Instituto de Informática - UFRGS

Sistemas Operacionais

Gerência de Memória

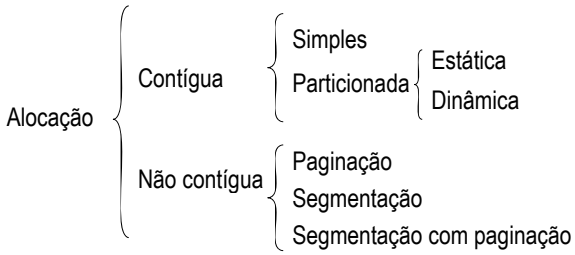
Alocação Particionada

Aula 12

Introdução

- Memória é um recurso limitado
  - Um programa (processo) para ser executado deve estar na memória
- Problemas:
  - Tamanho do programa excede a capacidade de memória
  - Vários processos devem compartilhar a memória (multiprogramação)
- Necessidades do gerenciamento de memória
  - Racionalizar a ocupação da memória (alocação de memória)
  - Determinação de áreas livres e ocupadas
- A alocação de memória depende:
  - Amarração de endereços é estática ou dinâmica
  - Necessidade de espaço contíguo ou não

Mecanismos para alocação de memória



Alocação contígua simples

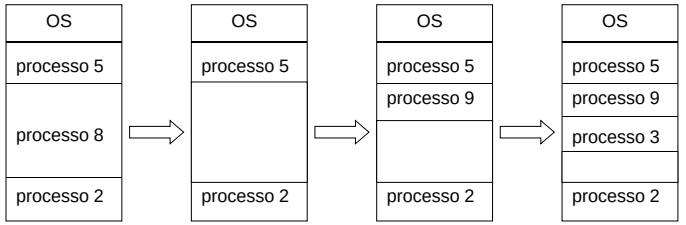
- Sistema mais simples
- Memória principal é dividida em duas partições:
  - Sistema operacional (parte baixa/alta da memória)
  - Processo do usuário (restante da memória)
- Usuário tem controle total da memória podendo inclusive acessar a área do sistema operacional
  - e.g. DOS (não confiável)
- Evolução:
  - Inserir proteção através de mecanismos de *hardware* + *software*
    - Registradores de base e de limite
    - *Memory Management Unit* (MMU)

Alocação contígua particionada

- Extensão natural do sistema de duas partições
  - Definição de múltiplas partições
- Necessidade imposta pela multiprogramação
- Filosofia:
  - Dividir a memória em blocos (partições)
  - Cada partição pode receber um processo (programa em execução)
  - Grau de multiprogramação é limitado pelo número de partições
- Duas formas básicas:
  - Alocação contígua com partições fixa (estática)
  - Alocação contígua com partições variáveis (dinâmica)

Alocação contígua particionada (cont.)

- O sistema operacional é responsável pelo controle das partições mantendo informações como:
  - Partições alocadas
  - Partições livres
  - Tamanho das partições

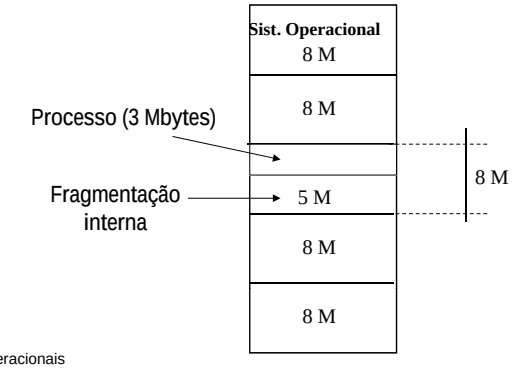


Alocação contígua particionada fixa

- Memória é dividida em partições de tamanho fixo:
  - Definido na inicialização do sistema (não muda com o tempo)
    - Difícil estimar e prever a real necessidade do sistema
  - Tamanhos idênticos ou não
- Questões:
  - Processos podem ser carregados em qualquer partição?
    - Depende se código é absoluto ou relocável e do tipo de amarração
  - Como tratar área de *heap* e pilha
    - Pré-alocação de uma área para ser compartilhada entre elas
  - Número de processos que podem estar em execução ao mesmo tempo
  - Programa é maior que o tamanho da partição
    - Não executa a menos que se empregue um esquema de *overlay*

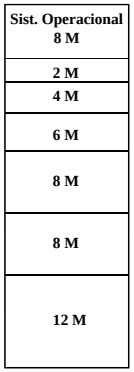
Fragmentação Interna

- Problema da alocação fixa é uso ineficiente da memória principal
- Um processo, não importando quão pequeno seja, ocupa uma partição inteira
  - Fragmentação interna



Paliativo para reduzir fragmentação interna

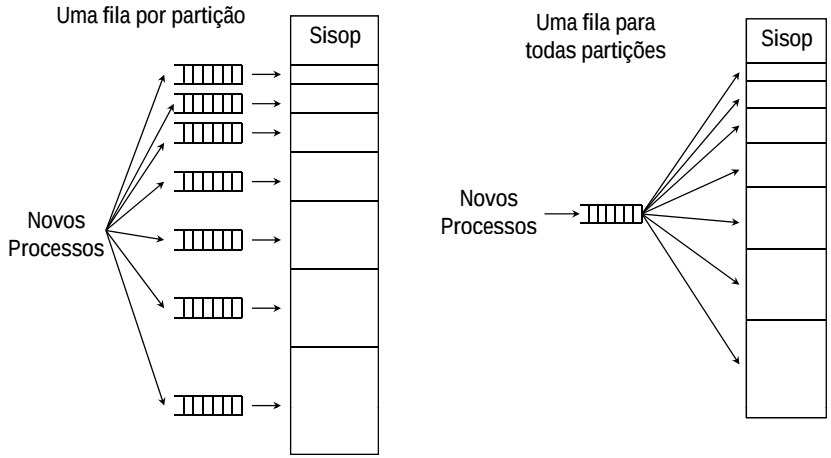
- Partições de tamanho diferentes
  - Processo é alocado para a menor partição possível que satisfaça seus requisitos de memória



Partições fixas e amarração

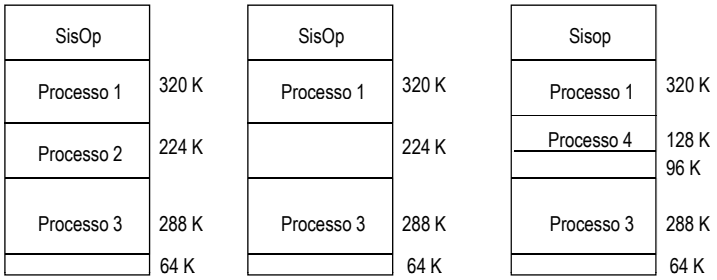
- Com código absoluto sem amarração dinâmica
  - Um processo só pode ser carregado em uma determinada partição
  - Pode haver disputa por uma partição mesmo tendo outras livres
    - Processo é mantido no escalonador de longo prazo (termo)
    - Empregar *swapping* (escalonamento de médio prazo)
- Com códido absoluto com amarração dinâmica
  - Processo pode ser carregado em qualquer partição de maior ou igual tamanho
  - Se todas as partições estão ocupadas, duas soluções:
    - Processo é mantido no escalonador de longo prazo (termo)
    - Empregar *swapping* (escalonamento a médio prazo)
- Código realocável após carregado apresenta mesmos problemas de um código absoluto sem amarração dinâmica

Gerenciamento de partições fixas



Alocação particionada dinâmica

- Objetivo é eliminar a fragmentação interna
- Processos alocam memória de acordo com suas necessidades
- Partições são em número e tamanho variáveis

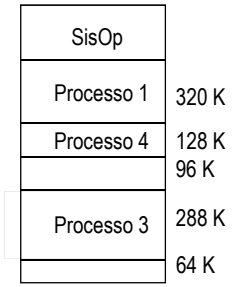


Fragmentação externa

- A execução de processos pode criar pedaços livres de memória
  - Pode haver memória disponível, mas não contígua
    - Fragmentação externa

Exemplo:

Criação processo 120K



Soluções possíveis fragmentação externa

- Concatenar espaços livres adjacentes em memória
  - De duas ou mais partições obter uma partição única
- Reunir partições livres para criar uma única área contígua
  - Técnica de compactação
    - Custo computacional elevado (processador e disco)
  - Acionado somente quando ocorre fragmentação
  - Possível apenas com código absoluto com amarração dinâmico

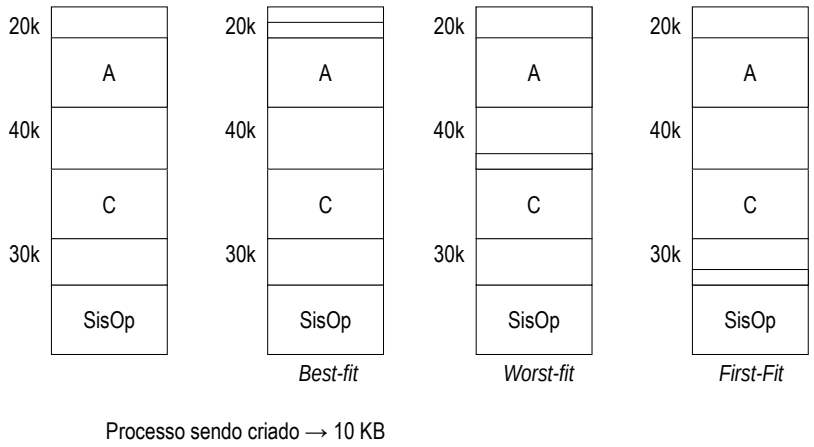
Gerenciamento de partições dinâmicas

- Determinar qual área de memória livre será alocada a um processo
- Sistema operacional mantém uma lista de lacunas
  - Pedaços de espaços livres em memória
- Necessidade de percorrer a lista de lacunas sempre que um processo é criado
  - Como percorrer essa lista??

Algoritmos para alocação contígua dinâmica

- *Best fit*
  - Minimizar tam_processo - tam_bloco
  - Deixar espaços livres os menores possíveis
- *Worst fit*
  - Maximizar tam_processo - tam_bloco
  - Deixar espaços livres os maiores possíveis
- *First fit*
  - tam_bloco > tam_processo
- *Circular fit*
  - Variação do *first-fit*

Exemplos: Algoritmos alocação contígua dinâmica



Processo sendo criado → 10 KB

Sistema *buddy*

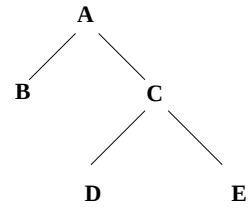
- Compromisso entre partição fixa e partição variável
- Memória é dividida em um certo número de blocos de espaços livre
  - Tamanho em potência de 2
  - Um bloco de tamanho 2^i é dividido em 2 blocos de tamanho 2^{i-1} (*buddies*)
- Custo "gerencial"
  - Necessário manter a lista de blocos livres e ocupados, concatenar buddies etc
- Exemplo de emprego:
  - Operações de E/S transferem diretamente dados de periféricos para memória via DMA, curto-circuitando a gerência de memória
  - Uma solução é alocar contiguamente uma área de memória para E/S

Swapping

- Obtenção de área de memória livre através do armazenamento temporário de um processo em disco (área de *swap*)
  - Candidatos potenciais são aqueles em estado bloqueado ou que já executaram demasiadamente no escalonador de curto prazo
- Operação de *swapping* é sobreposta ao processamento
  - E/S realizada através de DMA
- Procedimentos de *swap-in* e *swap-out*
  - Realizado por um processo especial (*swapper deamon*)
- Otimizações:
  - Copiar para o disco apenas as áreas que foram modificadas (dados e pilha)
  - Partição específica para o *swap*

Overlay

- Mecanismo que permite executar processos maiores que a capacidade total de memória ou de uma partição
- Princípio básico é dividir um processo em módulos
  - Um modelo substitui outro
  - Manter em memória apenas os módulos que necessitam residir simultaneamente
  - Necessário definir uma árvore de dependência



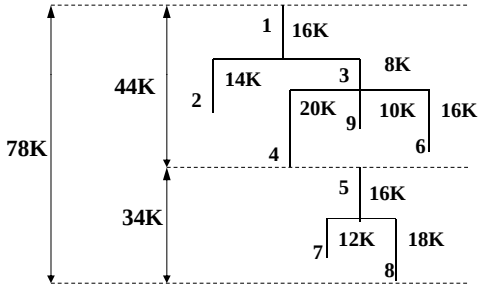
Interpretação:

- Módulo A e B devem estar em memória OU
- Módulo A e C devem estar em memória
  Neste caso, D OU E devem estar em memória

Exemplo: *overlay*

- Exemplo:
  - Programa 132K

1	2	3	4	5	6	7	8	9
16	14	8	20	16	16	12	18	12



- Espaço necessário 78K + supervisor de *overlay*

Leituras complementares

- A. Tanenbaum. *Sistemas Operacionais Modernos* (3ª edição), Pearson Brasil, 2010.
  - Capítulo 3: seção 3.2.3
- A. Silberchatz, P. Galvin; *Sistemas Operacionais*. (7ª edição). Campus, 2008.
  - Capítulo 8 (seção 8.3)
- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Bookman 4ª edição, 2010
  - Capítulo 6 (seção 6.2 e 6.4)