## Gerenciamento de Memória

**Gerenciamento básico de memória:** Estes podem ser divididos em duas classes: Sistemas que, durante a execução levam e trazem processos entre a memória principal e o disco (troca de processos e paginação), e sistemas mais simples, que não o fazem.

**Monoprogramação sem troca de processo ou paginação:** O sistema operacional (SO) pode estar na base do espaço de endereçamento em RAM, ou estar no topo do espaço de endereçamento, em ROM, ou os *drivers* de dispositivos podem estar no topo do espaço de endereçamento, em ROM, e o restante do sistema mais embaixo, em RAM.

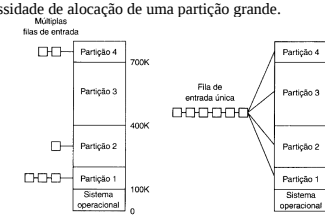
**Multiprogramação com partições fixas:** A multipogramação aumenta a utilização da CPU. A maneira mais comum de realizar a multipogramação consiste em simplesmente dividir a memória em *n* partições (provavelmente de tamanhos diferentes).

Ao chegar, um job pode ser colocado em uma fila de entrada associada à menor partição, grande o suficiente para armazená-lo.

A desvantagem da ordenação em filas separadas dos *jobs* que estão chegando torna-se evidente quando a fila para uma grande partição está vazia, mas a fila para uma pequena partição está cheia. Nesse caso, *jobs* pequenos têm de esperar pela liberação de memória, embora exista muita memória disponível.

Uma organização alternativa é manter uma única fila. Então, sempre que uma partição se torna disponível, o *job* mais próximo ao início da fila e que caiba nessa partição pode ser nela carregado e executado. Como é indesejável desperdiçar uma grande partição com um *job* pequeno, outra estratégia seria pesquisar em toda a fila de entrada e alocar a partição disponível ao maior *job* que nela pudesse ser carregado. Observe que essa última solução discrimina os *jobs* pequenos como não merecedores de uma partição completa, enquanto normalmente o desejável seria alocar aos *jobs* menores o melhor serviço, e não o pior.

Uma solução seria haver pelo menos uma partição pequena. Essa partição permitiria a execução de jobs pequenos sem necessidade de alocação de uma partição grande.




**Modelagem de multiprogramação:** O uso da multiprogramação pode melhorar a utilização da CPU. Um modelo consiste em considerar um ponto de vista probabilístico, onde: Utilização da CPU = 1 – p^n.

Relocação e proteção: A multiprogramação introduz dois problemas essenciais que devem ser resolvidos – realocação e proteção. Deferentes jovs serão executados em diferentes endereços de memória. Quando um programa é ligado, o ligador (linker) tem de saber em que endereço o programa deve começar na memória.

Para executar a relocação do programa, ao carregá-lo em uma partição de memória, o ligador deve incluir no código binário uma lista ou um mapa informando quais palavras do programa são endereços que necessitam de realocação e quais são código de operação, constantes ou outros intens que não devem ser relocados.

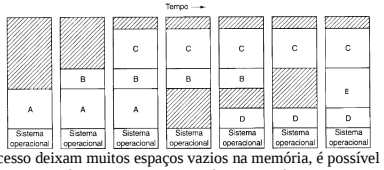
A relocação de um progrma durante sua carga não resolver o problema da proteção. Uma solução alternativa para ambos os problmeama, de realocação e proteção, é fornecer o processador com dois registradores epsciais em hardware denominados ***registrador-base*** e ***registrador-limite.*** Quando um porcesso é escalonado – ou sjea, escolhido para ser executado –, o registrador-base é carregado com o tamanho dessa partição. Cada endereço de memória gerado é automaticamente somado ao conteúdo do registrador-base antes de ser enviado à memória. Os endereços gerados são também verificados em relação ao registrador-limite para certificar-se de que não tentarão endereçar memória fora da partição alocado ao processo em execução.

Uma desvantagem desse esquema é a necessidade de executar uma adição e uma comparação em cada referência a memória.

**Troca de processos:** Em relação a sistemas com compartilhamento de tempo ou computadores gráficos pessoais, a situação é diferente. Às vezes não há memória principal suficiente para conter todos os processos ativos, de modo que os excedente devem ser mantidos em disco e trazidos dinamicamente para a memória a fim de serem executados.

Dois métodos gerais para o gerenciamento de memória podem ser usados, dependendo (em parte) dos recursos de hardware disponíveis. A estratégia mais simples, denominada ***troca de processos*** (*swapping*), consiste em trazer totalmente cada processo para a memória, executá-lo durante um certo tempo e então devolvê-lo ao disco. A outra estratégia denominada ***memória virtual***, permite que programas possam ser executados mesmo que estejam apenas parcialmente carregados na memória principal.

Neste contexto, é possível perceber que o tamanho e a localização das partições variam dinamicamente à medida que os processo entram e saem da memória. Isto melhora a utilização da memória; entretanto, complica a alocação e liberação de memória e o gerenciamento dessas trocas.

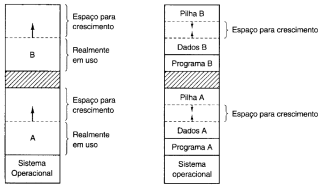



Quando as trocas de processo deixam muitos espaços vazios na memória, é possível combiná-los todos em um único espaço contíguo de memória, movendo-os, o máximo possível, para os endereços mais baixo. Essa técnica é denominada ***compactação de memória***.

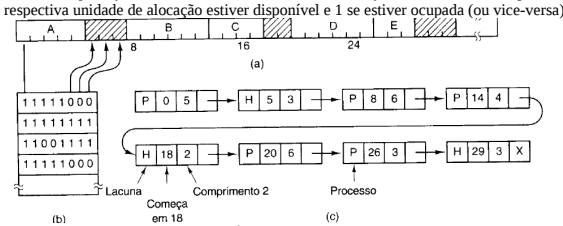
Um ponto importante a ser considerado relaciona-se com a quantidade de memória que deve ser alocado a um processo quando este for criado ou trazido do disco para a memória. Se processos são criados com um tamanho fixo, inalterável, então a alocação é simples: o SO alocará exatamente aquele que é necessário, nem mais nem menos.

Contudo, se a área de dados do processo puder crescer, problemas poderão ocorrer sempre que um processo tentar crescer. Por outro lado, se estiver adjacente a outro processo, o processo que necessita crescer poderá ser movido pra uma área de memória grande o suficiente para contê-lo ou um ou mais processos terão de ser transferidos para o disco a fim de criar essa área disponível. Se o processo não puder crescer na memória e a área em disco (área de troca) para a troca de processos entre a memória e o disco estiver cheia, o processo terá de esperar ou ser exterminado.

Se o esperado é que a maioria dos processos cresça durante a execução, provavelmente será uma boa ideia alocar uma pequena memória extra sempre que se fizer a transferência de um processo para a memória ou a movimentação dele na memória, a fim de reduzir o custo (extra) associado à movimentação ou à transferência de processos que não mais cabem na memória alocada a eles. Contudo, quando os processos forem transferidos de volta para o disco, somente a memória realmente em uso deverá ser transferida, pois é desperdício efetuar a transferência de memória extra também.




**Gerenciamento de memória com mapa de bits:** Quando a memória é alocada dinamicamente, o SO deve gerenciá-lo. Em termos gerais, existem duas maneiras de fazer isso: com mapa de bits e lista de disponíveis. Com um mapa de bits, a memória é divida em unidade de alocação, que podem conter apenas poucas palavras ou ter vários quilobytes. Associado a cada unidade de alocação exite um bit no mapa de bits, o qual vale 0 se a respectiva unidade de alocação estiver disponível e 1 se estiver ocupada (ou vice-versa).




Quanto menor a unidade de alocação maior será o mapa de bits. Contudo, mesmo uma unidade alocação tão pequena quanto 4 bytes, 32 bits, de memória, necessitará de somente um bit no mapa de bits.

O principal problema com essa técnica é que, quando se decide carregar na memória um processo com tamanho de k unidades, o gerenciador de memória precisa encontrar espaço disponível na memória procurando no mapa de bits uma sequencia de k bits consecutivos em 0. Essa operação de busca de 0s 'muito lenta, o que constitui um argumento contra os mapas de bits.

**Gerenciamento de memória com listas encadeadas:** Mantem uma lista encadeada de segmentos de memória alocados e de segmentos de memória disponíveis. Um segmento é uma área de memória alocada a um processo ou uma área de memória livre situada entre áreas de memória de dois processos.

Essa ordenção apresenta a vantagem de permitir uma atualização rápida e simples da lista sempre que um processo terminar sua execução ou for removido da memória. Um processo que termina sua execução tem geralmente dois vizinhos na lista encadeada de segmentos de memória. Esses vizinhos podem sr ou segmentos de memória alocados a processo ou segmentos de memória livres. Como a entrada da tabela de processos referente a um processo em estado de término de execução geralmente aponta para a entrada da lista encadeada associada a esse referido processo, uma lista duplamente encadeada pode ser mais adequada, em vez da lista com encadeamento.

Existem alguns algoritmos que lidam com tal estrutura, são eles:

- ***First fit***: O gerenciador de memória procura ao longo da lista de segmentos de memória por um segmento livre que seja suficientemente grande para esse processo. Esse segmento é então quebrado em duas partes, uma parte é alocada ao processo e a parte restante transforma-se em um segmento de memória livre de tamanho menor. Este algoritmo é rápido pois pesquisa o minimo possível.
- ***Next fit***: funciona da mesma maneira que o *firt fit*, exceto pelo fato de sempre memorizar a posição em que encontra um segmento de memória disponível de tamanho suficiente., quando o algoritmo *next fit* tornar a ser chamado para encontrar um novo segmento de memória livre, ele iniciará sua busca a partir desse ponto, em vez de procurar novamente a partir do início. Seu desempenho é inferior ao *firt fit*.
- ***Best fit***: Esse algoritmo pesquisa a lista inteira e escolhe o menor segmento de memória livre que seja suficiente ao processo.

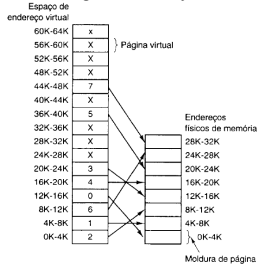
**Memória Virtual:** Dividir o programa em módulos, denominados *overlays* (módulos de sobreposição). O *overlay* 0 seria o primeiro a ser executado. Quando esse *overlay* terminasse, ele chamaria outro *overlay*, que seria carregado em seu lugar na memória a fim de ser executado.

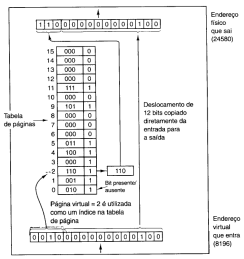
A ideia básica por trás do conceito de memória virtual é que o tamanho total do programa – ou seja, seu código mais seus dados e a pilha – pode exceder a quantidade de memória física disponível para ele. O SO mantém as partes ativas do programa na memória e o restante em disco.

**Paginação:** A maioria dos sistemas com memória virtual utiliza uma técnica denominada paginação. Em qualquer computador existe um conjunto de endereços de memória que os programas podem gerar ao serem executados. Endereços podem ser gerados com o uso da indexação, de registradores-base, registadores de segmento ou outra técnica.

Esses endereços gerados pelo programa são denominados endereços virtuais e constituem o espaço de endereçamento virtual. Em computadores sem memória virtual, o endereço virtual é idêntico ao endereço físico e, assim, para ler ou escrever uma posição de memória ele é colocado diretamente no barramento da memória. Em computadores com memória virtual, ele geralmente não é idêntico ao endereço físico e, desse modo, não é colocado diretamente no barramento da memória. Em vez disso, ele vai a uma MMU (*memory management unit* – unidade de gerenciamento de memória), que mapeia endereços virtuais em endereços físicos.

O espaço de endereçamento virtual é dividido em unidades denominadas páginas (*pages*). As unidades correspondentes em memória física são denominadas molduras de página (*page frame*). As páginas e as molduras de página são sempre do mesmo tamanho. A memória desconhece a existência da MMU e somente enxerga uma solicitação de leitura ou escrita no endereço 8192, a qual ela executa.




No hardware atual, um bit ***presente/ausente*** em cada entrada da tabela de páginas informa se a página está fisicamente presente ou não na memória. A MMU constata que essa página virtual não esta mapeada e força o desvio da CPU para o SO. Essa interrupção (*trap*) é denominada falta de página (*page fault*). O SO então escolhe uma moldura de página (*page frame*) pouco usada e a salva em disco, ou seja, reescreve seus conteúdos de volta no disco. Em seguida, ele carrega a página virtual e reinicia a instrução causadora da interrupção.

Primeiramente, o SO marcará, na tabela de páginas virtuais, a entrada da página virtual 1 como "não mapeada" e, consequentemente, qualquer acesso futuro a endereços virtuais entre 4K e 8K provocará uma interrupção (*trap*) do tipo falta de página. Em seguida, ele marcará, na tabela de páginas virtuais, a entrada da página virtual 8 como "mapeada", de modo que, quando a instrução causadora da interrupção for reexecutada, a MMU transformará o endereço virtual X em endereço físico Y.

O endereço virtual de 16 bits que chega à MMU é nela dividido em um número de página de 4 bits e um deslocamento de 12 bits.

O número da página é usado como um índice para a tabela de páginas a fim de se obter a moldura de página física correspondente àquela página virtual. Se o bit *presente/ausente* da página virtual estiver em 0, ocorrerá uma interrupção (*trap*) do tipo falta de página, desviando-se, assim, para o SO. Se o bit *presente/ausente* estiver em 1, o número da moldura de página encontrado na tabela de páginas será copiado para os três bits mais significativos do registrador de saída, concatenado-os ao deslocamento de 12 bits, que é copiado sem alteração do endereço virtual de entrada. Juntos constituem o endereço físico de 15 bits da memória. O registrador de saída envia esse endereço físico de 15 bits à memória por meio de um barramento.