Gerenciamento de Memória

A base do funcionamento da Memória Virtual é o Princípio da Localidade que estabelece que há uma tendência que os futuros endereços de memória de instruções e dados sejam próximos a endereços de memória recentemente acessados. Esse comportamento se deve as características peculiares aos programas, que frequentemente fazem uso de endereços em sequência (vetores), localizados em blocos de código bem definidos e frequentemente invocados (funções), ou de códigos repetitivos (laços de repetição).

A ideia básica da memória virtual é que o tamanho combinado do programa, dos seus dados e da pilha pode exceder a quantidade de memória física disponível para ele, ou seja, neste caso, a simples troca, vista anteriormente, não resolveria o problema. O Sistema Operacional, então, mantém partes do programa atualmente em uso, em forma de páginas ou segmentos, na memória principal e o restante em disco. Essas páginas/segmentos são "trocados" entre memória principal e secundária conforme o SO as solicita, conforme a demanda do programa.

A memória virtual também pode trabalhar em um sistema de multiprogramação, com pedaços de vários programas na memória simultaneamente. Enquanto um programa está esperando parte dele próprio ser trazido para a memória (ele fica esperando a E/S e não pode executar) a CPU pode ser dada a outro processo, assim como em qualquer sistema de multiprogramação.

Para a implementação desta técnica, alguns recursos mínimos são necessários: localização da página através do hardware MMU, carga de página, substituição de página e área de troca, partição ou arquivo especial de troca (swap ou página) destinada a armazenar páginas.

Muitos sistemas de Memória Virtual utilizam uma técnica denominada paginação, vista mais adiante.

Troca (Swapping)

Em algumas situações não é possível manter todos os processos na memória e uma solução para essas situações é o mecanismo conhecido como swapping (troca). A gerência de memória reserva uma área do disco para esse mecanismo, que é utilizada para receber processos da memória. A execução desse processo é suspensa, com isso é dito que o mesmo sofreu uma swap-out. Mais tarde, esse mesmo processo será copiado do disco para a memória, mecanismo conhecido como swap-in. Esse mecanismo de trocas de processos no disco tem como objetivo permitir que o sistema operacional consiga executar mais processos do que caberia na memória.

Esse processo gera um grande custo de tempo de execução para os programas. Fazer a cópia do processo da memória para o disco e depois fazer o inverso é demorado.

Paginação

O espaço de endereço virtual é dividido em unidades chamadas páginas. As unidades correspondentes na memória física são chamadas molduras de página (ou quadros). As páginas e as molduras (quadros) têm sempre exatamente o mesmo tamanho.

No espaço físico (memória) tem-se várias molduras de página. Por exemplo, podem existir 05 páginas situadas no espaço de endereço virtual que são mapeadas na memória física. No entanto, o espaço de endereço virtual é maior que o físico. As outras páginas não são mapeadas. No hardware real, um bit presente/ausente em cada entrada monitora se a página é mapeada ou não.

Quando um programa tenta utilizar uma página não mapeada em uma moldura, a MMU detecta o acontecimento (que a página não está mapeada) e gera uma interrupção, passando a CPU para o Sistema Operacional. Tal interrupção é chamada falha de página. O S.O., então, seleciona uma moldura de página pouco utilizada e grava o seu conteúdo de volta ao disco, substituindo-a pela página requisitada.

Quanto à forma como a paginação pode ser implementada, podemos considerar a paginação simples e a paginação por demanda. Na primeira, todas as páginas lógicas do processo são mapeadas e carregadas para a memória física, isso supondo-se que o espaço de endereçamento de memória para um processo tenha o tamanho máximo igual à capacidade da memória física alocada para processos. No caso da paginação por demanda, apenas as páginas lógicas efetivamente acessadas pelos processo

são carregadas. Nesse caso, uma página marcada como inválida na tabela de páginas de um processo pode tanto significar que a página está fora do espaço lógico de endereçamento do processo ou que simplesmente a página ainda não foi carregada. Para descobrir qual das situações é a verdadeira basta conferir o descritor de processo, que contém os limites de endereçamento lógico do processo em questão.

Tabelas de Página

O propósito de tabelas de página é mapear páginas virtuais em molduras de página. No entanto, existem duas questões que devem ser consideradas:

A tabela de páginas pode ser extremamente grande, sendo que cada processo necessita de sua própria tabela de páginas;

O mapeamento deve ser rápido: o mapeamento do virtual para o físico deve ser feito em cada referência da memória, o que pode ocorrer diversas vezes em uma única instrução, não devendo tomar muito tempo para não se tornar um gargalo na execução da instrução.

Neste caso, há a necessidade de um mapeamento de páginas rápido e grande. Existem duas formas básicas de projetar tabelas de páginas:

ter uma única tabela de páginas, através de uma matriz de rápidos registradores de hardware, com uma entrada para cada página virtual, indexada pelo número da página. Esta é uma solução cara se a tabela de páginas é grande;

manter a tabela de páginas inteiramente na memória principal, sendo que o hardware precisa de apenas um registrador que aponta para o início da tabela de páginas.

Esta última solução possui algumas variações que têm desempenho melhor. Uma das propostas que busca evitar o problema de ter enormes tabelas de páginas na memória todo tempo é a de Tabelas de Páginas Multinível. Neste caso, existe uma tabela de primeiro nível e diversas tabelas de segundo nível. O importante aqui é que somente as tabelas mais utilizadas estão presentemente na memória. As demais se encontram em disco. Apesar de permitir um espaço de endereçamento muito grande, o espaço ocupado de memória principal é muito pequeno.

O sistema de tabela de páginas de dois níveis pode ser expandido para três, quatro ou mais níveis, sendo que níveis adicionais dão mais flexibilidade, mas a complexidade da implementação acima de três níveis dificilmente vale a pena. Em relação aos detalhes de uma única entrada de uma tabela de páginas, seu arranjo pode depender da máquina, mas os tipos de informação usualmente são os mesmos.

O campo mais importante é o número da moldura de página, pois o objetivo do mapeamento de páginas é localizar este valor. Ao lado dele, tem-se o bit de presente/ausente. Se este bit for 1, significa que a entrada é válida e pode ser utilizada. Se for 0, a página virtual a que esta entrada pertence não está atualmente na memória. Ao acessar uma entrada da tabela de páginas com este bit configurado como zero ocorrerá uma falha de página.

Os bits "proteção" informam que tipos de acesso são permitidos. Em sua forma simples, este campo contém apenas um bit, com 0 para leitura/gravação e 1 par leitura somente. Um arranjo mais sofisticado é manter 3 bits, cada um para habilitar leitura, gravação e execução da página. Os bits "modificada" e "referenciada" monitoram a utilização da página. Ambos os bits tem como objetivo auxiliar no momento da substituição de páginas. O último bit permite que o cache seja desativado para a página, recurso importante para páginas que são mapeadas em registradores de dispositivo em vez da memória.

Alguns bits auxiliares são, normalmente, adicionados na tabela de páginas para facilitar na substituição de páginas quando a memória física estiver cheia e for necessário retirar uma página lógica da memória física para alocar outra página lógica.

Tem-se o bit de modificação (dirty bit) assim que a página for carregada tem valor zero se a página for alterada, na memória física, altera-se o valor para 1, portanto se a página for a página vítima e o bit de modificação for zero não será necessário fazer a cópia da página lógica em memória para a página lógica em disco pois as duas são iguais. Bit de referência: é zero assim que a página lógica é alocada

na página física, se a página for acessada altera o valor para um (a MMU altera o valor). Bit de trava: usado em páginas que não podem sair da memória física, o Sistema Operacional "tranca" uma página lógica na memória física ativando esse bit.

TLBs – Translation Lookside Buffers – Memória Associativa

A TLB, também conhecida como memória associativa, é um dispositivo de hardware cujo propósito é mapear endereços virtuais em endereços físicos sem passar pela tabela de páginas. Usualmente, ela faz parte da MMU.

Ela constitui-se de um banco de registradores que armazenam um pequeno número de entradas, muito rápidas, contendo as tabelas de páginas mais utilizadas. Quando um endereço virtual é enviado a MMU, ela primeiramente verifica se o seu número de página virtual está presente na TLB. Se o resultado for positivo (hit), a moldura de página é tomada diretamente da TLB sem a necessidade de passar pela tabela de páginas na memória (mais lento). Caso contrário (miss), a pesquisa é feita normalmente na tabela de páginas presente na memória. Então, uma das entradas é removida da TLB e a entrada da tabela de páginas pesquisa é colocada em seu lugar.

A TLB melhora bastante o desempenho no acesso à tabela de páginas, visto que registradores são muito mais rápidos que a memória RAM. Suas desvantagens estão em seu custo (registradores são caros), seu tamanho limitado e o fato de existir uma única TLB na MMU, sendo esta compartilhada por todos os processos.

Para calcularmos o desempenho da TLB, podemos tomar h como taxa de acerto (taxa em que a página necessária estará na TLB). Então o erro seria 1-h. Sendo assim, a tempo de acesso hit (tempo necessário para pegar a página da TLB) é dada pelo tempo de acesso à TLB mais o tempo de acesso à memória uma única vez. Enquanto o tempo de acesso miss (quando falta a página na TLB) é dada pelo tempo de acesso à TLB mais o tempo de dois acessos à memória. O tempo de acesso médio é dado pelo tempo de acesso hit multiplicado por h somado do tempo de acesso miss multiplicado por (1-h). Resumindo em fórmulas, temos:

TacessoHit = TacessoTLB + TacessoMemoria

TacessoMiss = TacessoTLB + TacessoMemoria + TacessoMemoria

TacessoMedio = h x TacessoHit + (1-h) x TacessoMiss

A taxa de acerto (h) depende do tamanho da TLB e do algoritmo que a controla, mantendo as páginas mais utilizadas.

Tabelas de Páginas Invertidas

Outra abordagem para trabalhar com páginas é manter uma tabela onde cada entrada representa uma moldura de página ao invés de um endereço virtual. Desta forma, a entrada deve monitorar qual página virtual está associada àquela moldura de página.

Embora as tabelas de páginas invertidas economizem quantidade significativas de espaço (pelo menos nas situações em que o espaço de endereço virtual é muito maior que a memória física), elas tem a desvantagem de que o mapeamento (tradução) do endereço virtual para o físico é mais complexo e potencialmente mais lento. Uma forma de facilitar a tradução do virtual para o físico é a utilização da TLB pesquisada por software. A pesquisa pode ser feita a partir de um encadeamento de páginas virtuais que possuam um mesmo endereço hash.

Tamanho de Página

Um ponto do qual o projetista deve se preocupar é com o tamanho da página. Conforme visto anteriormente, se esse tamanho de página for grande, pode ocorrer de o processo utilizador não ocupar todo o espaço a ele destinado. Se a página tiver um tamanho demasiadamente pequeno, a tabela de páginas será muito grande.

É possível uma modelagem matemática. Considere que cada processo tenha tamanho de s bytes, e cada página tenha tamanho de p bytes. A tabela de páginas terá um espaço de e bytes por entrada.

Assim, o número de páginas que um processo precisará é de s/p. O espaço que esse processo ocupa na tabela de páginas é s*e/p. O tamanho perdido na última página devido a fragmentação interna será de p/2.

Assim, haverá um custo de s*e/p + p/2 da tabela de páginas. Para que o tamanho da página seja ideal, o custo será zero. Dessa forma, derivando a expressão anterior e igualando a zero obtemos que o tamanho ideal de página é de s*e*sqrt.

Thrashing

Estado no qual o sistema operacional ao invés de executar instruções efetivas "gasta" tempo efetuando a troca de páginas entre memória física e memória lógica, em outras palavras desperdiça um tempo significante trazendo ou removendo páginas da memória. Para o usuário a sensação é que o sistema está travado ou congelado e para o hardware há um significante acesso ao disco ao invés de processamento

A memória como um recurso importante deve ser gerenciado com cuidado, sendo o gerenciador de memória a parte do sistema operacional que gerencia a sua hierarquia. Seu trabalho é controlar que partes da memória que estão em uso e que partes não estão, alocar memória para os processos quando eles necessitam e desalocarem quando eles terminarem, gerenciar a troca entre a memória principal e o disco quando a memória principal é muito pequena para armazenar todos os processos; Esses são alguns papeis do gerenciador de memória.

Unidade de Gerenciamento de Memória (Memory Management Unit - MMU)

É um modulo de hardware que faz o mapeamento entre os endereços lógicos(end. da memória virtual) e os endereços físicos da memória (RAM), ou seja, é um dispositivo que transforma endereços virtuais em endereços físicos. A MMU normalmente traduz número de páginas virtuais para número de página físicas utilizando a cache chamada TLB-Translation LooKaside Buffer.

Gerenciamento Básico de Memória

São divididos em duas classes:

Monoprogramação sem Troca ou Paginação: esquema mais simples de gerenciamento, executando apenas um programa por vez, compartilhando a memória entre o programa e o sistema operacional. Apenas um programa é carregado por vez.

Multiprogramação com partições fixas: aumentando a utilização da CPU, a multiprogramação permite que enquanto um processo é bloqueado esperando a E/S acabar, outro pode utilizar a CPU, melhorando a utilização da CPU e assim evitando desperdícios de ciclos de processamento. Na multiprogramação a memória é dividida em N partes, normalmente quando o sistema é iniciado, Os jobs são colocados em filas de entrada associadas à menor partição capaz de armazena-lo, por usar partições de tamanho fixo, todos o restante de espaço de memória não utilizado pelo Job será perdido, tal desperdício é chamado de Fragmentação Interna.

Outro tipo de fragmentação existente é a Fragmentação Externa que acontece na seguinte ocasião: imagine que exista duas partições livres, uma de 50 e outra de 80 kbytes, nesse instante é criado um processo de 90 Kbytes que não poderá ser carregado em memória pela forma como ela é gerenciada, sendo assim acontece uma fragmentação externa para que processo possa "interagir" com a memória. Na Multiprogramação com partições fixas os processos são colocados em filas únicas ou não, esperando alguma partição que possa armazena-los.

Multiprogramação com partições Variáveis: o tamanho dos processos na memória podem variar dinamicamente com o passar do tempo, o tamanho das partições é ajustado dinamicamente às necessidades exatas dos processos. A imagem abaixo ilustra o funcionamento deste algoritmo, considerando a ocorrência de swapping(trazer um processo do disco para a memória [swap in] executá-lo durante um intervalo de tempo e depois devolvê-lo ao disco [swap out].

Diferentemente do esquema de partições fixa, na multiprogramação com partições variáveis o tamanho e a localização dos processos varia, a medida que o mesmo deixa e retorna à memória. A gerencia dos

espaços vazios é mais complicada, bem como a alocação e liberação das partições. O sistema operacional mantém uma lista de espaços livres na memória física. Sempre que um novo processo é criado esta lista é percorrida e será usada uma lacuna maior ou igual ao tamanho do processo em questão. O espaço que ultrapassar o tamanho do processo pode dar origem a uma nova partição. As forma de percorrer esta lista é :

Fisrt-fit: Inicia a procura a partir da primeira página de memória e vai varrendo até encontrar a primeira lacuna suficientemente grande para armazenar o processo.

Best-fit: varre toda a memória e escolhe a página mais ajustada ao tamanho do processo.

Worst-fit: varre toda a memória e escolhe a página menos ajustada ao tamanho do processo.

Next-fit: segue a mesma idéia do firt-fit, mas somente a primeira busca é iniciada na parte da memória, as outras iniciam onde terminou a última. Usa-se uma lista circular para permitir que, eventualmente, toda a memória seja percorrida.

Como já mencionado a troca cria lacunas na memória é possível juntar todas elas em um grande espaço, movendo-os para a mesma direção. Técnica chamada de compactação de memória.

Como já mencionado, existem processos que tentem a necessitar mais tamanho de memória conforme são executados, para isso, nada melhor que alocar uma pequena memória extra para esses processos, essa memoria extra somente deve constar no momento em que o processo esta na memória, quando ele retorna para o disco ele deve conter o tamanho real.

Gerenciamento de memória com mapas de Bits: a cada unidade de alocação da memória é atribuído um bit para dizer se a posição está livre ou ocupada. Assim, o conjunto de todos os bits é representado em uma tabela, denominada mapa de bits, que mapeia todas as posições de memória dizendo o estado de cada uma. Devemos ressaltar que o tamanho da unidade de alocação é muito importante e quanto menor as unidades, maior será o mapa de bits. Como o mapa de bits também é armazenado em memória seu tamanho ocupará espaço útil e, consequentemente, uma parte da memória será desperdiçada. Quando um processo de k bits necessitar ser armazenado em memória a MMU deverá procurar no mapa k bits consecutivos indicando que a posição está vazia (pode ser o bit 0 ou 1). Como varrer o mapa de bits é lento este método quase não é usado.

Gerência de memória com lista ligada: As representações dos espaços livres e ocupados são feitos através de uma lista ligada, onde P indica uma região ocupada por um processo e H um espaço livre de memória (imagem abaixo). A lista pode estar ordenada por endereços de memória, conforme ilustrado na imagem. Assim como no mapa de bits, qualquer alteração nas posições de memória deve gerar uma alteração no mapeamento promovido pela lista ligada. Se a lista estiver ordenada por endereço uma atualização mais rápida é permitida sempre que um processo terminar de executar suas instruções ou for retirado da memória. A utilização de uma lista duplamente encadeada facilita no processo de atualização da mesma.

Existem alguns algoritmos que podem ser utilizados para alocar as informações na memória:

algoritmo da primeira alocação (first fit): procura-se pelo primeiro espaço na lista o suficientemente grande para armazenar o processo. É um algoritmo rápido pois ele gasta o tempo mínimo em procura. Se o processo não ocupa todo o espaço o restante é disponibilizado como buraco na lista. A pesquisa por espaço sempre inicia na parte baixa de memória, independentemente dos locais escolhidos para alocar os dados.

algoritmo da melhor alocação (best fit): busca em toda a lista o espaço cujo o tamanho seja o mais próximo possível do tamanho do processo. Este algoritmo é mais lento que o anterior pois precisa pesquisar em toda a lista para descobrir qual a melhor opção.

algoritmo da próxima alocação (next fit): semelhante ao first-fit, só que a próxima alocação inicia com uma busca a partir da página onde terminou a alocação anterior e não da parte baixa da memória.

algoritmo da pior alocação (worst fit): procura pelo maior espaço capaz de armazenar o processo, de tal forma que o espaço restante seja grande o suficiente para armazenar outro processo.

Memória Virtual

Técnica utilizada para executar um programa que não cabe na memória existente, consiste em manter partes do programa, dos dados e da pilha no disco, sendo que existe uma forma de decisão de quais processos devem permanecer no disco e quais na memória. Técnica realizada de forma automática pelo computador, podemos alocar diversos processos na memória virtual, de forma que cada um pensa ter uma quantidade de memória que somadas ultrapassam a quantidade real de memória.

Então:

É um espaço variável e reservado no disco onde o Sistema Operacional continua armazenado os dados que não couberam na memória RAM. Ou seja, na memória RAM ficam os dados temporários usados enquanto o computador está ligado, se ela enche, os dados vão sendo gravados no HD.

Devido aos Sistemas Operacionais apresentarem características variadas, determinados procedimentos tornam-se por vezes semelhantes ou próprios de cada Sistema. O gerenciamento de memória é um procedimento fundamental na objetividade de todos os Sistemas Operacionais.

A maioria dos computadores trabalha com o conceito de hierarquia de memória, possuindo uma pequena quantidade de memória cachê, muito rápida, uma quantidade de memória principal (RAM) e uma quantidade muito grande de memória de armazenamento em disco (HD), considerada lenta.

A memória é um dos itens mais importantes dentro de um computador, por isso a importância de um bom gerenciamento.

Gerenciamento (ou gestão) de memória é um complexo campo da ciência da computação e são constantemente desenvolvidas várias técnicas para torná-la mais eficiente. Em sua forma mais simples, está relacionado em duas tarefas essenciais:

Alocação: Quando o programa requisita um bloco de memória, o gerenciador o disponibiliza para a alocação;

Reciclagem: Quando um bloco de memória foi alocado, mas os dados não foram requisitados por um determinado número de ciclos ou não há nenhum tipo de referência a este bloco pelo programa, esse bloco é liberado e pode ser reutilizado para outra requisição.

ou melhor

Os que não realizam paginação ou troca entre disco e memória

O que não realizam este processo

O ideal seria, uma memória que não precisasse deste artificios, uma memória rapida, de grande capacidade e a um custo pequena. o que não acontece.

O que se vê são vários tipos de memória trabalhando paralelamente com os recursos computqacionais.

Assim, devido a suas similaridade e características distintas, as melhorias precisam de um meio para que possam ser gerenciadas da melhor forma possível.

É uma estrutura que possui uma hierarquia devido suas características e atividades desenvolvidas.

Cache (rápida e pequena, volátil, custo alto)

RAM (velocidade média, tamanho médio e volátil, custo médio)

Memorias Secundarias (velocidade lenta, tamanho grande e não volátil, custo baixo)

Também quando discutimos sobre gerenciamento de memória(GM), vimos que, apesar de seu crescimento exponencial com o tempo, cada vez temos mais memorias nos computadores modernos, temos do outro lado os programas cada vez mais exigentes com espaço, ou seja, cada vez temos programas maiores que a memória disponível.

A tendencia é que, com as interfaces gráficas, os programas ocupem ainda mais as memorias. É um problema para os projetistas de Sistemas Operacionais que devem prever mecanismo de Gerenciamento de memória mais eficientes possíveis.

Outra questão está relacionada as modalidades de programação distinta, tais como:

Monoprogramação sem troca de processos ou paginação

Um esquema mais simples de gerenciamento de memória, onde a memória é compartilhada entre o SO e os programas.

Podemos neste caso, possui três configurações:

A 1ª configurações esteve presente nos Mainframes ou computadores de grande porte

A 2ª esta presente nos palmtops e outros computadores de mão

E a 3ª configuração esteve presente nos sistemas MS-DOS

Multiprogramação com Partições Fixas

Atualmente, a monoprogramação não é mais utilizada, hoje, se um processo fica bloqueado esperando um recurso de E/S, outro processo poderá está utilizando a CPU naquele instante. Em sistemas Operacionais de rede, com vários usuários startando processos, isto bem evidente.

Esta habilidade presente somente neste sistemas operacionais rede, agora também faz parte dos sistemas dos usuários.

Como isso acontece, é realizado a divisão da memória em n partições que pode ser feita manualmente. Quando um job chega, há duas possibilidades: ele é colocado em uma fila de entrada da menor partição capaz de armazená-lo ou ele é colocado em uma fila de entrada única.

Como o tamanho das partições são fixas, se um job não ocupar todo o espaço da partição, este é pedido

Troca de Processos

Em sistemas computacionais modernos, usualmente, não há memória principal suficiente para armazenar os processos atualmente ativos. Desta forma, os processos excedentes são mantidos em disco e trazidos para a RAM quando necessário. Há duas formas de resolver este problema: troca e memória virtual, (sendo a última explicada posteriormente).

A operação de troca consiste em manter na memória alguns processos por um determinado período e, a seguir, trocar estes processos pelos demais que estão esperando em disco.

A partição utilizada para cada processo pode ser fixa ou definida dinamicamente (de acordo com o tamanho do processo). Durante a operação de troca, é possível surgirem lacunas de memória que podem ser reunidas em um espaço maior através da técnica de compactação de memória.

No entanto, a atribuição dinâmica de memória conduz a um problema adicional: o gerenciamento de memória em relação a quais parcelas estão em uso e/ou quais estão livres. Existem duas formas de resolver este problema.

Gerenciamento de Memória com Mapas de Bits

Com mapas de bits, a memória é dividida em unidades de alocação. Cada bit do mapa representa uma unidade de alocação, sendo que se o bit for 0, a unidade está livre; caso contrário, a unidade está ocupada.

Quanto menor for a unidade de alocação, maior será o mapa de bits e vice-versa. O maior problema com os mapas de bits é que procurar uma lacuna (sequência de 0s) suficientemente grande para um determinado processo pode ser uma operação muito lenta.

Gerenciamento de Memória com Listas Encadeadas

Neste caso, é mantida uma lista encadeada com os segmentos de memória livres e encadeados. Uma possível configuração seria manter, em cada entrada, o endereço em que inicia, o seu comprimento e, evidentemente, o ponteiro para a próxima entrada.

Memória Virtual

Quando os programas excedem o espaço de memória física disponível para eles, é possível dividir o programa em pedaços chamados overlays. No entanto, isto exige que o programador seja responsável por esta divisão. De forma a facilitar o desenvolvimento de programas, passando esta tarefa para o Sistema Operacional, foi criado o método conhecido como Memória Virtual.

A ideia básica da memória virtual é que o tamanho combinado do programa, dos seus dados e da pilha pode exceder a quantidade de memória física disponível para ele, ou seja, neste caso, a simples troca, vista anteriormente, não resolveria o problema. O Sistema Operacional, então, mantém partes do programa atualmente em uso na memória principal e o restante em disco.

Paginação

Em qualquer computador, existe um conjunto de endereços que um programa pode produzir. Estes endereços gerados por programas são chamados endereços virtuais e formam o espaço de endereço virtual.

A MMU (Memory Management Unit) é responsável por mapear os endereços virtuais para endereços físicos de memória, sendo que consiste de um chip ou uma coleção de chips.

O espaço de endereço virtual é dividido em unidades chamadas páginas. As unidades correspondentes na memória física são chamadas molduras de página. As páginas e as molduras têm sempre exatamente o mesmo tamanho.

No espaço físico (memória) tem-se várias molduras de página. Por exemplo, podem existir 05 páginas situadas no espaço de endereço virtual que são mapeadas na memória física. No entanto, o espaço de endereço virtual é maior que o físico. As outras páginas não são mapeadas. No hardware real, um bit presente/ausente em cada entrada monitora se a página é mapeada ou não.

Quando um programa tenta utilizar uma página não mapeada em uma moldura, a MMU detecta o acontecimento (que a página não está mapeada) e gera uma interrupção, passando a CPU para o Sistema Operacional. Tal interrupção é chamada falha de página. O S.O., então, seleciona uma moldura de página pouco utilizada e grava o seu conteúdo de volta ao disco, substituindo-a pela página requisitada.

raduzir não é mais “apenas” transferir, com o mínimo de perdas e distorções, ideias de um idioma para outro. Em função das necessidades dos consumidores de hoje (e, por consequência, dos fornecedores), traduzir agora inclui os conceitos de “traduzir com preço competitivo” e “traduzir com homogeneidade de vocabulário” – para não falar de qualidade do atendimento, rapidez na entrega e tantas outras coisas que o tradutor está aprendendo a entregar com seu trabalho.

As questões do “preço competitivo” e da “homogeneidade de vocabulário”, por si, já justificam, na área técnica, o uso das memórias de tradução e dos gerenciadores de terminologia.

Acreditar, porém, que o simples fato de comprar o software e saber usá-lo já resolveria tais questões – isso seria ingenuidade. Depois que aprendemos a “pilotar” o software, aparecem novas necessidades.

E a principal é: pensar mais. Pensar, decidir, organizar: ora, se passamos a ter um banco de dados de unidades de tradução e outro de terminologia, então precisamos saber utilizar esse patrimônio. Se aproveitamos itens de outros trabalhos, precisamos ter controle absoluto sobre o modo como os aproveitamos. A gestão das memórias de tradução e dos glossários aparece como uma nova tarefa, porém, mais que um trabalho adicional, ela otimiza o uso do nosso patrimônio e evita erros – às vezes caros – durante o uso

O presente trabalho irá tratar sobre gerenciamento de memória nos sistemas operacionais, que é um recurso que deve ser manuseado com cuidado. Serão apresentadas a evolução da memória ao longo dos anos, sua estrutura, como funciona e os diversos tipos de procedimentos inclusos no gerenciamento básico de memória, como a definição de mono e multiprogramação. Serão apresentados, também, os conceitos de realocação e proteção, troca e uma rápida introdução à memória virtual.

A memória é um recurso muito importante que requer atenção. Ela vem crescendo cada vez mais ao longo dos anos, além de melhorar seu desempenho (velocidade), o que é ideal para os programadores, visando também a característica de ser não-volátil, ou seja, o conteúdo não é perdido quando a memória perde contato com a energia. A maioria dos computadores possui uma hierarquia de memória, com uma pequena parte (cache) volátil rápida e cara, e outra parte, a maior, lenta, barata e não-volátil.

Em Sistemas Operacionais, a parte que cuida da memória é chamada de gerenciador de memória, que controla o que está e o que não está em uso, alocando a mesma para recursos que necessitem.

Os sistemas de gerenciamento podem ser divididos em duas classes: os que fazem troca e paginação, movendo processos entre memória principal e disco, e os que não fazem.

Monoprogramação sem Troca ou Paginação - esquema mais simples possível, executando somente um programa por vez, compartilhando memória entre esse programa e o sistema operacional. Quando organizado dessa maneira, somente um processo por vez pode ser executado. O usuário digita um comando, o sistema operacional cópia do disco para a memória e executa, sobrepondo o primeiro quando um novo comando é adicionado e executado.

Multiprogramação com Partições Fixas - permite vários processos simultâneos, aumentando a utilização da CPU. A maneira mais fácil de implementá-la é dividir a memória em n partições.

Quando um job chega, é 'alocado' para a menor partição capaz de armazená-lo (para que não haja perda), e assim sucessivamente, formando uma espécie de fila. Há também o modelo que utiliza partições fixas, o MFT (Multiprogramação com um número Fixo de Tarefas), onde cada job que chega aguarda uma partição livre adequada, onde são carregados e aguardam pela conclusão. Atualmente poucos sistemas operacionais suportam esse modelo.

A multiprogramação introduz problemas essenciais que devem ser resolvidos - realocação e proteção. A realocação procura assegurar que cada aplicação tenha seu próprio espaço de endereçamento, começando em zero. Faz uso do endereço absoluto, 100, por exemplo, somado com a localização da partição (partição 2, por exemplo, 100 + 200K). Para que haja uma realocação como essa durante o carregamento, o linkeditor inclui no programa binário uma lista ou mapa de bits, informando quais palavras do programa são endereços a serem realocados e quais são códigos de instruções, constantes ou outros itens.

No condizente à proteção, um programa malicioso poderia criar uma instrução que lê e grava qualquer palavra na memória e em seguida saltar para ela, já que trabalha com endereços absolutos. Em sistemas multiusuários, é indesejável que um processo leia ou grave memória pertencente a outros usuários. Para isso, a IBM dividiu a memória em blocos de 2KB e atribuiu um código de proteção de 4 bits para cada bloco. Apenas o SO poderia alterar os códigos e a chave de proteção.

Uma outra solução para ambos os problemas de realocação e proteção é equipar a máquina com registradores especiais de hardware, os chamados registrador base e registrador limite. Quando um processo é agendado, o base é carregado com o endereço do início da partição, e o de limite com o comprimento da partição.

Todo endereço gerado tem o registrador base automaticamente adicionado a si mesmo antes de ser enviado para a memória, assim uma instrução CALL 100 se transforma em CALL 100K + 100, sem que a instrução seja modificada. O registrador limite cerifica-se de que os endereços não tentarão endereçar memória fora da partição atual.

Um processo pode ser removido da memória temporariamente para um armazenamento auxiliar em seguida retornado para continuar sua execução. muitas vezes não há memória suficiente para armazenar todos os processos ativos. Os processos em excesso são mantidos no disco e trazidos de lá para execução dinamicamente.

Porém, um processo não pode invadir a memória de outro processo. No caso das partições variáveis a alocação da memória é dinâmica, ou seja, é alterada à medida que os processos entram e saem da memória. A vantagem é a melhor utilização da memória em relação às partições fixas, e a desvantagem é que as trocas de processos deixam muitos espaços vazios na memória fragmentando-a. Para este problema é possível a reorganização da memória, contudo esse processo é muito demorado.

A quantidade de memória que deve ser alocada a um processo quando ele for criado é um ponto muito importante a ser considerado. Se o processo não puder crescer então a alocação é simples, ou seja, o SO aloca somente o necessário. Uma solução alternativa é prevenir ou alocar uma memória extra para o seu possível crescimento. Pode-se criar uma memória para variáveis dinâmicas que são alocadas e desalocadas (Heap), e outra (Stack) para variáveis locais comuns e endereços de retorno. Para que isso seja possível , o SO precisa gerenciar esse processo, usando duas práticas para gerenciamento de memória, dependendo do hardware disponível.

Uma delas é a troca, que é a mais simples, que consiste em trazer o processo inteiro, executá-lo temporariamente e então devolvê-lo ao disco. A outra estratégia é a memória virtual, que permite que os programas executem mesmo quando estão apenas parcialmente na memória principal. A compactação de memória é quando múltiplas lacunas na memória são juntas em um grande espaço, movendo o mais baixo possível. Um ponto importante é quanta memória deve ser alocada para um processo quando é criado ou recuperado. Se processos são criados com tamanhos fixos, a alocação é simples, alocando-se somente o necessário.

Em relação à troca, ela pode ser dividida em dois processos: gerenciamento por mapas de bits e gerenciamento com listas encadeadas, que serão vistas a seguir.

Gerenciamento de Memória com Mapas de Bits

Com o mapa de bits há uma divisão da memória em unidades de alocação, tanto muito pequenas como muito grandes. Correspondente a cada unidade de alocação há um bit no mapa de bits, que é 0 se a unidade está livre ou 1 se estiver ocupada. Quanto menir a unidade de alocação, maior o mapa de bits. Um mapa de bits oferece uma maneira simples de monitorar palavras de memória em uma quantidade fixa de memória, já que o tamanho do mapa de bits depende do tamanho da memória e da unidade de alocação. O problema é que quando um processo de k unidades é enviado para a memória, o gerenciador deve fazer uma busca no mapa por uma lacuna de k unidades disponível. Essa pesquisa é uma operação lenta, sendo este um argumento contra os mapas de bits.

Gerenciamento de Memória com Listas Encadeadas

Uma outra forma de monitorar a memória é manter uma lista encadeada dos segmentos de memória alocados e livres, onde um segmento é um processo ou lacuna entre dois processos. Quando os processos e as lacunas são mantidos em uma lista classificada por endereço, vários algoritmos podem ser utilizados para alocar memória para um processo criado recentemente ou trocado a memória. O algoritmo mais simples é chamado de algoritmo do primeiro ajuste. O algoritmo do primeiro ajuste é rápido, pois faz o mínimo de pesquisa de lacunas na lista. Uma variação secundária do algoritmo do primeiro ajuste é a do próximo ajuste, que funciona da mesma maneira que o primeiro, exceto pela monitoração da posição em que ele está sempre encontra uma lacuna adequada.

Outro algoritmo conhecido é o do melhor ajuste, o qual pesquisa na lista inteira e pega a menor lacuna adequada. Antes de dividir uma lacuna grande (que pode ser necessária posteriormente), o melhor ajuste tenta localizar uma lacuna que é próxima do tamanho real necessário.

Esse algoritmo é mais lento do que o de primeiro ajuste, pois deve pesquisar na lista inteira uma vez que é chamado, além de ter um maior desperdício, já que tente a encher a memória de lacunas minúsculas e inúteis. Para que seja evitado o problema de dividir lacunas quase exatas entre um processo e uma lacuna minúscula, pode-se pensar no pior ajuste, que resume-se a sempre pegar a maior lacuna disponível, de modo que a lacuna resultante seja suficientemente grande para ser útil.

Todos esses quatro algoritmos podem ser acelerados ao se manter listas separadas para processos e lacunas, onde todos eles concentram-se em localizar as lacunas, não processos. O preço a ser pago por essa aceleração é a complexidade adicional e a queda de velocidade ao se deslocar memória. Ainda um outro algoritmo de alocação é o ajuste rápido, que mantém listas separadas para alguns dos tamanhos exigidos mais comuns.

Com o ajuste rápido, localizar uma lacuna do tamanho exigido é extremamente rápido, mas tem a mesma desvantagem dos outros que classificam por tamanho de lacuna.

Antigamente, era feita a divisão da memória em pedaços chamados overlays, onde o overlay 0 começava primeiro e, quando pronto, chamava outro overlay. Alguns desses sistemas eram altamente complexos, permitindo múltiplos overlays na memória simultaneamente. Embora o trabalho de comutação overlay fosse feito pelo sistema, o trabalho de dividir o programa em pedaços era feito pelo programador, o que gerava muito trabalho ao dividir grande programas em pequenos pedaços, até que uma solução foi encontrada. Essa solução veio a ser conhecida por memória virtual, cuja ideia básica é que o tamanho combinado do programa, dos dados e da pilha pode exceder a quantidade de memória física disponível para ele. O sistema operacional mantém essas partes do programa atualmente em uso na memória principal e o restante no disco.

A memória virtual pode também trabalhar em sistemas de multiprogramação, com pedaços de vários programas diferentes na memória ao mesmo tempo. Enquanto um programa está esperando parte dele mesmo serem trazidas para a memória, ele espera a E/S e não pode executar, então a CPU pode ser dada a outro processo, assim como em qualquer sistema de multiprogramação.

No Linux a memória funciona da seguinte maneira, processos que estão em execução têm prioridade na memória, quando termina um processo e se tiver espaço na memória, ficam resíduos desse processo na memória para uma futura volta desse processo ser mais rápida. Caso essa memória RAM esteja lotada com processos que estão em execução, aí começa a utilização da memória SWAP (troca)". (LIMA, 2007)

A memória virtual do Linux é paginada, ou seja, é possível que sejam executados programas que tenham o tamanho maior que a própria memória física do computador. É como se o sistema operacional assumisse a responsabilidade de manter em sua memória e sustentasse os programas em execução, deixando o resto no HD. Só que a utilização da memória virtual torna o computador mais lento, mas faz com ele que aparente ter mais memória RAM do que tem de fato.

Ou seja, quanto mais memória é requisitada, o Linux passa a transferir arquivos não usados há algum tempo, da memória RAM à memória virtual, também chamada de swap, liberando, assim, memória física para os aplicativos.

Em suma, o que acontece é que o Linux utiliza a energia ociosa (que não está sendo aproveitada) para cache, a fim de agilizar os processos solicitados pelo usuário naquele determinado momento. Durante esse período de ociosidade da energia, o sistema toma para si essa memória, mas logo que o usuário necessite do recurso, o Linux libera imediatamente.

É possível afirmar, então, que o Linux gerencia a energia e a memória de forma muito inteligente, pois ao invés de deixá-la "sem fazer nada" utiliza para agilizar outros processos. Esse é o gerenciamento de memória do Linux!

Gerenciamento de Memória

Gerenciador de Memória é a parte do SO que é responsável por cuidar de quais partes da memória estão em uso, quais estão livres, alocar memória a processos quando eles precisam, desalocar quando eles não necessitarem mais e gerenciar a troca dos processos entre a memória principal e o disco (quando a memória principal não é suficiente para manter todos os processos)

Maneiras de Gerenciar a Memória:

Gerenciamento sem Troca ou Paginação: troca e paginação são métodos utilizados de movimentação da memória para o disco e vice-versa durante a execução dos processos. Sem troca ou paginação é o caso mais simples.

Monoprogramação sem Troca ou Paginação: temos um único processo sendo executado por vez, de forma que o mesmo possa utilizar toda a memória disponível, com exceção da parte reservada ao SO (que permanece constante em local pré-determinado). O SO carrega um programa do disco para a memória executa-o e em seguida aguarda comandos do usuário para carregar um novo programa, que irá se sobrepor ao anterior.

Multiprogramação: a monoprogramação não é mais utilizada em sistemas grandes, pois:

Muitas aplicações são mais facilmente programáveis, quando as dividimos em dois ou mais processo;

Os grandes computadores em geral oferecem serviços interativos simultaneamente para diversos usuários (seria impossível trabalhar com um único processo por vez, pois representaria sobrecarga devido à constante necessidade de chavear de um processo para outro – constantemente lendo e escrevendo no disco);

É necessário que diversos processos estejam “simultaneamente” em execução devido as operações de E/S, que implica em grandes esperas nas quais por questão de eficiência a UCP deve ser entregue a outro processo.

Multiprogramação com Partições Fixas: consiste em dividir a memória existente em n partições fixas, podendo ser de tamanhos diferentes. Essas partições poderiam ser criadas ao inicializar o sistema pelo operador.

Uma maneira de se fazer isso seria: criar uma fila para cada partição existente e cada vez que um processo é iniciado, ele é colocado na fila de menor partição capaz de o executar. Os processos em cada partição são escolhidos de acordo com alguma forma de política, por exemplo, o primeiro a chegar é atendido antes. O problema é que pode ocorrer que uma partição grande esteja sem utilização, enquanto que diversos processos estão aguardando para utilizar uma partição menor. Para resolver isso podemos fazer o seguinte: estabelecer apenas uma fila para todas as partições e quando uma partição fica livre, um novo processo que caiba na partição livre é escolhido e colocado na mesma. A melhor forma de fazer a escolha seria percorrer a fila procurando o maior processo aguardando que caiba na partição livre, pois se a partição livre for entregue para o primeiro processo da fila, pode ocorrer que uma partição grande seja entregue a um processo pequeno.

Realocação e Proteção: há a necessidade de realocações, pois processos diferentes executam em posições diferentes de memória e com endereços diferentes. Uma possível solução é modificar as instruções conforme o programa é carregado na memória (quando o SO carrega o programa, adiciona a todas as instruções que se referenciarem a endereços, o valor do ponto inicial de carga do programa). Esta solução exige que o linker coloque no início do código do programa, uma tabela que apresente as indicações das posições no programa que devem ser modificadas no carregamento. Mas isso não resolve a proteção, pois um programa malicioso ou errado pode ler ou alterar posições na memória de outros usuários, já que as referências são sempre as posições absolutas de memória.

Uma solução adotada para isso foi dividir a memória em unidades de 2 KB e associar um código de proteção de 4 bits a cada uma dessas regiões. Durante a execução de um processo, o PSW contém um código de 4 bits que é testado com todos os acessos à memória realizados pelo processo, e gera uma interrupção se tentar acessar uma região de código diferente.

Uma solução alternativa para o problema da realocação e da proteção é a utilização de registradores de base e limite. Sempre que um processo é carregado na memória, o SO ajusta o valor do registrador de base de acordo com a disponibilidade de memória. Toda vez que um acesso é realizado na memória pelo processo, o valor do registrado é automaticamente somado, assim não há necessidade de que o código do programa seja modificado durante o carregamento. O registrador de limite indica o espaço de memória que o processo pode executar, então todo acesso realizado pelo processo à memória é testado com o valor do registrador limite para a validação do seu acesso. O método dos registradores permite que um programa seja movido na memória, mesmo após já estar em execução, o que antes não era possível sem antes alterar os endereços novamente.

Troca (swapping): num sistema de batch, desde que se mantenha a UCP ocupada o máximo de tempo possível, não há necessidade de se complicar o método de gerenciamento de memória. Mas num sistema de time-sharing, onde muitas vezes existe menos memória do que o necessário para manter todos os processos de usuário, então é preciso que uma parte dos processos seja temporariamente mantida em disco. Para executar processos que estão no disco, eles devem ser enviados para a memória, o que significa retirar algum que lá estava. Este processo é denominado troca.

Multiprogramação com Partições Variáveis: partições que variam conforme as necessidades dos processos, em tamanho, em localização e em número de partições, melhorando a utilização da memória (mas complica a alocação e desalocação da memória). Compactação de memória que é a combinação

de todos os buracos formados na memória em um único é raramente utilizada devido a grande utilização de UCP requerida.

Para determinarmos quanta memória deve ser alocada a um processo quando ele é iniciado, temos duas situações: se os processos necessitarem de uma quantidade pré-fixada e invariante de memória basta alocar a quantidade necessária a cada processo ativo. E o outro caso é quando os processos necessitam de mais memória durante o processamento (alocação dinâmica de memória). Neste caso pode existir um buraco de memória próximo ao processo bastando alocar a memória desse buraco ou o processo pode estar cercado por outros processos, ou o buraco que existe não é suficiente. Para os dois últimos casos temo que tomar algumas das seguintes ações: mover o processo par um buraco de memória maior e se não houver tal espaço, alguns processos devem ser retirados da memória para deixar espaço para esse processo e se não houver espaço no disco para outros processos, o processo que pediu mais espaço na memória deve ser morto. Quando se espera que diversos processos cresçam durante a execução, o melhor seria reservar espaço extra para esses processos quando eles são criados para eliminar a sobrecarga de lidar com movimentação ou troca de processos.

Gerenciamento de Espaço: as duas principais formas de cuidar da utilização de memória são:

Gerenciamento com Mapa de Bits: A memória é subdividida em unidades de um certo tamanho. A cada unidade é associada um bit que se for 0 indica que essa parte da memória está livre e se for 1 indica que está ocupada. O tamanho deve ser cuidadosamente escolhido. A desvantagem é que quando um novo processo que ocupa k unidades de memória deve ser carregado na memória, o gerenciador deve percorrer o mapa de bits para encontrar k bits iguais a zero consecutivos, o que não é um processo simples.

Gerenciamento com Listas Encadeadas: mantemos uma lista encadeada de segmentos alocados e livres, sendo que cada segmento é um processo ou um buraco entre dois processos. A lista apresenta-se em ordem de endereços, e quando um processo termina ou é enviado para o disco, e a atualização da lista ocorre da seguinte maneira: cada processo, desde que não seja nem o primeiro nem o último da lista, apresenta-se cercado por dois segmentos, que podem ser buracos ou outros processos. Os buracos adjacentes devem ser combinados num único. Para escolher o ponto em que deve ser carregado um processo recém criado ou que veio do disco por uma troca, vamos utilizar alguns algoritmos assumindo que o gerenciador de memória sabe quanto espaço alocar no processo:

First Fit (primeiro encaixe): percorrer a fila até encontrar o primeiro espaço em que caiba o processo. É um algoritmo rápido.

Next Fit (próximo encaixe): o mesmo que o algoritmo anterior, só que ao invés de procurar sempre a partir do início da lista, procura a partir do último ponto em que encontrou. Desempenho próximo ao anterior.

Best Fit (melhor encaixe): consiste em verificar toda a lista e procurar o buraco que tiver espaço mais próximo das necessidades do processo. É mais lento, e desempenho pior que o First Fit

Worst Fit (pior ajuste): pega sempre o maior buraco disponível. Desempenho ruim.

Esses algoritmos podem ter sua velocidade aumentada pela manutenção de duas listas separadas, uma para processos e outra para buracos. Quando temos duas listas separadas, outro algoritmo é possível. É o Quick Fit (ajuste rápido), que consiste em manter listas separadas para alguns dos tamanhos mais comuns especificados (ex. uma fila para 2k, outra para 4k, outra para 8k etc). Neste caso, a busca de um buraco com o tamanho requerido, é extremamente rápido, entretanto, quando um processo termina, a liberação de seu espaço é complicada, devido à necessidade de reagrupar os buracos e modificá-los de fila.

Alocação de espaço de troca (swap): espaço de troca é o espaço ocupado no disco pelos processos que aí estão guardados, pois foram retirados da memória devido a uma troca. Os algoritmos para gerenciar o espaço alocado em disco para swap são os mesmos apresentados para o gerenciamento de memória. A diferença é que em alguns sistemas, cada processo tem no disco um espaço reservado para o mesmo e na memória ele é constantemente mudado de lugar. Além disso, como os discos são dispositivos de bloco, a quantidade de espaço reservado para os processos no disco deverá ser múltipla do tamanho do bloco.

Memória Virtual: A primeira solução adotada para programas grandes demais para a quantidade de memória foi a utilização de overlays. Nesta técnica o programa era subdividido em partes menores (overlays), que podiam ser rodadas separadamente e quando um overlay terminava a execução um outro poderia ser carregado na mesma posição de memória utilizada pelo anterior. O problema é a divisão do programa em overlays não é simples e deve ser realizada pelo programador.

A técnica de memória virtual para executar um programa que não cabe na memória existente consiste em manter partes do programa, dos dados e da pilha no disco, sendo que existe uma forma de decisão de quais processos devem permanecer no disco e quais na memória. Esta técnica é realizada de forma automática pelo computador. Podemos alocar diversos processos na memória virtual, de forma que cada um pensa ter uma quantidade de memória que somadas ultrapassam a quantidade real de memória.

Técnicas Utilizadas em Sistemas com Memória Virtual:

Paginação: espaço virtual é o espaço de memória que pode ser referenciado por um programa qualquer em dado processador. Ex: um processador com endereçamento de 16 bits, possui um espaço virtual de 64 kbytes. Quando uma instrução como: LD A,(1000h) o 1000h corresponde a um endereço virtual, de um espaço de endereçamento virtual de 64k bytes. Em um computador sem memória virtual, o endereço virtual corresponde ao endereço efetivamente colocado no duto de endereçamento da memória. Quando o computador possui memória virtual, esse endereço virtual é enviado para uma unidade de gerenciamento de memória (MMU), que corresponde a um chip ou um conjunto de chips que translada esse endereço virtual em um endereço físico, de acordo com uma tabela.

O espaço de endereços virtuais é dividido em unidades chamadas páginas e o espaço de memória física é dividido em unidades chamadas quadros de página, de mesmo tamanho das páginas. A MMU tem uma tabela que indica para cada página, qual o quadro de página que corresponde à mesma. Se o processador tenta acessar o endereço 0, a MMU verifica que isto corresponde ao primeiro endereço da primeira página, verifica então que essa primeira página está alocada no terceiro quadro de página. Converte então esse endereço para 8192 (decimal) e envia o endereço convertido para a memória (nem a memória e nem o processador precisam ficar sabendo da existência de paginação).

Como nem todas as páginas do espaço virtual podem estar residentes na memória simultaneamente, ocorrer o caso de que um acesso seja realizado para um página que não está na memória. Para saber isso a MMU mantém na tabela de translação um bit para cada página que indica se a mesma está presente ou não na memória. Se um acesso for realizado a uma página ausente, é gerada uma falta de página, que chama uma rotina de tratamento de interrupção específica para o SO, que então se encarrega do carregamento da página faltante e o ajuste correspondente na tabela de translação.

A forma mais comum de implementação da MMU, é escolher alguns dos bits mais significativos do endereço virtual como indicadores do número de página e o restante dos bits como um deslocamento dentro dessa página. Ex: na figura acima, de 16 bits do endereço virtual, 12 serão utilizados para o deslocamento e 4 serão utilizados como um índice para qual das 16 páginas está sendo referenciada. A MMU pega os 4 bits do índice da página, acessa a posição correspondente da tabela de translação, verifica se a página está presente na memória, se não estiver, gera uma interrupção para carregamento e depois verifica o valor colocado nessa entrada da tabela de translação e os junto aos 12 bits de deslocamento dentro da página.

A paginação fornece uma forma de se conseguir grande espaços de endereçamento lineares em uma quantidade finita de memória física.

Segmentação: Na segmentação temos um espaço bidimensional no sentido de que é dividido em uma um certo número de segmentos, cada um com dado número de bytes. Dessa forma, um endereçamento é sempre expresso da forma (segmento, deslocamento). Os diferentes segmentos são associados a diversos programas ou mesmo arquivos, de forma que neste caso, os arquivos podem ser acessados como se fossem posições de memória. Na segmentação os programadores tentam colocar entidades diferentes em segmentos diferentes para facilitar o compartilhamento de objetos entre processos diferentes.

A maioria dos computadores trabalha com o conceito de hierarquia de memória, possuindo uma pequena quantidade de memória cache, muito rápida, uma quantidade de memória principal (RAM) e uma

quantidade muito grande de memória de armazenamento em disco (HD), considerada lenta. O problema básico para o gerenciamento de memória é que os programas atuais são muito grandes para rodarem, completamente, na memória cache. O gerenciador de memória deve ser capaz de controlar parte da memória que está em uso (e quais não estão), alocar memória para processos quando eles necessitam e desalocar quando eles terminam e, principalmente, gerenciar a troca entre a memória principal e o disco, quando a memória principal é muito pequena para armazenar todos os processos.

Existem dois tipos de memória principal: a memória lógica e a memória física. A memória lógica é aquela manipulada pelos programas, ela é visível para os programas; sempre que um programa necessita alocar um espaço na memória esse espaço é alocado em memória lógica. A memória física é a memória implementada pelos circuitos integrados é nela que os espaços alocados em memória lógica vão realmente residir, portanto a memória física tem tamanho menor que a memória lógica, geralmente. Para isso é necessário realizar uma tradução de endereços lógicos para endereços físicos, pois assim um programa que aloca uma memória lógica possa ter de fato uma memória física alocada para si. Esse processo de tradução de endereços lógicos em endereços físicos é realizado por uma unidade de gerência de memória chamada MMU

Os processos não enxergam a memória física, hardware usado para endereçar os circuitos integrados de memória, e sim a memória lógica, que é a memória capaz de ser endereçada e acessada pelo conjunto de instruções do processador, sendo que cada processo possui a sua memória lógica que é independente da memória lógica dos outros processos. A memória física é implementada pelos circuitos integrados de memória, pela eletrônica do computador. Ela possui espaço de endereçamento físico que é o conjunto formado por todos os endereços dos circuitos integrados que formam a memória, normalmente esses endereços são em formato hexadecimal. A memória lógica possui um espaço de endereçamento lógico, maior que o espaço de endereçamento físico, é formado por todos os endereços lógicos gerado pelo processo, sendo gerado pela CPU e único por processo.

Como o processo "enxerga" endereço de memória lógico e o hardware "enxerga" endereço de memória físico é necessário ter a conversão de endereço de memória lógico para endereço de memória físico. Esse mapeamento de endereço lógico em endereço físico é feito pela MMU, unidade de gerência de memória. Basicamente a MMU que vai mapear os endereços lógicos gerados pelos processos nos correspondentes endereços físicos que serão enviados para a memória.

Existem duas formas bem simples de transformação do endereço lógico para o endereço físico:

A MMU verifica se o endereço lógico é maior que o registrador limite inferior e menor que o registrador limite superior, se sim encaminha o acesso com esse endereço válido, se não, gera uma interrupção de endereçamento inválido. Nesse caso o endereço lógico é igual ao endereço físico.

A MMU verifica se o endereço lógico é menor que o registrador limite superior, se sim adiciona o registrador base ao endereço lógico e encaminha o acesso com esse endereço resultante, se não gera interrupção de endereçamento inválido. Nesse caso o endereço lógico é diferente do endereço físico.

A MMU Consiste de um Chip ou uma Coleção de Chips

Para que um programa seja executado ele precisa ser transformado em processo(s), assim é necessário alocar o descritor de processos, alocar espaço na memória para o código (área conhecida como TEXT, onde se localiza o programa principal, as funções e as bibliotecas estáticas), os dados (Data, área onde as variáveis são alocadas - globais, locais estáticas, buffers internos)e a pilha (que possui o HEAP, área onde se localiza as variáveis dinâmicas, e o STACK, endereços de retorno de chamadas e parâmetros de funções).

A atribuição de endereço físico para as áreas de código e áreas de dados pode ser feita de três formas: em tempo de compilação, em tempo de carga e em tempo de execução. Em tempo de compilação o programador já faz a conversão de endereço lógico em endereço físico, pois ele tem conhecimento de qual área da memória ira utilizar.

Em tempo de carga o código precisa ser relocável de forma que todas as referências a memória sejam corrigidas para que o endereço de carga corresponda (carregador relocador), em outras palavras no momento da carga o programa executável é interpretado e os endereços corrigidos, dispensando a MMU. Em tempo de execução tem-se o código absoluto e é realizada uma relocação dinâmica usando a MMU, não sendo necessário corrigir os endereços no momento da carga do programa em memória.

Além de gerenciar quais espaços em memória estão em uso, é também necessário controlar os espaços de memória livres. Existem duas técnicas mais utilizadas para resolver este problema, estas serão vistas nas próximas seções.

Gerenciamento de Memória Livre com Mapas de Bits

Com mapas de bits, a memória é dividida em unidades de alocação. Cada bit do mapa representa uma unidade de alocação, sendo que se o bit for 0, a unidade está livre; caso contrário, a unidade está ocupada. Quanto menor for a unidade de alocação, maior será o mapa de bits e vice-versa. O maior problema com os mapas de bits é que procurar uma lacuna (seqüência de 0s) suficientemente grande para um determinado processo pode ser uma operação muito lenta.

O SO mantém um 1 bit para indicar se cada bloco da memória (ou unidade de alocação) está ocupado (1) ou livre (0). A Memória é dividida em unidades de alocação. Considerações sobre o tamanho do bloco de memória:

Quanto menor a unidade de alocação, maior será o mapa de bits.

- Pequeno: necessidade de muitos bits ⇒ uso ineficiente da memória. Exemplo: se tamanho do bloco = 1 byte, 1/9 da memória serão utilizados para o mapa de bits.

- Grande: memória sub-utilizada, pois se o tamanho do processo não for múltiplo do tamanho da unidade de alocação, uma quantidade de memória considerável será desperdiçada no último bloco.

Vantagens do uso de Mapa de Bits:

Simplicidade: o tamanho do mapa depende apenas do tamanho da memória e das unidades de alocação.

Desvantagens:

Quando um processo necessita de k unidades de alocação, o gerenciador de memória deve encontrar uma sequência de k bits 0, o que se constitui um processo lento.

Gerenciamento de Memória Livre com Listas Encadeadas

Neste caso, é mantida uma lista encadeada com os segmentos de memória livres e encadeados. Uma possível configuração seria manter, em cada entrada, o endereço em que inicia, o seu comprimento e, evidentemente, o ponteiro para a próxima entrada.

A principal vantagem de utilizar uma lista encadeada classificada por endereço é que sua atualização é simples e direta. Vários algoritmos podem ser utilizados para encontrar uma lacuna de memória para alocação de um processo:

Primeiro ajuste: varre a lista desde o início e aloca no primeiro espaço (lacuna) suficientemente grande;

Próximo ajuste: varre a lista da posição atual e aloca no primeiro espaço suficientemente grande;

Melhor ajuste: varre a lista completamente e aloca no espaço que gerar a menor lacuna de memória;

Pior ajuste: varre a lista completamente e aloca no espaço que gerar a maior lacuna de memória disponível, de modo que a lacuna resultante possa ser suficientemente grande para ser útil;

Ajuste rápido: mantém diversas listas separadas para os tamanhos de processos mais comuns.

A base do funcionamento da Memória Virtual é o Princípio da Localidade que estabelece que há uma tendência que os futuros endereços de memória de instruções e dados sejam próximos a endereços de memória recentemente acessados.

Esse comportamento se deve as características peculiares aos programas, que frequentemente fazem uso de endereços em sequência (vetores), localizados em blocos de código bem definidos e frequentemente invocados (funções), ou de códigos repetitivos (laços de repetição).

A ideia básica da memória virtual é que o tamanho combinado do programa, dos seus dados e da pilha pode exceder a quantidade de memória física disponível para ele, ou seja, neste caso, a simples troca, vista anteriormente, não resolveria o problema. O Sistema Operacional, então, mantém partes do programa atualmente em uso, em forma de páginas ou segmentos, na memória principal e o restante em disco. Essas páginas/segmentos são "trocados" entre memória principal e secundária conforme o SO as solicita, conforme a demanda do programa.

A memória virtual também pode trabalhar em um sistema de multiprogramação, com pedaços de vários programas na memória simultaneamente. Enquanto um programa está esperando parte dele próprio ser trazido para a memória (ele fica esperando a E/S e não pode executar) a CPU pode ser dada a outro processo, assim como em qualquer sistema de multiprogramação.

Para a implementação desta técnica, alguns recursos mínimos são necessários: localização da página através do hardware MMU, carga de página, substituição de página e área de troca, partição ou arquivo especial de troca (swap ou página) destinada a armazenar páginas.

Muitos sistemas de Memória Virtual utilizam uma técnica denominada paginação, vista mais adiante.

Troca (Swapping)

Em algumas situações não é possível manter todos os processos na memória e uma solução para essas situações é o mecanismo conhecido como swapping (troca). A gerência de memória reserva uma área do disco para esse mecanismo, que é utilizada para receber processos da memória. A execução desse processo é suspensa, com isso é dito que o mesmo sofreu uma swap-out. Mais tarde, esse mesmo processo será copiado do disco para a memória, mecanismo conhecido como swap-in. Esse mecanismo de trocas de processos no disco tem como objetivo permitir que o sistema operacional consiga executar mais processos do que caberia na memória.

Esse processo gera um grande custo de tempo de execução para os programas. Fazer a cópia do processo da memória para o disco e depois fazer o inverso é demorado.

Paginação

O espaço de endereço virtual é dividido em unidades chamadas páginas. As unidades correspondentes na memória física são chamadas molduras de página (ou quadros). As páginas e as molduras (quadros) têm sempre exatamente o mesmo tamanho.

No espaço físico (memória) tem-se várias molduras de página. Por exemplo, podem existir 05 páginas situadas no espaço de endereço virtual que são mapeadas na memória física. No entanto, o espaço de endereço virtual é maior que o físico. As outras páginas não são mapeadas. No hardware real, um bit presente/ausente em cada entrada monitora se a página é mapeada ou não.

Quando um programa tenta utilizar uma página não mapeada em uma moldura, a MMU detecta o acontecimento (que a página não está mapeada) e gera uma interrupção, passando a CPU para o Sistema Operacional. Tal interrupção é chamada falha de página. O S.O., então, seleciona uma moldura de página pouco utilizada e grava o seu conteúdo de volta ao disco, substituindo-a pela página requisitada.

Quanto à forma como a paginação pode ser implementada, podemos considerar a paginação simples e a paginação por demanda. Na primeira, todas as páginas lógicas do processo são mapeadas e carregadas para a memória física, isso supondo-se que o espaço de endereçamento de memória para um processo tenha o tamanho máximo igual à capacidade da memória física alocada para processos.

No caso da paginação por demanda, apenas as páginas lógicas efetivamente acessadas pelos processo são carregadas.

Nesse caso, uma página marcada como inválida na tabela de páginas de um processo pode tanto significar que a página está fora do espaço lógico de endereçamento do processo ou que simplesmente a página ainda não foi carregada.

Para descobrir qual das situações é a verdadeira basta conferir o descritor de processo, que contém os limites de endereçamento lógico do processo em questão.

Tabelas de Página

O propósito de tabelas de página é mapear páginas virtuais em molduras de página. No entanto, existem duas questões que devem ser consideradas:

A tabela de páginas pode ser extremamente grande, sendo que cada processo necessita de sua própria tabela de páginas;

O mapeamento deve ser rápido: o mapeamento do virtual para o físico deve ser feito em cada referência da memória, o que pode ocorrer diversas vezes em uma única instrução,não devendo tomar muito tempo para não se tornar um gargalo na execução da instrução.

Neste caso, há a necessidade de um mapeamento de páginas rápido e grande. Existem duas formas básicas de projetar tabelas de páginas:

ter uma única tabela de páginas, através de uma matriz de rápidos registradores de hardware, com uma entrada para cada página virtual, indexada pelo número da página. Esta é uma solução cara se a tabela de páginas é grande;

manter a tabela de páginas inteiramente na memória principal, sendo que o hardware precisa de apenas um registrador que aponta para o início da tabela de páginas.

Esta última solução possui algumas variações que têm desempenho melhor. Uma das propostas que busca evitar o problema de ter enormes tabelas de páginas na memória todo tempo é a de Tabelas de Páginas Multinível. Neste caso, existe uma tabela de primeiro nível e diversas tabelas de segundo nível. O importante aqui é que somente as tabelas mais utilizadas estão presentemente na memória. As demais se encontram em disco. Apesar de permitir um espaço de endereçamento muito grande, o espaço ocupado de memória principal é muito pequeno.

O sistema de tabela de páginas de dois níveis pode ser expandido para três, quatro ou mais níveis, sendo que níveis adicionais dão mais flexibilidade, mas a complexidade da implementação acima de três níveis dificilmente vale a pena. Em relação aos detalhes de uma única entrada de uma tabela de páginas, seu arranjo pode depender da máquina, mas os tipos de informação usualmente são os mesmos.

O campo mais importante é o número da moldura de página, pois o objetivo do mapeamento de páginas é localizar este valor. Ao lado dele, tem-se o bit de presente/ausente. Se este bit for 1, significa que a entrada é válida e pode ser utilizada. Se for 0, a página virtual a que esta entrada pertence não está atualmente na memória. Ao acessar uma entrada da tabela de páginas com este bit configurado como zero ocorrerá uma falha de página.

Os bits "proteção" informam que tipos de acesso são permitidos. Em sua forma simples, este campo contém apenas um bit, com 0 para leitura/gravação e 1 par leitura somente. Um arranjo mais sofisticado é manter 3 bits, cada um para habilitar leitura, gravação e execução da página. Os bits "modificada" e "referenciada" monitoram a utilização da página. Ambos os bits tem como objetivo auxiliar no momento da substituição de páginas. O último bit permite que o cache seja desativado para a página, recurso importante para páginas que são mapeadas em registradores de dispositivo em vez da memória.

Alguns bits auxiliares são, normalmente, adicionados na tabela de páginas para facilitar na substituição de páginas quando a memória física estiver cheia e for necessário retirar uma página lógica da memória física para alocar outra página lógica.

Tem-se o bit de modificação (dirty bit) assim que a página for carregada tem valor zero se a página for alterada, na memória física, altera-se o valor para 1, portanto se a página for a página vítima e o bit de modificação for zero não será necessário fazer a cópia da página lógica em memória para a página lógica em disco pois as duas são iguais.

Bit de referência: é zero assim que a página lógica é alocada na página física, se a página for acessada altera o valor para um (a MMU altera o valor).

Bit de trava: usado em páginas que não podem sair da memória física, o Sistema Operacional "tranca" uma página lógica na memória física ativando esse bit.

TLBs – Translation Lookside Buffers – Memória Associativa

A TLB, também conhecida como memória associativa, é um dispositivo de hardware cujo propósito é mapear endereços virtuais em endereços físicos sem passar pela tabela de páginas. Usualmente, ela faz parte da MMU.

Ela constitui-se de um banco de registradores que armazenam um pequeno número de entradas, muito rápidas, contendo as tabelas de páginas mais utilizadas. Quando um endereço virtual é enviado a MMU, ela primeiramente verifica se o seu número de página virtual está presente na TLB. Se o resultado for positivo (hit), a moldura de página é tomada diretamente da TLB sem a necessidade de passar pela tabela de páginas na memória (mais lento). Caso contrário (miss), a pesquisa é feita normalmente na tabela de páginas presente na memória. Então, uma das entradas é removida da TLB e a entrada da tabela de páginas pesquisada é colocada em seu lugar.

A TLB melhora bastante o desempenho no acesso à tabela de páginas, visto que registradores são muito mais rápidos que a memória RAM. Suas desvantagens estão em seu custo (registradores são caros), seu tamanho limitado e o fato de existir uma única TLB na MMU, sendo esta compartilhada por todos os processos.

Para calcularmos o desempenho da TLB, podemos tomar h como taxa de acerto (taxa em que a página necessária estará na TLB). Então o erro seria 1-h. Sendo assim, a tempo de acesso hit (tempo necessário para pegar a página da TLB) é dada pelo tempo de acesso à TLB mais o tempo de acesso à memória uma única vez.

Enquanto o tempo de acesso miss (quando falta a página na TLB) é dada pelo tempo de acesso à TLB mais o tempo de dois acessos à memória. O tempo de acesso médio é dado pelo tempo de acesso hit multiplicado por h somado do tempo de acesso miss multiplicado por (1-h). Resumindo em fórmulas, temos:

TacessoHit = TacessoTLB + TacessoMemoria

TacessoMiss = TacessoTLB + TacessoMemoria + TacessoMemoria

TacessoMedio = h x TacessoHit + (1-h) x TacessoMiss

A taxa de acerto (h) depende do tamanho da TLB e do algoritmo que a controla, mantendo as páginas mais utilizadas.

Tabelas de Páginas Invertidas

Outra abordagem para trabalhar com páginas é manter uma tabela onde cada entrada representa uma moldura de página ao invés de um endereço virtual. Desta forma, a entrada deve monitorar qual página virtual está associada àquela moldura de página.

Embora as tabelas de páginas invertidas economizem quantidade significativas de espaço (pelo menos nas situações em que o espaço de endereço virtual é muito maior que a memória física), elas tem a desvantagem de que o mapeamento (tradução) do endereço virtual para o físico é mais complexo e potencialmente mais lento. Uma forma de facilitar a tradução do virtual para o físico é a utilização da TLB pesquisada por software. A pesquisa pode ser feita a partir de um encadeamento de páginas virtuais que possuam um mesmo endereço hash.

Tamanho de Página

Um ponto do qual o projetista deve se preocupar é com o tamanho da página. Conforme visto anteriormente, se esse tamanho de página for grande, pode ocorrer de o processo utilizador não ocupar todo o espaço a ele destinado. Se a página tiver um tamanho demasiadamente pequeno, a tabela de páginas será muito grande.

É possível uma modelagem matemática. Considere que cada processo tenha tamanho de s bytes, e cada página tenha tamanho de p bytes. A tabela de páginas terá um espaço de e bytes por entrada. Assim, o número de páginas que um processo precisará é de s/p. O espaço que esse processo ocupa

na tabela de páginas é s*e/p. O tamanho perdido na última página devido a fragmentação interna será de p/2.

Assim, haverá um custo de s*e/p + p/2 da tabela de páginas. Para que o tamanho da página seja ideal, o custo será zero. Dessa forma, derivando a expressão anterior e igualando a zero obtemos que o tamanho ideal de página é de s*e*sqrt(2).

Thrashing

Estado no qual o sistema operacional ao invés de executar instruções efetivas "gasta" tempo efetuando a troca de páginas entre memória física e memória lógica, em outras palavras desperdiça um tempo significante trazendo ou removendo páginas da memória. Para o usuário a sensação é que o sistema está travado ou congelado e para o hardware há um significante acesso ao disco ao invés de processamento