AULA 3 – GERENCIAMENTO DE MEMÓRIA

OBJETIVO DA AULA

Entender os mecanismos de controle e proteção de memória.

APRESENTAÇÃO

Olá!

Nesta aula estudaremos a segunda grande tarefa do sistema operacional enquanto gerente dos recursos do processador.

A memória é um recurso muito requisitado pelos processos, que precisam do máximo de espaço possível para trabalharem eficientemente. Porém, como qualquer recurso, tem seus limites físicos.

Assim, o sistema operacional precisa gerenciar essa demanda para permitir que todos os processos possam ser executados satisfatoriamente.

Então, vamos ver como isso é feito.

Mãos à obra!

1. INTRODUÇÃO

A função do gerenciamento de memória é fundamental porque qualquer programa precisa de alguma quantidade dela para ser executado.

Porém, a limitação de espaço e as permutas entre a memória principal e a secundária tornam essa tarefa crucial para o bom desempenho do computador.

A gerência de memória lida com quatro aspectos principais:

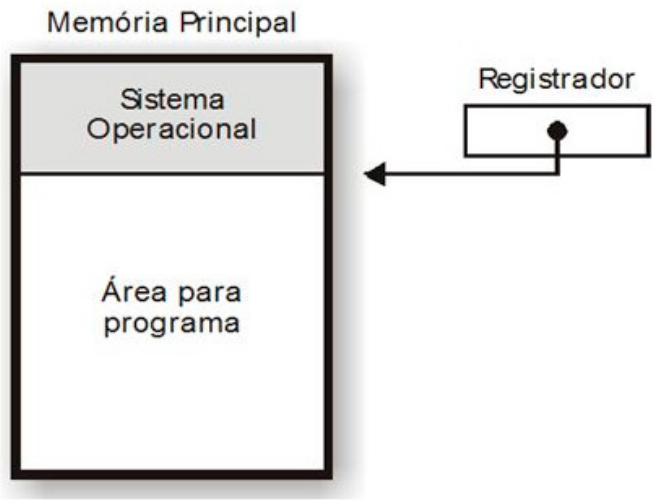
- Controle de ocupação – consiste em registrar os endereços das posições de memória livres e ocupadas;
- Alocação de memória – significa reservar uma faixa de endereços de memória (espaço de endereçamento) para o processo que acabou de ser criado;
- Proteção de memória – como há vários processos ocupando seus espaços de endereçamento, o sistema operacional precisa garantir que um processo não invadirá o espaço do outro;
- *Swapping* – é a permuta entre conteúdos da memória principal e da memória secundária. Esta função será estudada em detalhes quando abordarmos *memória virtual*.

É importante considerar que o sistema operacional deve buscar manter o maior número de processos possível na memória principal, bem como garantir que programas maiores do que o espaço de endereçamento disponível possam ser executados.

2. ALOCAÇÃO DE MEMÓRIA

Na *alocação contígua simples*, usada nos sistemas monoprogramáveis (ou monotarefa), a memória principal tem uma parte destinada ao próprio sistema operacional e o restante é reservado ao programa de usuário e, por haver somente um programa em execução, não há necessidade de proteger esse espaço de endereçamento. Há apenas um registrador que delimita os espaços reservados ao sistema operacional e ao programa de usuário.

FIGURA 1 | **Alocação contígua simples**



Fonte: https://slideplayer.com.br/slide/1235123/)

O principal problema deste modelo de alocação é a subutilização da memória, uma vez que, se o programa de usuário não ocupar todo o espaço disponível, o espaço não ocupado é desperdiçado.

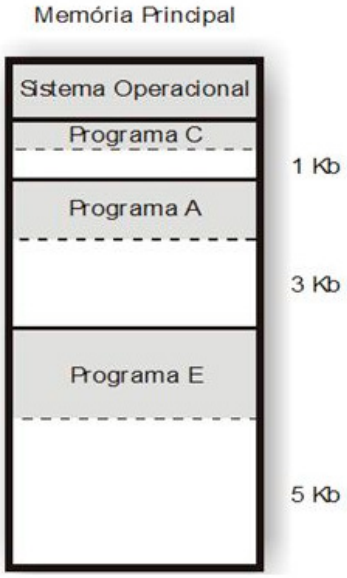
Além disso, se o programa for maior do que esse espaço, ele não caberá ali.

Para resolver isso, é utilizada a técnica de *overlay*, na qual os programas são divididos em módulos que possam ser carregados e executados de forma independente na memória.

A *alocação particionada,* para atender os sistemas multiprogramáveis (multitarefa), pode usar um esquema estático ou dinâmico.

Na alocação particionada estática o tamanho das partições é fixo e isso pode também causar algum desperdício de memória quando as partes em que os programas são divididos não ocupam as respectivas partições totalmente, causando a *fragmentação* da memória, conforme podemos ver na Figura 2 a seguir.

FIGURA 2 | **Fragmentação da memória**



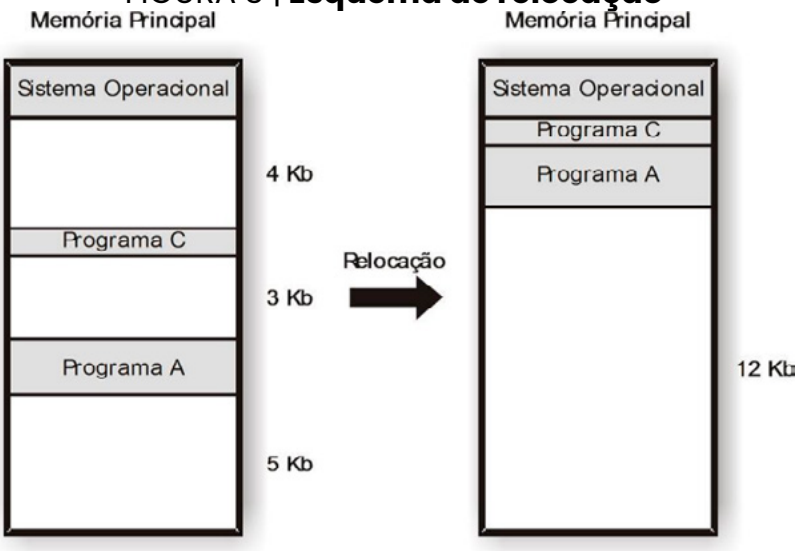
Fonte: https://slidleplayer.com.br/slide/1235123/.

Na Figura 2 as partes em branco são *sobras* ou fragmentos de memória não utilizados.

A *alocação particionada dinâmica* permite que os programas utilizem apenas o espaço necessário para sua execução. Porém, na medida em que vão terminando, os espaços de memória que vão liberando podem trazer também o problema da fragmentação.

A solução para esse problema é a utilização da alocação particionada com *relocação*, em que periodicamente os programas são movimentados na memória para liberar espaços livres.

FIGURA 3 | **Esquema de relocação**



Fonte: https://docplayer.com.br/73250663-Sistemas-operacionais-gerencia-de-memoria.html.

Como podemos observar na Figura 3, após a relocação todo o espaço vazio da memória foi reunido em um único espaço, resolvendo o problema da fragmentação.

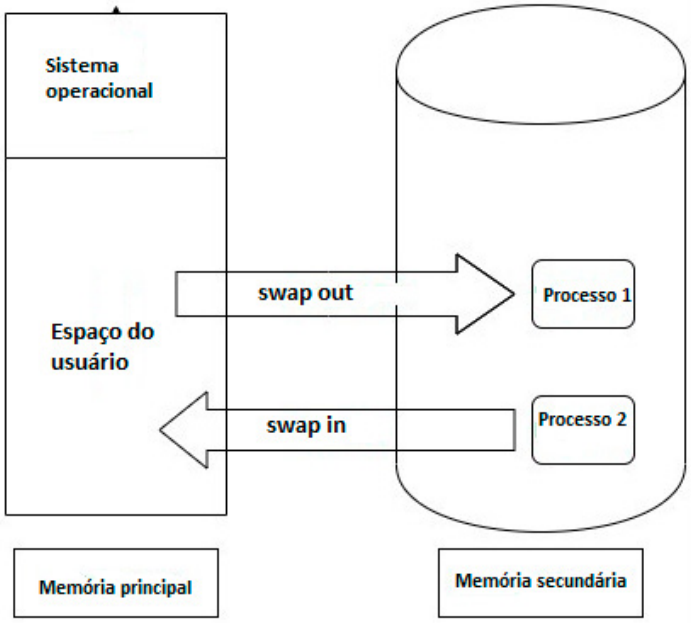
3. SWAPPING

Como sabemos, em geral, os programas são maiores do que o espaço de endereçamento reservado a eles. Assim, é impossível termos alguns programas carregados integralmente na memória principal enquanto estão sendo executados.

O sistema operacional precisa ter alguma forma de garantir que todos os programas – sejam eles do próprio sistema operacional ou do usuário – estejam de alguma forma disponíveis na memória virtual.

A técnica de *swapping* (troca, permuta) permite que o sistema operacional possa realizar as permutas entre a memória principal e a memória secundária, garantindo que todos os programas serão atendidos e que isso ocorrerá no momento em que for necessário.

FIGURA 4 | **Swapping**



Fonte: https://acervolima.com/diferenca-entre-paging-e-swapping-no-so/.

Na Figura 4 vemos que o movimento chamado *swap out* é o de ida da memória principal para a secundária. E o movimento *swap in* é o da memória secundária para a memória principal.

4. MEMÓRIA VIRTUAL

Como já vimos, os programas que demandam a memória são, na maioria das vezes, maiores do que o espaço oferecido a eles.

A memória virtual é uma técnica que utiliza as memórias principal e secundária combinadas de forma que, para o usuário, há a sensação de que a memória disponível é muito maior do que a memória principal.

Essa técnica tem a vantagem de permitir a execução de programas sem qualquer preocupação com seu tamanho, além de tornar possível uma quantidade maior de programas sendo atendidos.

Com a memória virtual os programas não fazem referência a endereços físicos, mas, sim, a endereços *virtuais*. No momento em que é necessário acessar um desses endereços, o sistema operacional se encarrega de traduzir o endereço virtual em um endereço físico e essa operação é denominada *mapeamento*.

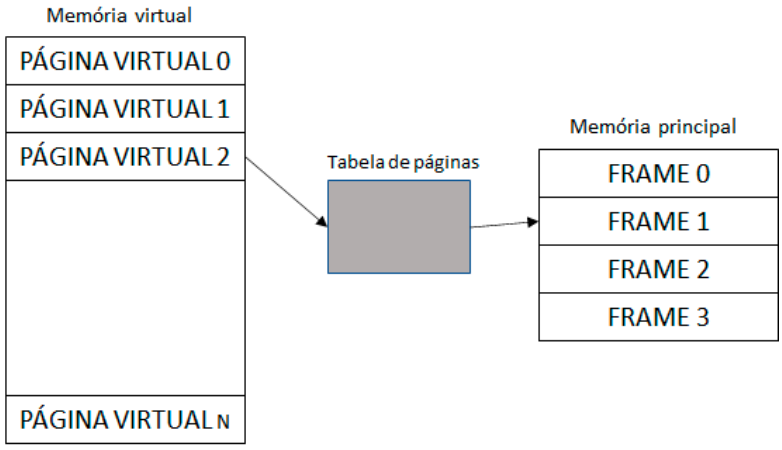
A implementação dessa técnica se dá de duas formas: a *paginação* e a *segmentação*.

Na técnica de paginação, os espaços de endereçamento virtual e real são divididos em pedaços do mesmo tamanho, chamados *páginas*.

Na paginação o sistema operacional utiliza uma estrutura chamada *tabela de páginas*, na qual armazena todas as informações sobre cada página do respectivo processo. Cada processo tem sua própria tabela de páginas.

Quando um programa é executado, suas páginas são transferidas por demanda para *frames* (molduras) na memória principal. Nesta técnica um endereço virtual é formado pelo número da página a que ele pertence mais um deslocamento e o endereço físico é calculado a partir do endereço do frame onde a página se encontra mais o deslocamento. O endereço do frame é encontrado na tabela de páginas.

FIGURA 5 | **Sistema de paginação**



Elaborado pelo autor.

Observe a Figura 5. Pelo que indica a tabela de páginas, a página virtual 2 (PV2) foi carregada no frame 1 (F1). Queremos saber qual o endereço físico correspondente ao endereço virtual PV2 + 4. Suponha que o início de F1 esteja no endereço 1048. Como PV2 está carregada nesse frame, o endereço físico correspondente a PV2 + 4 será 1052 (1048 + 4).

Para o sistema de paginação funcionar corretamente, é necessário que o sistema operacional lide com algumas situações.

A primeira delas é o fato de que há mais páginas nos programas do que frames na memória. Assim, o sistema operacional precisará considerar quais páginas estarão carregadas na memória principal e, quando todos os frames estiverem ocupados, será necessário escolher uma das páginas da memória principal por outra que está chegando da memória secundária.

Há três políticas de substituição de páginas:

1) FIFO (*First In First Out*) – substitui a página que está carregada há mais tempo;

2) LFU (*Least Frequently Used*) – substitui a página que sofreu a menor quantidade de acessos;

3) LRU (*Least Recently Used*) – substitui a página que está há mais tempo sem ser acessada.

Outro aspecto que precisa ser considerado pelo sistema operacional é o controle de quais páginas estão carregadas na memória principal e quais não estão carregadas. Para isso ele usa, na tabela de páginas, um *bit de validade* para cada página, indicando se ela está (1) ou não está (0) carregada na memória.

Finalmente, o sistema operacional também precisa saber se uma determinada página sofreu ou não alteração em seu conteúdo na memória principal. Para isso ele usa o *dirty bit,* que é atualizado para 1 se a página sofreu alterações e permanece 0 se ela não tiver sofrido alterações.

Embora o sistema de paginação seja simples de implementar, como ele não considera a estrutura lógica dos programas ao dividi-los em páginas, isso pode não ser eficiente quando uma estrutura está em parte numa página e em parte em outra.

Na técnica de segmentação, o espaço de endereçamento virtual é dividido em blocos de tamanhos diferentes, de acordo com a estrutura dos programas e há a possibilidade de que os segmentos variem de tamanho para suportar estruturas de dados dinâmicas.

Quanto ao mapeamento de endereços virtuais em endereços físicos, os procedimentos são semelhantes aos da paginação, com os endereços sendo formados pelo número do segmento somado a um deslocamento, no espaço virtual e ocorrendo o mesmo no espaço real, considerando o endereço inicial onde o respectivo segmento foi carregado.

CONSIDERAÇÕES FINAIS

Chegamos ao final dessa aula, na qual vimos como é feito o gerenciamento da memória pelo sistema operacional.

Vimos que são quatro as funções de gerência de memória: controle de ocupação, alocação, proteção e swapping.

Também vimos como o sistema operacional aloca memória, abordando os sistemas de alocação.

Finalmente vimos a técnica de memória virtual, fundamental para os sistemas atuais, uma vez que permite que programas maiores do que o espaço de endereçamento disponível possam ser executados sem problemas.

A memória virtual pode ser implementada basicamente por duas técnicas: a paginação e segmentação.

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá recordar e aprofundar o que vimos aqui a respeito de paginação e segmentação. Disponível: https://www.youtube.com/watch?v=TB19DZBVWRw

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.