Sistemas operacionais

Aula 4 - Gerência de memória

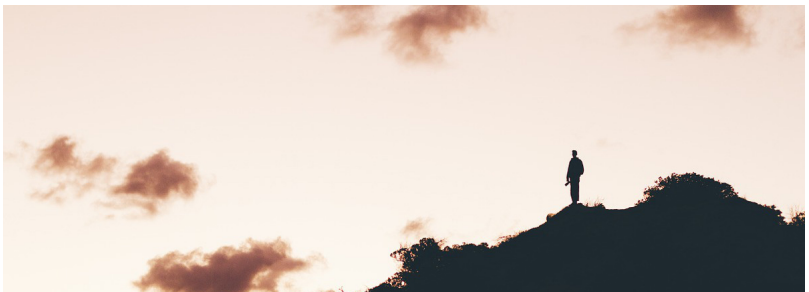
INTRODUÇÃO

A gerência de memória tornou-se um elemento importante nos sistemas de computação devido ao alto custo e a exiguidade desse recurso. Por outro lado, a necessidade de alocação de mais de um processo, na memória, nas propostas de sistemas que operam em multitarefa, tornam indispensável a existência deste gerenciamento.

Segundo a arquitetura de Von Neumann, programas e dados devem estar alocados na memória principal quando em execução, para que possam ser diretamente referenciados (acessados). O imperativo de gerenciar os espaços de memória decorre então da necessidade do sistema em saber onde estão localizadas as áreas de instruções e de dados dos processos que em execução.

Como fazer este gerenciamento é o objeto de estudo desta aula.

OBJETIVOS



Descrever a alocação particionada.

Analisar o funcionamento da paginação.

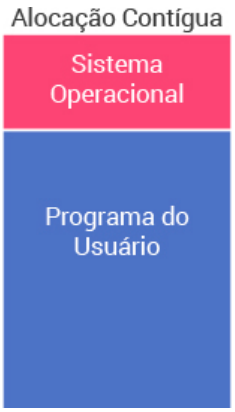
Distinguir paginação e segmentação.

GERÊNCIA DE MEMÓRIA

A gerência de memória possui duas grandes abordagens:

Manter os processos na memória principal durante toda a sua execução;
Mover os processos entre a memória principal e secundária (tipicamente disco), utilizando técnicas de *swapping* (permuta) ou de paginação.

GERENCIAMENTO SEM PERMUTA



Fonte da Imagem:

Alocação Contígua

Os primeiros sistemas implementaram uma técnica muito simples onde a memória principal disponível é dividida entre o Sistema Operacional e o programa em execução.

Como este esquema de gerenciamento é utilizado em sistemas monoprogramáveis, temos apenas um processo em execução por vez.

Nesta técnica, o tamanho máximo dos programas é limitado à memória principal disponível. Para superar esta limitação foi desenvolvido o conceito de overlay.

OVERLAY

A técnica de overlay utiliza o conceito de sobreposição, ou seja, a mesma região da memória será ocupada por módulos diferentes do processo.

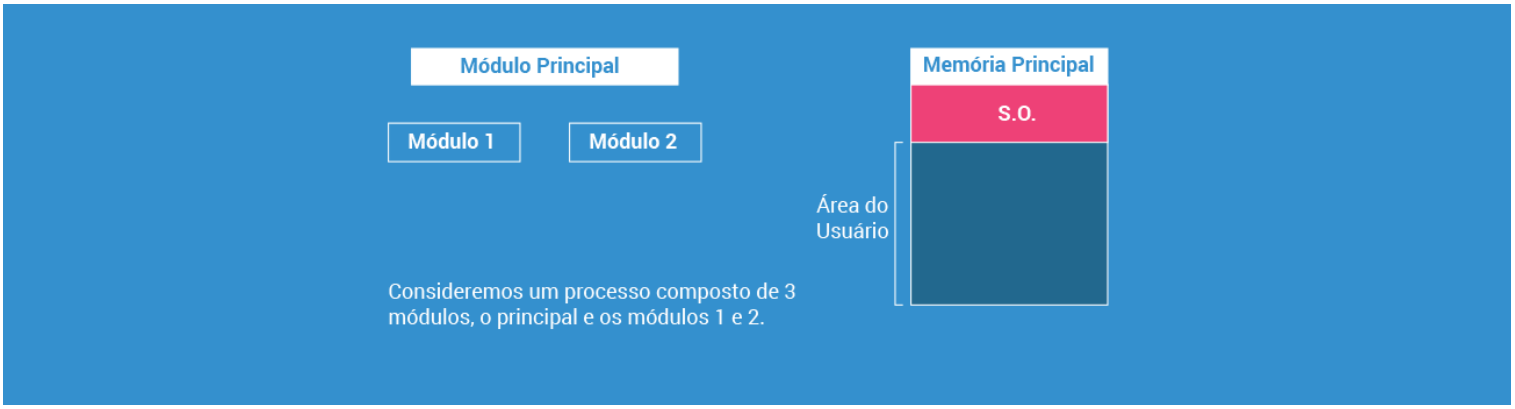
Neste caso, um programa que exceda o tamanho de memória disponível para o usuário, é particionado em segmentos de código de overlay que permanecem no disco até que sejam necessários, quando então são trazidos para memória e alocados a um espaço predeterminado pelo usuário quando da construção do módulo executável.

Esse gerenciamento é de responsabilidade do usuário (glossário) e, geralmente, é oferecido como um recurso da linguagem.

Em um programa com estrutura de overlay, um módulo fica sempre residente (Módulo Raiz) e os demais são estruturados como em uma árvore hierárquica de módulos mutuamente exclusivos, em relação a sua execução, colocados lado a lado, em um mesmo nível da árvore, de modo que o mesmo espaço possa ser alocado para mais de um módulo, os quais permanecem no disco e serão executados um de cada vez por meio de comandos de chamada (call).

Um módulo residente faz chamadas a um módulo de overlay em disco e este é carregado no espaço de overlay sobre outro módulo do mesmo nível que lá estava (daí a necessidade de serem mutuamente exclusivos).

Complicado? Veja o exemplo a seguir.

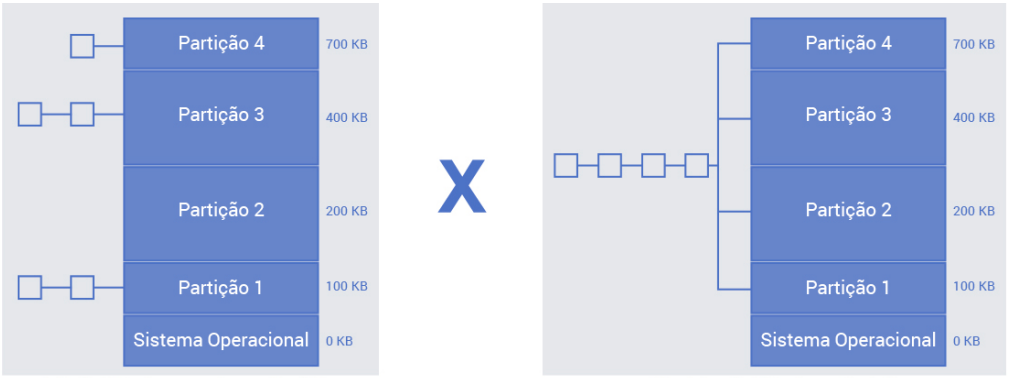


ALOCAÇÃO PARTICIONADA FIXA

Como o advento dos sistemas multitarefa surgiu, a necessidade de manter mais de um processo de usuário, na memória, ao mesmo tempo. A forma mais simples de se conseguir isso consiste em dividir a memória em *n* partições (possivelmente diferentes) e alocar os diferentes processos em cada uma delas.

Nos primeiros sistemas, estas partições eram estabelecidas na configuração do sistema operacional e seu tamanho e sua localização somente podiam ser alterados realizando um novo boot.

Quando um processo inicia, este deve ser alocado em uma partição com tamanho suficiente para acomodá-lo. Duas estratégias podem ser adotadas para alocar o processo: uma única fila de entrada (glossário) ou uma fila por partição (glossário). Observe, na imagem a seguir, um exemplo de cada estratégia.



Fila de Entrada separadas por Partição

Fila única de Entrada por Partição

Atenção!

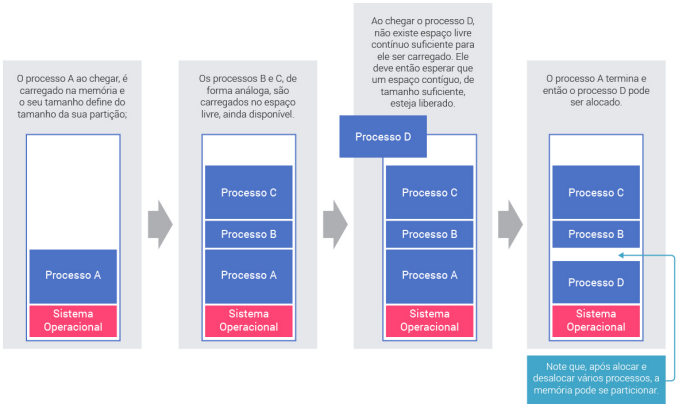
, Este método de gerência de memória baseado em partições fixas, de uma forma ou de outra, gera um grande desperdício de memória, posto que, um processo ao ser carregado em uma partição, se ele não ocupa todo o seu espaço, o restante não poderá ser utilizado por nenhum outro processo., , Para evitar esse desperdício, foi desenvolvido um esquema de gerenciamento e alocação de memória dinamicamente, dependendo da necessidade do processo. Este esquema é conhecido como alocação com partições variáveis.

ALOCAÇÃO COM PARTIÇÕES VARIÁVEIS

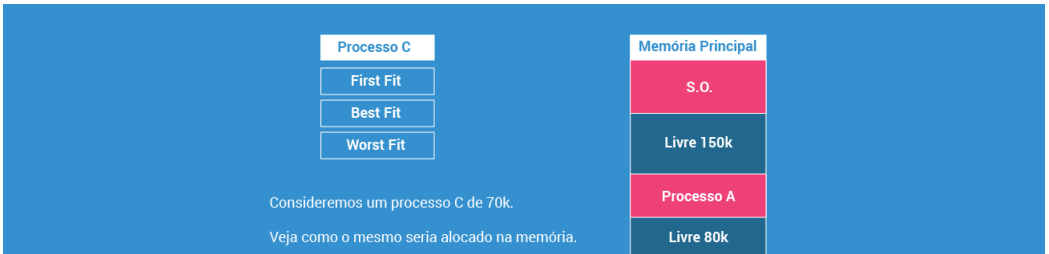
Nessa técnica, a memória principal não é particionada em blocos de tamanhos predeterminados, tal como na partição fixa. Aqui, inicialmente apenas uma partição existe e ocupa toda a memória disponível para usuários. À medida que os processos vão sendo alocados, essa partição vai diminuindo até que não caibam mais processos no espaço restante, resultando em um único fragmento.

Quando um processo termina e deixa a memória, o espaço ocupado por ele se torna uma partição livre que o gerente de memória utilizará para alocar outro processo da fila por memória. Uma lista de partições livres é mantida com ponteiros indicando os buracos disponíveis para alocação.

Para facilitar a sua compreensão, veja o exemplo a seguir:



A escolha da partição para o próximo job da fila deverá ser feita de acordo com uma das seguintes políticas:



Para nosso exemplo, consideremos um Processo C de 70k.

FIRST-FIT

Procura alocar o job na primeira partição onde ele couber(a busca termina assim que a primeira região é encontrada). É a mais rápida. Produz no nosso exemplo uma região resto de 80k.

Justificativa: probabilisticamente agrupa os jobs pequenos separando-os dos grandes.

BEST-FIT

Aloca o job na partição disponível de tamanho mais próximo ao seu (a lista de áreas livres de memória tem que ser totalmente pesquisada). Produz a menor região resto, no caso 10k.

Justificativa: busca deixar fragmentos livres menores.

WORST-FIT

Procura alocar o job na maior partição disponível onde ele couber (a lista de áreas livres de memória tem que ser totalmente pesquisada). Produz a maior região resto, no caso 130k. É a pior das três.

Justificativa: Intuitivamente sempre caberá mais um job pequeno no espaço resultante.

REALOCAÇÃO E PROTEÇÃO

As políticas de alocação de partição introduzem dois problemas essenciais que devem ser resolvidos: realocação e proteção.

Vamos entender esses problemas...

Fonte da Imagem:

Um processo, quando alocado na memória principal, faz referências a posições da memória, ou seja, a endereços físicos de memória. Por essa razão, quando um módulo foi link-editado e seus endereços já estão associados às posições físicas do espaço de memória onde ele será alocado, diz-se que esse módulo está em Imagem de Memória, ou que ele está com Endereçamento Absoluto.

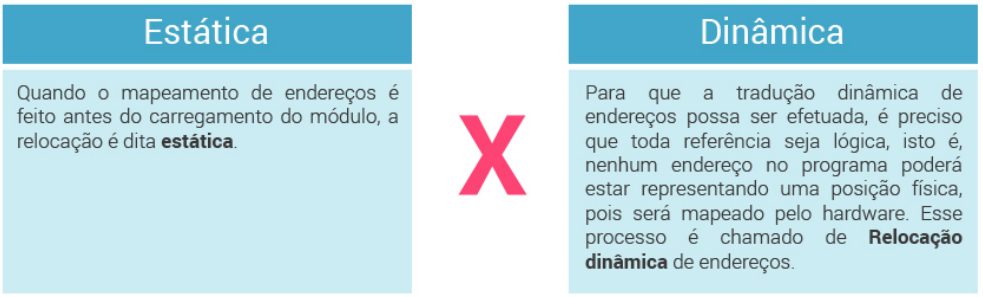
Para que um processo possa ser alocado, em qualquer posição da memória, ele não pode estar com endereçamento absoluto e, nesse caso, um mapeamento de endereços deverá ser feito entre os endereços dos objetos (glossário) referenciados pelo processo (endereços lógicos) e os endereços absolutos (físicos) no espaço que eles estarão ocupando na memória principal.

Supondo “N” o espaço de endereços lógicos e “M” o espaço de endereços físicos, então o mapeamento de endereços pode ser denotado por uma função do tipo: f: N → M

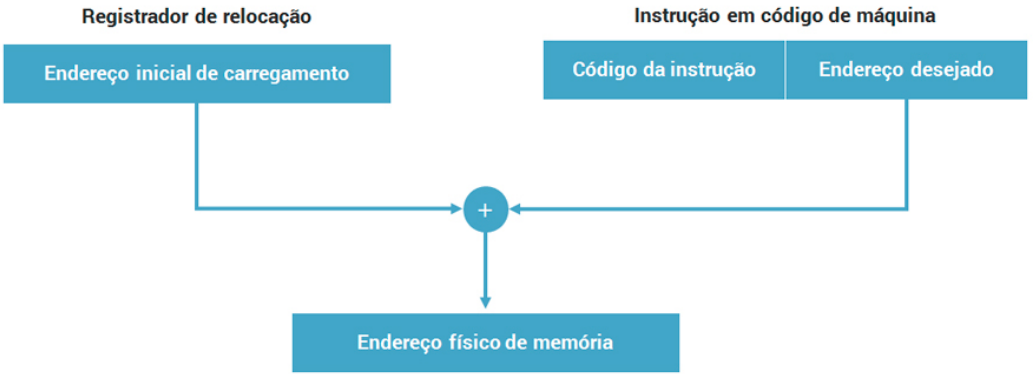
Mas, o que é a relocação de memória?

É a função que mapeia os endereços lógicos em endereços físicos. Quando ela é feita pelo link-editor, durante a resolução das referências em aberto, na geração do módulo de carga, os endereços são resolvidos em relação a uma base inicial e o processo só poderá ser alocado a partir dessa base, ou seja, sempre rodará no mesmo lugar da memória. Alguns sistemas deixam essa tarefa de relocação para o carregador (loader) ou ligador-carregador (link-loader) que faz a resolução das referências externas e a relocação de endereços no instante de carregar o processo na memória para sua execução.

Note que, no primeiro caso (link-editor), a carga em imagem de memória sempre roda no mesmo lugar da memória, enquanto que, no segundo caso, pode rodar em qualquer lugar. Assim, a relocação poderá ser:

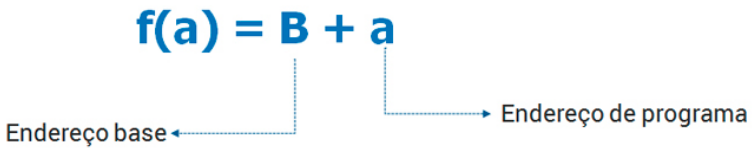


Veja o esquema a seguir:

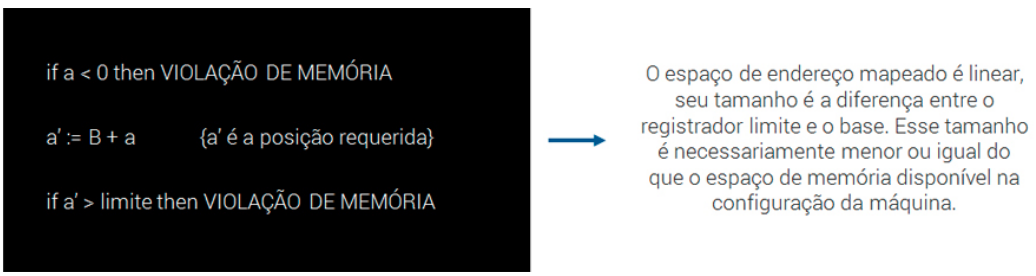


REGISTRADORES BASE E LIMITE

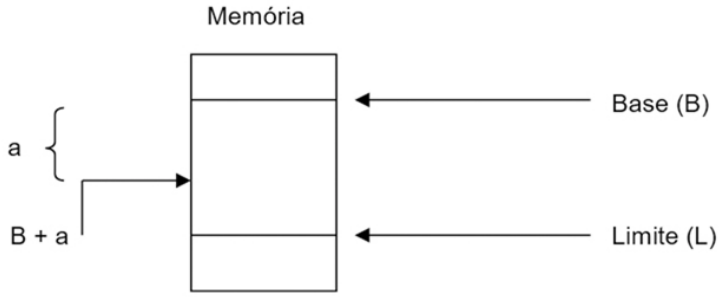
Quando um processo é carregado na memória, o endereço “inicial” é colocado em um registrador base e todos os endereços de programa são interpretados como sendo relativos ao endereço contido no registrador base. O mecanismo de transformação, neste caso, consiste em adicionar ao endereço do programa o endereço base, o que produz a localização da palavra de memória correspondente. Isto é:



Consegue-se relocação dinâmica facilmente: basta mover o programa e corrigir o conteúdo do registrador base. Proteção pode ser obtida com a inclusão de um segundo registrador (registrador limite) contendo o endereço da última posição de memória que o processo pode acessar. O mapeamento de endereço realizado pelo mecanismo (por “hardware”) de transformação é assim:



Veja a ilustração a seguir:



Atenção!

, Para reduzir o tempo de mapeamento, é conveniente que os registradores base e limite sejam de processo rápido. O custo dos registradores pode ser reduzido e a velocidade aumentada se os bits menos significativos desses registradores forem removidos. Se tal ocorrer, então o espaço de endereços deve ser múltiplo de 2n, onde *n* é igual ao número de bits removidos.

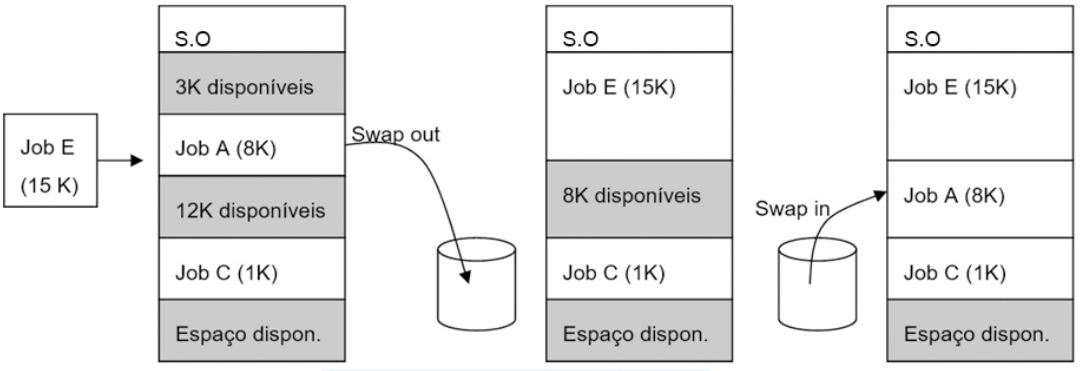
GERENCIAMENTO COM PERMUTA

Agora, conheceremos alguns conceitos ligados ao gerenciamento com permuta. Vamos lá!

Swap de memória

Denominamos swapping a política de remover um processo da memória toda vez que ele fica bloqueado. Assim que sua condição estiver satisfeita e ele retornar à fila de prontos ele é novamente carregado na memória.

Veja a situação a seguir onde temos um processo esperando por uma partição e que não consegue rodar porque não há uma partição suficientemente grande. Neste caso, um sistema que faça swapping retira o processo que está no meio de uma área livre de memória (swap out), coloca-o no disco, atribui a nova partição disponível ao processo que estava esperando e, posteriormente, atribui outra partição (swap in) ao processo que foi desalojado.



Naturalmente que, para fazer swap, é necessário haver relocação dinâmica durante a carga dos jobs.

O swap de memória possui algumas vantagens e desvantagens. São elas:

VANTAGENS	DESVANTAGENS
Maior compartilhamento da memória (maior *throughput* de tarefas); Menor fragmentação de memória; Boa técnica para ambientes com processos pequenos e poucos usuários.	Custo alto para fazer o swap, por ser uma operação com memória auxiliar (disco).

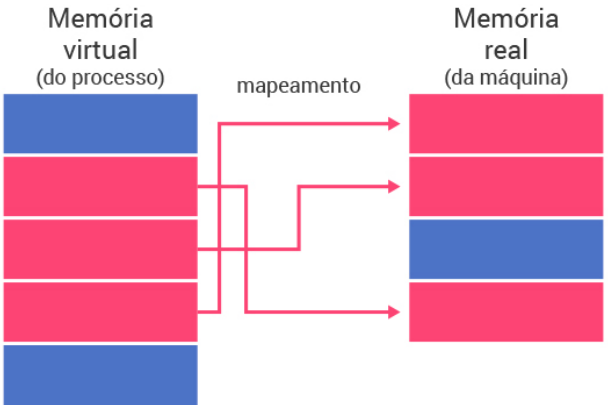
, Exemplo de S.O. que usou esta técnica: OS/360 da IBM.

Memória virtual

Como vimos anteriormente, o espaço de endereçamento lógico de um programa tem que ser mapeado em um espaço de endereçamento físico de

memória. De fato, essa correspondência pode ser deixada totalmente a cargo do S.O., de forma que o programa não tenha nenhuma responsabilidade em se ater a um determinado tamanho de memória física disponível. Assim, o conjunto de endereços que um programa pode endereçar, pode ser muito maior que a memória física disponível para o processo, em um determinado momento, ou mesmo que o total de memória fisicamente disponível na máquina. A esse conjunto de endereço chamamos de espaço de *endereçamento virtual*. Ao conjunto de endereços reais de memória chamamos espaço de *endereçamento real*.

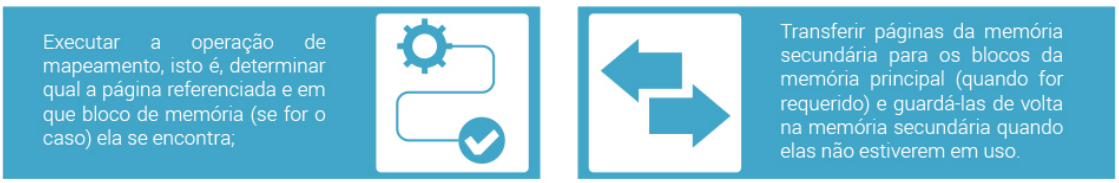
Como os programas podem ser maiores que a memória física, somente uma parte de cada programa pode estar na memória durante a execução. As partes que não são necessárias em um determinado instante, ficam em disco e só são carregadas quando se tornarem necessárias. Todo o processo é transparente para o usuário e mesmo para os compiladores e link-editores, pois estes trabalham apenas com o espaço de endereçamento virtual. Apenas o S.O. se incube de carregar ou descarregar as partes necessárias e mapeá-las no espaço de endereçamento real durante a execução.



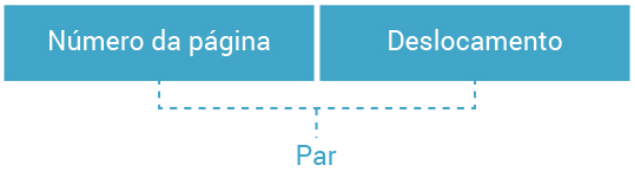
Como consequência desse mapeamento o programa não precisa estar nem mesmo em regiões contíguas de memória. Naturalmente, o nível de fracionamento do programa deve ser escolhido de forma a não comprometer o desempenho, à medida que a tarefa de mapeamento é feita pelo S.O., juntamente com recursos de hardware. Assim, a memória (tanto virtual como real) é dividida em blocos e o S.O. mantém tabelas de mapeamento para cada processo, que relacionam os blocos da memória virtual do processo com os blocos da memória real da máquina.

Paginação

Na paginação, o espaço de endereços é dividido em blocos de igual tamanho que chamamos de páginas. A memória principal também é dividida em blocos de mesmo tamanho (igual ao tamanho da página) denominados molduras.
Esses blocos de memória real são compartilhados pelos processos, de forma que a qualquer momento, um determinado processo terá algumas páginas residentes na memória principal (as páginas ativas), as restantes na memória secundária (as páginas inativas). O mecanismo da paginação possui duas atribuições:

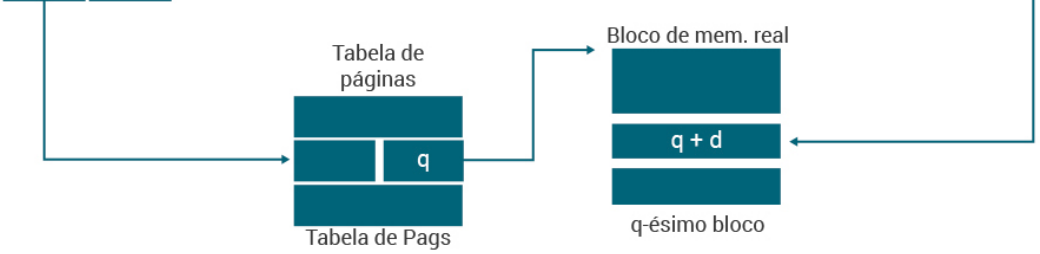


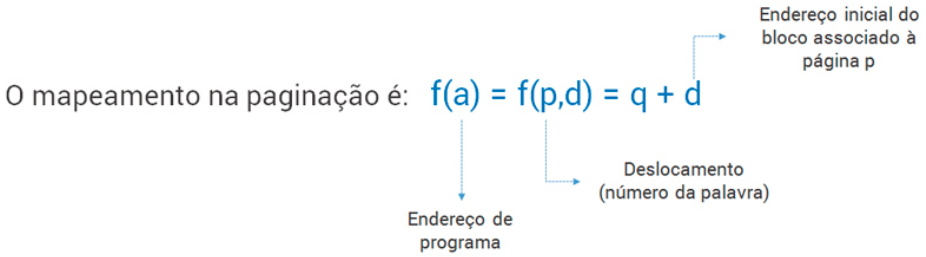
Com o objetivo de determinar qual a página referenciada por um endereço de programa, interpreta-se os bits mais significativos do endereço virtual como sendo o número da página dentro de uma tabela de páginas que é mantida pelo S.O. para cada processo ativo. Já os bits menos significativos indicam a localização da palavra na página selecionada:



Se o tamanho da página é 2^n palavras, então, os *n* bits menos significativos do endereço de programa representam o deslocamento, e os bits restantes indicam o número da página. O número total de bits do endereço de programa é suficiente para endereçar inteiramente a memória virtual.

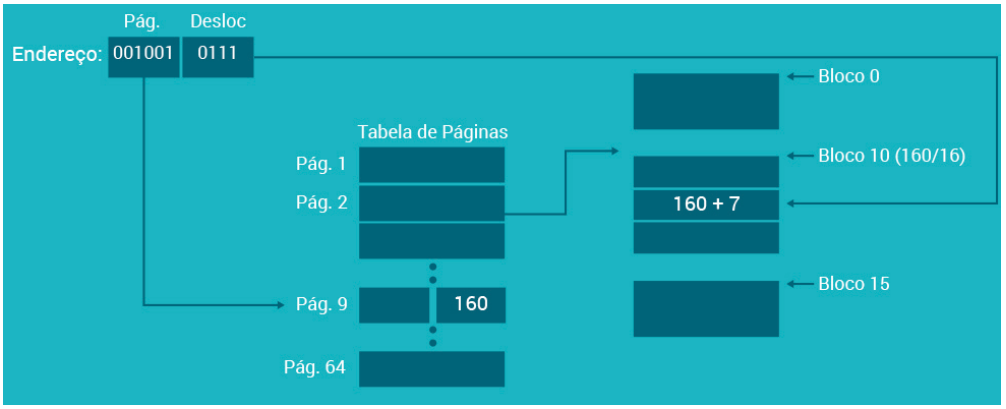
A divisão de um endereço de programa no par (número de página, deslocamento) é feita pelo hardware e é completamente transparente ao programador. Para transformar o par no endereço de uma posição de memória, o mecanismo conta com a tabela de páginas: a p-ésima entrada da tabela, aponta para o bloco “q” da memória que contém a página de número “p”. O número da palavra “d” é somado a “q”, formando o endereço.





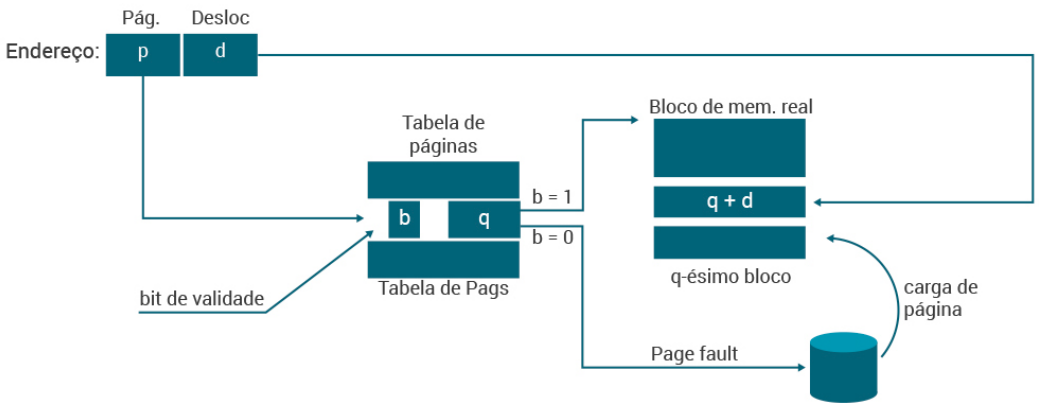
Se chamarmos de Z o tamanho da página, então: “p” é o quociente e “d” o resto da divisão de “a” por Z.

, Vamos supor que tivéssemos:
Páginas com 16 endereços, logo, 4 bits de endereçamento;
Endereços virtuais de até 10 bits, logo, 1024 endereços;
Memória física de 256 endereços.
, , Então, se quiséssemos acessar o endereço binário 0010010111 a composição, pelo esquema acima, seria:
Os quatro últimos bits (0111) são o deslocamento *d* dentro da página;
Os seis primeiros bits (001001) são o número da página.
, , Como temos 1024 endereços e páginas de 16 endereços, então temos 1024/16= 64 páginas na tabela de páginas., , Como temos 256 endereços reais, então temos 256/16 = 16 blocos físicos de memória (blocos de 0 a 15)., ,



Page fault

Normalmente, o número de blocos alocados a um processo é menor do que o número de páginas que ele usa. Por isso é possível que um endereço de programa referencie uma página ausente. Nesse caso, a entrada (da tabela) correspondente estará “vazia” e uma interrupção (glossário) do tipo *falta de página* (page fault) é gerada sempre que uma página inativa é requerida. Observe a representação a seguir:



Entendeu o que aconteceu? Vamos explicar...

A interrupção faz com que o mecanismo da paginação inicie a transferência da página ausente da memória secundária para um dos blocos da

memória principal e atualize a tabela de páginas. O processo muda de estado, ficando bloqueado até que a transferência chegue ao fim.

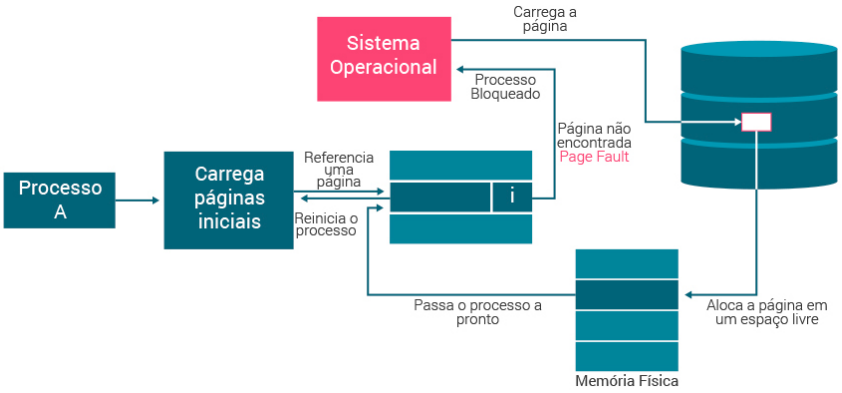
As páginas são transferidas da memória secundária para a principal apenas quando são referenciadas. Esse mecanismo é chamado de paginação por demanda. Em outra técnica, conhecida como paginação antecipada, o sistema tenta prever quais páginas serão referenciadas pelo programa, trazendo-as para a memória antecipadamente, evitando assim a ocorrência do page fault.

A posição ocupada por uma página na memória secundária é guardada em uma tabela separada ou na própria tabela de páginas. No último caso, é necessário um "bit de presença" para cada entrada da tabela de páginas, indicando se a página está ativa (está na memória física) ou não.

Se não houver nenhum bloco de memória disponível então é necessário desocupar um dos blocos para a página cuja presença está sendo solicitada. A escolha do bloco é função do algoritmo de troca (glossário), conforme veremos.

A transformação de endereços é função do hardware, enquanto que o algoritmo de troca é feito pelo software.

O esquema a seguir mostra esse processo:



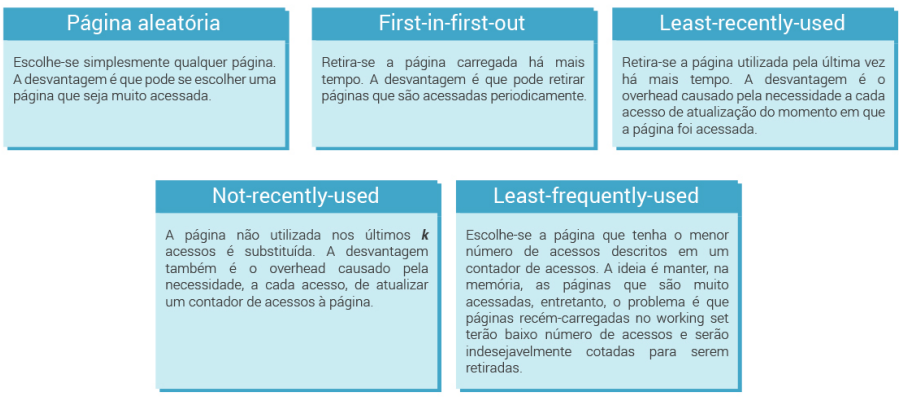
Políticas de paginação

Para minimizar a ocorrência de page faults surgiu o conceito de working set. Define-se workink set como o conjunto de páginas mais acessadas por um determinado processo. Assim, quando um programa começa a executar, a possibilidade de que ele requisite páginas que não estão na memória é muito grande. Entretanto, à medida que mais páginas vão sendo carregadas, a ocorrência de page faults vai diminuindo, devido ao princípio da localidade dos programas, já explicado quando falamos de memória cache.

Assim, cabe ao S.O. definir um conjunto de páginas que devem ficar carregadas, sabendo que se trata de um compromisso entre capacidade e velocidade:

Definido o working set, resta definir qual deve ser a página a retirar da memória quando ocorre um page fault e uma página precisa ser carregada. Esta decisão pode obedecer a várias políticas, como veremos a seguir. Entretanto, antes de descartar qualquer página, o S.O. precisa saber se houve alguma alteração na página enquanto ela esteve na memória. Se houve, então ela precisa ser salva em disco antes de ser descartada, para que nenhum dado se perca. Para tanto, o S.O. mantém um bit na tabela de páginas chamado *bit de modificação*. Caso o bit indique que houve mudança, a página é salva em um arquivo de paginação, onde poderá ser futuramente resgatada.

As políticas de liberação de páginas são:



Existem vantagens e desvantagens no uso da paginação. São elas:

VANTAGENS	DESVANTAGENS
Aumenta o espaço de endereçamento do processo;	Overhead devido ao excesso de page faults tratados pelo sistema;
Admite código reentrante e compartilhamento de código pelos processos.	Ocupa um arquivo só para E/S de páginas;
	Ocupa muita memória do S.O. para as tabelas de páginas dos processos.

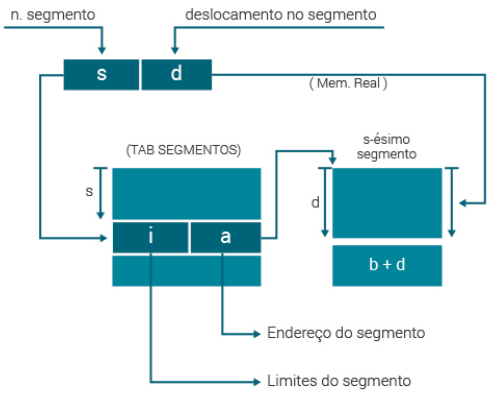
Administração de Memória por Segmentação

Devido ao grande número de page faults da paginação, a segmentação surgiu como uma alternativa de gerência de memória onde o programa agora não mais é dividido em blocos de comprimentos fixos (as páginas), mais sim, em segmentos de comprimentos variados, mas com um sentido lógico, isto é, a paginação procedia a uma divisão física do programa e, às vezes, isso leva a um número muito grande de page faults por não observar suas características lógicas.

A segmentação busca aproveitar exatamente essas características, agrupando em um segmento partes do programa que se referenciam mutuamente e que quando trazidas à memória estarão todas juntas não causando, o que nesse caso é chamado de segment faults, com tanta frequência.

Nesse esquema, o espaço de endereços torna-se bidimensional: endereços de programa são denotados pelo par (nome do segmento, endereço dentro do segmento). Para facilitar a implementação do mecanismo de transformação de endereços, o S.O. troca o nome do segmento por um nº, quando o segmento é referenciado pela primeira vez.

Considerando o par (s,d) como sendo um endereço de programa generalizado, onde s = número do segmento e d = endereço dentro do segmento, então o esquema de geração seria:



Conforme podemos ver, a transformação de endereço é realizada por intermédio de uma tabela de segmentos (uma tabela para cada processo). A s-ésima entrada da tabela contém o tamanho (l) e a posição inicial (a) da memória, onde o s-ésimo segmento foi carregado.

As entradas da tabela são chamadas de “descritores de segmento”. O algoritmo de mapeamento é:

extrair endereço de programa (s,d);
usar “s” para indexar tabela de segmentos;
retirar o endereço inicial do segmento (a);
se d < 0 ou d > l então “violação de memória”;
a + d é o endereço requerido.

A busca de espaços para um segmento a ser trazido à memória nos leva novamente aos problemas de gerência por partições variáveis. A diferença aqui é que o espaço contíguo só é necessário para o segmento e não para todo o processo.

A segmentação possui algumas vantagens e desvantagens. São elas:

VANTAGENS	DESVANTAGENS
Reduz o número de page faults dentro de um segmento;	A soma de s + d, na tradução de endereços, é mais lenta que a concatenação ***p*** e ***d*** na

Aproveita as características lógicas do processo;

Dispensa uso de arquivo de paginação (ou segmentação).

paginação.

A administração das áreas livres de memória é mais complicada (partições variáveis) e, portanto mais lenta;

As page faults (tratadas mais rapidamente, pois as páginas eram menores) são substituídas por segment faults que são maiores em comprimento.

, Sistemas que usam segmentação pura: B-6700 e PD- 11/45.

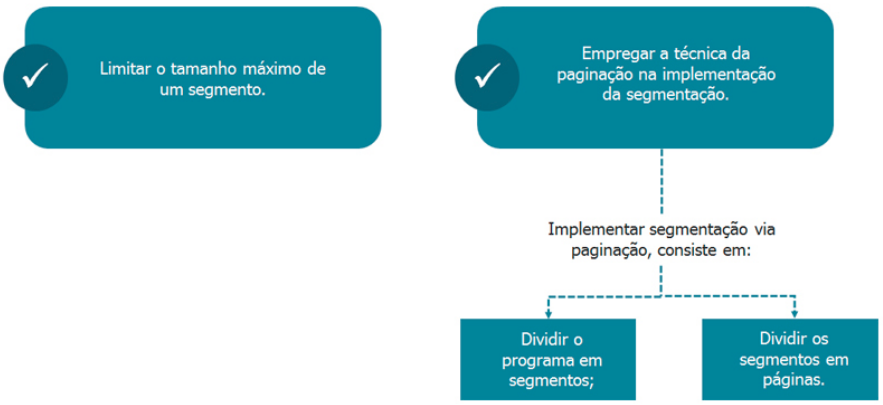
Você deve ter notado que a paginação e a segmentação possuem algumas similaridades. Mas, você percebeu qual é a diferença entre elas? Com base no que estudamos até aqui, responda a pergunta: Qual é a principal diferença entre segmentação e paginação?

Resposta Correta

Administração por Segmentação com Paginação

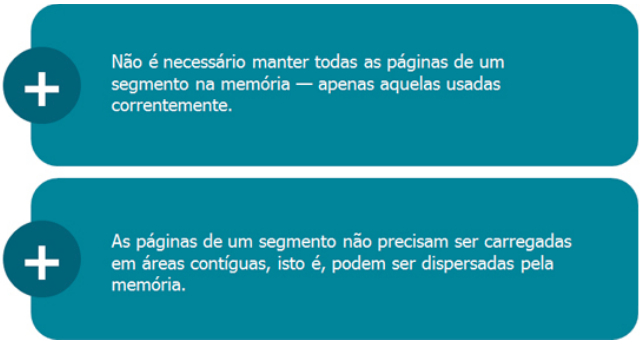
Na administração por segmentos, nem sempre é possível manter todos os segmentos na memória principal. Segmentos muito grandes poderiam, eventualmente, ultrapassar a memória disponível da máquina. Então, o que fazer?

Nesse caso há duas saídas:



Em tempo de execução, alguns trechos de um segmento estariam na memória principal e outros não. A entrada da tabela de segmentos, isto é, o descritor de um segmento, apontaria para uma tabela de páginas (do segmento) ou então estaria vazia.

Vantagens do uso de paginação na implementação da segmentação:



ATIVIDADE

Faremos mais uma atividade utilizando o Sosim, sistema que já utilizamos nas atividades das aulas passadas.

Clique aqui (glossário) para acessar a atividade.

Após realizar a atividade, clique aqui (glossário) e compare com o gabarito.

Glossário

RESPONSABILIDADE DO USUÁRIO

A técnica de overlay foi o primeiro uso do conceito de mover partes do processo, entre o disco e a memória principal, que acabaria por gerar as modernas técnicas de gerenciamento de memória onde a divisão dos módulos é totalmente transparente para o usuário.

UMA ÚNICA FILA DE ENTRADA

Todos os processos ficam na mesma fila e vão sendo alocados na menor partição livre que os possa acomodar.

UMA FILA POR PARTIÇÃO

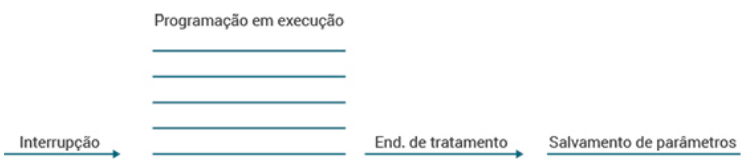
Os processos são divididos em várias filas de acordo com o seu tamanho e alocados quando a partição atendida pela fila está disponível.

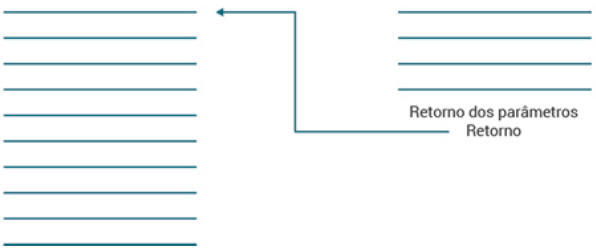
ENDEREÇOS DOS OBJETOS

Ao conjunto de endereços que um processo faz referências, sejam eles de dados ou de instruções, chamamos de Espaço de Endereçamento. Assim, definimos por Espaço Lógico de Endereçamento ao conjunto de objetos ou endereços lógicos por ele referenciados e por Espaço Físico de Endereçamento ao conjunto de endereços físicos correspondentes.

INTERRUPÇÃO

Uma interrupção é um evento externo que faz o processador parar a execução do programa corrente e desviar a execução para um bloco de código chamado *rotina de interrupção*. Ao terminar o tratamento de interrupção o controle retorna ao programa interrompido, exatamente, no mesmo estado em que estava quando ocorreu a interrupção.





Veremos mais a respeito de interrupções na próxima aula.

ALGORITMO DE TROCA

As informações requeridas pelo algoritmo estão contidas na tabela de página e poderiam ser, por exemplo:
Contador de acessos à página;
Quando a página foi acessada pela última vez;
Se a página foi alterada ou não.