MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 12 de Novembro de 2020

Roteiro

Abstração de memória e Swap

Gerência de espaço livre

Abstração de memória e Swap

Swap

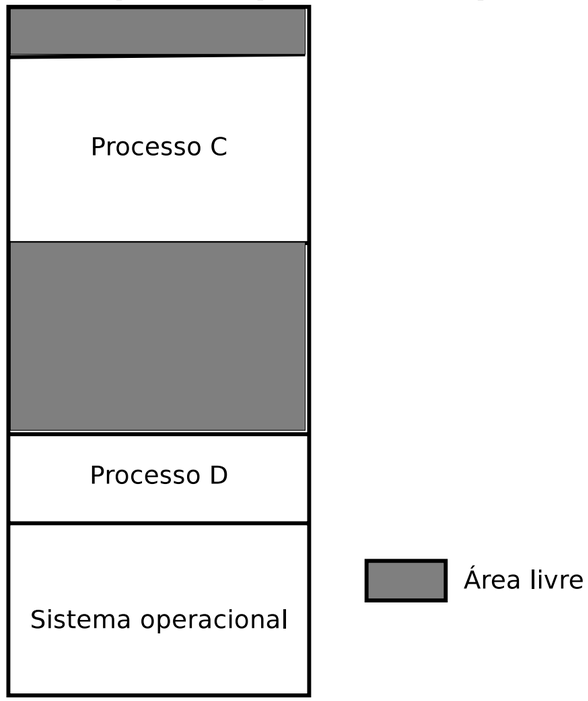
- Sem Swap: funciona se a memória física do computador for suficiente para comportar todos os processos (o que é raro)
- O mais realista é ter quantidades de memória dos processos (a soma) maior que a capacidade de RAM do computador
- Como lidar com a escassez da memória e o grande número de processos em execução simultânea? Swapping ou Memória virtual

Swap

- Mantém o processo no disco, traz ele para a memória quando é a vez dele executar, envia de volta para o disco após o quantum estourar (ou antes, se houver alguma operação de E/S).

Compactação de memória

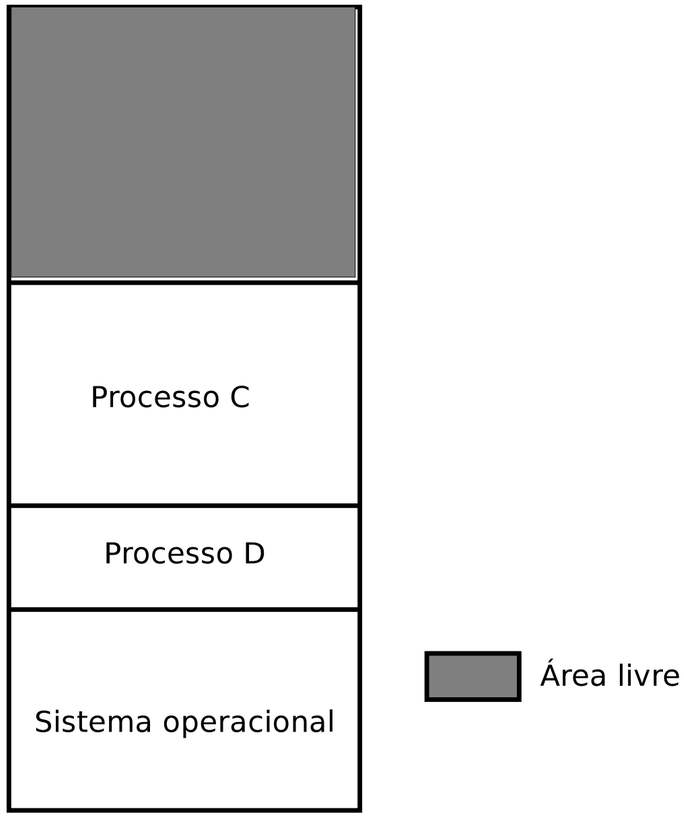
- Após alguma sequência de envios e remoções de processos do swap, suponha que o estado da memória é:



- Se um processo E que venha no futuro precisa de um espaço que seja a soma dos dois espaços livres acima ele não vai conseguir usar por causa da forma como os espaços estão

Compactação de memória

- Se o processo C fosse movido para o início, agora o processo E consegue ter acesso a um espaço contíguo da memória. Esse processo de mover os espaços de endereço para o início da memória chama-se **compactação de memória**



Compactação de memória

- Agora o processo E caberia após o processo C.
- Importante lembrar que a compactação exige que 1 ou mais processos sejam copiados dentro da memória e que os espaços de endereço desses processos (as bases e os limites) sejam atualizados nas estruturas de dados de cada processo no SO.

Gerência de espaço livre

Unidades de alocação

- Cada posição de memória (identificada por um endereço) tem um determinado espaço (por exemplo, 64 bits, 8 bits, etc...)
- Dificilmente um programa carregado na memória vai ocupar uma única posição
- Faz sentido fazer a alocação da memória pensando em espaços maiores do que uma única posição para tornar a gerência mais eficiente. A quantidade mínima de memória que pode ser alocada por um processo equivale a 1 **unidade de alocação**

Unidade de alocação

- Unidades de alocação maiores implicam que para ocupar o espaço exato da memória necessária, o processo precisa ter um tamanho que seja múltiplo do tamanho da unidade de alocação. Como consequência, isso leva a um desperdício de memória.
- Exemplo: Um processo precisa de 4 bytes mas a unidade de alocação é 3 bytes. Ele vai acabar usando 2 unidades de alocação (6 bytes) sendo que 2 desses bytes alocados não serão usados
- Unidades de alocação maiores entretanto reduzem o overhead de espaço necessário para manipular a memória pois com menos bits é possível mapear a memória inteira.

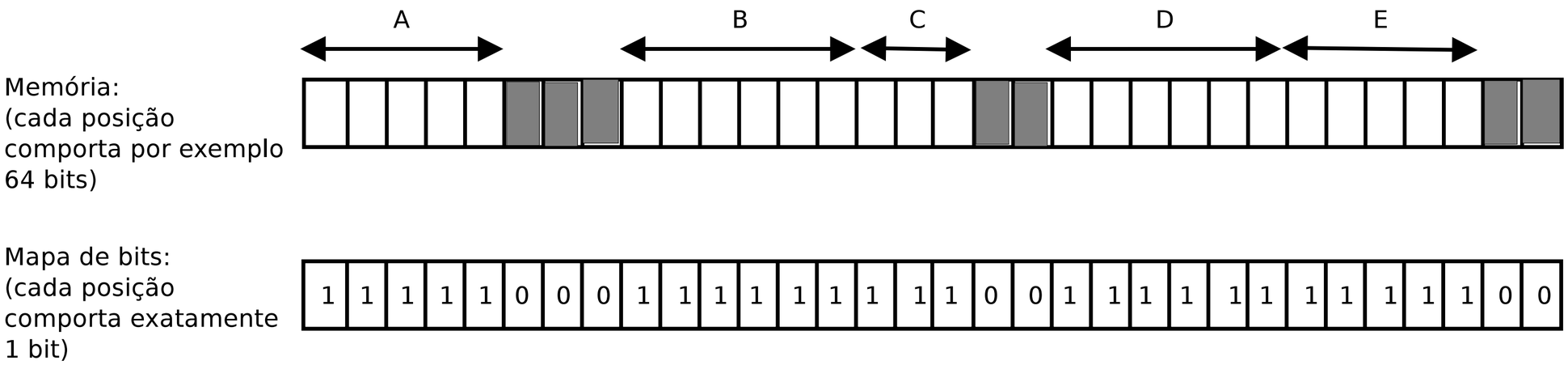
Unidade de alocação

- Unidades de alocação menores implicam que fica mais fácil ocupar o espaço exato da memória necessária, reduzindo o desperdício de memória.
- Exemplo: Um processo precisa de 4 bytes mas a unidade de alocação é 2 bytes. Ele vai acabar usando 2 unidades de alocação (4 bytes) sem gastar nada a mais. Mesmo que o valor não fosse par, apenas 1 byte seria desperdiçado
- Unidades de alocação menores entretanto aumentam o overhead de espaço necessário para manipular a memória pois serão necessários mais bits para mapear a memória inteira.

Mapa de bits

- Mapa de bits é o primeiro método que pode ser usado para gerenciar o espaço livre da memória
- A ideia aqui é ter um conjunto de bits (a quantidade de bits = quantidade de unidades de alocação da memória inteira) que marque como está cada endereço da memória. 1 se aquela área está ocupada. 0 se aquela área está livre.

Mapa de bits



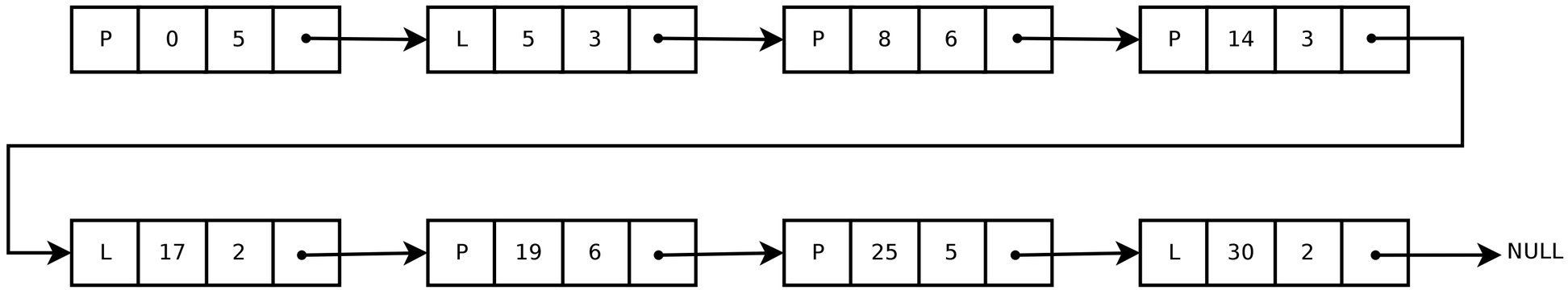
Mapa de bits

- Um problema do mapa de bits é que ele aumenta com o aumento da memória, mesmo que a quantidade de processos utilizando a memória não aumente. O ideal seria ter um método que fosse diretamente proporcional à quantidade de processos
- Outro problema é que a busca é mais custosa pois encontrar um “0” exige que sejam buscados outros “0” na sequência até verificar se o espaço é suficiente.

Lista encadeada

- A ideia nesse segundo método é ter uma estrutura mais eficiente em espaço para marcar espaços ocupados e livres
- Melhor porque tende a ser proporcional aos processos em execução e não mais ao tamanho da memória
- A busca seria mais eficiente se em cada espaço livre fosse armazenado junto, no nó da lista encadeada, a quantidade de espaço total (cada nó terá uma informação se há processo ou espaço vazio, a posição inicial e a quantidade de unidades de alocação)

Lista encadeada



Lista ligada (cada
nó da lista representa
um espaço usado ou livre)

Lista encadeada

- Quando um novo processo chega, um novo nó na lista será criado para esse processo e a depender da localização dele, outros nós precisarão ser modificados
- Quando um processo termina, um novo nó na lista precisará ser adicionado e outros poderão ser modificados
- Sempre que há modificações nos nós da lista, a base e o limite dos processos afetados precisarão ser atualizados na estrutura de dados dos processos no SO