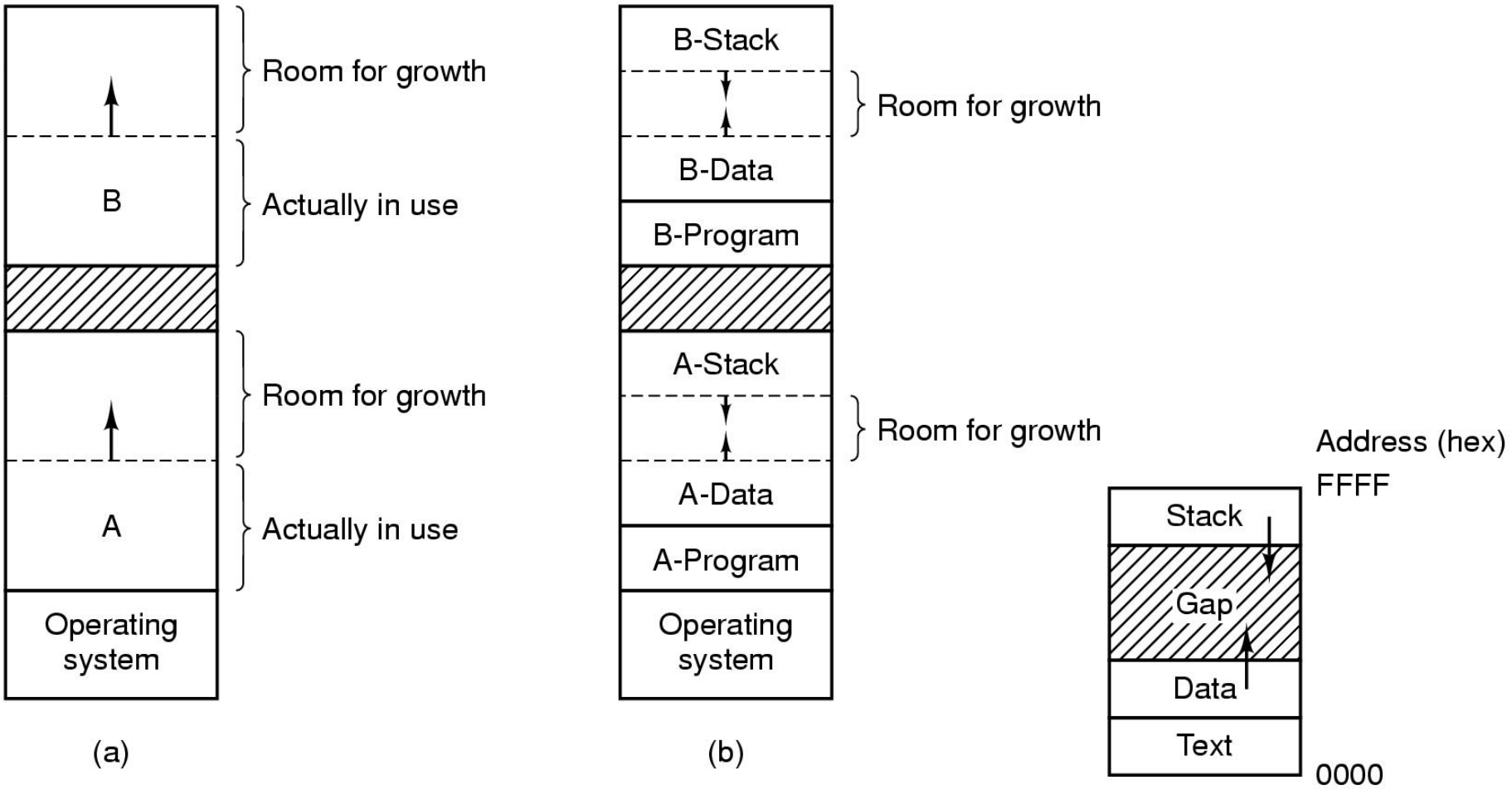
- Relocação
  Um programa deve poder rodar em endereços físicos distintos.
- Proteção
  Um programa não pode fazer acesso à área de memória reservada a outro programa.
- Relocação durante a carga do programa
  - Todos os endereços precisam ser identificados e alterados
  - Não resolve o problema da proteção

Registradores base e limite



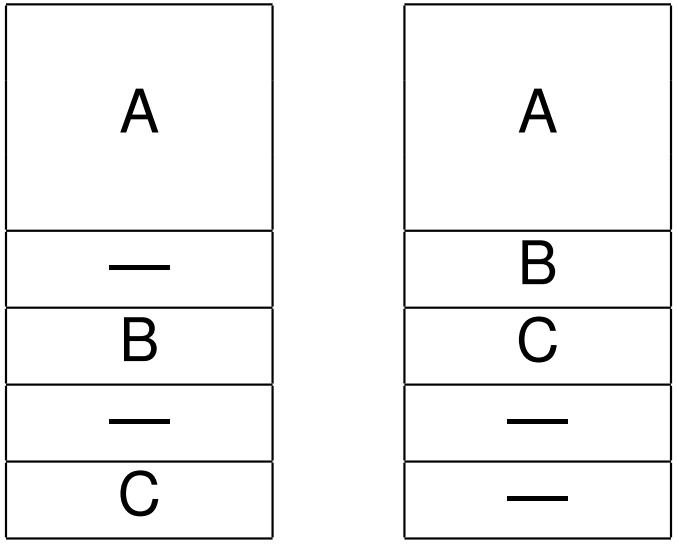
Tanenbaum: Figura 1-9

Espaço para crescimento



Tanenbaum: Figuras 4-6 e 1-20

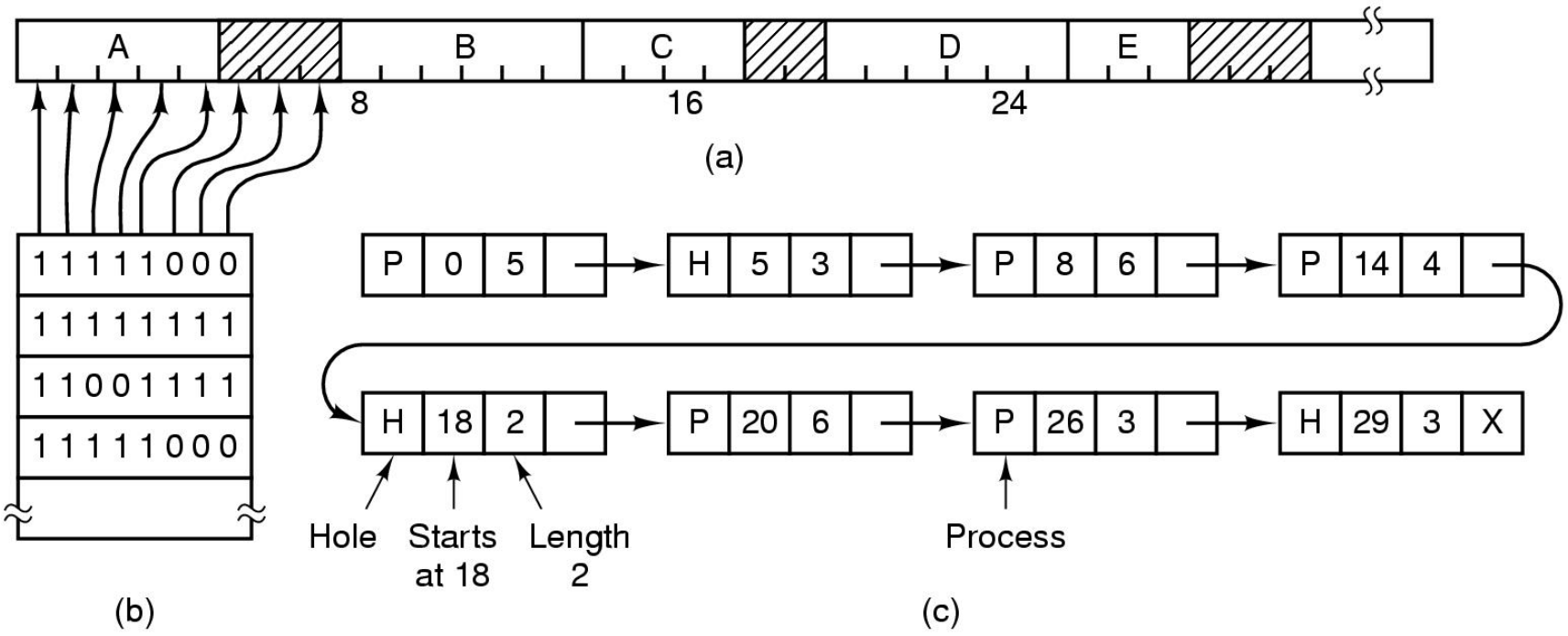
Compactação de memória



Malloc, free e realloc

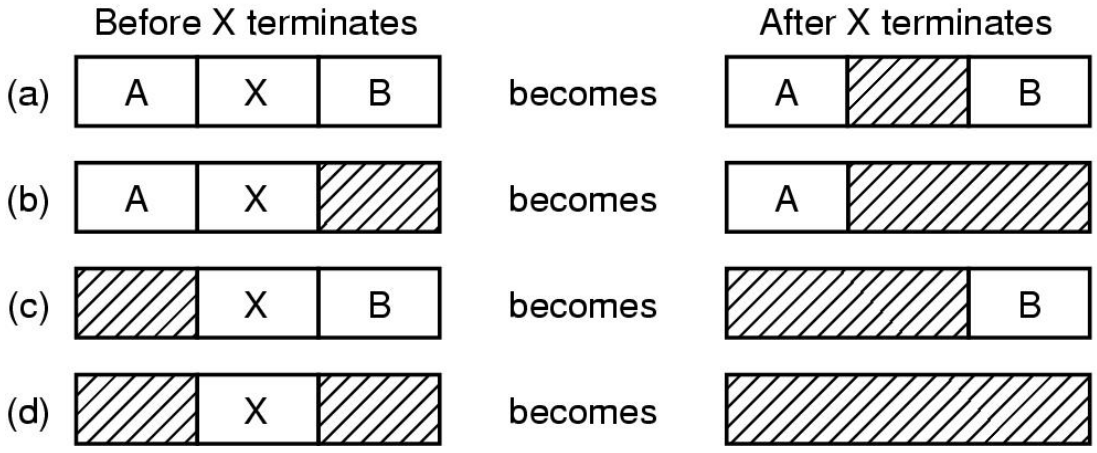
```
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

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?

Alocação de memória no kernel Linux

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

/* ...  */

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