

Gerenciamento de memória

Swapping



Memória disponível

- Como visto na aula anterior, em um mundo ideal a memória seria grande o bastante para todos os processos em execução
- Mas o mundo ideal não existe
- Em um sistema desktop comum há mais memória usada que memória disponível



Consumo de memória

- Em um sistema desktop é comum que executem simultaneamente dezenas a centenas de processos
- Se cada processo consome parte da memória como um sistema com uma quantidade limitada de memória poderia executar tantos processos?
- Duas estratégias são comuns:
 - Swapping
 - Memória Virtual



Swapping

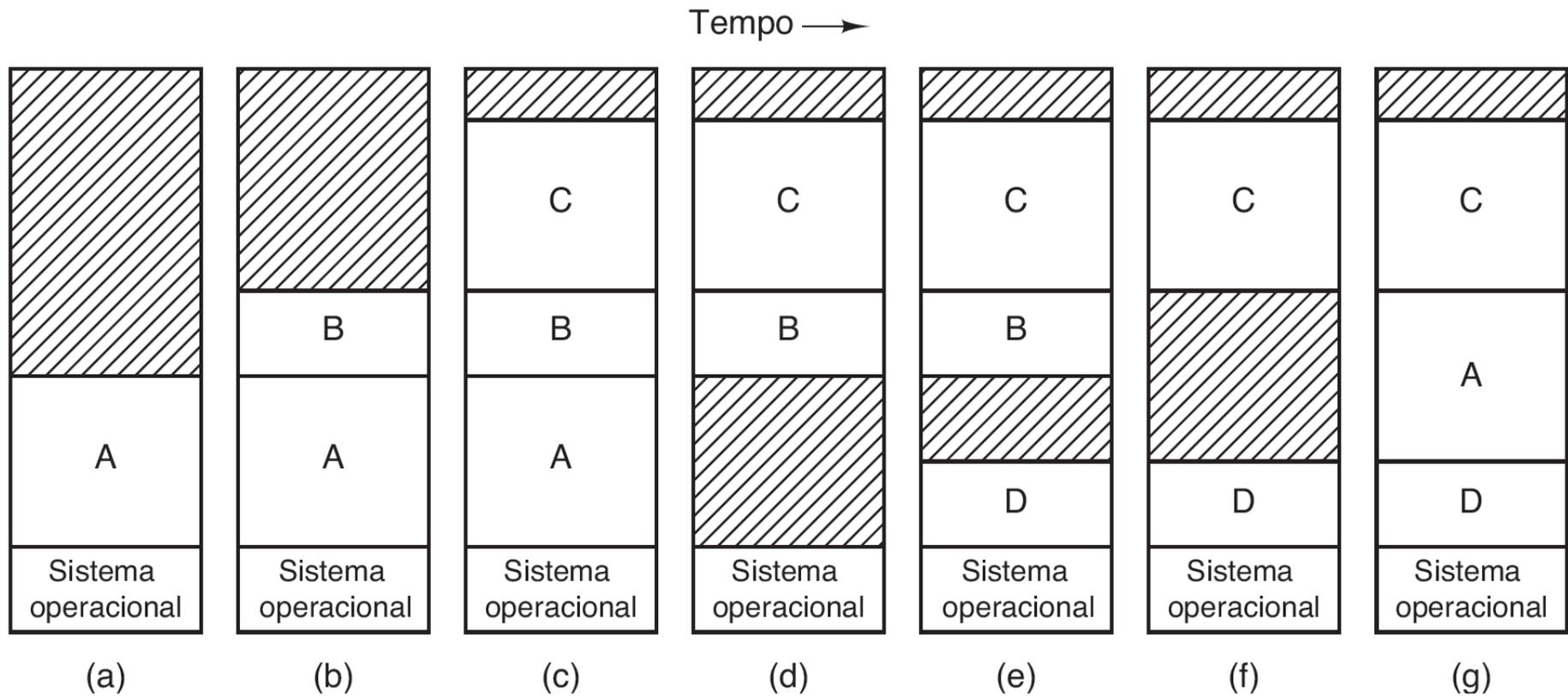
- A estratégia mais simples para executar vários processos que consomem acima do limite físico da memória RAM (também chamada memória principal)
- A ideia é simples:
 - **Carregue todos os processos que couberem na memória**
 - **Se faltar espaço retire um processo da memória principal (RAM) e copie seu conteúdo na memória secundária(HD/SSD)**
 - **Quando o processo for executar traga-o de volta da memória secundária(HD/SSD) para a primária(RAM)**

SWAP OUT

SWAP IN



Exemplo de Swap

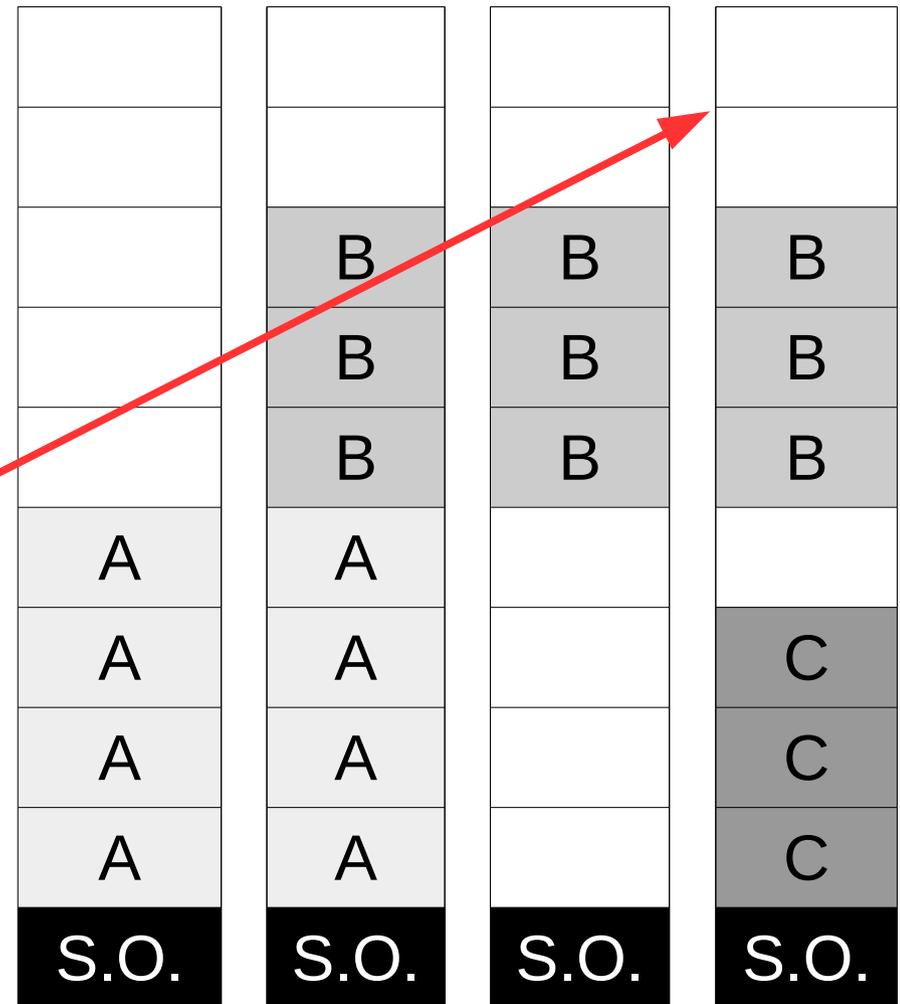


O problema da fragmentação

- Uma estratégia para resolver a fragmentação é a compactação da memória

A-4
B-3
C-3
D-3

Processo D que precisa de 3 espaços de memória não pode ser carregado

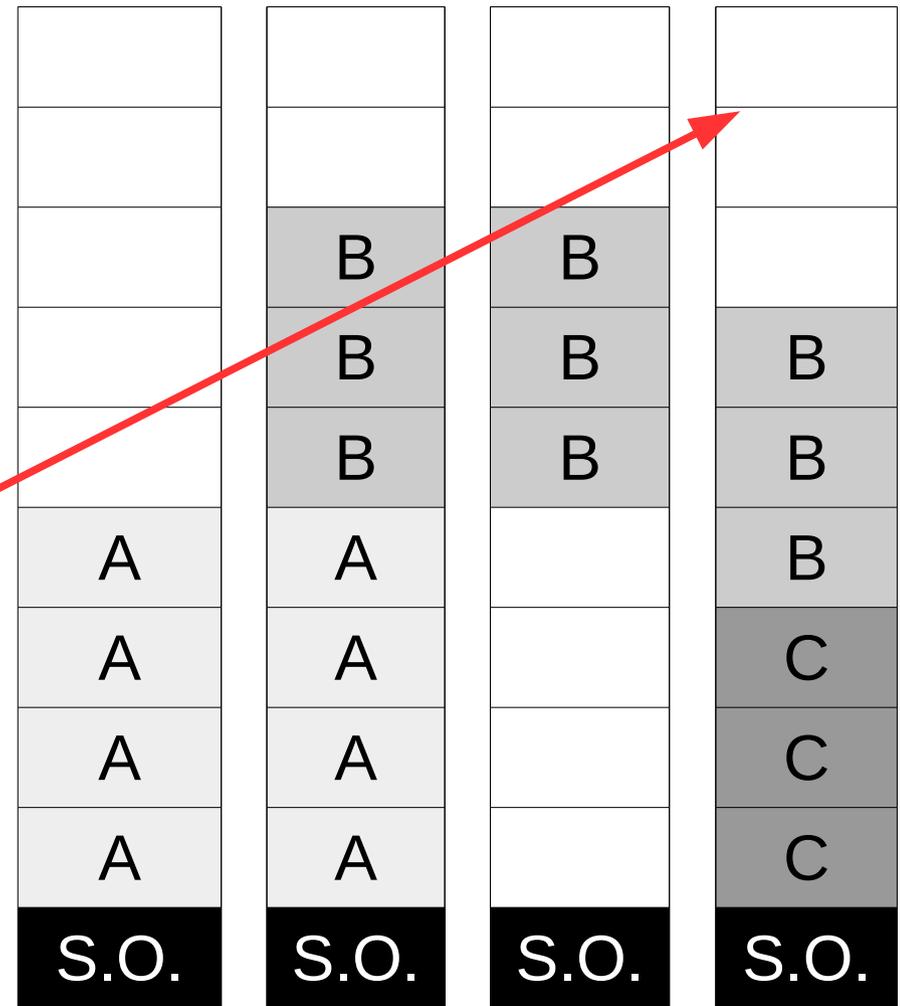


O problema da fragmentação

- Uma estratégia para resolver a fragmentação é a compactação da memória

A-4
B-3
C-3
D-3

Agora o Processo D que precisa de 3 espaços de memória pode ser carregado



Compactação da memória

- A estratégia é simples:
 - Mova cada processo para baixo na memória, fazendo com que ele ocupe os espaços vazios abaixo do seu endereço base
- Apesar de otimizar o uso da memória esta estratégia normalmente não é usada
- O tempo necessário para mover todos os processos para o fim da memória seria muito grande, comparado com o tempo de troca de contexto.



Alocação dinâmica

- Muitas linguagens permitem que o programador crie variáveis dinamicamente
- Para que isso funcione o processo precisará de mais espaço de memória do que foi alocado inicialmente
- A estratégia mais comum para implementar isto é usando uma pilha (stack)
- A medida que o processo solicita memória (cria novas variáveis) estas são alocadas em uma área pré-reservada para o crescimento do processo



A pilha

- Espaço de crescimento para o processo

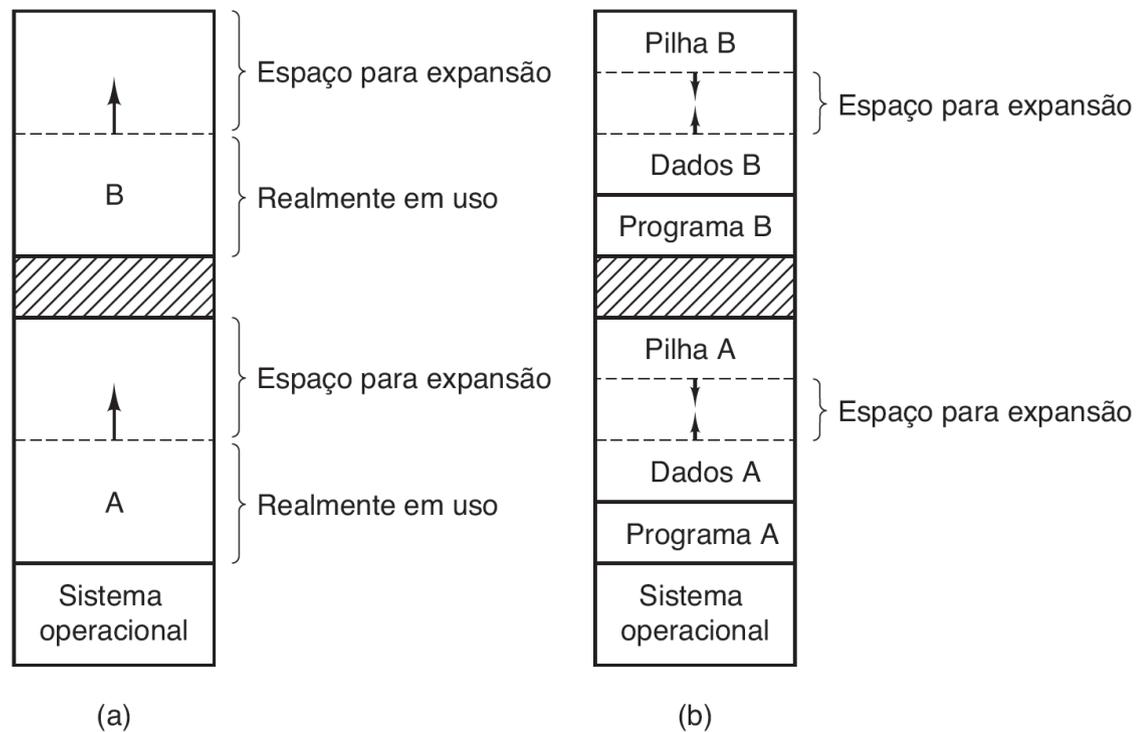


Figura 3.5 (a) Alocação de espaço para um segmento de dados em expansão. (b) Alocação de espaço para uma pilha e um segmento de dados em crescimento.

Alocação dinâmica

- O que acontece quando um processo excede o tamanho alocado da sua área de crescimento?
- Ele precisará ser realocado em uma área de memória com espaço contínuo que caiba o seu novo tamanho.
- Isso pode levar muito tempo, pois pode ser necessário fazer swap-out de outros processos e ainda mover os dados do processo para uma nova área de memória

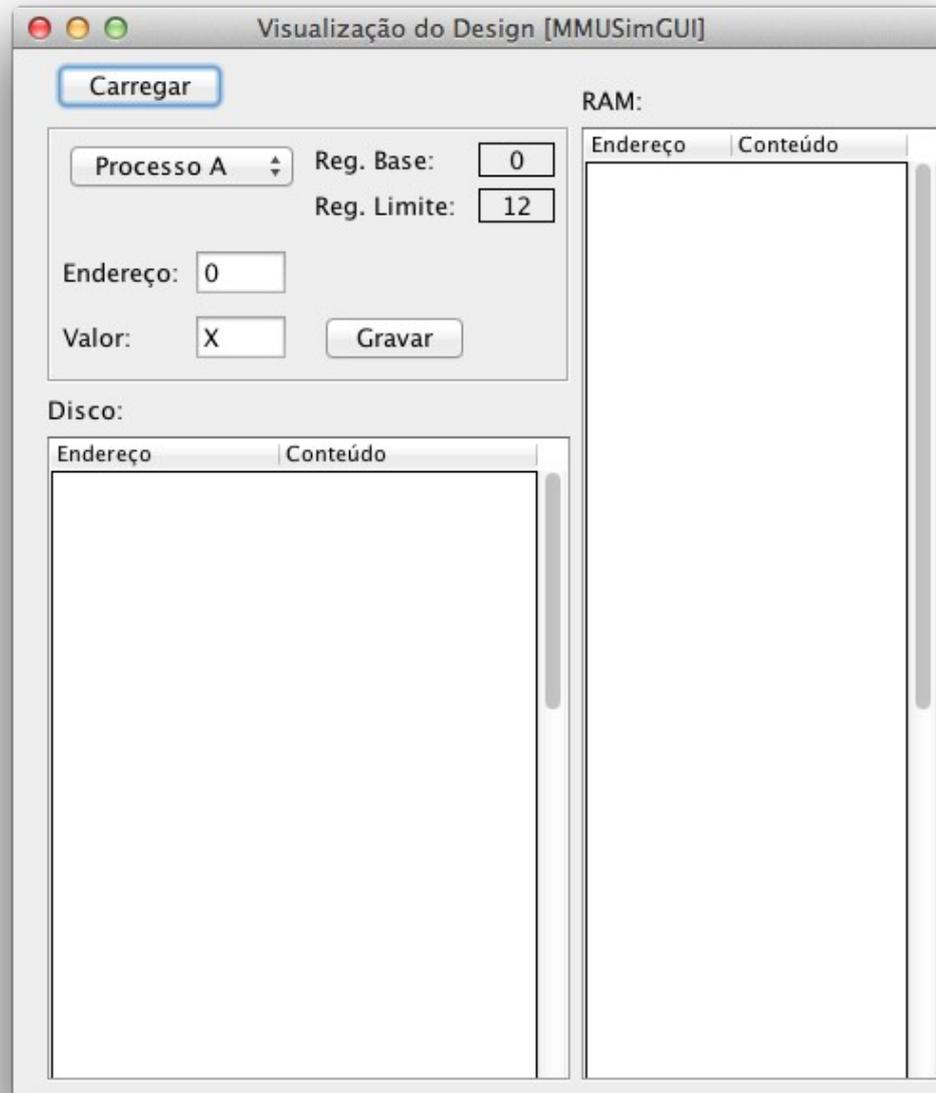


Atividade

- Utilizando o programa de alocação de memória da aula anterior crie um segundo vetor que será a memória secundária. Com 5 vezes o tamanho da memória principal.
- Altere o programa para que caso um processo ao ser carregado não caiba na memória principal seja feito swap-out de processos até que o novo processo caiba na memória principal.
- Se o usuário solicitar gravar em um processo que não esteja na memória principal, fazer o swap-in desse processo e realizar a operação gravar



Atividade



Endereço de entrega: <https://goo.gl/PFbjcv>

