http://scama.norbis.com.br - rodrigoscama@yahoo.com.br

Memória Virtual

Objetivo

O principal objetivo da memória virtual é disponibilizar aos usuários uma capacidade de memória independente da quantidade de memória física (RAM) instalada em um computador. Para cumprir esse objetivo, o SO, através da implementação do suporte à memória virtual, cria a ilusão de que cada processo possui um espaço de endereçamento contíguo em memória e, em princípio, sem restrição de tamanho. Dessa forma, a memória virtual elimina duas limitações impostas pela memória física (real): um processo ter seu tamanho máximo determinado pela capacidade de memória física instalada e, o somatório do espaço de memória ocupado por n processos não exceder essa mesma capacidade. Essa segunda limitação restringe o grau de multiprogramação, ou seja, o número de processos ativos em um sistema.

A memória virtual expande a memória total disponível em um sistema adicionando a memória real um espaço em disco (memória secundária). Essa área em disco corresponde ao que se denomina **área de swap**. A área de swap pode ser implementada através de um arquivo específico, como nos SO da Microsoft ou em partições do disco dedicadas a essa finalidade na família Linux/Unix.

O SO gerencia o espaço de endereçamento da memória virtual subdividindo-o em porções que são mapeadas em regiões da memória física ou do disco (área de swap). Quaisquer acessos feitos a posições de memória mapeadas em regiões da RAM são realizados diretamente. Já os acessos a posições mapeadas da memória na área do disco implicam que essas sejam primeiramente carregadas para a memória RAM para posteriormente serem acessadas. Nesse ponto, salienta-se dois aspectos. Primeiro, todos os acessos são feitos exclusivamente na memória RAM, alguns diretamente, outros, após a carga da região correspondente em memória. Segundo, caso a memória RAM não possua mais espaço disponível para receber a região que está armazenada em disco, torna-se necessário abrir espaço na memória RAM.

O emprego de memória virtual apresenta como vantagens a possibilidade de executar aplicações maiores que o tamanho real da RAM e o aumento do número de processos ativos em memória (grau de multiprogramação). Sua maior desvantagem é o desempenho, pois, parte dos acessos na área de endereçamento de um processo dependem de acessps à área de swap(disco), o que é mais lento que acessar diretamente a memória RAM.

Implementação de memória virtual

Em porções de memória a questão que se apresenta neste momento é como definir essa porção de memória. A resposta surge

através dos mecanismos básicos de gerência de memória, ou seja, paginação, segmentação e segmentação paginada.

A implementação do suporte à memória virtual implica em controlar o fluxo de páginas (ou segmentos) entre memória RAM e a área de swap (disco). Isso implica em gerenciar áreas livres e ocupadas na memória, determinar poslíticas para alocação de memória física, substituir páginas (ou segmentos) em memória e controlar o compartilhamento e a proteção de memória entre processos. Essas tarefas, se executadas exclusivamente por software, representam um ônus muito grande de processamento, ou seja, o desempenho global do sistema é afetado pela execução da gerência de memória virtual. Para reduzir esse impacto, a maioria dos procesadores são projetados de forma a incluir em seu hardware mecanismos que auxiliem e acelerem as tarefas executadas pelo software. Normalmente, esses mecanismos fazem parte da MMU (memory Management Unit).

Princípio da localidade e referência

Este princípio está relacionado com a constatação de que a execução de um programa tende a se concentrar, em determinados intervalos de tempos, a apenas certas regiões de seu código e dados, ou seja, a regiões bem determinadas do espaço de endereçamento do processo.

Essa execução concentrada em uma região do código(feita por loops tipo for, while e do-while) é denominada de **localidade temporal**. Analogamente, um programa é escrito de forma a acessar frequentemente algumas variáveis de controle e empregar estruturas de dados do tip de vetores e matrizes. Essa característica concentra os acessos a regiões de memória onde esses dados estão armazenados (**localidade espacial**).

A consequencia imediata da localidade de referência (temporal ou espacial) é que, em um determinado intervalo de tempo, apenas certos trechos de código e de dados de um programa (processo) necessitam estar carregados em memória RAM. Essa característica é explorada na implementação do mecanismo de memória virtual para manter em RAM apenas os trechos realmente necessários a execução de um processo, liberando assim memória física para outros processos. Caso o processo necessite executar uma instrução que pertença a um trecho que não está carregado na memória, o hardware detecta esse acesso como inválido e gera uma interrupção. O tratamento de interrupção providencia a carga em memória do trecho “faltante” e, após o mesmo ter sido carregado, permite que o processo retome sua execução.

Paginação sob demanda

A paginação sob demanda é a implementação de memória virtual baseada no mecanismo de paginação simples, isto é, cada processo possui uma memória lógica, contígua, a qual é dividida em

páginas lógicas de mesmo tamanho. As páginas lógicas são carregadas em páginas físicas, que são áreas de igual tamanho na memória física (RAM). A associação (mapeamento) de qual página física corresponde a uma determinada página lógica é feita através da tabela de páginas. Na paginação simples, todas as páginas lógicas de um processo são sempre carregadas para a memória física, resultando em uma entrada válida na tabela de páginas para cada página lógica do processo. O bit de válido/inválido é usado para indicar quais páginas estão fora da memória lógica. Quando uma página marcada como inválida é acessada, o processo é abortado por acesso ilegal à memória.

Na paginação por demanda, apenas as páginas efetivamente acessadas pelo processo são carregadas para a memória física. Agora, o bit válido/inválido é usado para indicar quais páginas lógicas foram carregadas. Dessa forma, na paginação por demanda, uma página marcada como inválida na tabela de páginas pode significar que ela realmente está fora do espaço lógico do processo ou pode significar apenas que essa página ainda não foi carregada para a memória física. A situação exata é determinada através de uma consulta ao descritor do processo em questão, no qual é mantido o tamanho do seu espaço lógico de memória.

Na paginação por demanda, então, um acesso à memória pode ter dois tratamentos distintos. Quando a página lógica acessada pelo processo está marcada como válida na tabela de páginas, o endereço lógico é transformado em endereço físico e o acesso transcorre normalmente. Caso ocorra um erro e, por consequência uma **interrupção de proteção**, temos uma **falta de página** (page fault).

Quando o SO é acionado em função de uma falta de página, as seguintes ações são realizadas:

- O processo que gerou a interupção de falta de página é suspenso e seu desrcitor de processo é inserido em uma fila especial, a “dos processos esperando página lógica”
- Uma página física livre deve ser alocada
- A página lógica acessada deve ser localizada no disco
- Uma operação de leitura do disco deve ser solicitada, indicando o endereço da págin alógica no disco e o endereço de página alocada.

Quando a operação de leitura do disco for concluída, a gerência de memória concluirá o atendimento à falta de página realizando as seguintes ações:

- A tabela de páginas do processo é corrigida para indicar que a página lógica causadora da interrupção é agora válida e está na página física que fora alocada antes
- O descritor do processo é retirado da “fila dos processos esperando página lógica” e reinserido na fila de aptos do escalonador.

Observe que o processo deverá repetir a instrução que causou a falta de página (esta instrução não foi executada devia a falta de página) repetir uma instrução após a carga da página faltante requer uma arquitetura adequada, hardware, portanto, projetado para suportar memória virtual.

Na “paginação por demanda pura”, somente são carregadas para a memória física aquelas páginas realmente referenciadas pelo processo (a carga das páginas é feita à medida que ocorrem as faltas de página).

Nas primeiras instruções que o processo executa certamente ocorrem várias faltas de página. Após algum tempo, o processo consegue se estabilizar e iniciar efetivamente sua execução. Para evitar essa rajada inicial de interrupções por falta de página, alguns sistemas carregam páginas automaticamente, antes de iniciar um processo. O sistema procura carregar páginas que serão realmente usadas. Um “bom chute” é carregar as páginas iniciais na memória lógica.

A parte do sistema operacional responsável por carregar páginas do disco para a memória principal é denominado **Pager**. É importante fazer distinção entre o Pager e o Swapper. O Pager carrega uma página específica de um processo e está associado com o mecanismo de memória virtual. O swapper carrega sempre um programa inteiro do disco para a memória e está associado com o mecanismo de swapping.

Alocação de memória

Um processo para ser executado necessita estar carregado na memória o que, no caso da paginação por demanda, implica em se ter em memória uma certa quantidade de páginas lógicas. Isso significa alocar uma quantidade equivalente de páginas físicas na memória RAM para esse processo. Algumas questões então estão associadas a essa alocação.

O primeiro ponto é determinar a quantidade mínima de páginas físicas que devem ser alocadas a um processo. Essa quantidade pode ser estimada a partir do próprio conjunto de instruções do processador em que o SO executa.

Uma solução é dividir entre os processos ativos a quantidade total de páginas físicas existentes no sistema (**alocação igualitária**). Dessa forma, cada processo recebe uma quantidade de páginas físicas equivalente ao número total de páginas físicas existentes dividido pela quantidade total de processos ativos. Entretanto, essa solução provoca algumas distorções, já que nem todos os processo possuem as mesmas necessidades de memória. Uma forma de corrigir essa distorção é considerar uma constante de proporcionalidade (**alocação proporcional**), como, por exemplo, o tamanho dos processos. Nesse caso, processos maiores recebem mais páginas físicas que processos menores, mantendo a relação

páginas carregadas sobre tamanho uma constante independentemente do tamanho do processo.

Entretanto, a alocação igualitária e a alocação proporcional apresentam um mesmo problema: tanto o número de processos como o número de páginas que cada um ocupa são dinâmicos, isto é, os processos são criados, executados e finalizados e, durante suas execuções, páginas são alocadas e liberadas. O problema ocorre quando um processo necessita alocar mais memória e não há páginas físicas disponíveis.

Uma solução para isso seria o SO suspender um processo, liberando o espaço de memória física que ele ocupa para atender a demanda por outros processos. O SO pode decidir entre suspender o processo que provocou essa situação e retomar sua execução quando houver memória suficiente. Essa técnica é chamada **swapping**.

Alternativamente, o SO pode remover da memória apenas uma ou algumas páginas lógicas de um processo, liberando memória e permitindo assim uma nova página ser carregada. Todos os processos continuam executando. A determinação da página a ser removida é uma tarefa realizada exclusivamente pelo sistema de gerência de memória virtual e é feita com base em uma política de substituição de páginas.

Substituição de páginas na memória

A medida que os processos vão sendo carregados para a memória, é possível que todas as páginas físicas acabem ocupadas. Nesse caso, para atender à **falta de página**, é necessário antes liberar uma página física ocupada. Isso significa escolher uma página lógica que está na memória, copiar seu conteúdo de volta para o disco e marca-la como inválida na tabela de páginas do seu processo. A página escolhida para ser copiada de volta ao disco é chamada de **página vítima**.

O algoritmo de substituição de páginas é responsável pela escolha da página vítima. Ele é muito importante para a eficiência do mecanismo como um todo. Um escolha errada significa que a página removida será novamente acessada em seguida, gerando uma nova falta de página. É importante que o algoritmo usado seja capaz de remover da memória física páginas que provavelmente não serão necessárias logo em seguida.

Vários **bits auxiliares** são normalmente adicionados às tabelas de páginas.

O **bit de sujeira** (dirty bit) indica quando uma página foi alterada durante a execução do processo. Esse bit é zerado pelo SO quando a página é carregada para a memória. A MMU automaticamente liga esse bit quando o processo realiza uma operação de escrita nessa página.

Dessa forma, o SO é capaz de determinar se essa página está alterada com relação à sua cópia em disco. Essa informação é importante no momento em que uma página é escolhida como

vítima. Caso a página não esteja alterada em relação à cópia em disco,seu conteúdo não precisa ser salvo, economizando assim um acesso ao disco. A maioria das páginas de um processo (código e constantes) é apenas lida e nunca escrita. Dessa forma, graças ao bit de sujeira, na maioria das substituições de páginas haverá apenas um acesso ao disco, não dois.

O **bit de referência** (reference bit) indica quando uma página foi acessada pelo processo. Esse bit é feito igual a zero pelo sistema operacional quando a página é carregada para a memória. Ele também será zerado em determinadas situações, ditadas pelo algoritmo de substituição de página empregado. A MMU liga automaticamente esse bit sempre que a página correspondente é acessada. Sua função é permitir que o SO tenha uma noção de quando cada página é usada. Se o bit é zerado no instante T1 e lido no instante T2, é possível determinar se a página foi ou não acessada no intervalo de tempo [T2-T1].

O **bit de tranca** (lock bit) serve para o SO "trancar" uma página lógica na memória física. Existem situações nas quais uma determinada página lógica não deve ser escolhida como vítima. Então, quando o SO envia ao controlador de disco um comando de leitura informando como destino uma página de processo, ele também liga o bit de tranca da respectiva página. O algoritmo de substituição é feito de forma que uma página com o bit de tranca ligado jamais será escolhida como vítima.

O bit de tranca também pode ser usado para impedir que páginas lógicas muito usadas pelo processo sejam escolhidas como vítima. Caso o programador saiba que determinadas páginas serão usadas intensamente, ele pode solicitar ao SO (via chamada de sistema) que aquelas páginas permaneçam sempre na memória física.

Algoritmos de substituição de páginas na memória

Em um sistema multiprogramado a memória é compartilhada por um certo número de processos, a determinação da página a ser substituída é feita por um **algoritmo de substituição de páginas** que recai em uma de duas classes genéricas possíveis: **classe global** e **classe local**.

Independente de classe, global ou local, o ideal é selecionar uma página que não seja mais útil ou que, pelo menos, não venha a ser necessária em um curto intervalo de tempo. Logo, o **algoritmo ótimo** de substituição é aquele que escolhe uma página lógica que já foi utilizada por um programa e que não será mais necessária ou aquela página que será utilizada no futuro mais distante. A implementação do algoritmo ótimo não é viável por exigir o conhecimento prévio do comportamento dos processos.

Algoritmos globais

FIFO (First In – First Out) – Escolhe para substituição a página que está a mais tempo residente em memória. A principal vantagem do FIFO é sua implementação simples: basta uma lista de até M posições e um ponteiro para indicar a frente da fila. Entretanto, essa estratégia peca por não considerar a utilização de uma página, ou seja, a página apontada como primeira da fila pode ser uma página acessada muito frequentemente ou que será necessária em breve.

LRU (Least Recently Used) – Escolhe como página vítima aquela que foi acessada a mais tempo, isto é, a que foi "menos recentemente usada". O LRU parte da premissa que as páginas acessadas recentemente por um processo continuarão a ser necessárias em um futuro próximo. A implementação básica de um algoritmo LRU é feita mantendo-se uma lista de páginas carregadas em memória com o registro do tempo do último acesso. Ao ser necessário escolher uma página vítima o SO percorre essa lista buscando a página lógica que possui a "hora" de acesso mais antiga. A principal desvantagem de um algoritmo LRU é seu custo computacional pois, a cada acesso à memória, é necessário atualizar o tempo de acesso da página e/ou reordenar a lista de páginas. Pode-se aproximar o comportamento do algoritmo LRU através de um histórico de bits de referência, ounde cada página possui um bit de referência associado. A MMU liga esse bit cada vez que a página é acessada. A gerência de memória pode implementar um histórico, amostrando periodicamente os bits de referência. A cada amostragem, esses bits são copiados da tabela de páginas para o histórico e então zerados na tabela de páginas. Como o histórico possui uma capacidade limitada, é necessário descartar uma amostragem antiga antes de inserir uma nova.

Segunda chance – Também baseado em bit de referência, porém mais simples que o histórico de bits. Nesse caso, a gerência de memória considera que todas as páginas lógicas presentes na memória formam uma lista circular. Um apontador percorre a lista circular formada por todas as páginas e indica qual a próxima página a ser usada como vítima. Quando uma página vítima é necessária, o algoritmo verifica o bit de referência da página indicada pelo apontador. Caso este bit esteja desligado, essa página é efetivamente escolhida como vítima, e o apontador avança uma posição na lista circular. Caso o bit de referência da página apontada esteja ligado, o bit de referência é desligado, e ela recebe uma segunda chance. Devido a esse comportamento circular, o algortimo segunda chance também é conhecido como **algoritmo do relógio** (clock algorithm), em analogia aos ponteiros de um relógio.

Algoritmos Locais

Uma outra aboradgem para o problema da substituição de páginas vêm da análise de dois comportamentos apresentados durante a execução de processos. O primeiro, cacda processo

necessita de um conjunto mínimo de páginas para executar de forma eficiente, isto é, sem provocar uma alta taxa de falta de página. O segundo, esse conjunto de páginas é dinâmico, isto é, evolui com a execução do processo. Considerando isso, o ideal é manter em memória, para cada processo, um conjunto ótimo de páginas necessárias a sua execução. Com esse raciocício, a determinação de quais páginas lógicas de um processo devem permanecer em memória é dado pela sua própria execução e não deve afetar a política de páginas de outro processo.

O algoritmo ótimo, então, ajusta o número de páginas em memória de acordo com as suas futuras referências. A idéia consiste em verificar se uma página lógica P acessada em isntante de tempo T, será ou não acessada futuramente dentro de TT unidades de tempo. Caso a página lógica P venha a ser acessada dentro desse intervalo futuro de tempo, ela é considerada necessária e é mantida em memória, caso contrário, o espaço em memória que ocupa (página física) pode ser liberado. O intervalo de tempo dado por T+TT fornece uma janela temporal definindo, por processo, quais páginas devem ser mantidas em memória. Devido, porém, a impossibilidade de se conhecer previamente as páginas de um processo, esse algoritmo serve apenas para estimativas.

O **Modelo de working set** é uma aproximação do algoritmo ótimo que estima quais as páginas serão necessárias no futuro com base em quais páginas goram acessadas no passado. O principio é manter em memória, em um instante T, conjunto de páginas P, acessadas por um processo durante o intervalo de tempo (T-TT; T). Sua lógica é considerar que se uma página foi acessada em um passado recente (T-TT), ela tyem boa chance de continuar sendo utilizada em um futuro próximo.

O problema do modelo working set é estimar o valor adequado para a constante TT (largura da janela temporal). Um valor pequeno para TT pode não englobar toda a localidade de referência de um processo, provocando faltas de páginas. Por outro lado, um valor muito grande para TT, tende a manter em memória páginas que não são mais referenciadas, provoacando um desperdíciio de memória, e consequentemente, reduzindo o número de processos aptos a executar.

Outra desvantagem do working set é a necessidade de reavaliá-lo a cada unidade de temp (acesso), o que representa um custo de processamento elevado.

O algoritmo de **frequência de falta de páginas (FFP)** ajust adinamicamente a quantidade de páginas físicas alocadas a um processo contabilizando o número de faltas de página que ele provoca dentro de um intervalo de tempo. Esse método utiliza três parâmetros de configuraçõa: taxa de falta de página, máxima e mínima, admissíveis no sistema e um período de contabilização de faltas de página.

Tanto no algoritmo working set como no algoritmo de frequência de falta de páginas, se um processo necessitar mais memória do que já tem alocado e não houver mais espaço disponível no sistema, o SO suspende um processo (transfere para a área de swap) liberando sua área de memória para satisfazer as novas requisições de alocação.

Trashing

O tratamento de uma falta de página é várias ordens de grandeza mais lento que um processo normal à memória. Logo, o impacto da taxa de falta de página sobre o tempo de execução de um processo é muito grande. Quando um processo possui um número muito pequeno de páginas físicas para executar, a sua taxa de falta de página aumenta. Á medida que a taxa de falta de página aumenta o processo pára de realizar qualquer trabalho útil. Tudo que o processo faz é esperar pelo atendimento de faltas de páginas. Nesse momento, é dito que está ocorrendo Thrashing.

Para os usuários, o thrashing se apresenta como o sintoma de um congelamento no sistema.

Para retirar o sistema do estado de thrashing é necessário suspender temporariamente alguns processos, librando suas páginas físicas. O mecanismo natural para isso é o swapping.

Obviamente, a gerência de memória realiza um revezamento entre os processos suspensos e aqueles executando. O swapping não é algo desejável, pois aumneta o tempo de resposta dos processos. Em condições normais, ele não deve ser usado.

A solução normalmente empregada para solucionar o problema de thrashing é a prevenção: o sistema de gerência virtual deve agir de forma pró-ativa e evitar que o sistema entre em thrashing. Um sistema entrará em thrashing sempre que o somatório das páginas lógicas necessárias pelos processos do sistema for superior ao número de páginas físicas disponíveis no sistema.

O modelo de working set, por exemplo, pode ser empregado para controlar o thrashing mantendo a soma dos tamanhos dos working sets (cardinalidade do conjunto) de todos os processos inferior a quantidade de páginas físicas disponíveis no sistema.

Outra forma de prevenir o thrashing é empregar o método de frequência de falta de páginas. Nesse caso, se muitos processos apresentarem uma alta taxa de falta de páginas, isso significa que há mais demanda de memória, por parte dos processos que executam, que memória disponível. Novamente, através do escalonador de médio prazo, o SO seleciona um ou mais processos para suspender, evitando assim a situação de thrashing.