

Sistemas Operacionais

Notas de Aula

Francisco Rapchan
rapchan@writeme.com
<http://www.inf.ufes.br/~rapchan>

Capítulo 3 – Gerência de Memória

Gerência de memória é o segundo dos grupos de funcionalidades fundamentais de um sistema operacional. Antigamente memória era um bem precioso e seu aproveitamento seguindo um gerenciamento austero mostrou-se fundamental para o bom desempenho dos sistemas.

Embora o desenvolvimento da tecnologia tenha nos propiciado mais memória do que antigamente isto, entretanto, não diminuiu a importância do bom gerenciamento dela. De fato, há uma tendência de termos sempre programas que, independentemente da quantidade de memória que temos, consomem mais memória do que temos disponível. (É provavelmente culpa da mesma lei que diz que temos sempre mais livros do que prateleiras, mais arquivos do que HD ou mais contas para pagar do que salário.)

Algumas palavras chaves que serão tratadas neste capítulo são as seguintes:

- Swapping
- Relocação
- Paginação
- Segmentação

3.1 GERÊNCIA SIMPLES DE MEMÓRIA

O sistema de memória pode ser simples e usar apenas a memória física disponível ou ser mais sofisticado e estender esta memória de forma virtual, usando partes do disco rígido.

O sistema de **gerência simples de memória** usa apenas a memória disponível no computador, sem fazer movimentação dos processos entre a memória e o disco rígido.

Já vimos as **vantagens da multitarefa**. Nela, um conjunto de processos é colocado na memória e sua execução é gerenciada pelo escalonador. Assim o processador fica mais tempo ocupado.

A questão é: Como organizar a memória de forma que **vários processos possam estar presentes** nela?.

PARTIÇÕES FIXAS

Uma solução é dividir a memória em **partições** fixas com tamanhos diferentes. O número e o tamanho das partições seriam estabelecidos pelo operador. Esta técnica foi usada no sistema operacional dos mainframes da série /360 da IBM na década de 60 (OS/360 ou MFT).

A idéia é termos um **enfileiramento** de processos e colocar cada processo que tem que ser executado em uma partição. Quando o processo acaba, ele deixa a partição e esta pode ser carregada com outro processo.

Dois problemas devem ser levados em conta com relação a partições: relocação e proteção.

Relocação e Proteção

Se o compilador estabelecesse **endereços absolutos** ao gerar o código executável, quando o programa fosse executado em uma partição, provavelmente os endereços não fariam sentido.

Uma solução é fazer a relocação durante a **carga do programa** ou seja, modificar os endereços relocáveis de forma que ficassem relativos ao início da partição.

A relocação durante a carga não resolve o problema de proteção pois um programa pode gerar um endereço dinamicamente pertencente a qualquer área da memória.

Outra solução é ter dois **registradores**, um para guardar o endereço da **base** e outro o **tamanho** da partição. Desta forma o processo poderia ser deslocado dentro do espço de memória se fosse necessário.

O antigo IBM PC tinha apenas o registrador de base e não o de limite. Com isto o DOS teve que ser desenvolvido levando em consideração esta limitação.

2 SWAPPING

Swapping é o movimento de processo entre a memória e o disco e vice-versa.

Em sistemas multiprogramados, um algoritmo de swapping com **partições variáveis** é mais interessante do que um com partições fixas.

[ver figura 3.5]

Esta abordagem pode acabar **fragmentando muito a memória** (fenômeno conhecido como tabuleiro de xadrez ou checkerboarding). Uma solução é **compactar a memória** de vez em quando. Esta operação normalmente é cara.

Outra questão é: Que **tamanho de partição** fornecer a um processo? Uma solução é fornecer segmentos diferentes para **código, dados e pilha** e fazer pilha e dados crescerem em direções opostas. Este é o modelo usado nos sistemas operacionais modernos (UNIX, Windows, etc).

[ver figura 3.6]

Há 3 formas de gerenciar o uso da memória: mapas de bits, listas ligadas e os sistemas buddy.

FORMAS DE GERÊNCIA DE MEMÓRIA

Mapas de bits

Memória é dividida em **unidades de alocação**.

Associa-se a cada unidade de alocação um **bit do mapa de bits** que será **0** se a unidade estiver livre e **1** se estiver ocupada ou vice-versa.

Raramente são usados mapas de bits porque é muito caro buscar um espaço de alocação neles.

Listas ligadas

A idéia é gerenciar a alocação de memória através de uma **lista encadeada dos segmentos livres e ocupados** da memória.

[Ver Fig3.7 pag58]

Há vários **algoritmos de alocação** possíveis para encontrar um local na memória para colocar o processo:

- **Best fit:** busca na lista inteira a **melhor alocação** para armazenar o processo que está precisando de espaço. Gera muita fragmentação (fragmentação externa - buracos entre segmentos).
- **Fist fit:** busca na lista pela **primeira alocação** que seja suficiente para conter o processo. Knuth provou, por incrível que pareça, que este é o melhor algoritmo.
- Pior alocação.
- **Alocação rápida:** mantém uma lista separada para alguns tamanhos de buracos mais requisitados.

Sistema Buddy

A idéia é manter **listas de blocos livres** de tamanho 1, 2, 4, 8, ... bytes até o tamanho da memória. Com uma memória de 1Mbyte, serão então necessários 21 destas listas. Com a memória vazia, teríamos a lista de 1Mbyte com uma entrada e todas as outras listas vazias.

Exemplo:

Suponha que chegue um processo de 60 Kbytes. A memória de 1M seria retirada da lista e teríamos duas entradas na lista de 512. Como ainda é um tamanho muito grande, uma destas entradas é retirada gerando duas entradas na lista de 256. O processo se repete até que um tamanho adequado de memória seja obtido.

[Ver fig3.9 pag60]

Gera muita **fragmentação interna** (desperdício de memória dentro do segmento alocado).

Apesar de extremamente **eficiente sob o aspecto da velocidade**, este sistema **não é eficiente na utilização da memória**.

3 MEMÓRIA VIRTUAL

Programas serem maiores do que toda a memória disponível para roda-lo é um problema real.

Uma primeira solução para isto uma técnica que consistia em dividir o programa em módulos denominados **overlays**. Nesta técnica o programador era responsável por esta divisão. O sistema operacional apenas gerenciava quais overlays deveriam estar na memória e quais poderiam ficar em disco.

Em 1961 um grupo de pesquisadores de Manchester, Inglaterra, propôs um método para realizar o processo de overlay automaticamente. Este método ficou conhecido como **memória virtual**.

A memória virtual consiste em permitir que o **programa enderece um tamanho qualquer de memória**. O sistema operacional fica responsável por manter na memória as partes do programa que está em execução. Quando é feita um referência a uma parte do programa que não está na memória, esta parte é carregada do disco no lugar de uma parte que não esteja mais em uso.

Observação. Naturalmente que enquanto um processo está esperando que outra parte dele seja carregada na memória, ele fica bloqueado e não poderá passar para o estado de executando (relembre os estados de um processo: executando, bloqueado, pronto).

Exemplo

- Um programa de 1M pode rodar numa máquina com 256K de memória, escolhendo-se cuidadosamente os 256K do programa que deverão ser mantidos na memória a cada instante.
- Da mesma forma 8 programas de 1M podem ser alocados a partições de 256K de uma memória de 2M, com cada um deles operando como se possuísse sua própria máquina como 256K de memória.

PAGINAÇÃO

Paginação é uma técnica muito usada de memória virtual. A idéia é dividir o processo em unidades denominadas **páginas** (pages) e a memória física em **page frames**. Ambos são sempre do mesmo tamanho (usualmente de 512 bytes a 8 Kbytes).

O hardware deverá ter uma **unidade de gerência de memória (MMU)** que faz o mapeamento entre o endereço virtual da página do processo e o endereço real da page frame.

[Ver fig3.11 pag63]

Quando o MMU verifica que a página não está nos page frames ela gera um trap para o sistema operacional indicando **page fault**. O sistema operacional então se responsabiliza por obter a página.

O MMU usa uma estrutura chamada **Tabela de Página** cujo objetivo é mapear paginas virtuais em molduras de páginas do processo. O índice desta tabela é o próprio número da página de memória virtual. Cada processo tem sua tabela de página.

Exemplo da figura 3.12:

Suponha que chegue ao MMU o seguinte endereço: 8196. Em binário: 0010000000000100. Os 4 primeiros bits podem formar o índice da tabela de páginas: 0010 que é a terceira entrada. O valor do page frame correspondente é 110 e ele está presente (valor do campo Presente é 1). Assim o endereço físico será: 1100000000000100. Ou 24.580 em decimal.

Índice	Page Frame	Presente
0000	010	1
0001	001	1
0010	110	1
0011	000	1
...	...	...

TABELAS DE PÁGINAS

A função da tabela de páginas é auxiliar o MMU a mapear páginas virtuais em molduras de páginas. Cada processo deve ter sua própria tabela de páginas.

Normalmente a tabela de página fica em um **conjunto de registradores de acesso rápido**. Carregar a tabela de páginas a cada troca de contexto pode vir a degradar bastante a performance do sistema.

Manter a **tabela toda na memória convencional** também pode tornar o trabalho do MMU muito lento. Para evitar que, no caso de uma memória virtual muito grande, a tabela de páginas ocupe muito espaço, usa-se **tabelas de páginas multinível**.

Exemplo (pag65, penúltimo parágrafo)

Suponha um computador com endereçamento de 32 bits. Isto equivale a uma memória virtual de 4Giga. Pode-se dividir o endereço virtual em 3 partes:

Tabela nível 1 10 bits	Tabela nível 2 10 bits	Deslocamento 12 bits
---------------------------	---------------------------	-------------------------

Os primeiros 10 bits endereçam uma tabela de 1024 entradas. Cada uma destas entradas possui um endereço para uma outra tabela, de segundo nível, cada uma também com 1024 bits.

Os 10 bits seguintes endereçam a tabela de segundo nível. Esta tabela também tem 1024 entradas e cada uma destas possui um endereço para uma página de 4K.

Os 12 bits restantes fornecem o deslocamento a partir do início da página.

Em um programa de 12 Megabytes com 4M de código, 4M de dados e 4M de pilha. Pode-se alocar a área de dados com a primeira entrada da tabela de nível 1, a de dados com a segunda e a de pilha com a última, deixando as entrada intermediárias para permitir que a área de dados e pilha possam crescer.

Neste caso, apesar do espaço de endereçamento possuir milhões de páginas, apenas 4 tabelas de páginas precisam ser armazenadas.

O exemplo foi feito com um sistema de tabelas de páginas em dois níveis. Podemos ter **sistemas também com três, quatro ou mais níveis**.

As tabelas de páginas contêm normalmente as seguintes informações:

- Número da moldura de página (page frame).
- Bit de presença (ou bit de residência).
- Bits de proteção (indicam se a página permite leitura, execução ou escrita).
- Bits de modificação (informa se a página precisa ser gravada no disco quando sair da memória).
- Bits de referência (informa se a página foi recentemente usada).

Observação: O endereço em disco usado para armazenar a página quando ela não estiver na memória não faz parte da tabela de páginas.

3.4 ALGORITMOS DE SUBSTITUIÇÃO DE PÁGINAS

O sistema operacional deve ter alguma estratégia para escolher quais páginas ficam na memória e quais devem ir para o disco, disponibilizando lugar para outras páginas.

Há muitos algoritmos para fazer esta tarefa, dentre eles:

Ótimo

Deve sair da memória a página que vai levar mais tempo para ser referenciada novamente. É muito difícil prever quando uma página irá ser referenciada novamente. Portanto este algoritmo é inviável.

Não usada recentemente - NUR

Sai da memória a página que não foi referenciada recentemente. Para isto há 2 bits na tabela de páginas:: R - Referência e M - Modificada. Assim, quando uma página for referenciada, R passa para 1 e quando uma página for modificada M passa para 1. Assim temos 4 classes de páginas:

- classe 0 - não referenciadas, não modificadas
- classe 1 - não referenciadas, modificadas.
- classe 2 - referenciadas, não modificadas.
- classe 3 - referenciadas, modificadas.

O algoritmo NUR remove randomicamente uma página da menor classe.

FIFO

Remove a página que está há mais tempo na memória. É implementado através de uma lista que controla a ordem de chegada das páginas.

Segunda chance

É uma combinação entre o algoritmo FIFO e o NUR. É verificada a página que está como primeira da fila (está há mais na memória). Se ela foi referenciada recentemente, ela vai para o fim da fila (como se tivesse chegado agora na memória). Senão ela é removida. Usa o bit R (Referência) da tabela de páginas para verificar se a página foi referenciada recentemente.

Relógio

É o algoritmo da segunda chance usando fila circular.

Página usada há mais tempo - LRU.

A idéia básica deste algoritmo é que páginas que foram muito usadas pelas últimas instruções executadas continuarão sendo usadas pelas próximas instruções e páginas que não foram referenciadas pelas últimas instruções provavelmente continuarão não sendo usadas por um bom tempo.

Uma das formas de implementar isto é ter um contador de instrução (longo, talvez com 64 bits). A cada referência a uma página, seria armazenado o número deste contador na tabela de páginas. Desta forma poderia-se descobrir qual a página usada há mais tempo.

Outra forma é ter uma matriz quadrada de bits, com cada página representando uma linha e uma coluna desta matriz. Quando uma página é referenciada, sua linha é toda colocada em 1 e sua

coluna é colocada em 0. Desta forma em cada instante, a linha com o menor valor binário armazenado será correspondente à página usada há mais tempo.

[Ver figura 3.26 página 78].

LRU implementado em software

A idéia é ter um contador associado a cada página. Este contador funciona da seguinte forma. Quando uma página é referenciada, seu contador é deslocado uma vez para a direita e seu bit mais significativo é setado em 1. Quando ocorre uma falta de página, a página cujo contador tem o valor mais baixo deve ser removida.

[Ver figura 3.27 página 79].

3.5 MODELANDO ALGORITMOS DE PAGINAÇÃO

Este item não será visto neste curso.

3.6 ASPECTOS DO PROJETO DE SISTEMAS PAGINADOS

Os primeiros tópicos deste item não serão visto neste curso.

Tratamento de falta de página

[Ver descrição do que acontece quando há a ocorrência de uma falta de página. pg 88, passos 1 a 10].

3.7 SEGMENTAÇÃO

A memória virtual por paginação é unidimensional, ou seja, os endereços virtuais vão de 0 até um valor máximo continuamente. Para muitos problemas, ter dois ou mais espaços de endereçamento virtuais separados pode ser muito melhor do que ter apenas um.

A solução é prover a máquina com muitos espaços de endereçamento independentes. Chamamos estes espaços de endereçamento de **segmentos**.

Segmentos são sequências lineares de endereço, começando com 0 e indo até um determinado valor máximo. Segmentos diferentes podem ter tamanhos diferentes. Segmentos distintos podem crescer ou diminuir de forma dinâmica, sem interferirem um nos outros.

Um segmento é uma entidade lógica, definida pelo programador. Ele pode conter um procedimento, uma matriz, uma pilha, um conjunto de variáveis escalares. Entretanto não deve-se misturar informações de caráter diferente em um mesmo segmento.

Para especificar um endereço em uma memória segmentada, o programa precisa gerar um endereço com duas partes, um número que especifica qual o segmento e um endereço dentro deste segmento.

A segmentação facilita o compartilhamento de procedimentos e de dados entre vários processos na forma por exemplo de bibliotecas. Além disto facilita também os mecanismos de proteção. Como cada segmento tem um uso específico (dados, pilha, código), é possível estabelecer o mesmo critério de proteção para todo aquele espaço de endereçamento.

Uma diferença fundamental entre segmentação e paginação é que as páginas têm tamanho fixo e os segmentos não.