... 4.4.4 - Troca de Página Menos Recentemente Usada

✘ **principal problema:** o algoritmo nunca esquece referências anteriores, ou seja, páginas muito (e não mais) refenciadas no começo da execução de um programa permanecem com um contador alto até o final da execução;

- ... felizmente, uma pequena modificação no NFU faz com que este seja capaz de simular LRU muito bem (o Algoritmo modificado é denominado *Aging*);

- NFU modificado = Algoritmo *Aging*:

❶ os contadores são cada um deslocados 1 bit para a direita, ou seja, são divididos por 2 antes do bit R ser incrementado;
❷ o bit R é incrementado no bit mais a esquerda.

- ... quando ocorre ume falta de página, a página de menor contador é removida.

... 4.4.4 - Troca de Página Menos Recentemente Usada

- ... é óbvio que a página que não tenha sido referenciada por, digamos, 04 *ticks* de relógio terá 04 zeros significativos em seu contador, tendo assim um valor mais baixo que o contador de uma página que tenha sido referenciada nos 04 últimos *ticks*.

✍ Uma diferença entre LRU e *Aging* é que, no último os contadores têm um número finito de bits (tipicamente 8). Portanto, não podemos classificar as páginas segundo referências anteriores à capacidade do contador.

... 4.4.4 - Troca de Página Menos Recentemente Usada

❶ após cada referência de memória, o corrente valor de C é armazenado na entrada da tabela de páginas para a página referenciada;

- ... quando ocorre uma falta de página, o sistema operacional examina todos os contadores na tabela de páginas para achar o menor deles;
- ... a página correspondente é a menos recentemente usada.

❷ quando o *page frame* k é referenciado, o *hardware* primeiro ativa todos os bits da linha k para 1, atribuindo a todos os bits da coluna k o valor 0;

- ... em algum instante, a linha na qual o valor binário é o menor é a menos recentemente usada; a linha na qual o valor é o próximo superior é a segunda menos recentemente usada, e assim por diante.

... 4.4.4 - Troca de Página Menos Recentemente Usada

- Embora os algoritmos apresentados sejam realizáveis, eles são dependentes de *hardware* especial, e são de pouco uso para o projetista de sistema operacional construindo um sistema para uma máquina que não dispõe deste *hardware*.
- ... uma solução que possa ser implementada em *software* faz-se necessária e uma possibilidade é o algoritmo chamado de *Not Frequently Used* ou NFU.
- **idéia** - usar um contador em *software* associado a cada página, inicialmente zero.
- ... em cada *tick* de relógio, o sistema operacional pesquisa todas as páginas na memória e para cada página, o bit R, que é 0 ou 1, é adicionado ao contador;
- ... em suma, os contadores são uma tentativa de guardar a frequência com que cada página tem sido referenciada e, assim, quando uma falta de página ocorre, a página com o menor contador é escolhida para substituição.

4.4.4 - Troca de Página Menos Recentemente Usada

- Uma boa aproximação para o algoritmo ótimo é considerar que as páginas muito usadas nas últimas instruções, provavelmente o serão nas próximas instruções;
- ... igualmente, páginas que não têm sido usadas por um longo tempo provavelmente continuarão não sendo usadas (referenciadas);
- ... esta observação sugere um algoritmo realizável, ou seja, quando ocorre uma falta de página, retira-se a página que não tem sido usada por um tempo longo;
- ... esta estratégia é chamada *Least Recently Used* ou LRU.

... 4.4.4 - Troca de Página Menos Recentemente Usada

✯ Embora o algoritmo LRU seja teoricamente realizável, seu custo é alto, pois é necessário manter uma lista ligada de todas as páginas em memória, com a página mais recentemente usada no início e a menos recentemente usada no final;

✘ ... a dificuldade é que a lista deve ser atualizada para toda referência feita à memória;

✘ ... adicionalmente, encontrar a página na lista, removê-la de sua posição corrente, e movê-la para o início representa um esfôrço não desprezível.

✍ por ser proibitivo manipular uma lista ligada a toda instrução, mesmo em *hardware*, há 02 maneiras de implementar o LRU utilizando um *hardware* especial:

❶ equipar o *hardware* com um contador C de 64 bits que é automaticamente incrementado após a execução de cada instrução;

❷ para uma máquina com n *page frames*, o algoritmo LRU deve manter uma matriz de nxn bits, inicialmente todos zeros.

4.4.3 - Troca de Página FIFO

- **idéia** - de modo análogo ao NRU, o sistema operacional mantém uma lista de todas as páginas correntes na memória, sendo a página da cabeça da lista a mais velha e a página do fim a mais recente;

- ... em uma falta de página, a página da cabeça é removida e a nova página é acrescentada no final da lista ocupando, então, a posição da mais recente.

- ... para evitar o problema da retirada de uma página muito usada, uma variação do algoritmo é examinar os bits R e M da página mais velha;

- ... se a página pertencer a Classe 0, ela é retirada, senão a próxima página mais velha é examinada, e assim por diante;

- ... se páginas da Classe 0 não estão presentes na memória, então o algoritmo é repetido procurando por páginas da Classe 1, 2 e 3.

... 4.4.3 - Troca de Página FIFO

- outra variação do FIFO é o algoritmo *Segunda Chance* - que primeiro examinar a página mais velha como uma vítima potencial;

- ... se seu bit R é 0, a página é trocada imediatamente, senão o bit R é zerado e a página é colocada no fim da lista, como se tivesse acabado de chegar à memória;

- ... a pesquisa pode então prosseguir.

- **Idéia do Segunda Chance** - procurar por uma página velha que não tem sido referenciada no *tick* de relógio anterior. Se todas as páginas tiverem sido referenciadas, o algoritmo degenera-se e torna-se simplesmente um FIFO.

- ... uma pequena variação do *Segunda Chance* é guardar todas as páginas numa lista circular, ou seja, em vez de colocar as páginas no fim da lista, o ponteiro da cabeça da lista é simplesmente avançado uma página, produzindo o mesmo efeito.

4.4.2 - Troca de Página Não Recentemente Usada

- **idéia** - considerando 02 bits, R e M indicando operação de leitura/escrita e de escrita na página, podemos construir um algoritmo de paginação utilizando esses bits;
- ... o **bit de referência** é ativado pelo *hardware* em qualquer leitura ou escrita na página, enquanto o **bit de modificação** é ativado pelo *hardware* quando uma página é escrita, p. ex., um byte é alterado e/ou adicionado à página;
- ... quando um processo é iniciado, ambos os bits de página para todas as páginas são declarados 0 pelo sist. oper. e, periodicamente, o bit R é zerado para distingüir páginas que não foram referenciadas recentemente daquelas que tenham sido;
- ... quando uma falta de página ocorre, o sist. oper. examina todas as páginas e as classifica em 4 categorias baseado nos valores correntes de seus bits R e M:

Classe 0: não referenciada, não modificada	Classe 1: não referenciada, modificada
Classe 2: referenciada, não modificada	Classe 3: referenciada, modificada

... 4.4.2 - Troca de Página Não Recentemente Usada

- Ainda que as páginas na Classe 1 pareçam impossíveis de existir, elas ocorrem quando as páginas da Classe 3 têm seu bit R zerado pela interrupção do relógio;
- ... interrupções de relógio não zeram o bit M porque esta informação é necessária para determinar se uma página terá que ser reescrita no disco ou não.
- O algoritimo *Não Recentemente Usada (Not Recently Used, NRU)*, remove uma página aleatória da classe não vazia de numeração mais baixa;
- ... implícito neste algoritmo é que é melhor remover uma página modificada que não foi referenciada pelo menos no último *tick* de relógio (tipicamente 20 mseg), que uma página não modificada mas muito usada;
- ... as características principais do NRU é que ele é fácil de entender, eficiente de se implementar, e gera um desempenho que, enquanto certamente não ótimo, é geralmente tido como adequado.

4.4 - Algoritmos de Troca de Páginas

- quando uma falta de página ocorre, o sistema operacional tem que escolher uma página para remover da memória a fim de dar lugar à que será trazida do disco;
- ... se a página a ser removida foi modificada enquanto estava na memória, ela deve ser reescrita no disco para manter a cópia em disco atualizada;
- ... se, todavia, a página não tiver sido alterada, a cópia do disco já está atualizada, não sendo necessária sua reescrita e assim a página a ser lida é simplesmente superposta à página que foi retirada.
- apesar de ser possível escolher uma página aleatória para dar lugar à página em demanda, o desempenho do sistema é melhorado se for escolhida uma página que tenha sido pouco referenciada/usada;
- ... se uma página muito usada é removida, ela provavelmente terá de ser trazida de volta em breve, resultando um esforço adicional e, portanto, algoritmos eficientes de troca de página visam minimizar este esforço.

4.4.1 - Troca Ótima de Página

- **idéia** - rotular cada página com o número de instruções que serão executadas antes que a própria páginas seja inicialmente referenciada;
- ... o algoritmo da página ótima simplesmente diz que a página com o maior rótulo deve ser removida, adiando-se o máximo possível a próxima falta de página.
- **problema do algoritmo** - ele não é realizável, pois no momento da falta de página, o sistema operacional não tem como saber quando cada página será referenciada (problema similar ocorre no algoritmo da menor tarefa primeiro);
- ... no máximo podemos executar um programa num simulador e, mantendo uma lista de todas as páginas referenciadas, implementar o algoritmo na segunda execução (usando as informações coletadas na primeira execução).

... 4.3.2 - Segmentação

- ... ao combiná-los, gera-se um único número 10 bits usado para indexar a tabela de segmentos e localizar o descritor de segmento relevante — note que existem exatos $2^{10} = 1024$ segmentos na tabela: 64 segmentos/processo X 16 processos;

❷ verifica-se os bits de proteção para ver se o acesso é permitido e, em sendo, a MMU compara o número da página extraída do endereço virtual com o tamanho do segmento para verificar se o segmento é grande o bastante;

- ... caso seja, o número da página é usado como um índice para a tabela de páginas, cujo endereço é fornecido no descritor de segmento (todas tabelas de páginas são mantidas numa memória rápida especial dentro da MMU);

- ... uma vez que o número da estrutura de página é encontrado, ele é combinado com o deslocamento do endereço virtual para formar o endereço físico da memória, o qual é então posto no barramento de endereço.

... 4.3.2 - Segmentação

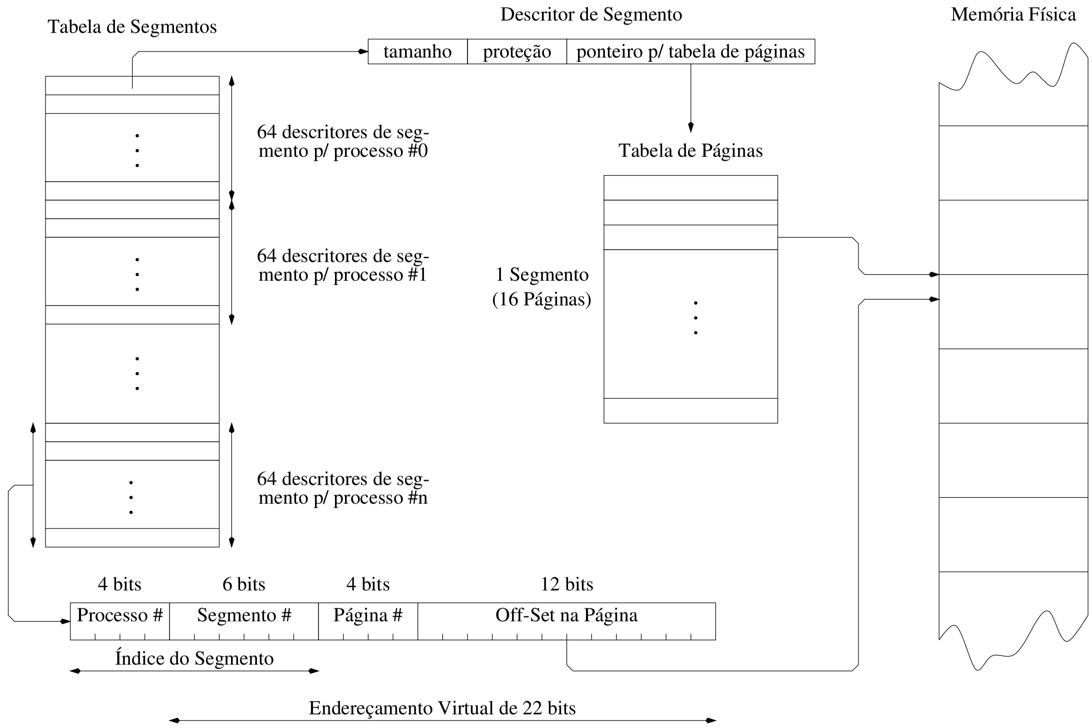
✔ **característica chave** - quando o sistema operacional escolhe um processo, tudo o que ele tem de fazer é mudar o registrador do número de 4 bits do processo e, assim, não precisa recarregar todas as tabelas de segmento ou de página;

- ... 02 ou mais processos podem dividir um segmento se seus descritores de segmentos apontarem para a mesma tabela de páginas, assim, qualquer mudança feita numa página deste segmento é automaticamente visível para os outros processos.

✔ **expansão do espaço de endereçamento** - para espaços de endereço mais largos paga-se o preço de uma tabela maior dentro da MMU;

- ... para o exemplo apresentado, se tivéssemos quatro vezes mais memória para as tabelas de páginas dentro da MMU e seus conteúdos com 4 processos em vez de 16, o MMU poderiamos suportar 64 segmentos de 1 MB por processo.

... 4.3.2 - Segmentação



... 4.3.2 - Segmentação

- ... cada um dos descritores de segmento contém um tamanho de segmento (0 à 16 páginas), bits de proteção dizendo se um segmento pode ser lido ou escrito, e um ponteiro para a própria tabela de páginas;

- ... cada uma das tabelas de páginas contêm 16 entradas, cada entrada apontando para uma *page frame* na memória (guardando o número da *page frame*);

- ... quando o sistema operacional começa um processo, ele carrega um número de 4 bits do processo num registrador especial do *hardware*;

✯ ... quando que um referência a memória é feita, o gerenciador de memória traduz o endereço virtual como se segue:

❶ toma-se o número de 4 bits do processo e os 6 bits de mais alta ordem dos 22 que compõem o endereço virtual (necessários para o endereço de 4M);

4.3.2 - Segmentação

- Paginação provê uma técnica para implementação de um grande espaço endereçável numa memória física limitada, no entanto, para algumas aplicações, um espaço endereçável bi-dimensional é mais conveniente.

- Idealmente, cada programa deve ter um número muito grande de segmentos (i.e. $2^{32} = 4G$), cada um consistindo de grande númeto de bytes (i.e. também 2^{32});

- ... os primeiros 64K de segmentos podem ser revervados para procedimentos, dados, e pilhas, pertencentes ao programa em execução;

- ... os segmentos restantes podem conter um arquivo por segmento, tal que processos possam diretamente endereçar todos seus arquivos, sem ter que abrí-los e usar primitivas especiais de entrada/saída para lê-los e escrevê-los;

- ... cada arquivo cresce ou reduz-se completamente, independentemente dos outros nesta disposição, com cada byte da memória sendo endereçado por um par formado pelo segmento e offset.

... 4.3.2 - Segmentação

- Projeto MULTICS introduziu a idéia de memória segmentada que sobrevive até hoje.

- ... uma implementação típica provê suporte de *hardware* para até 16 processos, cada um com espaço de endereçamento virtual de 1024 páginas de 2K ou 4K;

- ... se introduzirmos páginas de 4K para este exemplo, cada processo terá um espaço de endereçamento virtual de 4M, consistindo de 1024 páginas de 4K cada.

- Este esquema poderia ser implementado dando a cada processo sua própria tabela com 1024 números de *page frames*, no entanto, esta técnica raramente é empregada;

- ... em vez disto, o *hardware* da MMU contém uma tabela com 16 seções, uma para cada um dos 16 processos e cada uma com 64 descritores de segmento;

- ... então o espaço de endereçamento para cada processo de 4M é dividido em 64 segmentos, cada um contendo 16 páginas de 4K.

... 4.3.1 - Paginação

- ✯ O que acontece se o programa tentar usar uma página não mapeada ?

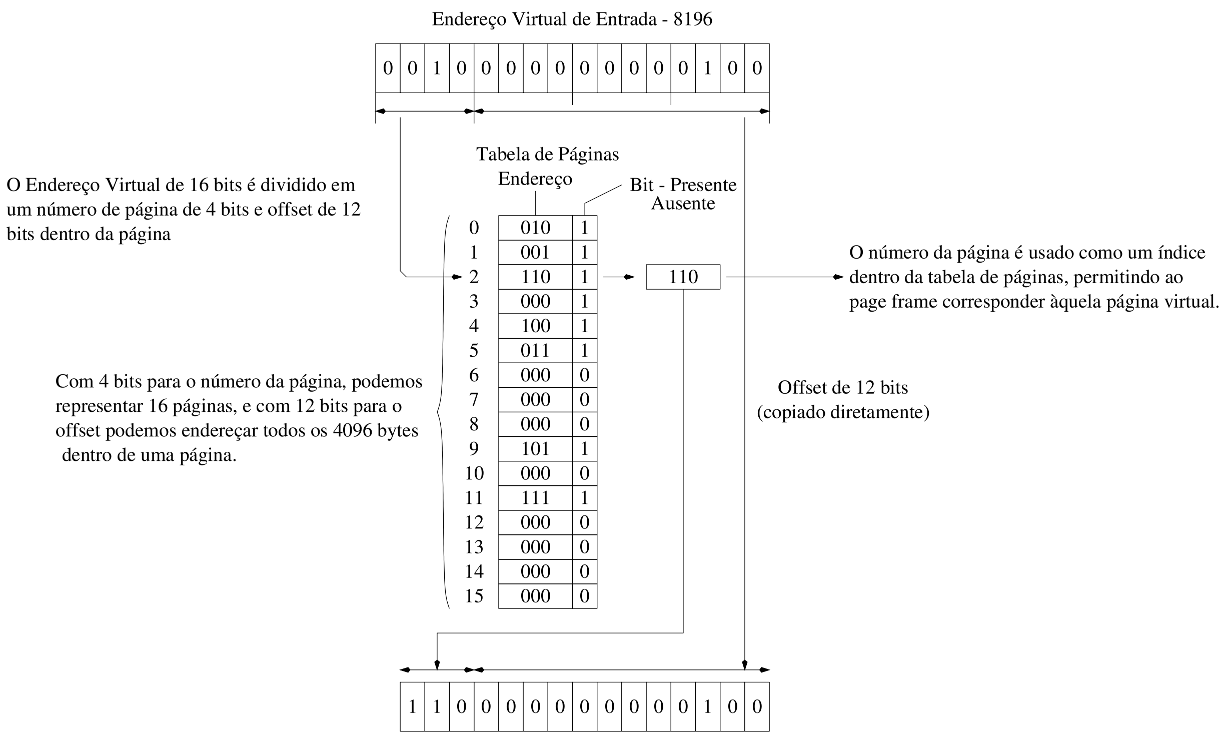
- ... como exemplo, seja uma instrução que referencia o byte número 12 dentro da página virtual 8 (começando em 32768) — a MMU observa que a página está sem mapeamento e força a CPU a causar um *trap* no sistema operacional;

- ... o sistema operacional remove o *page frame* menos usado e escreve seu conteúdo de volta no disco. Ele então busca a página referenciada para o *page frame* liberado, atualiza o mapa, e retorna à instrução interrompida.

- ✯ Agora, vejamos como o gerenciador de memória trabalha e porque temos de escolher um tamanho de página como uma potência de 2.

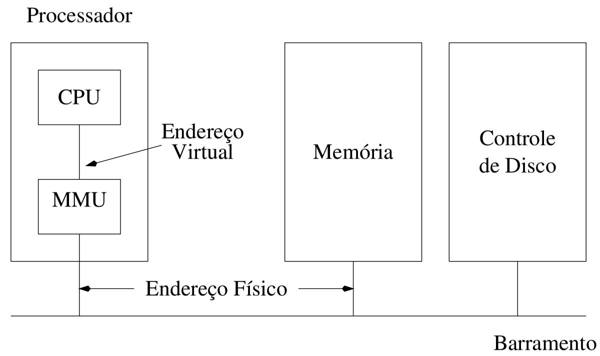
- ... para isto considere o endereço virtual 8196 (0010 0000 0000 0100) sendo mapeado usando o mapa ou tabela de páginas do gerenciador de memória.

... 4.3.1 - Paginação



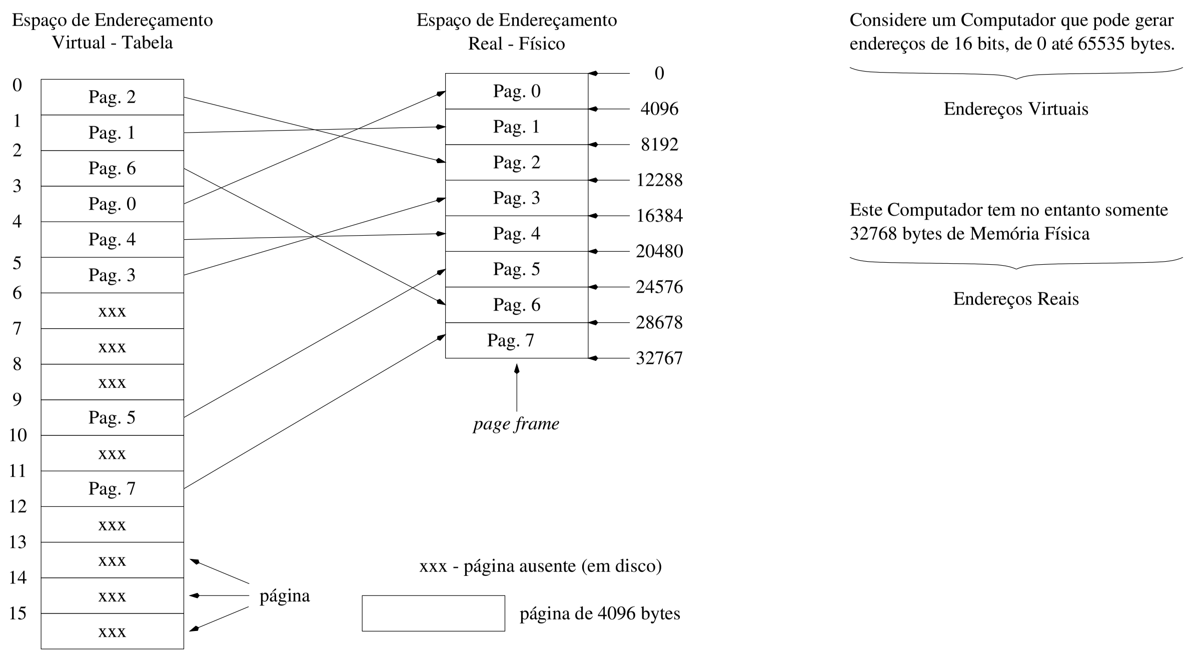
4.3.1 - Paginação

- ... endereços gerados pelos programas são chamados *endereços virtuais* e formam o *espaço virtual de endereçamento* do processo;
- ... em computadores sem memória virtual, o endereço virtual representa o próprio endereço físico e, portanto, não se faz necessário o mapeamento;
- ... quando memória virtual é usada, os endereços de memória não vão diretamente para o barramento de memória, ao invés disso, eles vão à unidade de gerenciamento de memória denominada MMU ou *Memory Management Unit*;



... 4.3.1 - Paginação

- vejamos como esse mapeamento funciona com o auxílio da figura abaixo:



4.3 - Memória Virtual

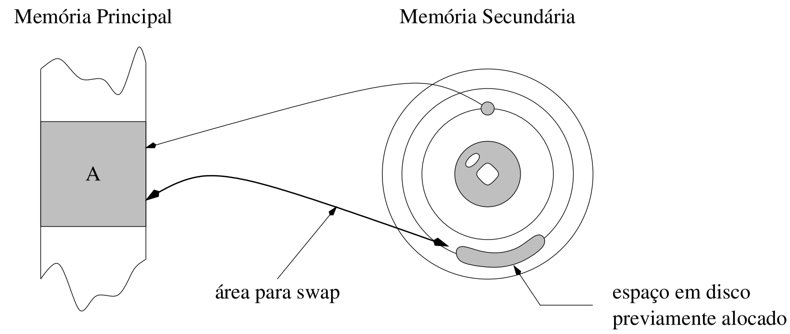
- desde os primórdios da informática, o tamanho dos programas vem superando a quantidade de memória disponível para abrigá-los;
- ... a solução usualmente adotada era dividir o programa em partes, chamados *overlays* e após a execução de cada *overlay* o próximo é executado;
- ... alguns sistemas com *overlay* eram relativamente complexos, permitindo múltiplos *overlays* na memória por vez, embora mantidos em disco e permutados para a memória principal pelo sistema operacional.
- **problema** - o particionamento do programa era deixada a cargo do programador.
- **proposta** - deixar o particionamento do programa a cargo do sistema operacional, ou seja, a combinação do tamanho do programa, dados e pilha, pode exceder a quantia de memória física disponível = **memória virtual**.

... 4.3 - Memória Virtual

- ... o sistema operacional mantém aquelas partes do programa correntemente em uso na memória principal e o resto no disco;
- ... por exemplo, um programa de tamanho 10M bytes pode ser executado em uma máquina com 1M bytes de memória escolhendo-se cuidadosamente quais dos 1M será mantido na memória a cada instante.
- memória virtual pode também ser utilizada em um sistema multiprogramado, por exemplo, 08 programas de 1 MB podem ser alocados em partições de 256 KB numa memória de 2 MB, com cada programa operando sobre 256 KB.
- ... de fato, memória virtual e multiprogramação estão intimamente relacionadas;
- ... enquanto um programa está esperando que parte de seu espaço de endereçamento seja trazido à memória, o programa é bloqueado até que a operação de E/S seja completada, e neste intervalo a CPU pode ser direcionada a outro processo.

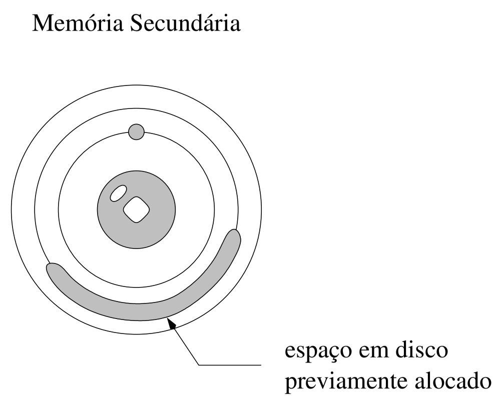
... 4.2.4 - Alocação de Espaço para Swapping

- ... adicionalmente, os algoritmos para gerenciamento de espaço para permuta serão os mesmo usados para gerenciamento da memória principal;
- em outros sistemas, quando um processo é criado, um espaço para permuta é alocado em disco utilizando um dos algoritmos descritos acima;
- ... e, assim, sempre que um processo em memória dá lugar a outro processo, ele é colocado no espaço em disco alocado previamente a ele;



... 4.2.4 - Alocação de Espaço para Swapping

- ... quando um processo termina, o seu espaço para permuta é desalocado;
- Esta técnica é mais eficiente que a anterior pois uma única alocação de espaço em disco por processo é necessária permitindo que um possa processo possa ser permutado várias vezes durante sua execução;
- ... no entanto, uma área maior do disco deve ser reservada para *swapping*;



... 4.2.3 - Gerenciamento de Memória com Listas Encadeadas

- **algoritmo quick-fit** - mantém listas separadas para tamanhos requeridos, ou seja, tem-se um tabela com n entradas onde cada entrada corresponde a um ponteiro para a lista de buracos de tamanho m bytes, sendo que os ponteiros seguintes apontam para listas de tamanho múltiplo de m bytes;

- ... é um algoritmo rápido, embora requeira mecanismos para classificar os buracos por tamanho; determinar seus vizinhos para uma possível fusão quando, por exemplo, um processo termina ou é movido para o disco;

- ... se fusões não forem feitas, rapidamente a memória se fragmenta em um grande número de pequenos buracos não utilizáveis;

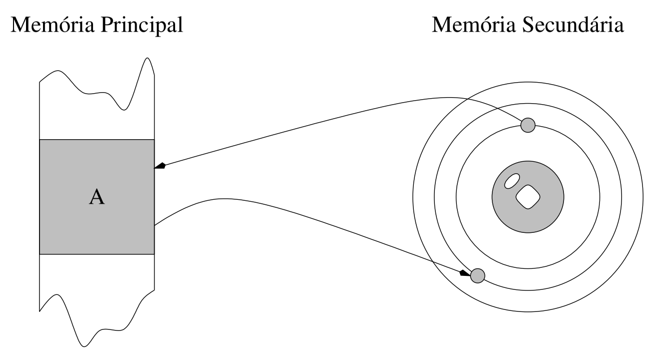
✭ Todos os 04 algoritmos podem ter sua performance melhorada mantendo-se listas separadas para processos e buracos, no entanto este aumento de velocidade na alocação promove a diminuição de velocidade quando se trata de liberar memória.

4.2.4 - Alocação de Espaço para Swapping

- Até agora, consideramos que a memória principal é rastreada quando processos retirados do disco, facilitando assim a alocação de espaço em memória;

- ... mas o que fazer em sistemas nos quais após um processo ter-se movido para a memória, nenhum espaço é mais reservado em disco para este processo ?

- ... quando do seu retorno ao disco, espaço deve ser alocado para abrigá-lo e, assim, em cada troca o processo poderá residir em diferentes regiões do disco;



... 4.2.3 - Gerenciamento de Memória com Listas Encadeadas

- quando os processos e buracos são mantidos na lista ordenada por endereços, vários algoritmos podem ser usados para alocar memória:

- **algoritmo first-fit** - procura ao longo da lista de segmentos até encontrar um buraco de tamanho superior ou igual ao segmento de memória requisitado (M bytes);

- ... caso o buraco tenha tamanho superior (N bytes), o buraco é dividido em 02 segmentos: um de M bytes e outro de N-M bytes;

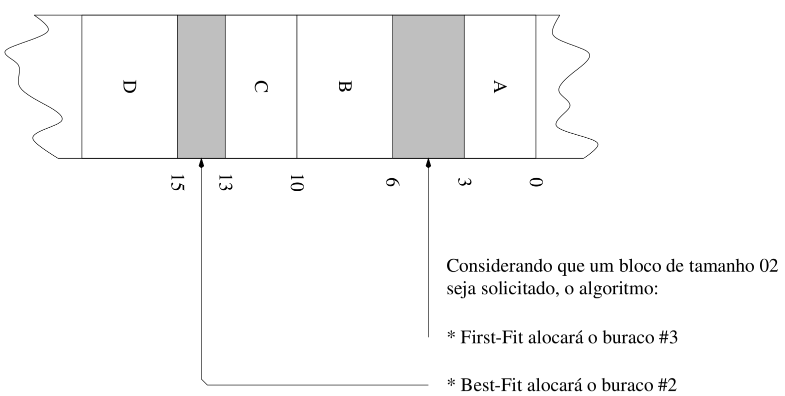
- ... *first-fit* embora seja o algoritmo mais simples é rápido, pois finaliza a busca por um segmento o mais cedo possível, ou seja, não busca pela solução ótima.

- **algoritmo next-fit** - trabalhando da mesma forma que o *first-fit*, com a ressalva de que armazena a posição da lista do último buraco alocado e, assim, na próxima chamada começa a busca a partir do ponto que parou.

... 4.2.3 - Gerenciamento de Memória com Listas Encadeadas

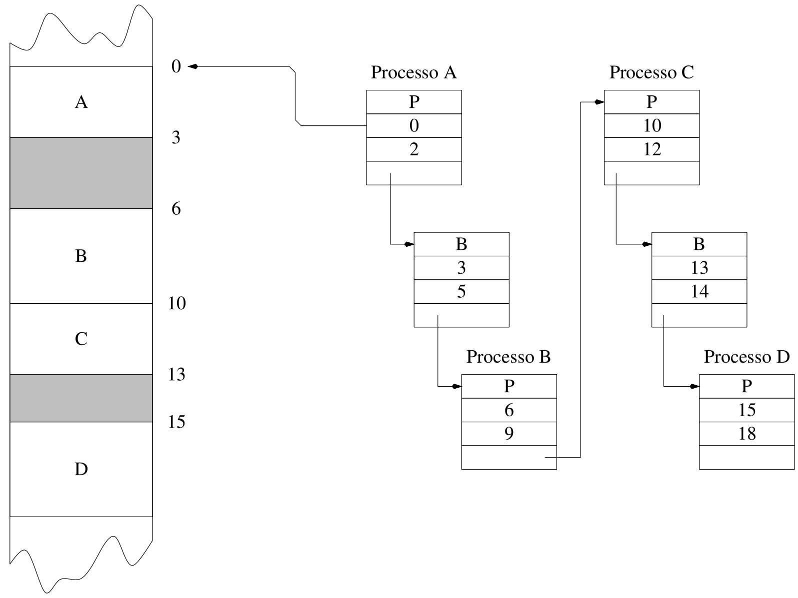
- **algoritmo best-fit** - procura pela lista inteira e toma o buraco de tamanho mais próximo do tamanho do segmento requisitado (M bytes);

- ... é um algoritmo lento que cria na memória buracos pequenos que dificilmente serão alocados, no entanto, quando o segmento procurado é grande, este algoritmo melhora as chances de encontrar um segmento de tamanho adequado;



4.2.3 - Gerenciamento de Memória com Listas Encadeadas

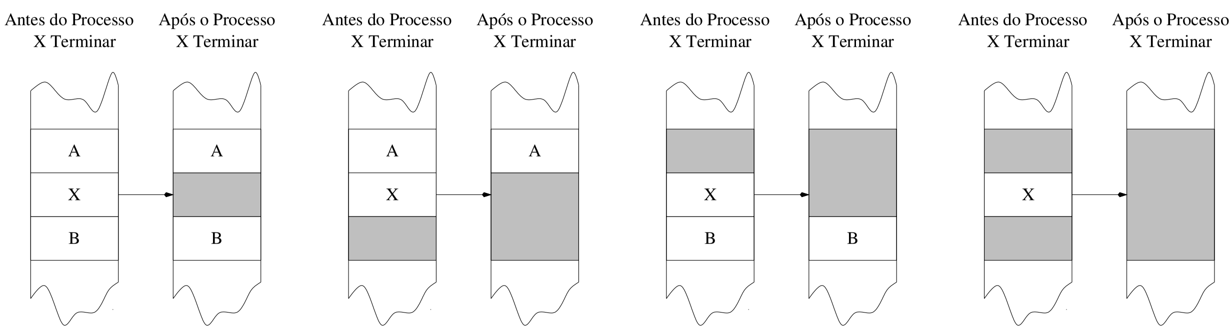
- outra maneira de gerenciar a memória é manter uma lista de alocações e segmentos de memória livre, onde um segmento é um processo ou um buraco entre 02 processos;



... 4.2.3 - Gerenciamento de Memória com Listas Encadeadas

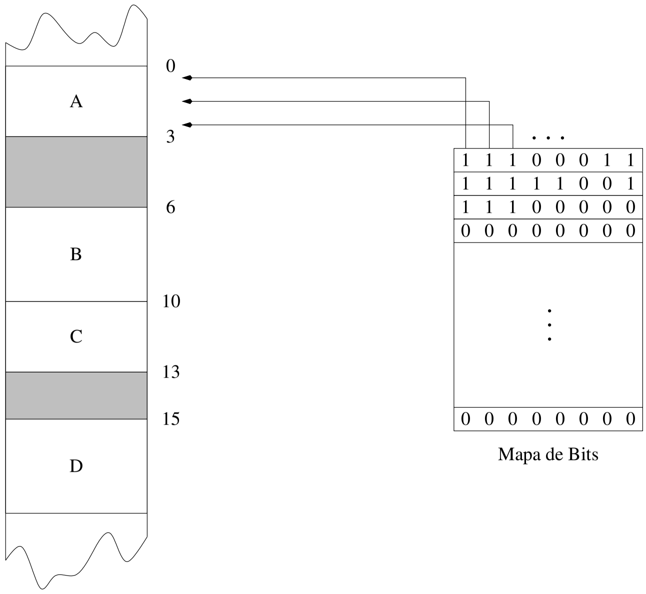
- neste exemplo, a lista de segmentos é mantida ordenada por endereços que traz a vantagem de permitir a atualização direta da lista quando do término de um processo ou mudança do mesmo para o disco;

- ... quando do término de um processo, 04 situações podem ocorrer:



4.2.2 - Gerenciamento de Memória com Mapa de Bits

- com um mapa de bits, cada unidade de alocação da memória tem um bit no mapa indicando se a unidade está livre ou sendo utilizada;



... 4.2.2 - Gerenciamento de Memória com Mapa de Bits

- ... o tamanho da cada unidade de alocação é uma importante fator de projeto — o que representa um gasto maior de memória no gerenciamento da própria memória;
- ... para pequenas unidades de alocação tem-se um mapa de bits maior, embora mesmo assim o montante gasto não necessariamente é grande, por exemplo, uma unidade de 32 bits irá requerer somente 3% da memória.
- ... se a unidade de alocação for escolhida grande, o mapa de bits será pequeno embora possa haver maior desperdício de memória caso o processo não seja um múltiplo exato da unidade de alocação;

✔ o mapa de bits depende somente do tamanho da memória e da unidade de alocação;

✘ para trazer um processo de k unidades de alocação para a memória, o gerenciador de memória deve pesquisar no mapa de bits uma seqüência de k consecutivos bits 0 no mapa — operação lenta, razão pela qual são evitados na prática.

... 4.2.1 - Multiprogramação com Partições Variáveis

- 02 situações são possíveis quando da necessidade de expansão:

  ❶ se o processo necessitar expandir sua memória e existir um buraco adjacente, simplesmente o buraco pode vir a ser incorporado ao espaço de endereçamento do processo;

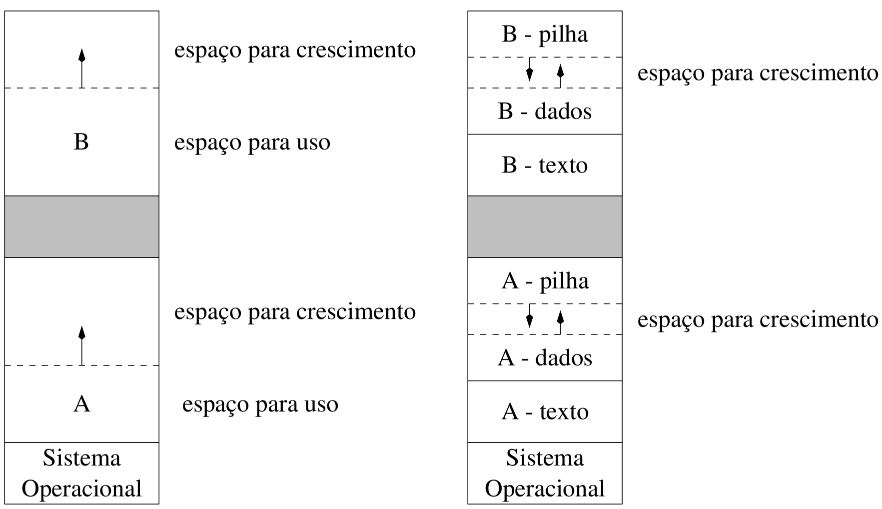
  ❷ de outra forma, se o processo está adjacente a outro processo, o primeiro deverá ser movido para um buraco grande o suficiente para armazená-lo, ou espaço de memória terá que ser criado com a remoção de algum processo.

- ... se for esperado que muitos processos cresçam na memória, uma boa política seria alocar uma pequena área extra toda vez que o processo é permutado ou movido;

- ... contudo, quando os processos são permutados para o disco, somente a área de memória atualmente em uso deve ser copiada, não havendo necessidade de efetuar cópia da área extra da memória reservada para o crescimento do processo.

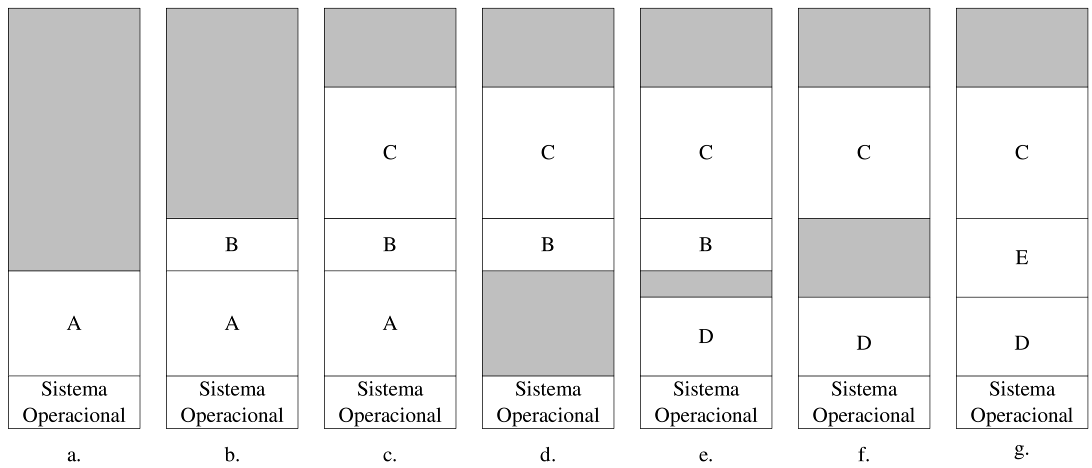
... 4.2.1 - Multiprogramação com Partições Variáveis

- ... a figura abaixo mostra a configuração da memória na qual a área para crescimento foi alocada para 02 (dois) processos:



... 4.2.1 - Multiprogramação com Partições Variáveis

- ... quando partições variáveis são usadas, o número e o tamanho dos processos na memória varia dinamicamente como pode ser visto no diagrama abaixo:



- ... a principal diferença entre partições fixas e variáveis é que o número, a localização e o tamanho das partições variam dinamicamente ao longo do tempo.

... 4.2.1 - Multiprogramação com Partições Variáveis

- É possível combinar todos os buracos disjuntos num único buraco movendo todos os processos para um lado da memória — esta técnica é conhecida como **compactação da memória**;

- ... esta técnica é pouco empregada pelo fato de requerer muito tempo de CPU, por exemplo, um microcomputador com 1 MB de memória e que pode copiar 1 byte por μs, gasta 1 s para compactar toda a memória.

- ... um ponto negativo neste método é saber o quanto de memória alocar para um processo, o que na verdade não é tão simples dado que os segmentos de dados e pilha de um processo tendem a crescer durante a sua execução;

- ... para exemplificar, considere os processos de alocação dinâmica de memória e a recursão presentes nos programas — ambos são típicos exemplos do crescimento dos segmentos de dados e segmento de pilha.

4.2 - Gerenciamento de Memória com Swapping

- ... como discutido anteriormente, em um sistema multiprogramado em lote, a organização em partições fixas é simples e efetiva — quanto mais *jobs* na memória, mais a CPU estará ocupada e não há razão para usar algo mais complicado;
- nos sistemas *time-sharing*, a situação é diferente uma vez que há mais usuários utilizando o sistema que memória disponível para os seus processos;
- ... neste contexto, mover temporariamente processos para o disco é a alternativa para atender os usuários em suas requisições;
- ... este processo de permuta entre a memória principal e o disco, como o intuito de criar espaço na memória para executar outros processos é denominado de *swapping*.

4.2.1 - Multiprogramação com Partições Variáveis

- em princípio, um sistema que utiliza *swapping* pode ser baseado em partições fixas, assim, sempre que um processo é bloqueado, haverá um permuta entre este e um outro processo que esteja no disco;
- ... na prática, partições fixas são pouco atrativas quando a área da memória é escassa fazendo com que muita memória seja perdida por programas cujo tamanho é bem inferior ao tamanho das partições;
- ... assim sendo, um novo sistema de gerenciamento de memória é proposto, o gerenciamento de memória com partições de tamanhos variáveis;
- **multiprogramação com partições variáveis** - o número, a localização e o tamanho das partições variam dinamicamente ao longo do tempo;
- ... a flexibilidade de não se ter um número fixo de partições aumenta a utilização da memória, mas também complica a tarefa de alocar, liberar e gerenciar a memória.

... 4.1.3 - Multiprogramação com Partições Fixas

- **proteção** - refere-se a necessidade de na execução de um programa binário, não haver violação de espaço de endereçamento de um outro programa;
- ... a realocação durante o carregamento não resolve o problema da proteção, pelo fato de que não é possível proibir um programa de ler ou gravar em qualquer posição da memória quando estes operarem endereços absolutos;
- ... outro aspecto importante é que em sistema multi-usuários é indesejável permitir que processos leiam ou escrevam em posições da memória alocadas a outros processos;
- **Solução adotada no IBM 360** - dividir a memória em blocos de 2 KB e atribuir um código de proteção de 4 bits para cada bloco que por sua vez é parte do PSW (*Program Status Word*);
- ... assim o *hardware* protege qualquer tentativa de programa em execução de acessar a memória cujo código de proteção difere daquele presente no PSW, dado que somente o sistema operacional pode mudá-lo;

... 4.1.3 - Multiprogramação com Partições Fixas

- **solução alternativa** - equipa-se o *hardware* com 02 registradores especiais, denominados registradores de base e de limite;
- ... quando o processo é escalonado, o registrador base é carregado com o endereço do começo da sua partição e o registrador limite é carregado com o tamanho da partição para a qual o processo foi alocado;
- ... cada endereço de memória referenciado tem o conteúdo do registrador base a ele adicionado antes de ser enviado para o barramento de acesso à memória;
- ... na seqüência, os endereços modificados são comparados com o registrador de limite para prevenir endereçamento fora do espaço alocado ao processo;
- ... adicionalmente, o *hardware* protege os registradores base e limite para que programas do usuário não o modifiquem.

... 4.1.3 - Multiprogramação com Partições Fixas

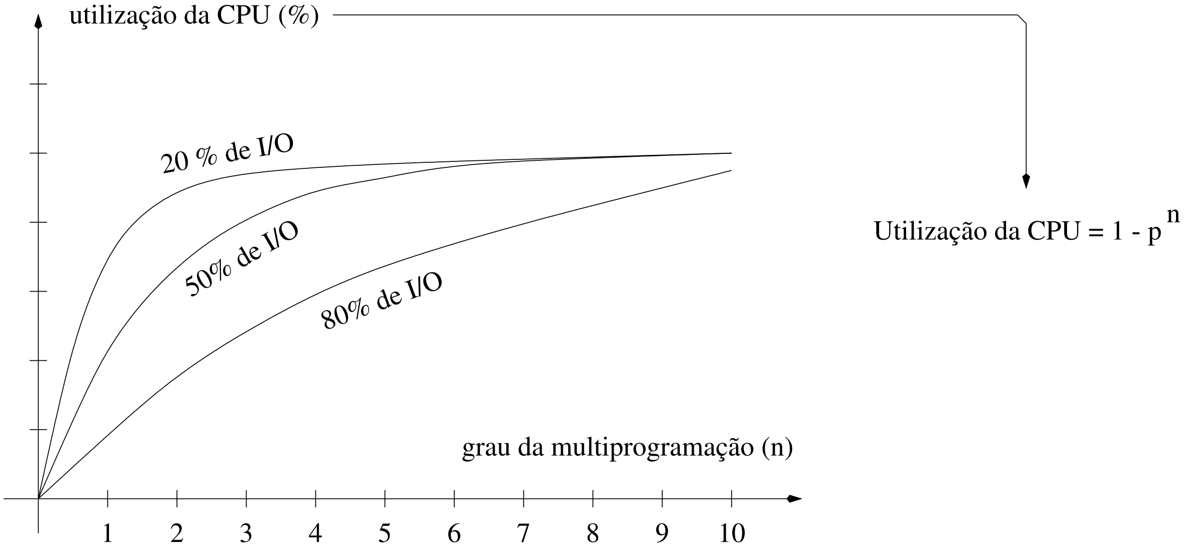
- ... a desvantagem de se ordenar os processos que chegam em filas separadas torna-se aparente quando a fila para a maior partição está vazia, mas a fila para a menor partição está cheia, como no caso das partições 1 e 4 da figura anterior;
- ... uma organização alternativa é manter uma fila única permitindo que quando uma partição for liberada, a mesma seja alocada ao primeiro processo da fila;
- ... esta estratégia pode causar desperdício no uso de partições e assim uma estratégia mais eficaz é procurar em toda fila de entrada a maior tarefa para a partição liberada;
- por outro lado, este último algoritmo também apresenta desvantagens pois discrimina os processos pequenos concedendo um melhor tratamento aos processos maiores;
- ... este algoritmo foi inicialmente utilizado no OS/360 do IBM, onde era chamado de MFT (*Multiprogramming with a Fixed number of Task*).

... 4.1.3 - Multiprogramação com Partições Fixas

- multiprogramação introduz 02 problemas essenciais que devem ser resolvidos: realocação do código e proteção do código;
- **realocação** - quando um arquivo binário é gerado pelo *linker*, o mesmo combina as subrotinas escritas pelo usuário com as bibliotecas em um espaço de endereçamento único e que para ser executado exige considerações quanto a alocação de memória;
- ... uma solução possível é modificar as instruções quando o programa é carregado para a memória principal, por exemplo, programas carregados para a partição 1 terão 100 K adicionados a seus endereços;
- ... isto requer que o *linker* inclua no programa binário uma lista contendo quais palavras do programa são endereços que necessitam realocação, quais são códigos, constantes e outros itens que não devem ser realocados;

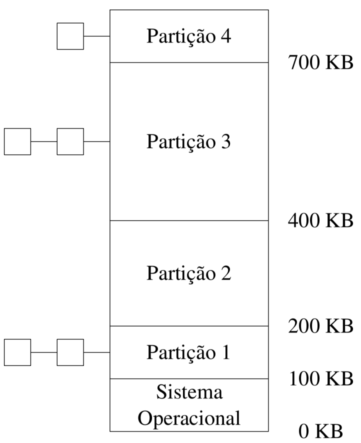
... 4.1.2 - Multiprogramação e Uso da Memória

- o melhor modelo é ver a CPU do ponto de vista probabilístico, ou seja, considere que cada processo gaste em média uma fração p do tempo a espera de entrada/saída, assim a probabilidade de n processos estarem esperando por entrada/saída é p^n.

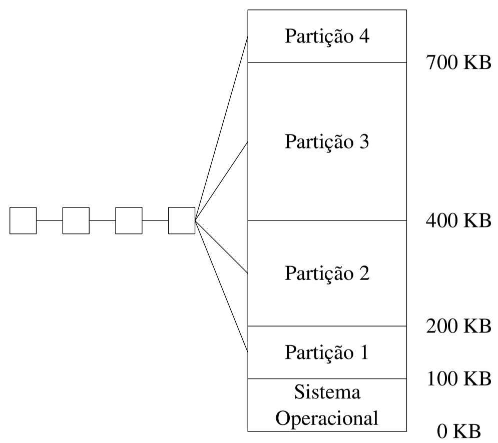


4.1.3 - Multiprogramação com Partições Fixas

- ao admitir mais de um processo na memória por vez, ou seja, multiprogramação, devemos estabelecer uma estratégia de organização da memória;
- ... a estratégia mais simples é dividir a memória em n partições, estabelecidas quando da configuração do sistema operacional, por exemplo;



Fila de Entrada separadas por Partição



Fila Única de Entrada por Patição

... 4.1.1 - Monoprogramação

- ... o programa na memória ROM é chamado de BIOS (*Basic Input Output System*;
- ... quando o sistema é organizado dessa maneira, somente um processo pode estar em execução por vez, ou seja, o usuário entra com um comando no terminal, e o sist. oper. carrega o programa requerido do disco para a memória e o executa.
- ... quando o processo termina, o sist. oper. reassume a CPU e espera por um novo comando para carregar um outro processo na memória já liberada pelo primeiro.
- O modelo apresentado para o IBM PC utiliza um bloco de 8 KB mais altos localizados dentro do espaço de endereçamento de 1 MB de endereçamento.

4.1.2 - Multiprogramação e Uso da Memória

- ... embora a monoprogramação seja usada em pequenos computadores, ela é proibitiva em grandes computadores com múltiplos usuários — dado a necessidade de prover serviços interativos para vários usuários simultaneamente;
- ... a multiprogramação também pode ser justificada pelo uso substancial de operações de entrada/saída nessas operações anteriormente descritas;
- ... por exemplo, se for gasto 40 ms para ler um bloco e a computação demanda apenas 10 ms, sem a multiprogramação a CPU ficará ociosa 80% do tempo.
- com a *multiprogramação*, o percentual de utilização da CPU aumenta pois mais processos podem estar na memória e utilizarem a CPU;
- ... que modelo utilizar para representar a multiprogramação ?

4.1 - Gerenciamento de Memória sem Swapping

- ... sistemas de gerenciamento de memória podem ser divididos em duas classes:

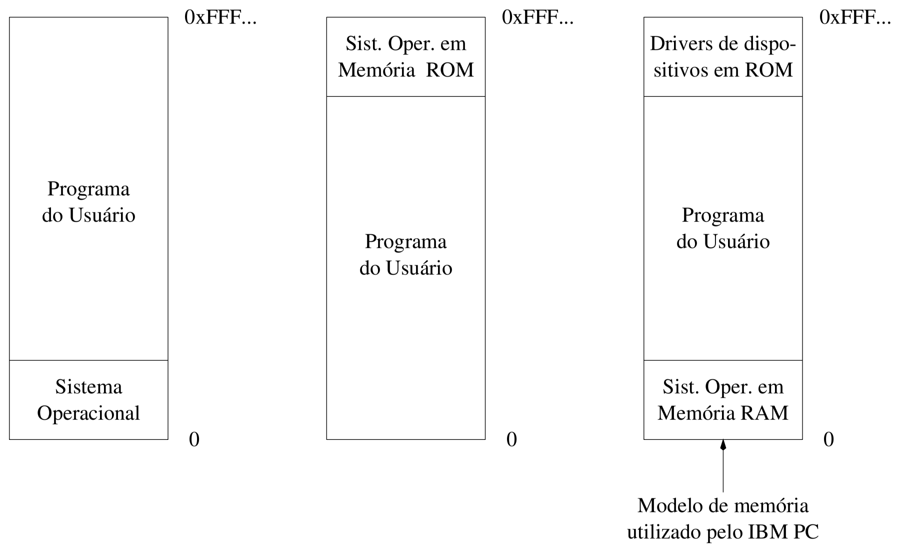
❶ processos são movidos da memória principal para a memória secundária — neste caso o gerenciamento é baseado em *swapping* ou em paginação;
❷ processos permanecem na memória primária.

- ... nas próximas seções, apresentamos com mais detalhes algumas das técnicas utilizadas para manter os processos na memória principal:

❶ monoprogramação;
❷ multiprogramação e uso da memória;
❸ multiprogramação com partições fixas.

4.1.1 - Monoprogramação

- ... o sistema mais simples de gerenciamento de memória consiste em ter-se somente um processo na memória durante toda a sua execução (meados da década de 60);



Cap. 4 - Gerenciamento de Memória

4.1 - Gerenciamento de Memória sem Swapping ou Paginação

- Monoprogramação; Multiprogramação e Uso de Memória; Multiprogramação com Partições Fixas.

4.2 - Gerenciamento de Memória com Swapping

- Multiprogramação com Partições Variáveis; Gerenciamento de Memória com Mapa de Bits; Gerenciamento de Memória com Listas Encadeadas; Alocação de Espaço para Permuta.

4.3 - Memória Virtual

- Paginação; Segmentação.

4.4 - Algoritmos de Troca de Páginas

- Troca Ótima de Página; Troca da Página Não Recentemente Usada; Troca de Página FIFO; Troca de Página Menos Recentemente Usada.

Cap. 4 - Sistema de Arquivo - Referências Bibliográficas

✯ Eleri CARDOZO; Maurício MAGALHÃES; Luís F. FAINA; "Sistemas Operacionais", Dep. de Eng. de Computação e Automação Industrial, Fac. de Engenharia Elétrica e de Computação, UNICAMP, 1996.

- Cap. 4 - Gerenciamento de Memória

✯ A.S. TANEMBAUM; "Modern Operating Systems", Prentice-Hall International Editions, 1984, ISBN: 0-13-637331-3

- Cap. 4 - Gerenciamento de Memória