Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação – IESTI
CCO 004 – Sistemas Operacionais – Prof. Edmilson Marmo Moreira

Capítulo 7 – Gerência de Memória

"Os homens sábios ensinaram-nos que não basta escolher entre os males o menor; mas também tirar deles todo o bem que possam conter."

Cícero

5.1 Introdução

A memória principal sempre foi vista como um recurso escasso e caro. O objetivo sempre foi desenvolver sistemas operacionais que não ocupassem muito espaço e, ao mesmo tempo, otimizassem a utilização dos recursos computacionais.

Nos sistemas monoprogramáveis a gerência da memória não é uma tarefa muito complexa, entretanto nos sistemas multiprogramáveis ela é crítica, devido à necessidade de se maximizar o número de usuários e aplicações utilizando eficientemente o espaço da memória principal.

Funções Básicas

O processador somente executa instruções localizadas na memória principal, assim, o sistema operacional deve sempre transferir programas da memória secundárias para a memória principal antes de serem executados.

Como o tempo de acesso à memória secundária é muito superior ao tempo de acesso à memória principal, o sistema operacional deve buscar reduzir o número de operações de E/S à memória secundária.

A gerência de memória deve tentar manter na memória principal o maior número de processos residentes, permitindo maximizar o compartilhamento do processador e demais recursos computacionais.

Mesmo na ausência de espaço livre, o sistema deve permitir que novos processos sejam aceitos e executados. Isso é feito através da transferência temporária de processos residentes na memória principal para a memória secundária, liberando espaço para novos processos.

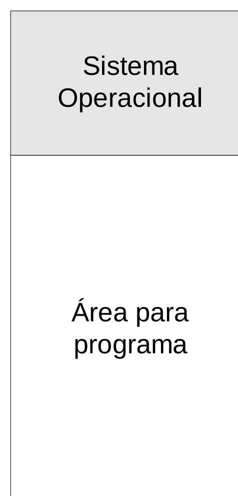
Além disso, a gerência de memória deve permitir a execução de programas que sejam maiores que a memória física disponível (*overlay* e memória virtual).

Outra preocupação da gerência de memória é a proteção das áreas de memória ocupadas por cada processo, além da área onde reside o próprio sistema operacional. Caso um processo tente realizar algum acesso indevido à memória, o sistema deve impedi-lo.

5.2 Alocação Contígua Simples

A Alocação Contígua Simples foi implementada nos primeiros sistemas operacionais, porém ainda está presente em alguns sistemas monoprogramáveis.

Nesse tipo de organização, a memória principal é subdividida em duas áreas: uma para o sistema operacional e outra para o programa do usuário.



Nesse esquema, o usuário tem controle sobre toda a memória principal, podendo ter acesso a qualquer posição da memória, inclusive a área do sistema operacional.

Para proteger o sistema desse tipo de acesso, que pode ser intencional ou não, alguns sistemas implementam proteção através de um registrador que delimita as áreas do sistema operacional e do usuário.

Apesar da fácil implementação e do código reduzido, a alocação contígua simples não permite a utilização eficiente dos recursos computacionais,

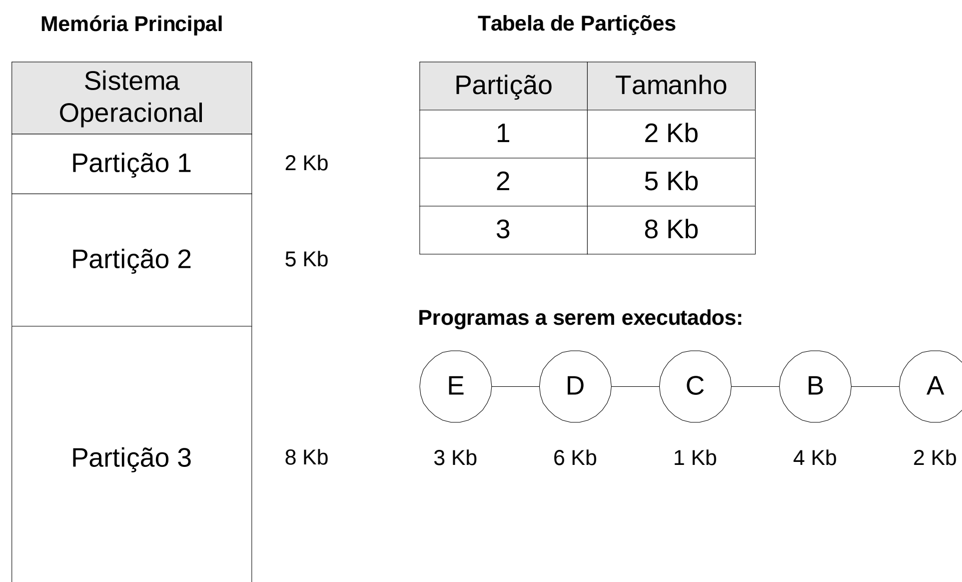
pois apenas um usuário pode dispor desses recursos. Em relação a memória principal, caso o programa do usuário não a preencha totalmente, existirá um espaço de memória livre sem utilização.

5.3 Partições Fixas

Nos primeiros sistemas multiprogramáveis, a memória era dividida em pedaços de tamanho fixo, denominado partições. O tamanho das partições era definido na inicialização do sistema.

Sempre que fosse necessária a alteração do tamanho de uma partição, o sistema deveria ser desativado e reinicializado com um nova configuração.

Esse tipo de gerenciamento de memória é também conhecido como alocação particionada estática.



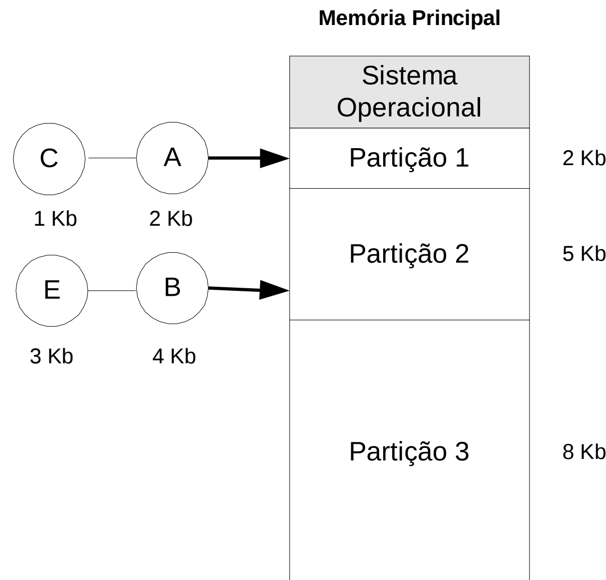
Partição	Tamanho
1	2 Kb
2	5 Kb
3	8 Kb

Inicialmente, os programas só podiam ser carregados e executados em apenas uma partição específica, mesmo se outras estivessem disponíveis. Essa limitação se devia aos compiladores e montadores, que geravam apenas código absoluto.

No código absoluto, todas as referências a endereços no programa são posições físicas na memória principal, ou seja, o programa só poderia ser carregado a partir do endereço de memória especificado no seu próprio código.

Exemplo:

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



Esse tipo de gerência de memória era conhecida como alocação particionada estática absoluta.

Com a evolução dos compiladores, o código gerado deixou de ser absoluto para ser relocável. No código relocável, todas as referências a endereços no programa são relativas ao início do código e não a endereços físicos de memória.

Assim, no exemplo anterior, caso os programas A e B terminassem, o programa C poderia ser executado em qualquer uma das duas partições. Esse tipo de gerência de memória é denominado alocação particionada estática relocável ou partições fixas relocáveis.

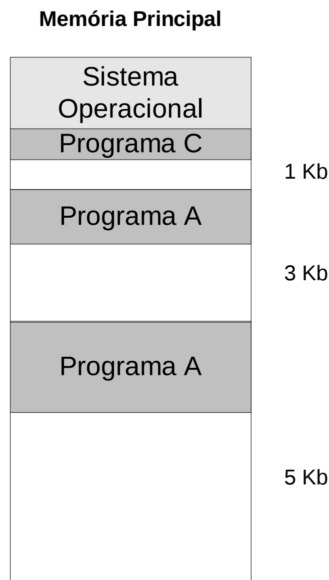
Para manter o controle sobre quais partições estão ocupadas, a gerência de memória mantém uma tabela com o endereço inicial de cada partição, seu tamanho, e se está em uso. Sempre que um programa é carregado para a memória, o sistema percorre a tabela, na tentativa de localizar uma partição livre, onde o programa possa ser carregado.

Tanto nos sistemas de alocação absoluta quanto nos de alocação relocável, os programas, normalmente, não preenchem totalmente as partições onde são carregados.

partições, estabelecidoA Alocação Contígua Simples foi implementada nos primeiros sistemas operacionais, porém ainda está presente em alguns sistemas monoprogramáveis.

Exemplo:

Os programas C, A e E não ocupam integralmente o espaço das partições onde estão alocados, deixando 1 Kb, 3 Kb e 5 Kb de áreas livres, respectivamente.



Esse tipo de problema, decorrente da alocação fixa das partições, é conhecido como fragmentação interna.

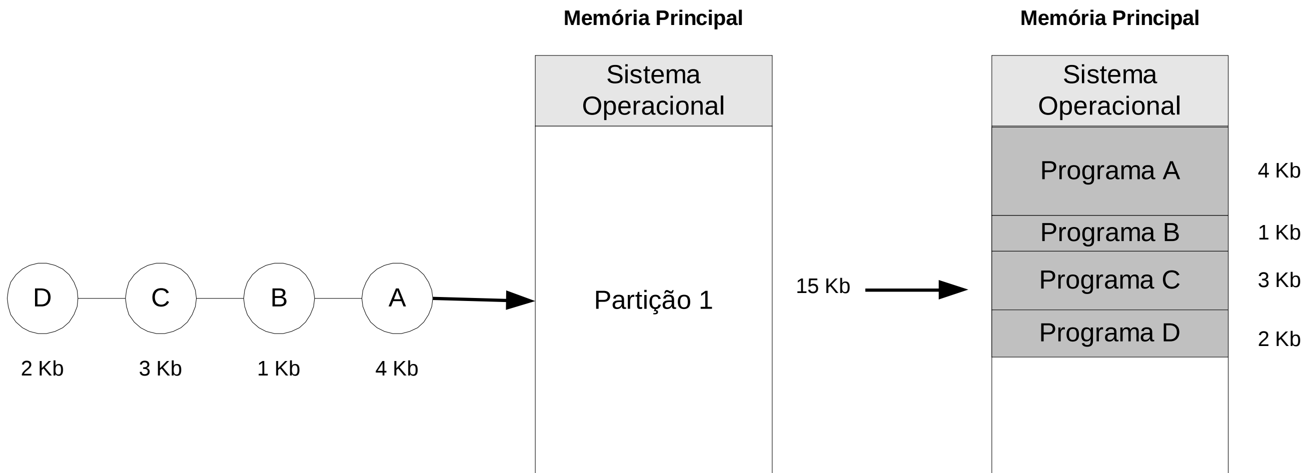
Um exemplo de sistema operacional que implementou esse tipo de gerência de memória é o OS/MFT da IBM.

5.4 Partições Variáveis

A alocação particionada estática deixou evidente a necessidade de uma nova forma de gerência de memória principal, onde o problema da fragmentação interna fosse resolvido.

O mecanismo de partições variáveis ou alocação particionada dinâmica eliminou o conceito de partições de tamanho fixo.

Nesse esquema, cada programa utiliza o espaço necessário, tornando essa área a sua partição.

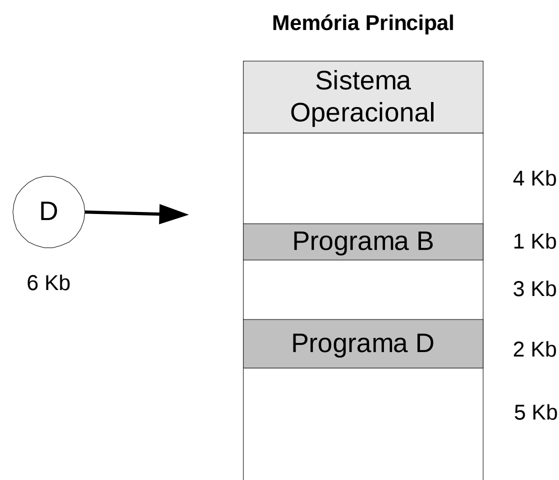


Como os programas utilizam apenas o espaço de que necessitam não ocorre fragmentação interna.

Entretanto, nesse tipo de gerência outro tipo de fragmentação pode ocorrer. Ela acontece quando os programas forem terminando e deixando espaços cada vez menores na memória, não permitindo o ingresso de novos programas.

Exemplo:

No exemplo abaixo, mesmo existindo 12 Kb livres de memória principal, o programa D, que necessita de 6 Kb de espaço, não poderá ser carregado para execução, pois este espaço não está disposto contiguamente.



Esse tipo de problema é chamado fragmentação externa.

Existem duas soluções para o problema da fragmentação externa. A primeira solução consiste em reunir os espaços adjacentes conforme os

programas forem terminando. No exemplo anterior, caso o programa B termine um espaço de 8 Kb será criado.

A segunda solução envolve a relocação de todas as partições ocupadas, eliminando todos os espaços entre elas e criando uma única partição contígua. Esse mecanismo de compactação, também conhecido como alocação particionada dinâmica com relocação, reduz o problema da fragmentação, porém a complexidade do seu algoritmo e o consumo de recursos do sistema, como processador e área em disco, podem torná-lo inviável.

Um exemplo de sistema operacional que implementou esse tipo de sistema de memória é o OS/MVT da IBM.

Estratégias de Alocação de Partição

Os sistemas operacionais implementam, basicamente, três estratégias para determinar em qual área livre um programa será carregado para execução.

As estratégias tentam diminuir ou evitar o problema da fragmentação externa.

A melhor estratégia a ser adotada por um sistema depende de uma série de fatores, sendo o mais importante o tamanho dos programas processados no ambiente.

- **Best-Fit:** Nessa estratégia, a partição escolhida é aquela em que o programa deixa o menor espaço sem utilização. Nesse algoritmo, a lista de áreas livres está ordenada por tamanho, diminuindo o tempo de busca por uma área desocupada.

- **Worst-Fit:** A partição escolhida é aquela em que o programa deixa o maior espaço sem utilização. Apesar de utilizar as maiores partições, a técnica de worst-fit deixa espaços livres maiores que permitem a um maior número de programas utilizar a memória.

- **First-Fit:** É escolhida a primeira partição livre de tamanho suficiente para carregar o programa. Nesse algoritmo, a lista de áreas livres está ordenada crescentemente por endereços. A vantagem dessa técnica é a rapidez na escolha da partição, consumindo menos recursos do sistema. Uma variação dessa técnica é a **Next-fit** que

seleciona a primeira partição livre de tamanho suficiente, mas inicia a procura pelo ponto interrompido pela última busca.

5.5 *Swapping*

Mesmo com o aumento da eficiência da multiprogramação e, particularmente, da gerência de memória, muitas vezes um programa não podia ser executado por falta de uma partição livre disponível.

A técnica de *swapping* foi introduzida para contornar o problema da insuficiência de memória principal.

Em todos os esquemas apresentados anteriormente, um processo permanecia na memória principal até o final da sua execução, inclusive nos momentos em que esperava por um evento, como um operação de leitura ou gravação.

O *swapping* é uma técnica aplicada à gerência de memória para programas que esperam por memória livre para serem executados.

Nesta situação, o sistema escolhe um processo residente, que é transferido da memória principal para a memória secundária (*swap out*), geralmente disco.

Posteriormente, o processo é carregado de volta da memória secundária para a memória principal (*swap in*) e pode continuar sua execução como se nada tivesse ocorrido.

Os primeiros sistemas operacionais que implementaram esta técnica surgiram na década de 1960, como o CTSS do MIT e OS/360 da IBM. Com a evolução dos sistemas operacionais, novos esquemas de gerência de memória passaram a incorporar a técnica de *swapping,* como a gerência de memória virtual.

5.5 Gerência de Memória Virtual

Memória Virtual é uma técnica sofisticada e poderosa de gerência de memória, onde as memórias principal e secundária são combinadas, dando ao usuário a ilusão de existir uma memória muito maior que a capacidade real da memória principal.

O conceito de memória virtual fundamenta-se em não vincular o endereçamento feito pelo programa dos endereços físicos da memória principal. Desta forma, programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços associados à memória secundária.

Em adição, a memória virtual permite um número maior de processos compartilhando a memória principal, já que apenas partes de cada processo estarão residente. Isto leva a uma utilização mais eficiente do processador.

Outra vantagem dessa técnica é a minimização do problema da fragmentação da memória principal.

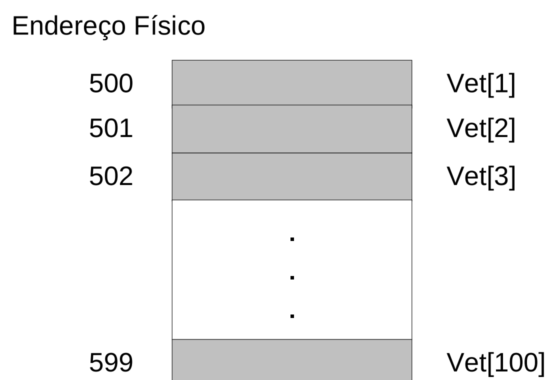
A primeira implementação de memória virtual foi realizada no início da década de 1960, no sistema Atlas, desenvolvido na Universidade de Manchester. Posteriormente, a IBM introduziu este conceito comercialmente na família System/370 em 1972.

Atualmente, a maioria dos sistemas implementa memória virtual, com exceção de alguns sistemas operacionais de supercomputadores.

Existe um forte relacionamento entre a gerência da memória virtual e a arquitetura de hardware do sistema computacional. Por motivos de desempenho, é comum que algumas funções da gerência de memória virtual sejam implementados diretamente no hardware.

Espaço de Endereçamento Virtual

O conceito de memória virtual se aproxima muito da idéia de um vetor. Quando um programa faz referência a um elemento do vetor, não há preocupação em saber a posição de memória daquele dado.



O compilador se encarrega de gerar instruções que implementam esse mecanismo, tornando-o totalmente transparente ao programador.

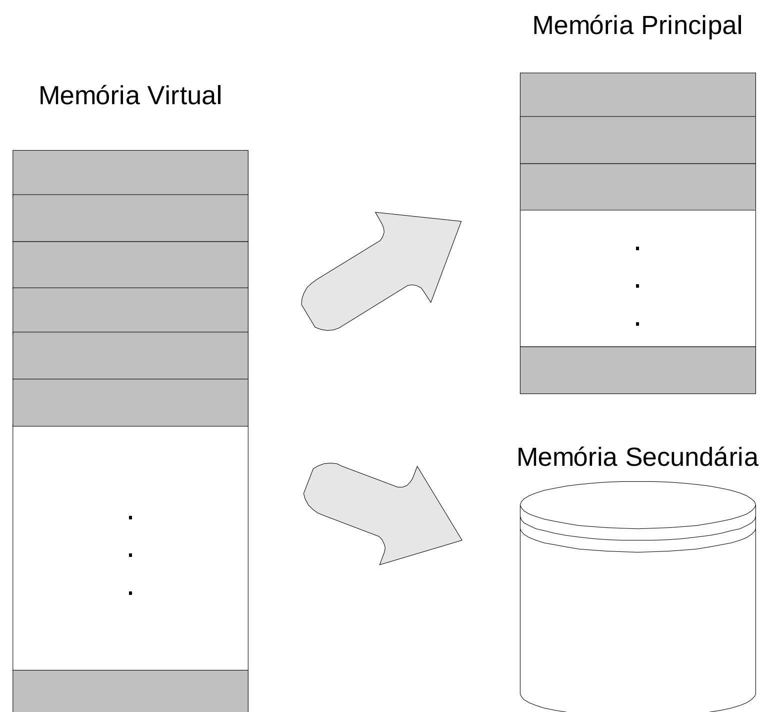
A memória virtual utiliza abstração semelhante, só que em relação aos endereços dos programas e dados. Um programa no ambiente de memória virtual não faz referência a endereços físicos de memória (endereços reais), mas apenas a endereços virtuais.

No momento da execução de uma instrução, o endereço virtual referenciado é traduzido para um endereço físico, pois o processador manipulado apenas posições da memória principal.

O mecanismo de tradução do endereço virtual para endereço físico é denominado mapeamento.

Um processo, conforme apresentado, é formado pelo contexto de hardware, contexto de software e pelo espaço de endereçamento. Em ambientes que implementam memória virtual, o espaço de endereçamento do processo é conhecido como espaço de endereçamento virtual e representa o conjunto de endereços virtuais que o processo pode endereçar.

Analogamente, o conjunto de endereços reais que o processador pode referenciar é chamado de espaço de endereçamento real.

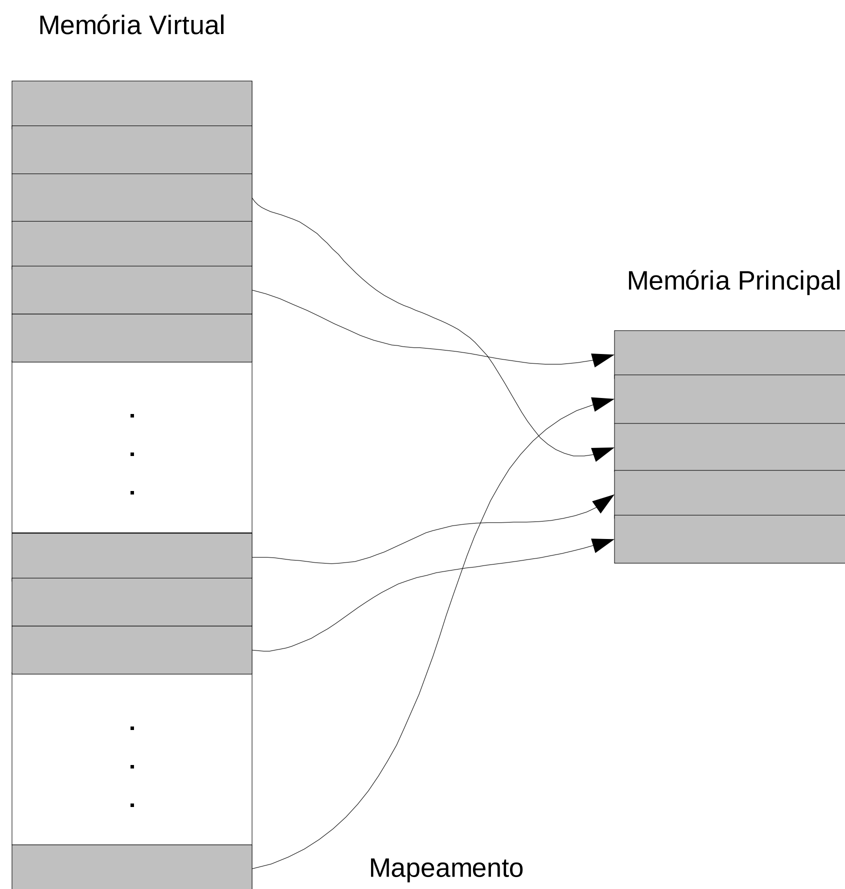


Como o espaço de endereçamento virtual não tem nenhuma relação direta com os endereços no espaço real, um programa pode fazer referência a endereços virtuais que estejam fora dos limites da memória principal, ou seja, os programas e suas estruturas de dados não estão mais limitados ao tamanho da memória física disponível.

Mapeamento

O processador apenas executa instruções e referencia dados residentes no espaço de endereçamento real; portanto, deve existir um mecanismo que transforme os endereços virtuais em endereços reais. Esse mecanismo é conhecido como mapeamento.

Como conseqüência do mapeamento, um programa não mais precisa estar necessariamente em endereços contíguos na memória principal para ser executado.

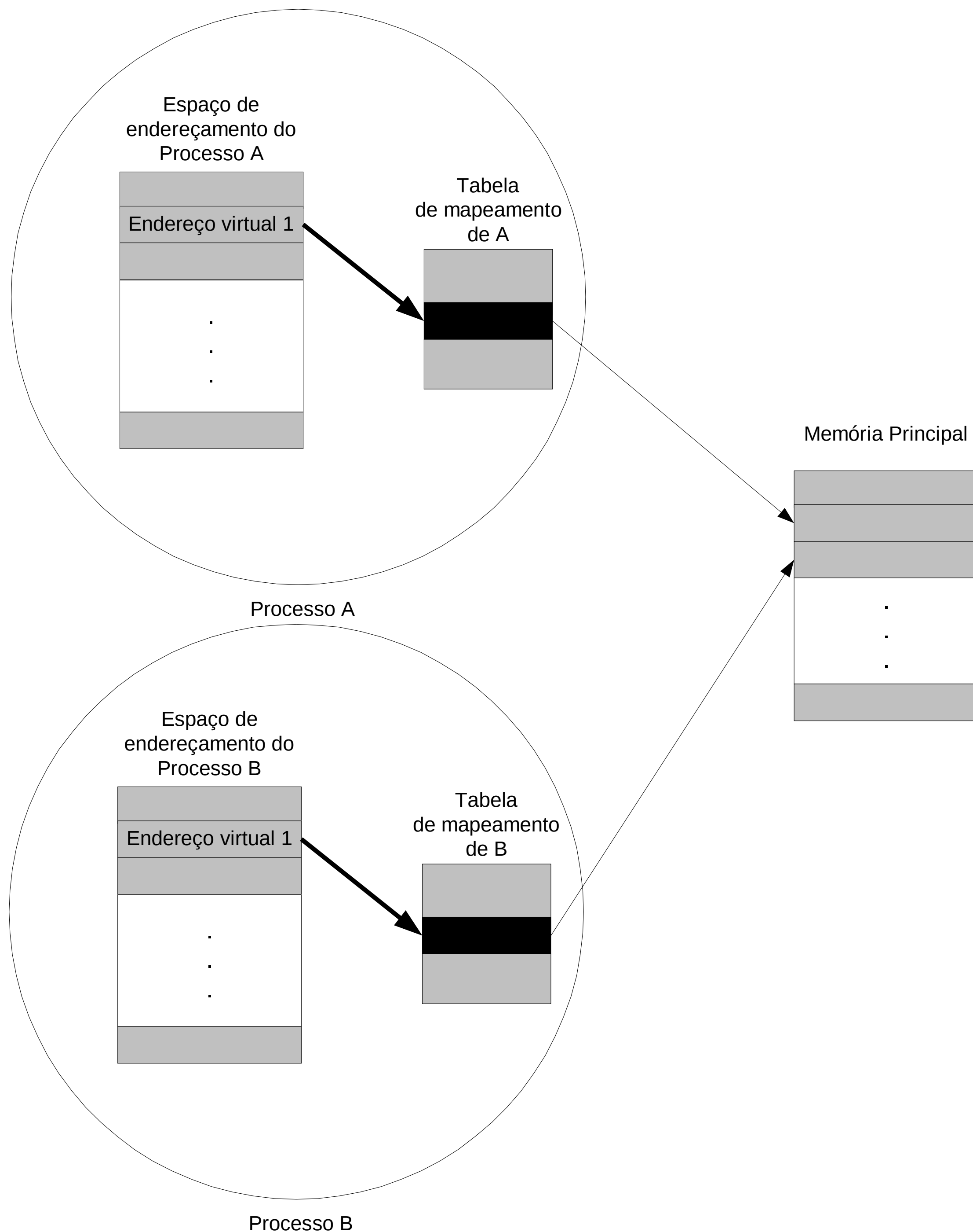


Nos sistemas modernos, a tarefa de tradução de endereços virtuais é realizada por hardware juntamente com o sistema operacional, de forma

a não comprometer seu desempenho e torná-lo transparente a usuários e suas aplicações.

O dispositivo de hardware responsável por esta tradução é conhecido como unidade de gerência de memória (*Memory Management unit* – MMU).

Cada processo tem o seu espaço de endereçamento virtual como se possuísse sua própria memória. O mecanismo de tradução se encarrega, então, de manter tabelas de mapeamento exclusivas para cada processo, relacionando os endereços virtuais do processo às suas posições na memória real.



A tabela de mapeamento é uma estrutura de dados existente para cada processo. Quando um determinado processo está sendo executado, o sistema utiliza a tabela de mapeamento do processo em execução para realizar a tradução de seus endereços virtuais.

5.5 Memória Virtual por Paginação

A memória virtual por paginação é a técnica de gerência de memória onde o espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos de mesmo tamanho chamados páginas.

As páginas no espaço virtual são denominadas páginas virtuais, enquanto as páginas no espaço real são chamadas de páginas reais ou *frames*.

Todo mapeamento de endereço virtual em real é realizado através da tabelas de páginas.

Cada processo possui sua própria tabela de páginas e cada página virtual do processo possui uma entrada na tabela (entrada na tabela de páginas – ETP), com informações de mapeamento que permitem ao sistema localizar a página real correspondente.

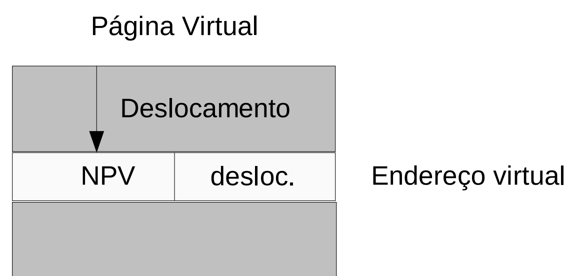
Quando um programa é executado, as páginas virtuais são transferidas da memória secundária para a memória principal e colocadas nos frames.

Sempre que um programa fizer referência a um endereço virtual, o mecanismo de mapeamento localizará na ETP da tabela do processo o endereço físico do *frame* no qual se encontra o endereço real.

Nessa técnica, o endereço virtual é formado pelo número da página virtual (NPV) e por um deslocamento.

O NPV identifica unicamente a página virtual que contém o endereço, funcionando como um índice na tabela de páginas.

O deslocamento indica a posição do endereço virtual em relação ao início da página na qual se encontra. O endereço físico é obtido, então, combinando-se o endereço do *frame*, localizado na tabela de páginas, com o deslocamento, contido no endereço virtual.



Além da informação sobre a localização da página virtual, a ETP possui outras informações, como o *bit* de validade (*valid bit*) que indica se uma

página está ou não na memória principal. Se o bit tem o valor 0, isto indica que a página virtual não está na memória principal, mas se é igual a 1, a página está localizada na memória.

Sempre que o processo referencia um endereço virtual, a unidade de gerência de memória verifica se a página que contém o endereço referenciado está ou não na memória principal.

Em caso negativo, ocorre uma falta de página (*page fault*). Neste caso, o sistema transfere a página da memória secundária para a memória principal (*page in*).

O número de *page faults* gerados pro cada processo em um determinado intervalo de tempo é definido como taxa de paginação do processo.

O *overhead* gerado pelo mecanismo de paginação é inerente da gerência de memória virtual, porém se a taxa de paginação dos processos atingir valores elevados, o excesso de operações de E/S poderá comprometer o desempenho do sistema.

5.6 Políticas de Busca de Páginas

A política de busca de páginas determina quando uma página deve ser carregada para a memória. Basicamente, existem duas estratégias para este propósito: paginação por demanda e paginação antecipada.

Paginação por demanda

As páginas dos processos são transferidos da memória secundária para a principal quando são referenciadas.

Este mecanismo é conveniente, na medida em que leva para a memória principal apenas as páginas realmente necessárias à execução do programa.

Desse modo, é possível que partes não executadas do programa nunca sejam carregadas.

Paginação antecipada

O sistema carrega para a memória principal, além da página referenciada, outras páginas que podem ou não ser necessárias ao processo ao longo do seu processamento.

A técnica de paginação antecipada pode ser empregada no momento da criação de um processo ou na ocorrência de um *page fault.*

5.7 Políticas de Alocação de Páginas

A política de alocação de páginas determina quantos *frames* cada processo pode manter na memória principal. Basicamente, existem duas alternativas: alocação fixa e alocação variável.

Alocação fixa

Cada processo tem um número máximo de *frames* que pode ser utilizado durante a execução do programa. Caso o número de páginas reais seja insuficiente, uma página do processo deve ser descartada para que uma nova página seja carregada.

O limite de páginas reais pode ser igual para todos os processos ou definido individualmente.

Apesar de sua simplicidade, a política de alocação fixa apresenta dois problemas. Se o número máximo de páginas alocadas for muito pequeno, o processo tenderá a ter um elevado número de *page faults,* o que pode impactar no desempenho do sistema. Por outro lado, caso o número de páginas seja muito grande, cada processo irá ocupar um grande espaço da memória principal, reduzindo o número de processo residentes e o grau de multiprogramação.

Alocação variável

O número máximo de páginas alocadas ao processo pode variar durante sua execução em função de sua taxa de paginação e da ocupação da memória principal.

Nesse modelo, processos com elevadas taxas de paginação podem ampliar o limite máximo de *frames*. Da mesma forma, processos com baixas taxas de paginação podem ter *frames* realocados para outros processos.

5.8 Políticas de Substituição de Páginas

Quando um processo atinge o seu limite de alocação de *frames* e necessita alocar novas páginas na memória principal, o sistema operacional deve selecionar, dentre as diversas páginas alocadas, qual deverá ser liberada.

Este mecanismo é conhecido como política de substituição de páginas.

Uma página real, quando liberada por um processo, está livre para ser utilizada por qualquer outro processo. A partir dessa situação, qualquer estratégia de substituição de páginas deve considerar se uma página foi ou não modificada antes de liberá-la para não perder os dados armazenados na página.

O sistema operacional consegue identificar as páginas modificadas através de um *bit* que existe em cada entrada da tabela de páginas, chamado *bit* de modificação (*dirty bit* ou *modify bit*). Sempre que uma página sobre uma alteração, o valor do *bit* de modificação é alterado, indicando que a página foi modificada.

5.9 *Working Set*

Como cada processo possui na memória principal apenas algumas páginas alocadas, o sistema deve manter um conjunto mínimo de *frames* buscando uma baixa taxa de paginação.

Além disso, o sistema deve impedir que os processos tenham um número excessivo de páginas na memória, de forma a aumentar o grau de compartilhamento da memória principal.

Caso os processos tenham na memória principal um número insuficiente de páginas para a execução, é provável que diversos *frames* referenciados ao longo do seu processamento não estejam na memória.

Essa situação provoca diversos *page faults.* Nesse caso, ocorre um problema conhecido como *thrashing*, provocando sérias conseqüências ao desempenho do sistema.

O conceito de *workint set* surgiu com o objetivo de reduzir o problema do *thrashing* e está relacionado ao princípio da localidade.

Existem dois tipos de localidade que são observados durante a execução da maioria dos programas.

A localidade espacial é a tendência de que após uma referência a uma posição de memória sejam realizadas novas referências a endereços próximos.

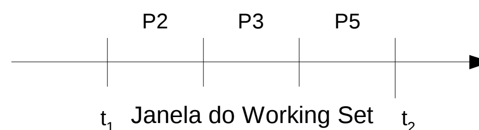
A localidade temporal é a tendência de que após a referência a uma posição de memória esta mesma posição seja novamente referenciada em um curto intervalo de tempo.

O princípio da localidade significa que o processador tenderá a concentrar suas referências a um conjunto de páginas do processo durante um determinado período de tempo.

O princípio da localidade é indispensável para que a gerência de memória virtual funcione eficientemente. Como as referências aos endereços de um processo concentram-se em um determinado conjunto de páginas, é possível manter apenas parte do código de cada um dos diversos programas na memória principal.

O conceito de *working set* é definido como sendo o conjunto das páginas referenciadas por um processo durante um determinado intervalo de tempo.

A figura abaixo ilustra que no instante t_2, o *working set* do processo, W(t_2, Δt), são as páginas referenciadas no intervalo Δt (t_2 – t_1), isto é, as páginas P2, P3 e P5.



O intervalo de tempo Δt é denominado janela do *working set.*

Dentro da janela do *working set,* o número de páginas distintas referenciadas é conhecido como tamanho do *working set.*

O modelo de *working set* permite prever quais páginas são necessárias à execução de um programa de forma eficiente.

Caso a janela do *working set* seja apropriadamente selecionada, em função da localidade do programa, o sistema operacional deverá manter as páginas do *working set* de cada processo residentes na memória principal.

Considerando que a localidade de um programa varia ao longo de sua execução, o tamanho do *working set* do processo também varia, ou seja, o seu limite de páginas reais deve acompanhar esta variação.

Apesar do conceito de *working set* ser bastante intuitivo, sua implementação não é simples, por questões de desempenho. Para implementar esse modelo, o sistema operacional deve garantir que o *working set* de cada processo permaneça na memória principal, determinando quais páginas devem ser mantidas e retiradas em função da última janela de tempo.

Em função disso, o modelo de *working set* deve ser implementado somente em sistemas que utilizam a política de alocação de páginas variável, onde o limite de páginas reais não é fixo.

Uma maneira de implementar o modelo é analisar a taxa de paginação de cada processo. Caso um processo tenha uma taxa de paginação acima de um limite definido pelo sistema, o processo deverá aumentar o seu limite de páginas reais na tentativa de alcançar o seu *working set.*

5.10 Algoritmos de Substituição de Páginas

Os algoritmos de substituição de páginas têm o objetivo de selecionar os *frames* que tenham menores chances de serem referenciados em um futuro próximo; caso contrário, o *frame* poderia retornar diversas vezes para a memória principal, gerando vários *pages faults*.

A partir do princípio da localidade, a maioria dos algoritmos tenta prever o comportamento futuro das aplicações em função do comportamento passado, avaliando o número de vezes que uma página foi referenciada, o momento em que foi carregada para a memória principal e o intervalo de tempo da última referência.

Ótimo

O algoritmo ótimo seleciona para substituição uma página que não será mais referenciada no futuro ou aquela que levará o maior intervalo de tempo para ser novamente utilizada.

Essa estratégia é utilizada apenas como modelo comparativo na análise de outros algoritmos de substituição, uma vez, que o sistema operacional não tem como conhecer o comportamento futuro das aplicações.

Aleatório

O algoritmo aleatório não utiliza critério algum de seleção. Todas as páginas alocadas na memória principal têm a mesma chance de serem selecionadas.

Apesar se ser uma estratégia que consome poucos recursos do sistema, é raramente implementada, em função de sua baixa eficiência.

FIFO (*First In First Out*)

No algoritmo FIFO, a página que primeiro foi utilizada será a primeira a ser escolhida, ou seja, o algoritmo seleciona a página que está há mais tempo na memória principal.

Esse algoritmo é raramente implementado sem algum outro mecanismo que minimize o problema da seleção de páginas antigas que são constantemente referenciadas.

LFU (*Least Frequently Used*)

O algoritmo LFU escolhe a página menos referenciada, ou seja, o *frame* menos utilizado. Para isso, é mantido um contador com o número de referências para cada página na memória principal. A página que possuir o contador com o menor número de referências será escolhida.

Inicialmente, esta parece ser uma boa estratégia, porém as páginas que estão há pouco tempo na memória principal podem ser, justamente, aquelas selecionadas, pois seus contadores estarão com o menor número de referências. É possível também que uma página muito utilizada no passado não seja mais referenciada no futuro.

LRU (*Least Recently Used*)

O algoritmo LRU seleciona a página na memória principal que está a mais tempo sem ser referenciada. Pelo princípio da localidade, uma página que não foi utilizada recentemente provavelmente não será referenciada novamente em um futuro próximo.

Para implementar esse algoritmo, é necessário que cada página tenha associado o momento do último acesso, que deve ser atualizado a cada referência a um *frame.*

Quando for necessário substituir uma página, o sistema fará uma busca por um *frame* que esteja há mais tempo sem ser referenciado.

NRU (*Not Recently Used*)

Este algoritmo é bastante semelhante ao LRU. Para implementação deste algoritmo é necessário um *bit* adicional, conhecido como *bit de referência* (BR).

O BR indica se a página foi utilizada recentemente e está presente em cada entrada da tabela de páginas.

Quando uma página é carregada para a memória principal, o BR é alterado pelo hardware, indicando que a página foi referenciada (BR=1). Periodicamente, o sistema altera o valor do BR (BR=0), e à medida que as páginas são utilizadas, o bit associado a cada *frame* retorna para 1.

Desta forma, é possível distinguir quais *frames* foram recentemente referenciados. No momento da substituição de uma página, o sistema seleciona um dos *frames* que não tenha sido utilizado recentemente, ou seja, com o *bit* de referência igual a zero.

O algoritmo NRU torna-se mais eficiente se o bit de modificação for utilizado em conjunto com o bit de referência. Nesse caso, é possível classificar as páginas em quatro categorias.

Categorias	Bits avaliados	Resultado
1	BR=0 e BM=0	Página não referenciada e não modificada
2	BR=0 e BM=1	Página não referenciada e modificada
3	BR=1 e BM=0	Página referenciada e não modificada
4	BR=1 e BM=1	Página referenciada e modificada

O algoritmo, inicialmente, seleciona as páginas que não foram utilizadas recentemente e não foram modificadas, evitando assim um *page out.*

O próximo passo é substituir as páginas que não tenham sido referenciadas recentemente, porém modificadas. Neste caso, apesar de existir um acesso à memória secundária para a gravação da página modificada, seguindo o princípio da localidade, há pouca chance de essa página ser novamente referenciada.

5.11 Memória Virtual por Segmentação

A memória virtual por segmentação é a técnica de gerência de memória onde o espaço de endereçamento virtual e o espaço de endereçamento

real são divididos em blocos de tamanhos diferentes chamados segmentos.

Na técnica de segmentação, um programa é dividido logicamente em sub-rotinas e estruturas de dados, que são alocadas em segmentos na memória principal.

Enquanto na técnica de paginação o programa é dividido em páginas de tamanho fixo, sem qualquer ligação com sua estrutura, na segmentação existe uma relação entre a lógica do programa e sua alocação na memória principal.

Normalmente, a definição dos segmentos é realizada pelo compilador, a partir do código fonte do programa, e cada segmento pode representar um procedimento, função, vetor ou pilha.

O mecanismo de mapeamento é semelhante ao de paginação. Os segmentos são mapeados através de tabelas de mapeamento de segmentos (TMS), e os endereços são compostos pelo número do segmento virtual (NSV) e por um deslocamento.

O NSV identifica unicamente o segmento virtual que contém o endereço, funcionando como um índice na TMS. O deslocamento indica a posição do endereço virtual em relação ao início do segmento no qual se encontra. O endereço físico é obtido, então, combinando-se o endereço do segmento, localizado na TMS, com o deslocamento, contido no endereço virtual.

Uma grande vantagem da segmentação em relação à paginação é a sua facilidade em lidar com estruturas de dados dinâmicas. Como o tamanho do segmento pode ser facilmente alterado na ETS, estruturas de dados, como pilhas e listas encadeadas, podem aumentar e diminuir dinamicamente.

Na técnica de segmentação, apenas os segmentos referenciados são transferidos da memória secundária para a memória principal. Se as aplicações não forem desenvolvidas em módulos, grandes segmentos estarão na memória desnecessariamente.

Enquanto na paginação existe o problema da fragmentação interna, na segmentação surge o problema da fragmentação externa. Este problema

ocorre sempre que há diversas áreas livres na memória principal, mas nenhuma é grande o suficiente para alocar um novo segmento.

Bibliografia:

Arquitetura de Sistemas Operacionais
Autores: Francis B. Machado e Luiz Paulo maia
Editora: LTC
Edição: 3ª.
Ano: 2002