Aula 7
Gerenciamento de memória

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

Professor João Victor de A. Oliveira

Hoje

- **Gerenciamento de memória (Capítulo 3)**
  - Introdução
    - Sem abstração de memória
    - Com Abstração de memória
  - Espaços de endereçamentos
  - Troca de processos
  - Gerenciando a memória livre
    - Mapas de bits
    - Listas encadeadas

Introdução

- Memória RAM - Recurso importante que deve ser cuidadosamente gerenciado
  - Computador pessoal médio tem 10.000 vezes mais memória que o maior computador do mundo da década de 1960!
    - Entretanto, os programas estão ficando maiores e mais rápidos do que as memórias!
    - **Lei de Parkinson:** *"Programas tendem a expandir-se a fim de preencher a memória disponível para contê-los"*
  - Estudaremos como o SO consegue criar uma abstração a partir da memória e como eles a gerenciam

How did game developers pack entire games into so little memory twenty five years ago?
https://qr.ae/pN614V

Introdução

- O que todo programador quer?
  - Memória privada, infinitamente grande e rápida, que fosse não volátil e fosse barata
    - Infelizmente ainda não conseguimos construir esse tipo de memória
- O que temos?
  - **Hierarquia de memória**
    - Registradores < cache multinível < memória RAM < Discos ou armazenamentos removíveis
  - É função do SO abstrair essa hierarquia em um modelo útil e gerenciar essa abstração

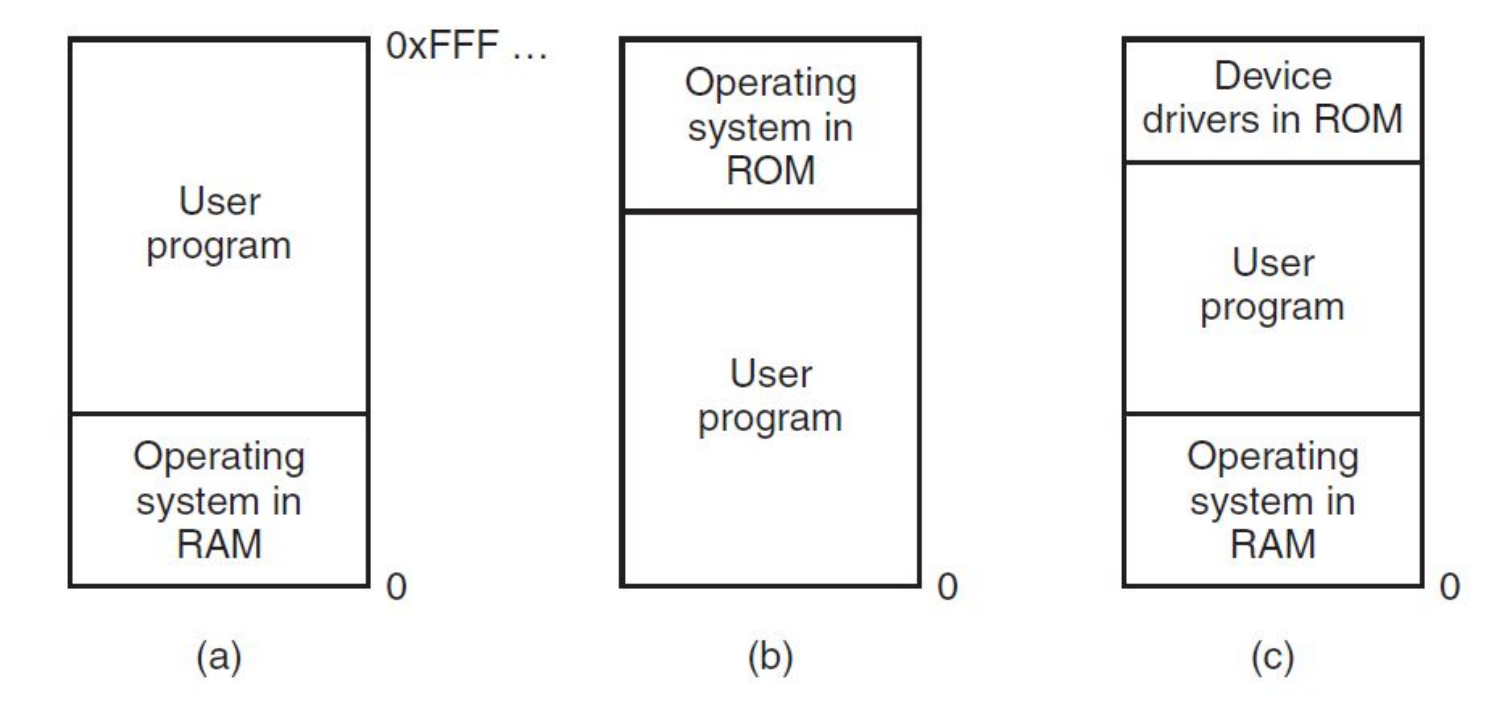
Introdução

- **Gerenciador de memória**
  - Parte do SO que gerencia (parte da) hierarquia da memória
  - Deve gerenciar eficientemente a memória:
    - Controlar quais partes estão sendo usadas
    - Alocar memória para processos quando eles precisam
    - Liberar memória quando algum processo tiver terminado
- Foco do nosso estudo
  - Investigar modelos de memória principal do programador e como ela pode ser gerenciada (e simplificada - **abstração**)

Modelo sem abstração de memória

- **Abstração de memória mais simples:** não ter abstração nenhuma
  - Primeiros computadores de grande porte (1960), minicomputadores (1970) e computadores pessoais (1980) **não tinham abstração de memória**
    - Cada programa enxergava a memória física!
  - A instrução **LW $s0, 1024($zero)**
    - Carrega o conteúdo da memória física da posição 1024 para $s0
  - Modelo de memória apresentado ao programador era apenas a **memória física**
    - Conjunto de endereços de 0 a algum máximo, onde cada endereço correspondia a uma célula contendo algum número de bits

Modelo sem abstração de memória



Três maneiras de organizar a memória: (a) SO na parte inferior da RAM; (b) SO no topo em memória não volátil ROM; (c) Drivers de dispositivos no topo da memória em uma ROM e o SO na parte inferior

Modelo sem abstração de memória

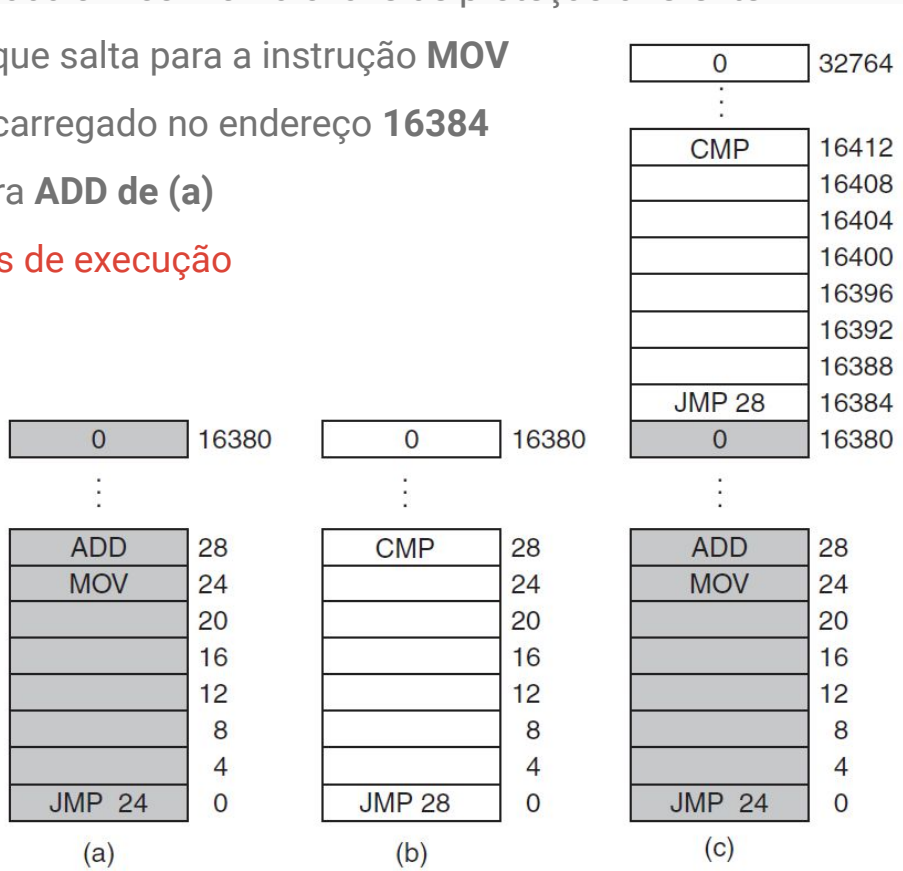
- **Note que não é possível executar dois programas ao mesmo tempo**
  - Tão logo o usuário digita um comando, o SO copia o programa solicitado do disco para a memória e o executa
  - Quando o processo termina, o SO exibe o prompt de comando e espera um novo comando do usuário
- Caso o SO esteja em RAM, é possível que um processo apague por completo um sistema operacional em execução, possivelmente com resultados desastrosos
- **Paralelismo é possível**
  - Multithread, mas possivelmente nenhum “sistema tão primitivo” a ponto de não proporcionar abstração de memória irá proporcionar abstração de threads...

Múltiplos programas sem uma abstração de memória

- **Múltiplos programas sem uma abstração de memória é possível?**
  - **Swapping (troca de processos)**
    - O SO salva o conteúdo inteiro da memória em um arquivo de disco e, então, introduz e executa o programa seguinte
    - Se houver apenas um processo em memória, não há conflitos
  - **Usando um Hardware adicional para executar múltiplos processos sem *swapping***
    - **Solução para os modelos da IBM 360 (1964)**
      - Memória dividida em blocos de 2KB
      - Para cada bloco é designada uma **chave de proteção** de 4 bits
      - Hardware impedia qualquer tentativa de um processo em execução de acessar a memória com um código de proteção diferente da chave armazenada em um registrador especial, o **PSW (Palavra de estado do programa)**

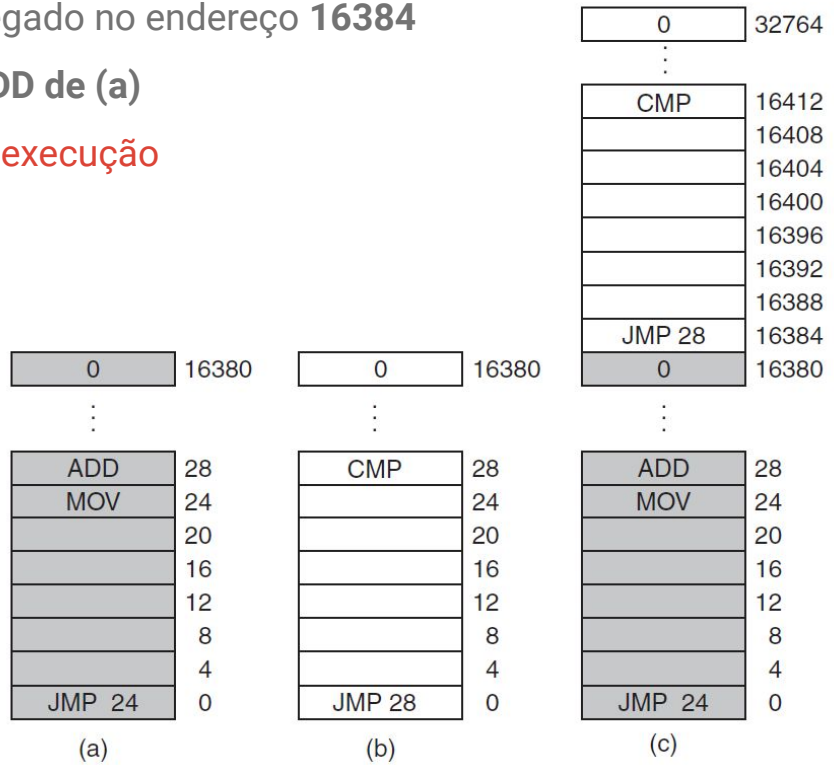
Múltiplos programas sem uma abstração de memória

- **Seja 2 programas com 16KB de tamanho (a) e (b),** cada um com uma chave de proteção diferente
  1. (a) inicializa, executando a instrução **JMP 24** que salta para a instrução **MOV**
  2. Após algum tempo, o SO decide executar (b), carregado no endereço **16384**
     - A primeira instrução é **JMP 28**, que salta para **ADD de (a)**
     - Programa entra em colapso bem antes de 1s de execução
- **Alguma solução?**



Múltiplos programas sem uma abstração de memória

- **Seja 2 programas com 16KB de tamanho (a) e (b),** cada um com uma chave de proteção diferente
  1. (a) inicializa, executando a instrução **JMP 24** que salta para a instrução **MOV**
  2. Após algum tempo, o SO decide executar (b), carregado no endereço **16384**
     - A primeira instrução é **JMP 28**, que salta para **ADD de (a)**
     - Programa entra em colapso bem antes de 1s de execução
- **Solução temporária**
  - Modificar o segundo programa dinamicamente
  - **Realocação estática**
    - A constante 16384 é acrescentada a cada endereço de programa durante seu carregamento (lento)
  - **Outro problema:** ambiguidade entre endereços e constantes (imediatos) na linguagem de máquina



Espaços de Endereçamento

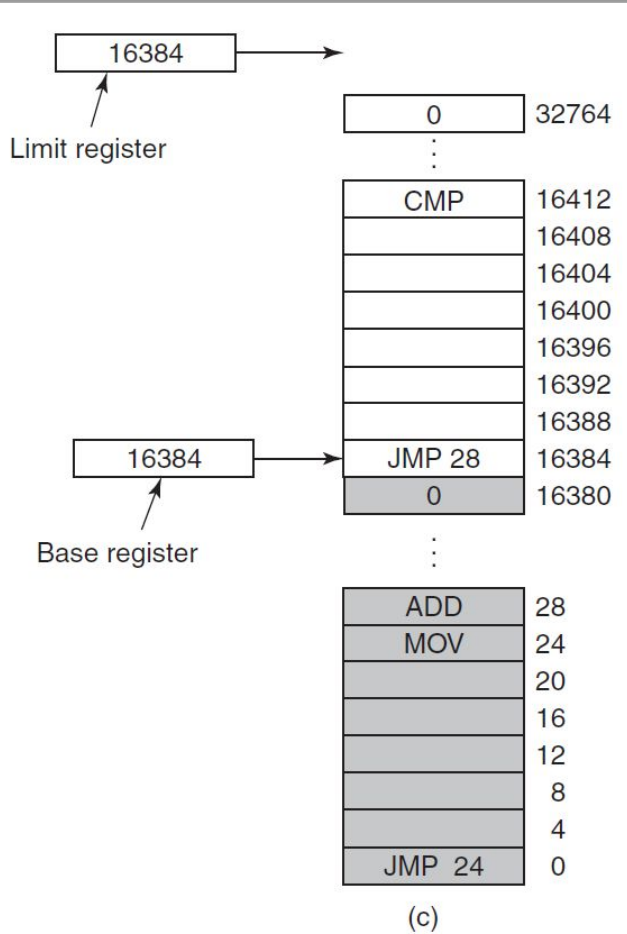
- 2 Problemas a serem solucionados no gerenciamento de memória
  - **Proteção e Realocação**
    - IBM 360 soluciona apenas a primeira, rotulando blocos de memória com as chaves de proteção
    - Uma solução melhor é inventar uma nova abstração de memória: **o espaço de endereçamento**
      - Conjunto de endereços que um processo pode usar para endereçar a memória
      - Cada processo tem o seu próprio espaço de endereçamento, independente daqueles que pertencem a outros processos

Registradores base e limite

- Como dar a cada programa seu próprio espaço de endereçamento?
  - Endereço 28 de um programa é uma localização física diferente do endereço 28 de outro programa
  - **Possível solução: registradores base e registradores limite**
    - **Realocação dinâmica**
      - Mapeia o espaço de endereçamento de cada processo em uma parte diferente da memória física
    - É equipada em cada CPU dois registradores especiais, chamados **registradores base e registradores limite**
      - Programas são carregados em posições de memórias consecutivas sempre que haja espaço e sem realocação durante o carregamento

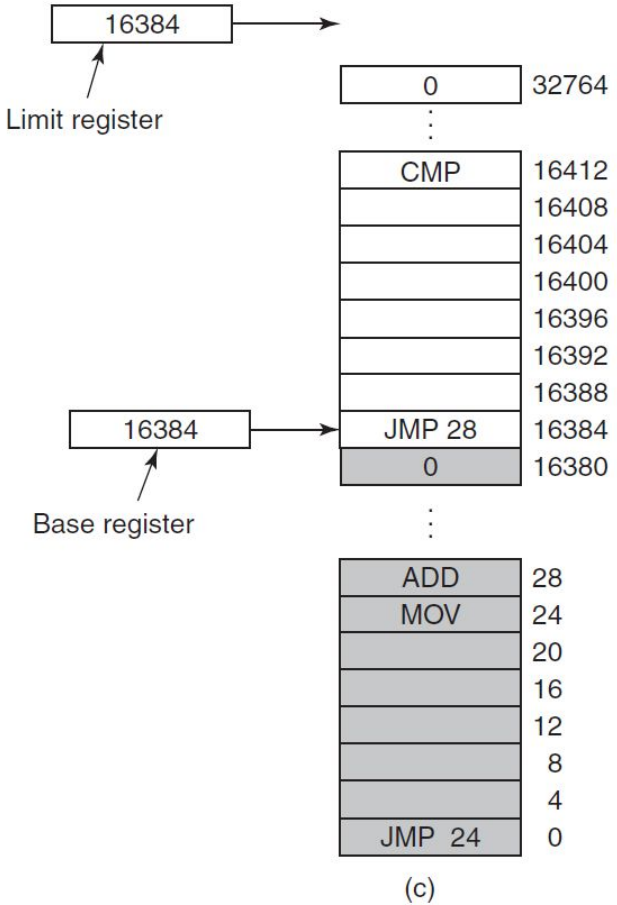
Registradores base e limite

- Em toda referência de memória, **o hardware da CPU adiciona o valor base ao endereço gerado** pelo processo antes de enviá-lo para o barramento de memória
- Ao mesmo tempo, confere se o **endereço fornecido é igual ou maior do que o registrador limite**
  - Caso o endereço não esteja na **faixa [base, limite],** o acesso é abortado
- Existe uma forma de acessar indevidamente outro espaço de endereçamento?
- Qual a desvantagem deste método?



Registradores base e limite

- Em toda referência de memória, **o hardware da CPU adiciona o valor base ao endereço gerado** pelo processo antes de enviá-lo para o barramento de memória
- Ao mesmo tempo, confere se o **endereço fornecido é igual ou maior do que o registrador limite**
    - Caso o endereço não esteja na **faixa [base, limite],** o acesso é abortado
- Existe uma forma de acessar indevidamente outro espaço de endereçamento?
- Qual a desvantagem deste método?
    - Necessidade de realizar uma adição e uma comparação em cada referência de memória.



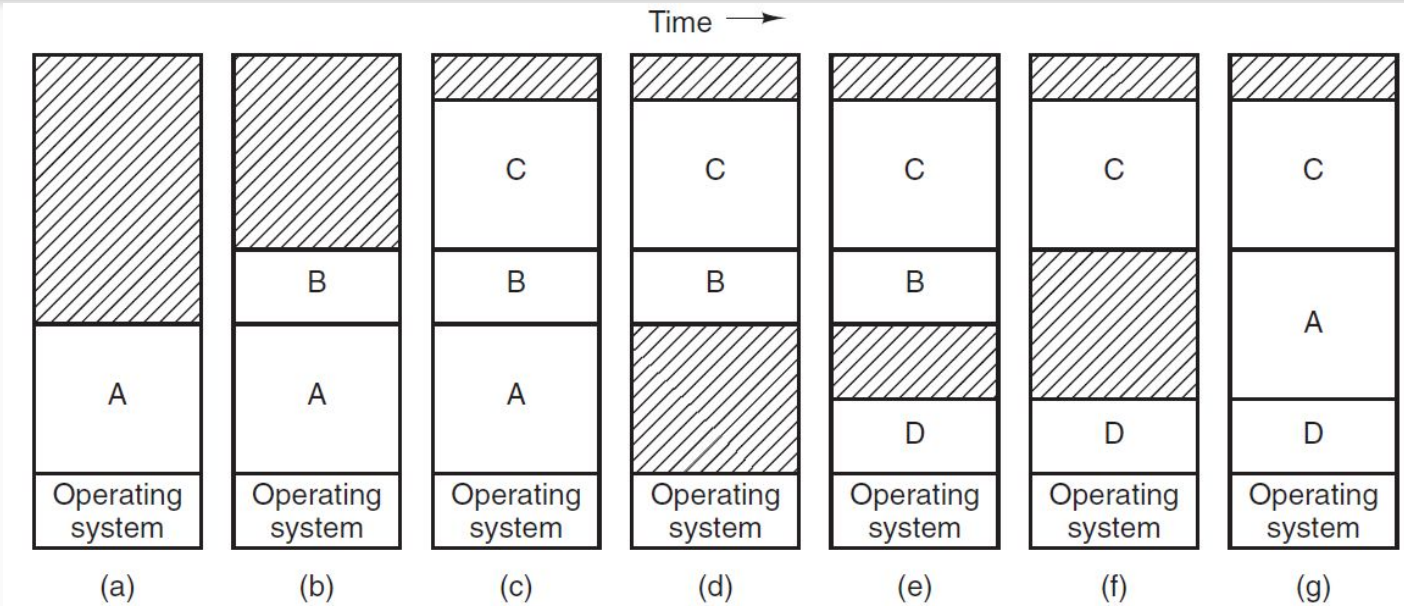
Troca de Processos (Swapping)

- O Total de RAM demandado por todos processos geralmente é muitas vezes maior do que pode ser colocado em memória
  - Manter todos os processos na memória o tempo inteiro exige uma quantidade de memória que muitas vezes não é disponível!
  - Duas abordagens para lidar com sobrecarga de memória:
    - **Swapping**
    - **Memória virtual**

Troca de Processos (Swapping)

- **Swapping (Troca de processos)**
  - Consiste em trazer cada processo em sua totalidade, executá-lo por um tempo, e então colocá-lo de volta no disco
- **Memória virtual**
  - Permite que os programas possam ser executados mesmo quando estão apenas parcialmente na memória principal

Mudanças na alocação de memória à medida que processos entram e saem dela

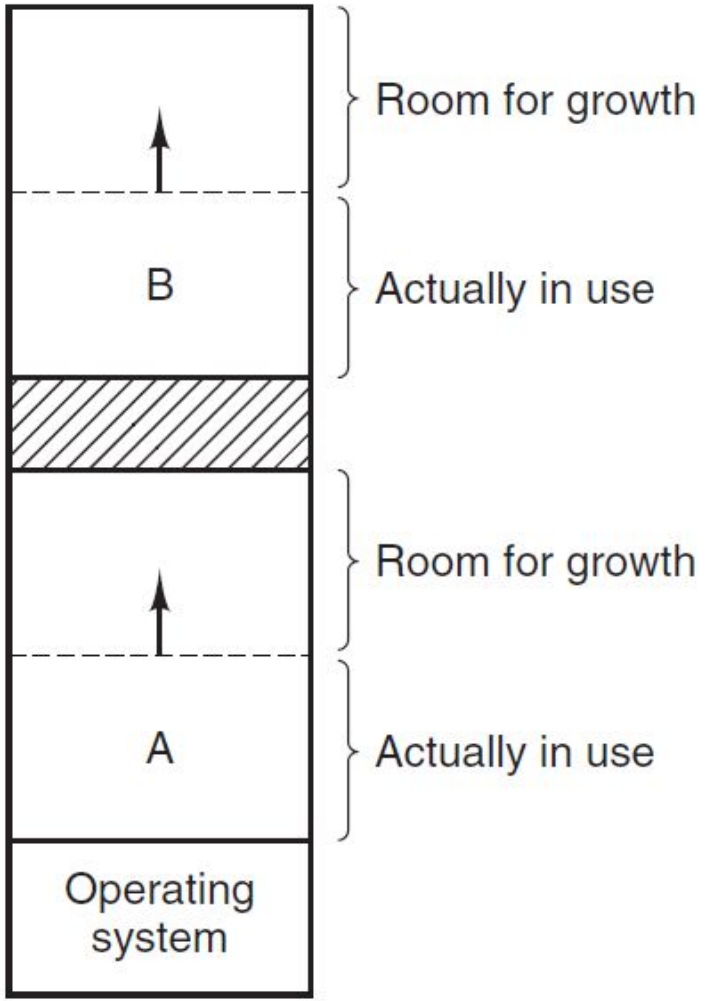


Trocas de processos criam múltiplos espaços na memória (fragmentação externa), sendo possível combiná-los em um grande espaço, movendo todos os processos “para baixo” o máximo possível. Essa técnica é conhecida como compactação de memória (muito lenta)

Troca de Processos (Swapping)

- Quanta memória deve ser alocada para um processo quando ele é criado ou trocado?
  - **Alocação simples (Tamanho fixo)**
  - **Alocação dinâmica**
    - Segmento de dados dos processos podem crescer, sendo necessário a alocação dinâmica de uma área de memória temporária
    - **Se o processo for adjacente a outro**
      - o que cresce deve ser movido para um espaço de memória grande o suficiente
    - **Caso o processo não puder crescer em memória e a área de troca no disco estiver cheia**
      - ele terá de ser suspenso, até que algum espaço seja liberado (ou ele pode ser morto...)
    - **Ideia:** alocar um pouco de memória extra sempre que o processo for trocado ou movido

Alocação de espaço para um segmento de dados em expansão

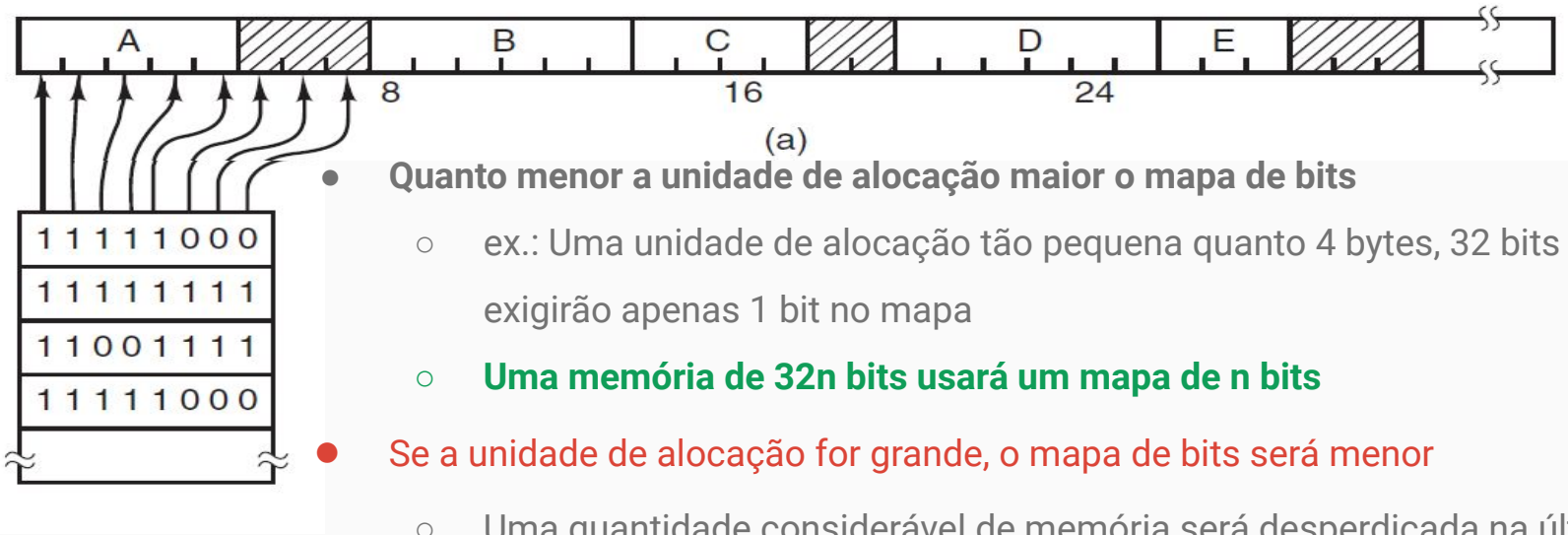


Gerenciando a memória livre

- **Dois modos de rastrear o uso de memória**
  - Mapa de Bits
  - Listas livres

Gerenciamento de memória com mapa de bits

- Mapa de bits
  - A memória é dividida em unidades de alocação de tamanho fixo
  - Cada unidade de alocação corresponde a 1 bit no mapa de bits:



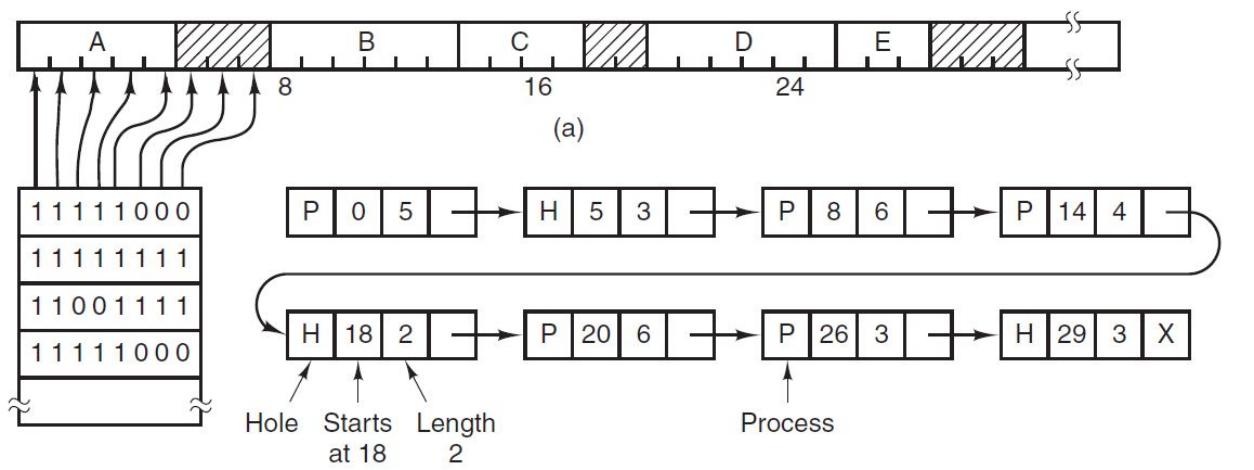
- **Quanto menor a unidade de alocação maior o mapa de bits**
  - ex.: Uma unidade de alocação tão pequena quanto 4 bytes, 32 bits de memória exigirão apenas 1 bit no mapa
  - **Uma memória de 32n bits usará um mapa de n bits**
- Se a unidade de alocação for grande, o mapa de bits será menor
  - Uma quantidade considerável de memória será desperdiçada na última unidade do processo se ele não for múltiplo exato da unidade de alocação (**Fragmentação interna**)

Gerenciamento de memória com mapa de bits

- Principal problema nos mapas de bits
  - Quando fica decidido carregar um processo com tamanho de ***k* unidades**, o gerenciador de memória deve procurar o mapa de bits para encontrar uma sequência de **zeros k bits consecutivos**
    - Operação lenta!

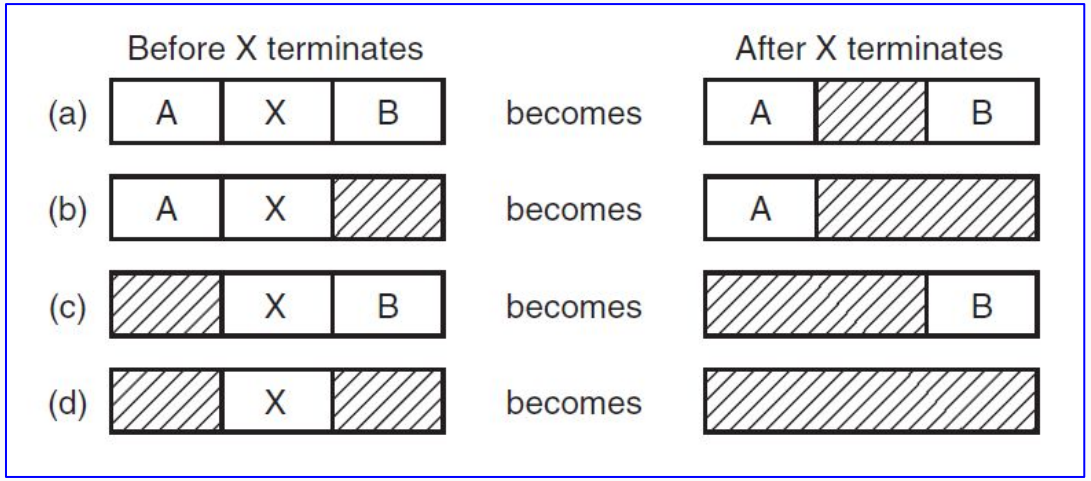
Gerenciamento de memória com listas encadeadas

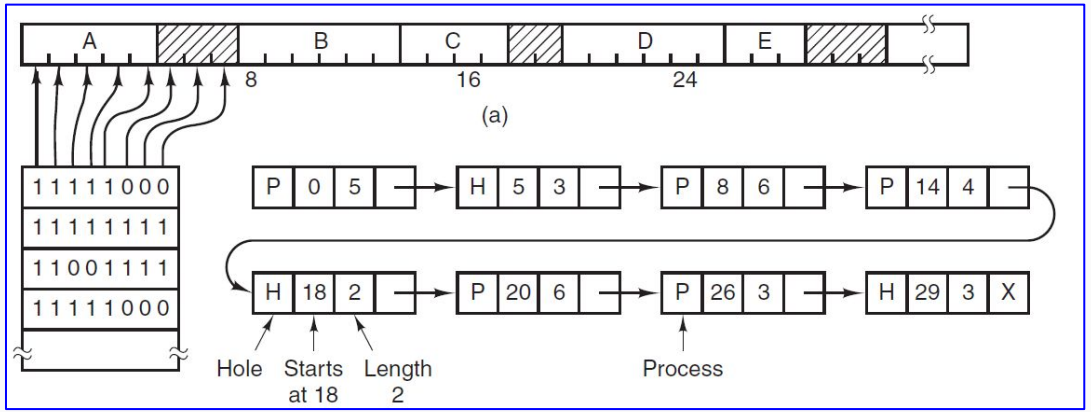
- **Lista encadeada de espaços livres e de segmentos de memória alocados a processos**
  - Cada entrada na lista especifica se é um **espaço livre (H ou L)** ou um **espaço alocado a um processo (P) (1º termo)**
  - **Endereço** no qual se inicia esse segmento **(2° termo)**
  - **Comprimento e um ponteiro** para o item seguinte **(3º e 4º termo)**



Gerenciamento de memória com listas encadeadas

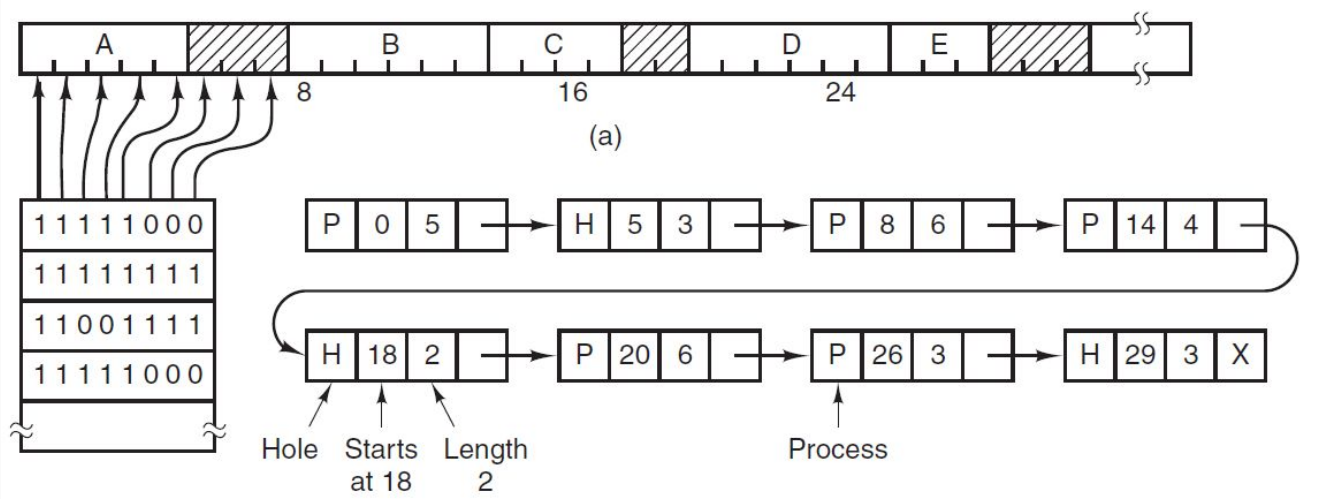
- **Nesse exemplo a lista é mantida ordenada pelos endereços**
  - **Vantagem**: quando um processo é terminado ou transferido atualizar a lista é algo simples de se fazer
    - **Quatro possíveis combinações de espaços livres após o término de um processo**
- Pode-se usar listas duplamente encadeadas para facilitar a manipulação da Estrutura de dados





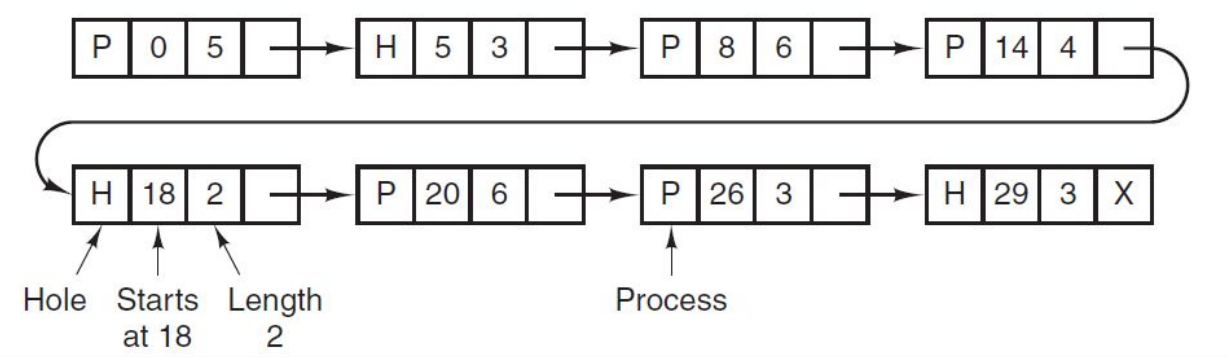
Gerenciamento de memória com listas encadeadas

- **Vários algoritmos podem ser usados para alocar memória a um processo criado**
  - First Fit
  - Next Fit
  - Best Fit
  - Worst Fit
  - Quick Fit



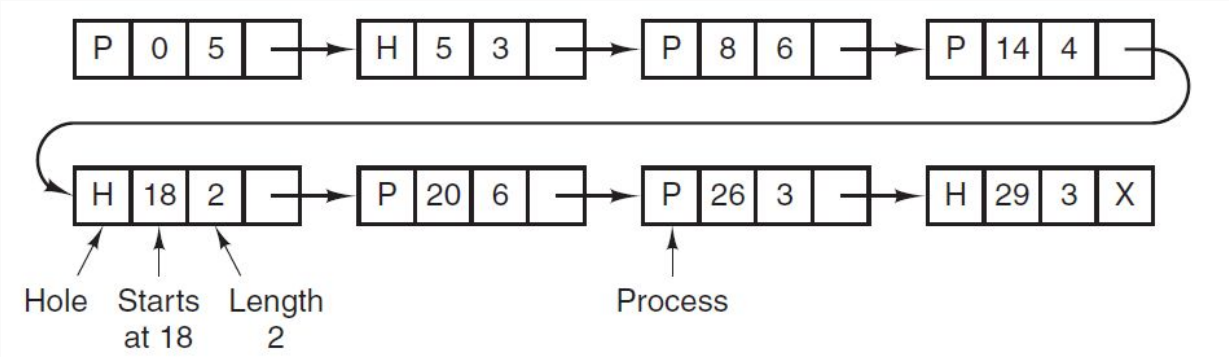
First Fit

- Algoritmo mais simples (**primeiro encaixe**)
  - O gerenciador de memória examina a lista de segmentos até encontrar um espaço livre que seja grande o suficiente
  - O espaço livre é então dividido em duas partes:
    - uma para o processo
    - outra para a memória não utilizada
  - First fit é um algoritmo rápido, pois procura fazer a menor busca possível!



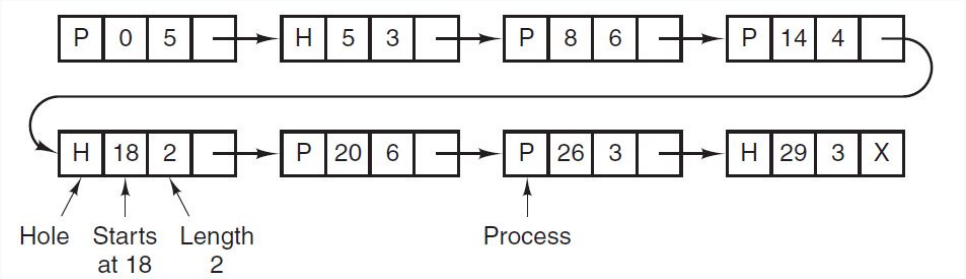
Next Fit

- Variação do First Fit
  - Funciona da mesma maneira que o **first fit,** exceto que memoriza a posição que se encontra um espaço livre adequado sempre que o encontra.
    - Da vez seguinte que for chamado para encontrar um espaço livre, ele começa **procurando na lista do ponto de onde havia parado**, em vez de sempre do princípio (como é o caso do first fit)
  - Simulações mostram que o **next fit** tem um desempenho ligeiramente pior que o *first fit*



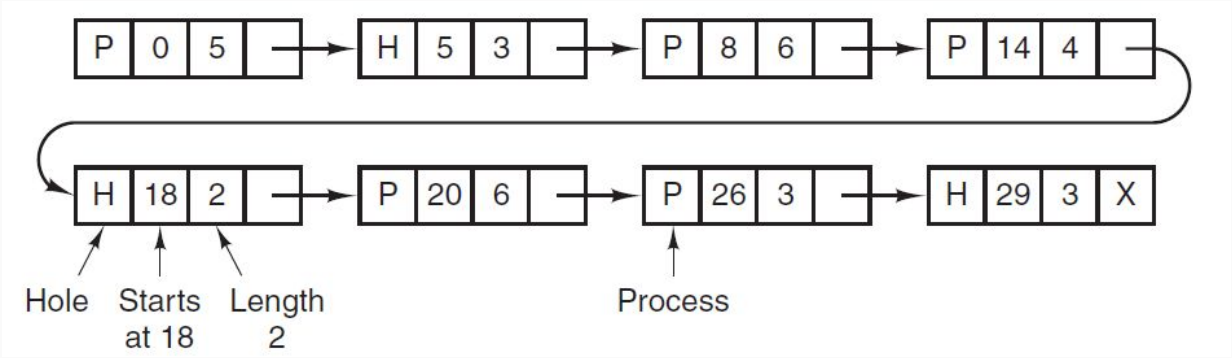
Best Fit

- Faz uma busca em toda a lista, do início ao fim
  - Escolhe o menor espaço livre que seja adequado
  - Tenta encontrar um espaço que seja de um tamanho próximo do tamanho real necessário, para casar da melhor maneira possível a solicitação com os segmentos disponíveis
  - É mais lento que o **first fit**
  - Gera um Desperdício maior de memória que o **first fit ou next first**, pois tende a preencher a memória com segmentos minúsculos e inúteis (fragmentação externa)
  - first fit gera espaços livres maiores em média



Worst Fit

- Sempre escolhe o maior espaço livre, de modo que o novo segmento livre seja grande o bastante para ser útil
  - Simulações demonstraram que o worst fit não é uma boa ideia



Gerenciamento de memória com listas encadeadas

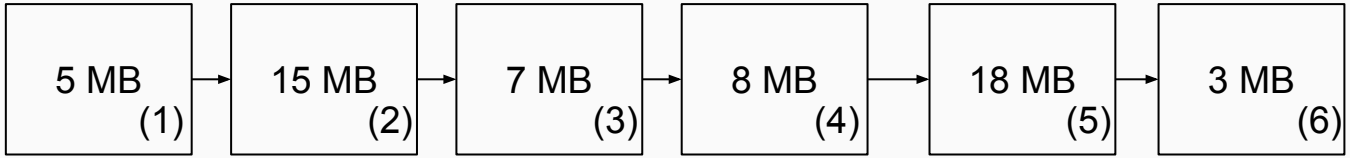
- De modo geral, todos os 4 algoritmos podem ser acelerados
  - Mantendo-se listas separadas para processos e espaços livres
  - Dessa forma os algoritmos devotam toda sua energia para **inspecionar espaços livres, não processos**
    - Preço a se pagar: complexidade e lentidão ao remover a memória, pois é necessário remover da lista de processos e incluir na de espaços livres
  - Listas de espaço livres podem ser mantidas ordenadas por tamanho
    - Best fit e first fit ficam igualmente rápidos (next fit fica irrelevante)

Quick Fit

- Mantém listas em separado para alguns dos tamanhos mais comuns solicitados
  - Ex.: pode ter uma tabela com n entradas, na qual a primeira é um ponteiro para início de uma lista de espaços livres de 4 KB, a segunda apontando para uma lista de espaços livres de 8KB, e assim por diante.
  - Encontrar um espaço livre do tamanho exigido é algo extremamente rápido
  - Problema: Preço a se pagar ao desalocar áreas de memória...

Exercício

Considere um sistema de troca no qual a memória consiste nos seguintes tamanhos de lacunas na ordem da memória:



Qual lacuna é pega para sucessivas solicitações de segmentos 7 MB, 8 MB, 1 MB, 17 MB, 4 MB e 2 MB de para os seguintes métodos:

- Primeiro encaixe (First Fit);
- Melhor encaixe (Best Fit);
- Pior encaixe (Worst Fit);
- Próximo encaixe (next Fit).

Obs.: Caso não seja possível alocar alguma solicitação de segmentos, explique por que isso aconteceu.

Próxima aula

- Na próxima aula:
  - Memória Virtual
  - Algoritmos de substituição de páginas

Referências

Tanenbaum, A. S. e Bos, H.. Sistemas Operacionais Modernos. 4.ed. Pearson/Prentice-Hall. 2016.