UniFil
EDUCAÇÃO A DISTÂNCIA

Unidade 2

Gerência de processador e memória

Introdução

Olá aluno, como vai? Pronto para mais um desafio? Nessa unidade iremos desvendar mais um pouco sobre os sistemas operacionais. A maioria das pessoas pouco entende como o sistema operacional executa vários programas ao mesmo tempo, geralmente pensam que é executado todos de uma vez só. Você que é aluno da área da computação verá que não é bem assim.

Sistemas multiprogramáveis tem como característica de a UCP (unidade central de processamento) ser compartilhada entre diversos processos existentes. Dessa forma, todo sistema multiprogramável possui algum critério para indicar qual a ordem na escolha dos processos que concorrem pela utilização do processador. Então, nessa unidade estudaremos quais técnicas são utilizadas pelos SO's para fazer o escalonamento dos processos concorrentes.

Ainda nessa unidade, também veremos a importância da memória principal, ou memória *RAM (Random Access Memory)* para os sistemas operacionais. Estudaremos os seus principais esquemas de organização e gerência. Como vê é um assunto curioso, que muitas vezes é restrito apenas a profissionais da área tecnológica.

Bons estudos!

Objetivos

- Aprender sobre a gerência do processador;
- Aprender sobre a gerência de memória.

Conteúdo programático

Aula 01 – Gerência do processador.
Aula 02 – Gerência de memória.

Você poderá também assistir às videoaulas em seu celular! Basta apontar a câmera para os **QRCodes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

Aula 01 – Gerência do processador

Sabemos que em sistemas multiprogramados é permitido que vários programas sejam executados ao mesmo tempo. Esses programas compartilham entre si diversos recursos como memória, CPU, dispositivos de entrada e saída.

Claro que com muitos programas rodando ao mesmo tempo é necessário que haja critérios para determinar qual a ordem ou prioridade nas escolhas dos processos para execução entre os programas que concorrem pelo uso do processador. Por isso, é essencial a gerência do processador.

Com a gerência do processador é possível garantir o uso adequado do processador para responder a dezenas de programas em execução pelo sistema operacional (SO). Para o SO escolher os processos que executa, utiliza-se de uma técnica para isso – conhecida como escalonamento (*scheduling)*. A parte do código do sistema operacional responsável pelo escalonamento é chamada de escalonador (*scheduler)* (MACHADO; MAIA, 2000).

As funções básicas da política de escalonamento de processos podem ser as seguintes, segundo (MACHADO; MAIA, 2000):

- Maximizar o rendimento: uma política correta seria atender ao maior número de processos por unidade de tempo;
- Maximizar a utilização de recursos: nesse caso, deve manter os recursos do sistema ocupado;
- Privilegiar a execução de aplicações críticas;
- Assegurar a previsibilidade: diminuindo os tempos de resposta aos processos, um sistema pode garantir que os processos recebam níveis de serviços previsíveis.

Um fator importante que devemos entender é que cada sistema operacional possui a sua própria política de escalonamento adequada ao seu projeto.

Critérios de escalonamento

Um algoritmo de escalonamento tem como objetivo principal escolher qual dos processos prontos para execução deve ser alocado na UCP. Obviamente como foi dito anteriormente, cada sistema operacional tem seu algoritmo de escalonamento adequado ao seu tipo de processamento. Os principais critérios de escalonamento são (MACHADO; MAIA, 2000):

- **Utilização do processador (UCP)**: na maioria dos sistemas, é esperado que o processador esteja a maior parte do tempo ocupado;
- ***Throughput*:** *Throughput* representa o número de processos executados em um determinado intervalo de tempo. Quanto maior o *throughput*, maior o número de tarefas executadas em função do tempo;
- **Tempo de *turnaround***: tempo que o processo leva da sua admissão no sistema até ao seu término, considerando-se o tempo de espera para alocação de memória, espera na fila de processos prontos para execução, processamento na UCP e operações E/S;
- **Tempo de Resposta**: tempo de resposta é o tempo decorrido entre uma requisição ao sistema e o instante em que a resposta é exibida;
- **Tempo de Processador / Tempo de CPU**: é o tempo que um processo leva no estado de execução durante seu processamento. A política de escalonamento não influencia este tempo que depende apenas do programa executado e do volume de dados processado.

Videoaula 1

Agora assista ao vídeo no qual abordaremos os primeiros conceitos sobre a gerência do processador.

Videoaula 1

Utilize o QRcode para assistir!

Agora assista ao vídeo no qual abordaremos os primeiros conceitos sobre a gerência do processador.

Escalonamento não preemptivo

Nos primeiros sistemas multiprogramáveis, onde que se usava apenas o processamento *batch,* o escalonamento utilizado era do tipo não preemptivo. Nesse tipo de escalonamento, quando um processo ganha o direito de utilizar a UCP, nenhum outro processo pode lhe tirar esse recurso.

Existem três tipos de algoritmos de escalonamento não preemptivo, vamos analisá-los (MACHADO; MAIA, 2000):

Escalonamento *First-In-First-Out* (FIFO): é o mais simples algoritmo de escalonamento. O processo que chegar primeiro (*first in*) é o primeiro a ser executado (*first out*). Seu funcionamento precisa apenas de uma fila, onde os processos que passam para o estado de pronto no seu final e são escalonados quando chegam ao seu início. Nesse sistema, quando o processo recebe o processador, ele utilizará a UCP sem ser interrompido. E quando o processo em execução entra em espera, o primeiro processo da fila de pronto é selecionado para execução.

Escalonamento *shortest-Job-First (*SJF): esse algoritmo de escalonamento associa cada processo ao seu tempo de execução no processador. Partindo dessa ideia, quando o processador ficar livre, o processo que estiver em estado de pronto e que precisar

de menos tempo de processador para terminar seu processamento será selecionado para execução.

Dessa forma, esse tipo de escalonamento favorece os processos que executam programas menores e reduz o tempo médio de espera em relação ao FIFO. O problema dessa técnica é saber exatamente, quanto tempo de UCP que cada processo necessita para terminar o seu processamento. Geralmente esse tempo não é determinado de forma precisa.

Os dois algoritmos não preemptivos visto até aqui, não são algoritmos de escalonamento aplicado a sistemas compartilhados, visto que nesses sistemas um tempo de resposta razoável deve ser garantido ao usuário.

Escalonamento Cooperativo: aqui, como o próprio nome diz, o algoritmo coopera para um melhor uso do processador. Nesse tipo de escalonamento não preemptivo, o processo que está executando a UCP pode liberar o processador, retornando à fila de pronto e possibilitando que um outro processo seja escalonado.

Sua característica está no fato de a liberação do processador ser uma tarefa realizada exclusivamente pelo processo em execução. Dessa forma, não temos a interferência do sistema operacional nessa questão. Porém, isso pode ocasionar sérios problemas, visto que um programa malicioso ou um programa mal escrito podem entrar em *looping*, monopolizando assim a UCP.

As primeiras versões do sistema operacional *Windows* 3.1 e 3.11 utilizavam este tipo de escalonamento, ficando conhecido como multitarefa cooperativo.

Videoaula 2

Agora assista o vídeo no qual será explicado sobre os escalonamentos não preemptivos.

Videoaula 2

Utilize o QRcode para assistir!

Agora assista o vídeo no qual será explicado sobre os escalonamentos não preemptivos.

Escalonamento Preemptivo

Um algoritmo é dito preemptivo quando o sistema operacional pode interromper um processo em execução para que outro o processo utilize o processador.

Usando a Preempção é possível ao sistema operacional priorizar a execução de processos, como no caso de aplicações de tempo real, onde o fator tempo é crítico.

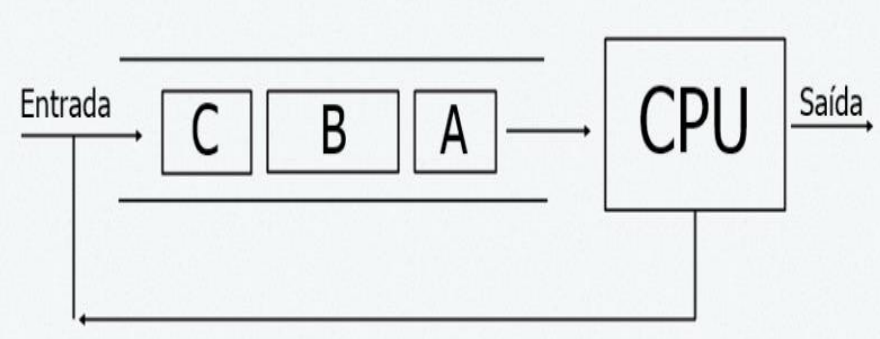
Outra vantagem na preempção é a possibilidade de implementar políticas de escalonamento que compartilhem o processador de uma maneira mais uniforme, distribuindo de forma balanceada o uso da UCP pelos processos.

Ao se fazer trocas de um processo por outro na UCP, é gerado um *overhead* (qualquer processamento ou armazenamento em excesso) ao sistema. Para que isso não se torne preocupante, o sistema deve estabelecer de forma correta os critérios de preempção. Agora vamos estudar alguns algoritmos preemptivos.

Escalonamento circular: esse escalonamento é implementado por um algoritmo que foi projetado exclusivamente para sistemas de tempo compartilhado. Nota-se nesse algoritmo uma semelhança com o algoritmo FIFO, mas, quando o processo passa para o estado de execução é lhe dado um tempo limite para sua utilização de forma contínua. Quando esse tempo, conhecido como *time-slice*, expira, sem ter a UCP liberada pelo processo, ele volta ao estado de pronto, fornecendo a vez para outro processo. Esse mecanismo é conhecido como preempção por tempo.

Os processos ficam em fila no estado de pronto, como mostra figura 1.0 abaixo. O escalonamento é feito, alocando a UCP para cada processo da fila no intervalo de tempo determinado pelo sistema (*quantum).*

Figura 1 - Exemplo de escalonamento circular



Fonte: o autor (2021)

Nesse tipo de escalonamento circular, nenhum processo poderá monopolizar a UCP, pois o mesmo obedece ao tempo determinado pelo sistema.

Escalonamento por prioridades: como vimos, o escalonamento circular, melhora a distribuição do tempo de UCP em relação aos escalonamentos não preemptivos. Mas ainda não consegue implementar um compartilhamento igualitário entre os diferentes tipos de processo, isso acontece porque o escalonamento circular trata todos os processos de maneira igual, o que na computação nem sempre é desejável (TANENBAUM, 2015).

Então basicamente, o escalonamento por prioridades funcionaria da seguinte forma: Sempre que um programa for para a fila de pronto com prioridade superior ao do processo que está em execução, o sistema deverá interromper o processo corrente, colocá-lo no estado de pronto e, escolher o de maior prioridade para ser executado. Finalizando essa ideia, neste tipo escalonamento o processo com maior prioridade no estado de pronto é sempre o escolhido para execução, e o processo com valores iguais são escalonados seguindo o critério de FIFO.

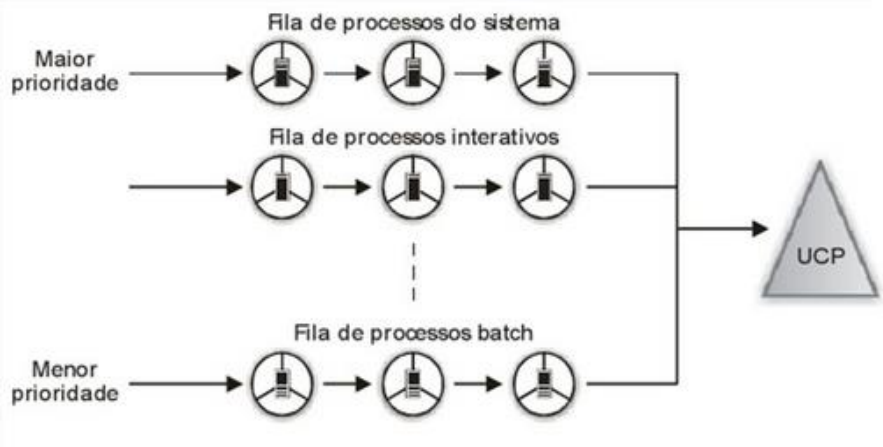
Escalonamento por múltiplas filas: processos tem características de processamento diferentes, é muito improvável que apenas uma técnica de escalonamento seja apropriada a todos tipos de processo. Nesse caso, seria ideal classificar em grupos os processos em função do processamento realizado e aplicar aos grupos mecanismos de escalonamento distintos. Pensando assim, criou-se o escalonamento por múltiplas filas.

Nessa técnica, os processos devem ser classificados, previamente, em função do tipo de processamento, para poderem ser encaminhados a uma determinada fila. Dessa forma, cada fila possui uma prioridade associada a ela que estabelece quais filas são prioritárias. Por fim, o sistema apenas escalona processos de uma fila se todas as outras de prioridade maior estiverem vazias (MACHADO; MAIA, 2000).

Para ilustrar esse escalonamento observe na figura 2, abaixo. Considere os processos que em virtude de suas características sejam divididos em três grupos: sistema, interativo, *batch*. Os processos de sistema podem ser colocados em uma fila superior de prioridade em relação as dos outros processos, usando dessa forma um escalonamento por prioridades, os processos interativos devem estar em uma fila de prioridade intermediária, implementando, por exemplo, o escalonamento circular. O mesmo tipo de escalonamento pode ser usado na fila de processo *batch*, contudo a fila deverá possuir uma prioridade mais baixa.

Então caro aluno, você pode notar que a característica deste escalonamento é permitir a convivência de mecanismos de escalonamento diferentes em um mesmo sistema operacional.

Figura 2 - Escalonamento por múltiplas filas



Fonte: Siqueira (2021)

Escalonamento em Sistemas de Tempo Real: um sistema de tempo real é aquele no qual o tempo tem uma função essencial. De maneira diferente dos sistemas de tempo compartilhado, onde o tempo de resposta rápida é esperado, porém não obrigatório, todo processamento em tempo real deve ser executado dentro de limites altamente rígidos de tempo, caso contrário, pode haver sérios problemas ao sistema (TANENBAUM, 2015).

Nesse tipo de escalonamento não existe o conceito de *time-slice.* O que existe é a utilização única do esquema de prioridades. Para cada processo é atribuída uma prioridade conforme a sua importância dentro do sistema. Essa prioridade deve ser estática, não podendo ser alterada no decorrer do processamento. Esse escalonamento deve ser

aplicado na solução de aplicações onde exista graus de exigência na execução de suas tarefas.

Sistemas de tempo real são utilizados em aplicações de controle de processos, sistemas que controlam produções nas indústrias, tráfego aéreo e sistemas hospitalares etc.

Videoaula 3

Agora assista o vídeo no qual será explicado sobre os escalonamentos preemptivos

Videoaula 3

Utilize o QRcode para assistir!

Agora assista o vídeo no qual será explicado sobre os escalonamentos preemptivos

Indicação de Vídeo

Assista a esse interessante vídeo em que é explicado o escalonamento utilizado pelo sistema operacional *Linux*.

Disponível em: https://www.youtube.com/watch?v=wiabvRrbB3s. Acesso em: 8 maio 2021.

Leitura Obrigatória

Vamos aprofundar o conhecimento a respeito do escalonamento de processos. Por isso, para conhecer mais, leia o capítulo 2 do livro **“Sistemas Operacionais Modernos”** de Andrew Tanenbaum, a partir do **tópico 2.4.1**. Essa leitura com certeza vai complementar seus estudos. Acesse o link:

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/74. Acesso em: 8 maio 2021.

Aula 02 – Gerência de memória

Estudamos na aula anterior, a importância de se ter uma gerência para os processadores. Agora veremos também, como é importante a gerência da memória para que um sistema funcione corretamente. Para que um programa funcione em um computador muitos componentes devem ser controlados, por meio do sistema operacional.

Na memória principal (*RAM – Random Acess Memory)* são colocados todos os programas e dados que serão executados ou referenciados pelo processador. A memória é considerada um recurso relativamente caro. Por esse motivo, é que existem formas de otimizar a sua utilização.

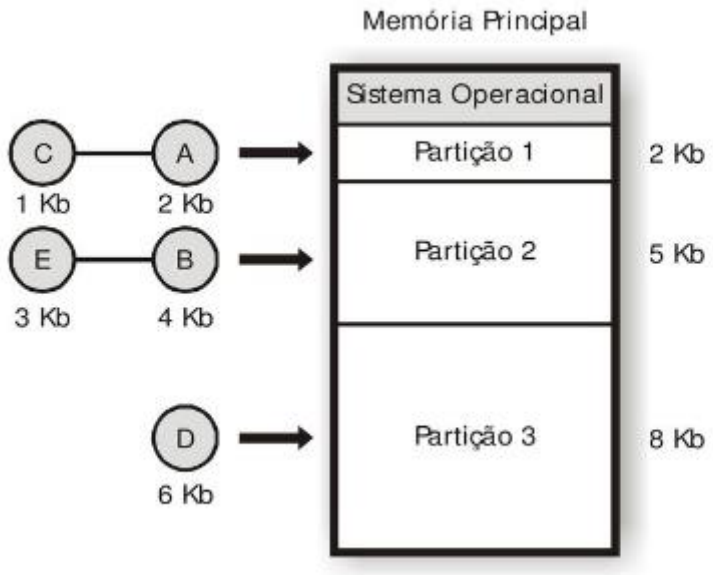
Para que se tenha eficiência em sistemas multiprogramáveis é importante que vários programas estejam em memória ao mesmo tempo, surgindo daí a necessidade de gerência da memória. Então, agora vamos estudar algumas técnicas de organização e gerência de memórias.

Alocações Particionadas Estática e Dinâmica

No começo dos sistemas multiprogramados, a memória foi dividida em blocos de tamanho fixo (embora de tamanho fixo, não tinham necessariamente o mesmo tamanho entre elas, possibilitando diferentes configurações para sua utilização), chamado de partições. O tamanho das partições era estabelecido na fase de inicialização do sistema, em função do tamanho dos programas que executariam no ambiente (MACHADO; MAIA, 2000). A alteração do tamanho de uma partição necessitaria a inicialização do Sistema Operacional. Esse esquema era conhecido como alocação particionada estática.

Nesse esquema os programas só podiam executar em apenas uma das partições, mesmo que outras estivessem disponíveis. Esse problema acontece devido aos compiladores e montadores que geravam códigos absolutos (carregável a partir de uma posição absoluta). Vamos exemplificar, observe a figura 1 abaixo:

Figura 1 - Alocação particionada estática



Fonte: o autor (2021)

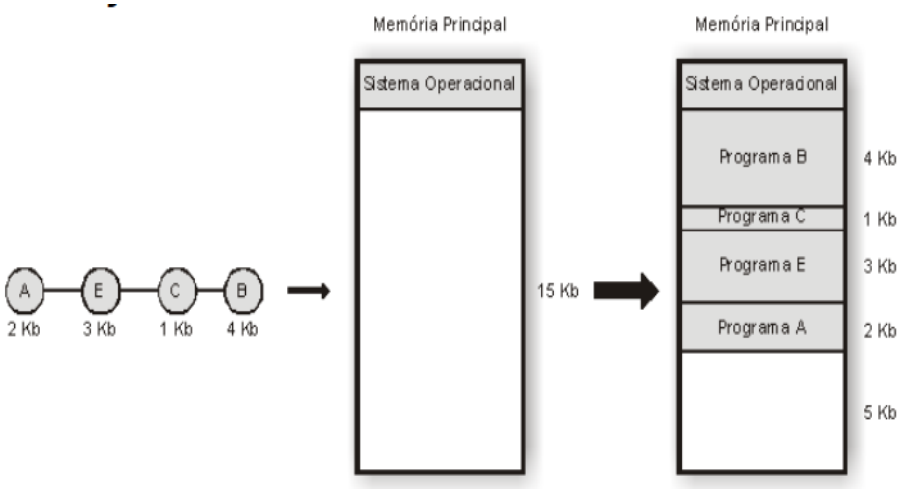
Então observe, se os programas A e B estivessem sendo executados e a terceira partição estivesse livre, os programas C e E não poderiam ser executados.

Como podem notar não é um método muito apropriado, haja vista que acarretava muitos problemas. Atente-se aos problemas trazidos pela alocação estática:

- Alguns programas são maiores que qualquer partição livre. Nesse caso, o programa fica esperando uma que o acomode-o, mesmo se existir duas ou mais partições adjacentes que, somadas, totalizassem o tamanho do programa. Este tipo de problema, em que pedaços de memória ficam impedidos de serem usados por outros programas, é chamado de **fragmentação**;
- Os programas só podem executar em uma das partições, mesmo com outras disponíveis.

Como o problema da fragmentação desperdiça muito espaço, foi necessário pensar outro tipo de alocação como solução que pudesse aumentar o compartilhamento da memória. Na alocação particionada dinâmica, foi eliminado o conceito de partições de tamanho fixo. Nesse novo esquema, cada programa pode utilizar o espaço que necessita, passando esse bloco ser a sua partição. Observe a figura 2 para compreender melhor esse conceito.

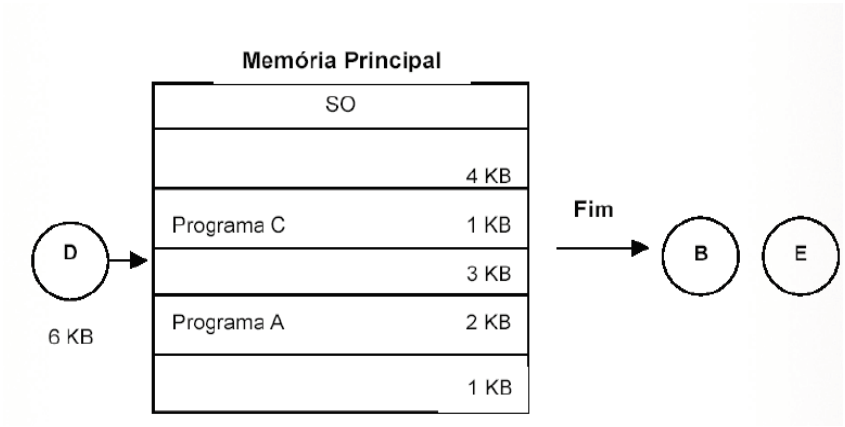
Figura 2 - Alocação particionada dinâmica



Fonte: o autor (2021)

Observando a figura acima, notamos que o problema da fragmentação da memória foi resolvido em grande parte. Mas à medida que os programas forem ocupando espaços, eles vão deixando espaços cada vez menores na memória, não permitindo o ingresso de novos programas, dessa forma, gerando fragmentação novamente. Veja o caso da figura 3, mesmo existindo espaço de 8kb livres o programa D não pode ser carregado devido à falta de espaço.

Figura 3 - Memória fragmentada

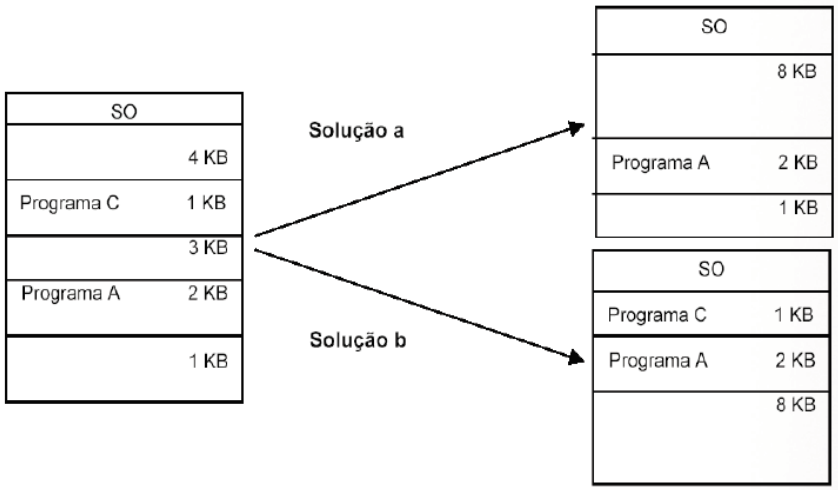


Fonte: Siqueira (2021)

Se caso for detectada a fragmentação nesse esquema, existem duas soluções para o problema, observe a figura 4.

a) Caso o programa C termine, pode-se reunir os espaços adjacentes produzindo um único espaço de tamanho maior de 8kb, *solução a*.
b) Outra maneira, envolve a relocação de todas as partições ocupadas, eliminando, dessa forma, todos os espaços entre elas e criando uma única área livre contígua. Porém, esse procedimento aumenta de maneira considerável a complexidade do algoritmo e acaba consumindo mais recursos do sistema, *solução b*.

Figura 4 - Soluções para a fragmentação



Fonte: Siqueira (2021)

Videoaula 1

Agora assista o vídeo no qual será discutido sobre a gerência de memória.

Videoaula 1

Utilize o QRcode para assistir!

Agora assista o vídeo no qual será discutido sobre a gerência de memória.

Leitura Obrigatória

Vamos aprofundar o conhecimento a respeito de Gerência de memória. Por isso, para conhecer mais, leia o capítulo 9 do livro "**Sistemas Operacionais",** de DEITEL, Harvey M. *et al.* os tópicos 9.8 e 9.9**.** Essa leitura com certeza vai complementar seus estudos. Acesse o link do livro:

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/315/pdf/12. Acesso em: 8 maio. 2021.

Indicação de Vídeo

Agora assista o vídeo do professor Eduardo Wenzel, ele apresenta alguns conceitos sobre a gerência de memória. Vale a pena conferir!

Disponível em: https://www.youtube.com/watch?v=hNVnosJsS5Q. Acesso em: 8 maio 2021.

A técnica de *Swapping*

Nos esquemas de alocação particionada estática e dinâmica, um programa permanecia na memória principal até o final da sua execução, até mesmo quando os processos estavam esperando por um evento, tal como uma operação de leitura ou gravação. Ou seja, o programa somente sairia da memória principal quando tivesse terminada sua execução.

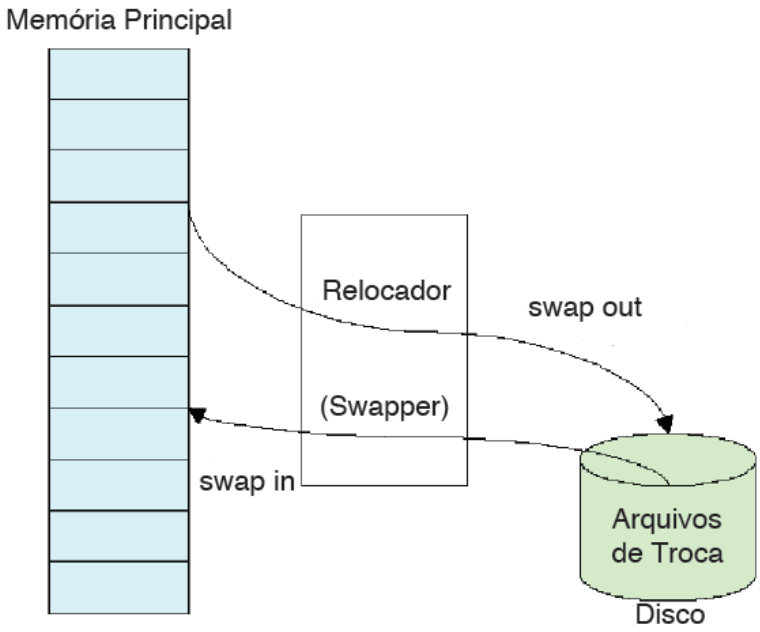
Ocorre que, muitas vezes existem mais processos para serem executados do que memória para manter todos os processos. Dessa forma, foi necessário deixar os processos que estão em excessos em um disco secundário.

O *swapping* é uma técnica aplicada à gerência de memória. Esse esquema consiste em o sistema operacional levar um programa da memória principal para um disco secundário (*swap out*), retornando posteriormente para a memória principal (*swap in),* sem que o usuário perceba essa manobra.

O *swapping* é realizado por rotinas especiais que estão presentes nos SO e que são chamadas de relocadores ou *swappers.* Cada SO fornece seu relocador dependendo do tipo de gerenciamento de memória.

Para executar essa técnica, o sistema operacional que dispõe do gerenciamento de memória e processos, envia um comando para o relocador retirar os dados de uma área de memória e armazená-los em disco. Então o relocador realiza uma cópia desta área de memória em arquivo especial denominado arquivo de troca ou *swap file.* Quando se copia a área de memória para o disco, tal área fica marcada como livre, tornando-a disponível para outros processos. Ainda é realizado um registro do que foi copiado para a memória tornando possível assim a sua recuperação. A figura 5 demonstra o esquema de *swapping* trabalhando com um processo no sistema operacional (TANENBAUM, 2015).

Figura 5 - Processo de *swapping*



Fonte: Siqueira (2021)

Ainda na esteira da técnica de *swap*, em sistemas multiprogramados podem existir vários processos em estado de pronto e espera, e somente um pode estar no estado de execução em determinado instante. Esse processo que está em execução pode solicitar áreas adicionais de memória e esses pedidos podem ser solucionados de duas formas conhecidas:

- As áreas livres podem ser cedidas ao processo, ou;
- Pode-se também acionar o **relocador** para a liberação de aéreas de memória que estão sendo usadas por outros processos através da remoção de seu conteúdo para os arquivos de troca (*swap*).

Com base em nossa leitura até aqui, podemos então tirar algumas conclusões sobre as vantagens e desvantagens desse esquema de *swap*. Primeiramente então, vamos analisar as vantagens:

- Tenta resolver o problema de insuficiência da memória para todos os usuários;
- Aloca espaço para programas que esperam por memória livre para serem processados;
- Essencial para a implementação de um sistema multiprogramável;
- Mais eficiente para programas onde existiam poucos usuários competindo por memória e em ambientes que trabalhavam com aplicações pequenas.

Agora como desvantagem desse sistema temos os seguintes itens:

- Problema da realocação dos programas. O *loader* realocável permite que um programa seja colocado em qualquer posição da memória, porém a realocação é realizada no momento do carregamento;
- Em certas situações o mecanismo é ineficiente em função do tempo gasto para carregamento;
- Seu maior problema é o elevado custo das operações de entrada/saída (*swapped in/out*).

Videoaula 2

Agora assista o vídeo no qual será discutido sobre a técnica SWAP.

Videoaula 2

Utilize o QRcode para assistir!

Agora assista o vídeo no qual será discutido sobre a técnica SWAP.

Indicação de Vídeo

Agora assista o vídeo do Douglas Mugnos, em que ele explica sobre a memória *SWAP* de uma maneira muito simples. Citando exemplos práticos, vale a pena conferir!

Disponível em: https://www.youtube.com/watch?v=5UUOXPVFHbA. Acesso em: 8 maio 2021.

Leitura Obrigatória

Vamos aprofundar o conhecimento a respeito do assunto *SWAP*. Por isso, para conhecer mais leia o capítulo 3 do livro "**Sistemas Operacionais Modernos"** de Andrew Tanenbaum, em especial o **tópico 3.2.2.** Essa leitura com certeza vai complementar seus estudos. Acesse o link abaixo:

Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233. Acesso em: 8 maio 2021.

Memória virtual

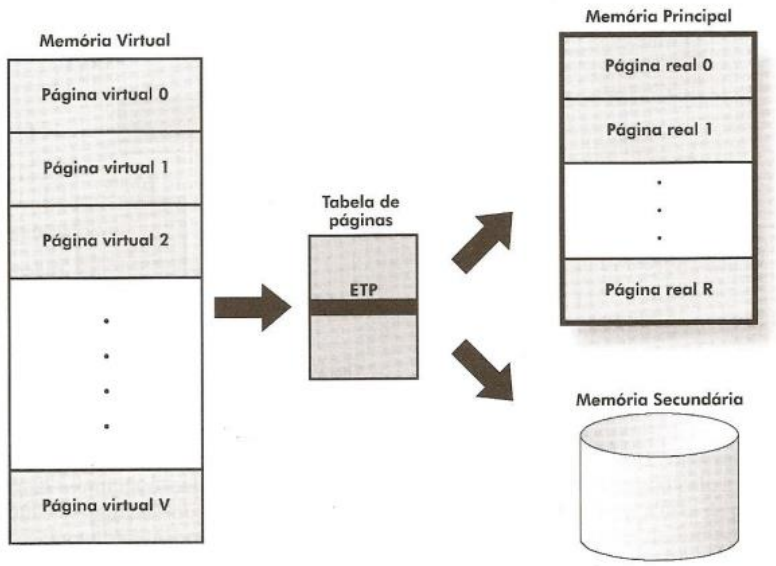
Nos tópicos anteriores estudamos algumas técnicas básicas de gerenciamento de memória e, cada uma dessas técnicas disputavam o espaço limitado da memória. Memórias principais maiores e rápidas poderiam ser a solução, mas como se sabe, o custo

é muito caro. Uma outra solução pensada é implementar uma ilusão de que existe mais memória. Essa é a ideia principal da memória virtual (MACHADO; MAIA, 2000).

Memória virtual é uma técnica primorosa e poderosa que é utilizada na gerência de memória, onde as memórias principais e a secundária são combinadas e, dessa maneira passam a impressão para o usuário que existe uma memória muito maior que a memória principal. Nessa técnica cada programa tem o seu próprio espaço de endereçamento que é dividido em blocos chamados de página. Cada página é uma série de endereços próximos.

Ocorre que quando os programas são maiores que a memória física, apenas parte deles pode estar residente na memória em um determinado instante. O sistema operacional utiliza a memória secundária como extensão da memória principal. Dessa forma, quando um programa está executando, uma parte apenas do código fica na memória principal, enquanto o restante permanece na memória secundária até o momento de ser referenciado, figura 6.

Figura 6 - Espaço de endereçamento virtual

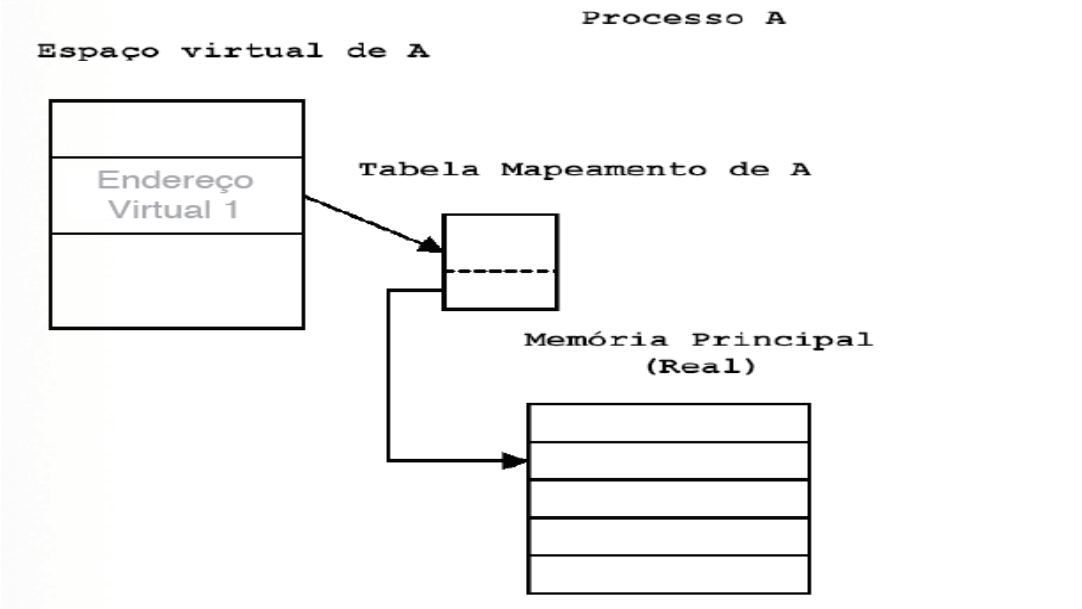


Fonte: Siqueira (2021)

Um programa que utiliza a memória virtual não faz referência a endereços físicos de memória (endereços reais), mas apenas a endereços virtuais. Quando se executa uma instrução, o endereço virtual é traduzido para um endereço físico. O mecanismo de tradução do endereço virtual para o endereço físico é chamado de mapeamento.

O mecanismo de mapeamento é responsável por manter as tabelas, figura 7, de mapeamento exclusivas para cada processo, relacionando os endereços virtuais do processo às suas posições na memória física.

Figura 7 - Tabela de mapeamento



Fonte: Siqueira (2021)

A memória virtual pode ser implementada através do uso dos mecanismos de paginação e segmentação. É mais comum o uso da paginação ao invés da segmentação.

Leitura Obrigatória

Vamos aprofundar o conhecimento a respeito da memória virtual. Por isso, para conhecer mais, leia o capítulo 3 do livro "**Sistemas Operacionais Modernos"** de Andrew Tanenbaum, o **tópico 3.3.** Essa leitura com certeza vai complementar seus estudos. Acesse o *link* abaixo:

Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233. Acesso em: 8 maio 2021.

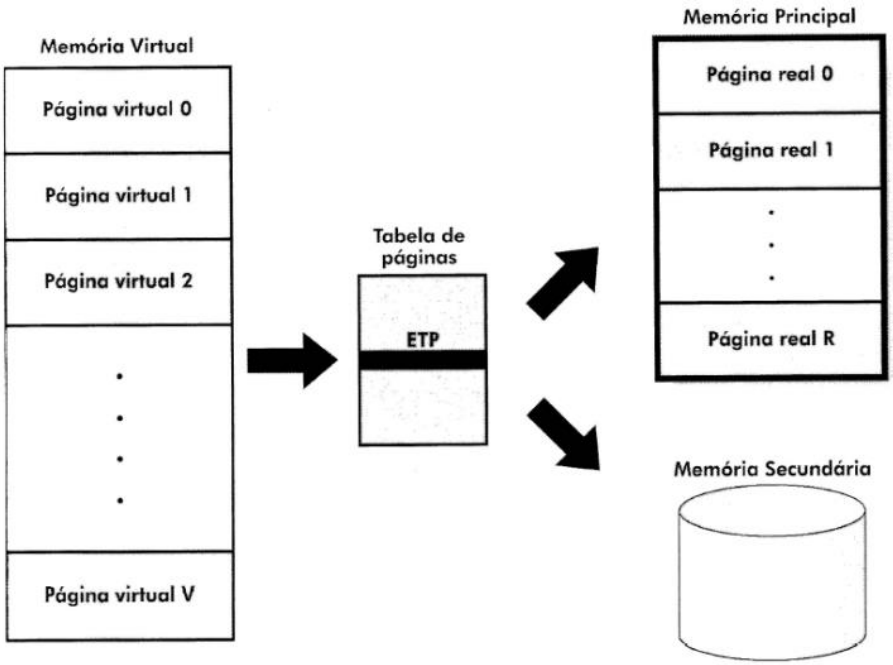
Paginação

A maioria dos sistemas que contêm memória virtual utiliza a técnica denominada paginação. É uma técnica de gerência de memória em que o espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos de mesmo tamanho. As páginas que pertencem ao espaço virtual são denominadas páginas virtuais, enquanto as páginas no espaço real são conhecidas como páginas reais ou *frames* (molduras) (MACHADO; MAIA, 2000).

As páginas virtuais possuem um número que as identifica e, também a memória física (molduras) é dividida em blocos iguais, tendo o mesmo tamanho das páginas virtuais. As molduras são identificadas por um número e correspondem a uma determinada área da memória física. Essas molduras são identificadas e começam a ser numeradas a partir do zero.

Todo o mapeamento de endereço virtual em real é realizado através de tabelas de páginas. Cada página virtual do processo possui uma entrada na tabela (entrada na tabela de páginas-ETP), que contêm informações advindas do mapeamento que fornecem a localização da página real correspondente, veja figura 8.

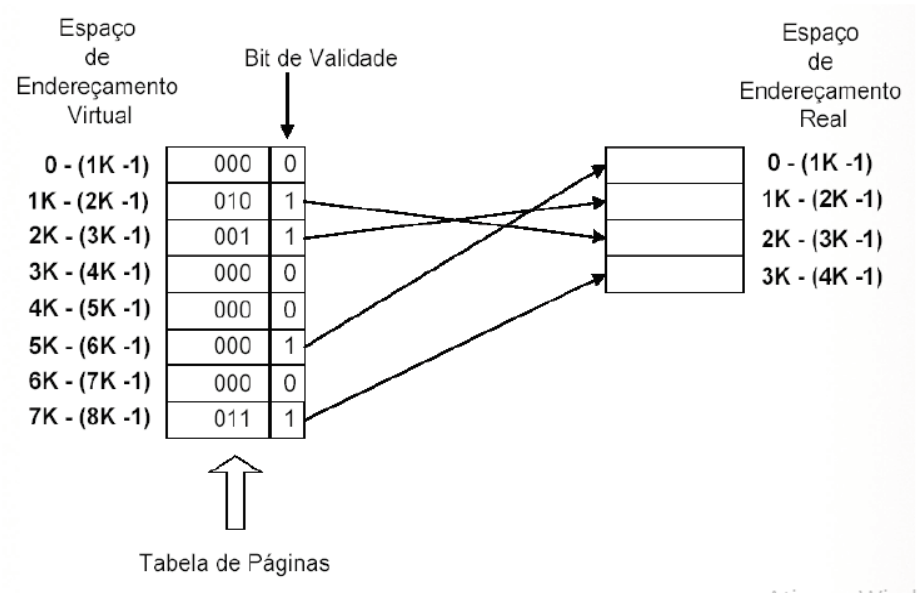
Figura 8 - Tabela de página



Fonte: Machado; Maia (2000)

A ETP não lida somente com as localizações das páginas virtuais, ela também possui outras informações para o processo de paginação. Uma das informações é o *bit* de validade, que indica se uma página está ou não na memória principal. Se o *bit* tem valor 0, indica que a página virtual não está localizada na memória principal, mas se é igual a 1, a página está localizada na memória. É o caso da página 6K-(7K-1), cujo *bit* de validade é 0, observe a figura 9 abaixo.

Figura 9 - Tabela de página com bit de paridade

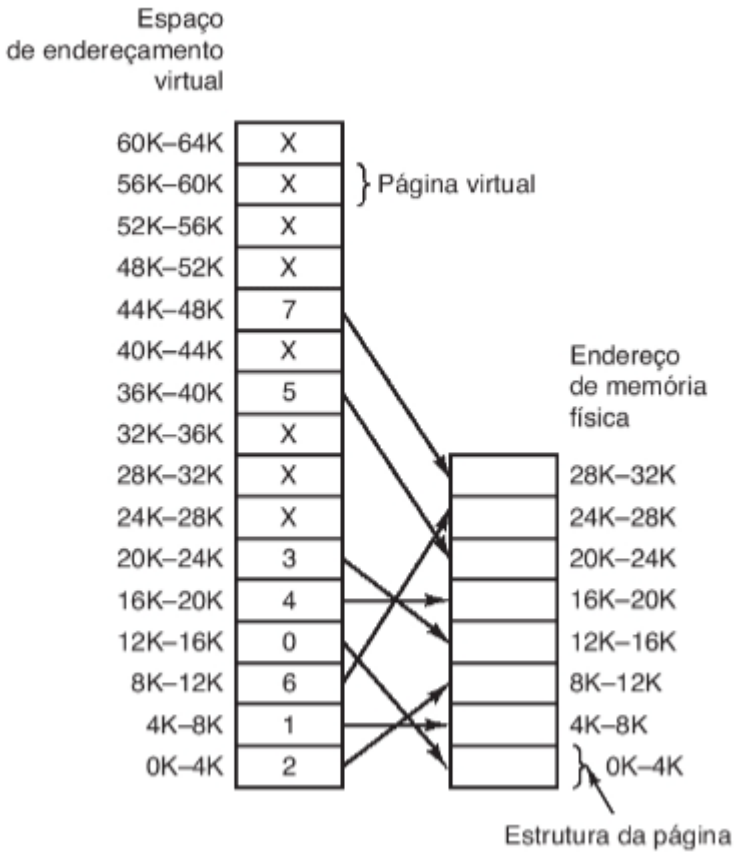


Fonte: Machado; Maia (2000)

Sempre que o processo referência um endereço virtual, a unidade de gerência de memória verifica, através do *bit* de validade, se a página que contém o endereço referenciado está ou não na memória principal (endereçamento Real). Se por acaso não estiver, o sistema deve transferir a página da memória secundária para a memória física. Toda vez que o sistema é solicitado para isso, dizemos que ocorreu uma falta de página *(page fault*).

Realmente meu caro aluno! É muita teoria a respeito desse assunto. Vamos tentar exemplificar. No exemplo da figura 10, temos um computador que pode gerar endereços virtuais de 16 *bits*, de 0 a 64K-1. Também podemos notar que esse computador tem apenas 32KB (28K – 32K) de memória física. Podemos notar ainda que temos **páginas de endereçamento virtuais** que podem ser chamadas apenas de **página**. E temos também **endereço de memória física** que são conhecidos na computação como **molduras** de página.

Figura 10 - Relação de endereços virtuais e endereços de memória física



Fonte: Tanenbaum (2015, p. 116)

Suponha agora que um programa tente acessar o endereço 0 (0K-4K), por exemplo. O endereço virtual 0 é enviado a MMU (*memory management unit* - unidade de gerenciamento de memória, que mapeia endereços virtuais em físicos) que identifica que esse endereço virtual se encontra na página virtual 0K-4K, que significa que os endereços físicos e virtuais nessa página são de 0 a 4095. E que de acordo com seu mapeamento corresponde a moldura de página 2 que é 8K – 12K (8192 a 12287). Então a MMU transforma o endereço virtual 0, que foi lhe foi entregue pela CPU, no endereço físico 8192 e o envia a memória por meio do barramento. Nesse caso, a memória não reconhece a existência da MMU e apenas enxerga uma solicitação de leitura ou escrita no endereço 8192, a qual ela executa. Assim sendo, a MMU mapeia todos os endereços virtuais de 0 a 4095 em endereços físicos localizados de 8192 a 12287 (TANENBAUM, 2015).

Videoaula 3

Agora assista o vídeo no qual será discutido sobre a técnica Paginação.

Videoaula 3

Utilize o QRcode para assistir!

Agora assista o vídeo no qual será discutido sobre a técnica Paginação.

Indicação de Vídeo

Agora assista o vídeo do professor Daniel Torrico da UNIFAL-MG que explica sobre a paginação nos computadores.

Disponível em: https://www.youtube.com/watch?v=2oFxmUCjDHo. Acesso em: 8 maio 2021.

Leitura Obrigatória

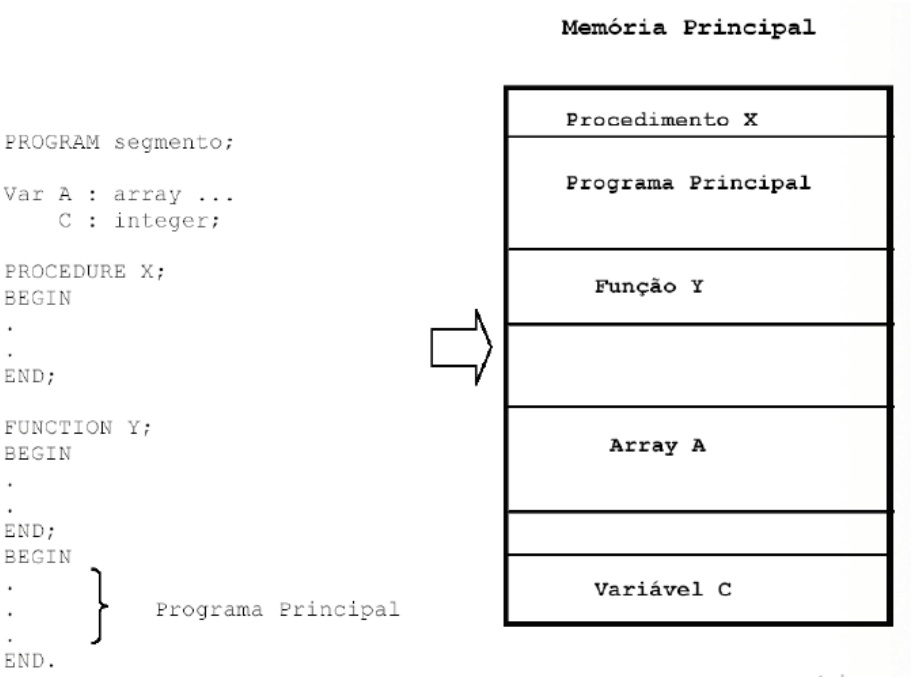
Vamos aprofundar o conhecimento a respeito da paginação. Por isso, para conhecer mais, leia o capítulo 3 do livro "**Sistemas Operacionais Modernos"** de Andrew Tanenbaum, a partir do **tópico 3.3.1** até **3.3.3.** Essa leitura com certeza vai complementar seus estudos. Acesse o *link* abaixo:

Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/1233. Acesso em: 8 maio 2021.

Segmentação

A segmentação é a técnica onde os dados e instruções de um programa são divididos em blocos chamados de segmentos. Os blocos têm tamanhos diferentes e cada um com seu próprio espaço de endereçamento, veja a figura 11 abaixo.

Figura 11 - Segmentação



Fonte: Machado; Maia (2000)

A diferença entre a paginação e a segmentação é que, a paginação divide os programas em partes de tamanho fixo, sem qualquer ligação com a estrutura do programa, enquanto que a segmentação permite uma relação entre a lógica do programa e sua divisão na memória. Sendo mais detalhista nesse conceito, podemos falar que a segmentação é um conceito lógico e não físico (MACHADO; MAIA, 2000).

Um sistema de segmentação de memória virtual deve ter a capacidade de manter na memória principal somente os segmentos de que uma aplicação necessita para ser executado em determinado instante; os demais segmentos residem em um armazenamento secundário (DEITEL *et al.*, 2005).

O mecanismo de mapeamento é muito semelhante ao da paginação. Além do endereço do segmento na memória física, cada entrada na tabela de segmentos possui informações sobre o tamanho do segmento e se ele está ou não na memória. Se as aplicações não estiverem divididas em módulos, grandes pedaços de código estarão na memória desnecessariamente, não permitindo que outros usuários também utilizem a memória.

O problema da fragmentação também ocorre nesse modelo, quando as áreas livres são tão pequenas que não acomodam nenhum dado que necessite ser carregado. Logo, temos de conviver e minimizar os problemas de fragmentação durante a alocação ou, ainda, ter programas que auxiliem na desfragmentação da memória.

Encerramento da Unidade

Chegamos ao final de mais uma unidade, caro aluno. Se chegou até aqui, foi porque tem perseverança e vontade de aprender. Não se preocupe se não assimilou o conteúdo de maneira integral, pois o aprendizado é algo progressivo e com certeza com um pouco mais de leitura tudo se encaixa.

Essa unidade foi dedicada a dois assuntos importantes para o sistema operacional: O controle do processador e a memória. Como você notou no transcorrer da unidade e da leitura obrigatória dos capítulos dos livros, que o processador e a memória não podem ser são usados de maneira desordenada, tudo tem uma técnica de controle.

No controle do processador estudamos algumas formas de escalonamento dos processos, dado que o uso do processador deve ser otimizado para que ele não seja subutilizado. Os principais tipos de escalonamento estudados foram o escalonamento **não preemptivo** e o **escalonamento preemptivo.**

Em relação a memória, vimos que se trata de um componente muito caro ainda, então sua expansão nos computadores não é algo trivial. Para solucionar esse problema, algumas técnicas que são usadas para gerenciar melhor o uso da memória principal foi apresentado no decorrer dessa unidade. Podemos destacar a técnica *SWAPPING* que é usada na maioria dos sistemas operacionais, onde se o sistema operacional leva dados da memória principal para a memória secundária e vice e versa. Outra técnica bastante relevante é a PAGINAÇÂO que também é utilizada atualmente nos computadores. Paginação é um mecanismo de gerenciamento de memória pelo qual um computador

armazena e recupera dados de um armazenamento secundário para uso na memória principal.

Bom! Espero que tenho gostado dessa unidade. É um tema muito importante na área de computação. Sabendo como funciona essas técnicas, projetos melhores de *softwares* podem ser desenvolvidos.

Um abraço a todos.

Referências

DEITEL, Harvey M. *et al*. **Sistemas Operacionais**. 3. ed. São Paulo: Pearson Education Brasil, 2005. 784 p.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais**. 2. ed. Rio de Janeiro: Ltc, 2000. 232 p.

SIQUEIRA, Fernando de. **Gerência de Memória**. Disponível em: https://sites.google.com/site/proffernandosiqueiraso/aulas/9-gerencia-de-memoria. Acesso em: 27 fev. 2021.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2015. 864 p.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR