Gerenciamento de Memória

Páginas compartilhadas: É nitidamente mais eficiente compartilhar páginas para evitar a situação de existirem duas cópias ou mais da mesma página presentes na memória. Um problema é que nem todas as páginas são compartilháveis. Em particular, as páginas somente de leitura- como as que contêm código de programa – são compartilháveis, mas páginas com dados alteráveis durante a execução não o são.

Se o sistema suportar os espaços I e D, o compartilhamento de programas será obtido de modo relativamente direto. Em geral, em uma implementação que suporte compartilhamento desse tipo, tabela de páginas são estruturas de dados independentes da tabela de processos. Cada processo tem assim dois ponteiros em sua tabela de processos: um aponta para a tabela de páginas do espaço I e outro aponta para a tabela de páginas do espaço D.

Quando dois ou mais processos compartilham o mesmo código, um problema ocorre com as páginas compartilhadas. Suponha que o processo A e B estejam ambos executando o editor e compartilhando sua páginas. Se o escalonador decide remover A da memória, descartando todas as suas áginas e carregando as molduras de página vazias com outro programa, ele leva o processo B a causar muitas faltas de página até que suas páginas estejam novamente presentes na memória.

Em geral, é muito trabalhos pesquisar todas as tabelas de páginas para descobrir se uma determinada página é compartilhada, de modo que são necessárias estruturas de dados especias para manter o controle das páginas compartilhadas.

Política de limpeza: A paginação funciona melhor quando existe uma grande quantidade de molduras de página disponíveis prontas a serem requisitadas quando ocorrerem faltas de página. Para garantir um estoque abundante de molduras de página disponível, muitos sistemas de paginação executam um processo específico, denominado ***daemon de paginação*** (*paging daemon)*, que dorme quase todo o tempo, mas que é acordado periodicamente par inspecionar o estado da memória. Se existirem apenas algumas molduras de página disponível, o *daemon* de paginação começa a selecionar as páginas a serem removidas da memória usando um algoritmo de substituição de páginas.

Em qualquer eventualidade, o conteúdo anterior da página é lembrado. No caso de uma página descarada da memória ser novamente referenciada antes de sua moldura ter sido sobreposta por uma nova página,l a moldura pode ser reclamada, ou seja, trazida de volta, retirando-a do pool (conjunto) de molduras de página disponível.

Uma maneira de implementar essa política de limpeza é usar um relógio com dois ponteiros. O ponteiro da frente é controlado pelo *daemo*s de paginação. Quando esse ponteiro aponta para uma página suja, essa página é escrita em disco e o ponteiro avança. Quando ele aponta para uma página limpa, ele apenas avança. O ponteiro de trás é usado para substituição de página, assim como no algoritmo-padrão do relógio, só que, agora, a probabilidade de o ponteiro de trás apontar para uma página limpa aumenta devido ao trabalho do *daemon* de paginação.

Interface da memória virtual: Uma razão para dar o controle do mapa de memória a programadores é permitir que dois ou mais processos compartilhem a mesma memória. Com dois ou mais processos compartilhando as mesmas páginas, surge a possibilidade de um compartilhamento em alta largura de banda.

Se processos puderem controlar seus mapeamentos, uma mensagem poderá ser trocada retirando-se página relacionadas do mapeamento do processo emissor e inserindo-a no mapeamento do processo receptor.

Envolvimento do Sistema Operacional (SO) com a paginação: Existem quatro circunstâncias em que o SO tem de se envolver com a paginação: na criação do processo, na execução do processo, na ocorrência de falta de página e na finalização do processo.

Quando um novo processo é criado em um sistema com paginação, o SO deve determinar qual será o tamanho (inicial) do programa e de seus dados e também criar uma tabela de páginas para ele. Um espaço precisa ser alocado na memória para a tabela de páginas, e esta deve ser inicializada. A tabela de página não precisa estar presente na memória quando o processo é levado para disco, mas ela tem de estar na memória quando o processo estiver em execução. Além disso, um espaço deve ser alocado na área de trocas em disco (*swap area)*, de modo que, quando uma página é devolvida ao disco, ela tem para onde ir. Alguns sistemas paginam o programa diretamente do arquivo executável, economizando assim espaço em disco e tempo de inicialização. Informações acerca da tabela de páginas e da área de trocas de processos em disco devem ser registradas na tabela de processos.

Quando um processo é escalonado para execução, a MMU tem de ser reinicializada para o novo processo, e a TLB, esvaziada para livrar-se de resíduos do processo executado anteriormente.

Quando uma falta de página ocorre, o SO tem de ler o(s) registrador(es) em hardware para determinar o endereço virtual causador da falta de página. A partir dessa informação, ele precisa calcular qual página virtual é requisitada e então localizá-la em disco. Em seguida, ele procura uma moldura de página disponível para colocar a nova página, descartando, se necessário, alguma página antiga. Por fim, o SO tem de salvar o contatador de programa para que ele aponte para a instrução que causou a falta de página, de modo que a mesma possa ser executada novamente.

Quando um processo termina, o SO deve liberar sua tabela de página.

Recuperação de instrução: Quando um programa referencia uma página não presente na memória, a instrução causadora da falta de página é bloqueada no meio de sua execução e ocorre uma interrupção, desviando-se assim para o SO. Após o SO buscar em disco a página necessária, ele deverá reiniciar a instrução causadora da interrupção.

Memória secundária: O algoritmo mais simples para a alocação de espaço em disco consiste na manutenção de uma área de troca (*swap aera*) em disco. Quando o SO é iniciado, essa área encontra-se vazia e é representada na memória como uma única entrada contendo localização e seus tamanho. Quando o primeiro processo é iniciado, reserva-se uma parte dessa área de troca, com o tamanho desse processo, e a área de troca restante fica reduzida dessa quantidade. À medida que os processos vão terminando, seus espaços em disco são gradativamente liberados. A área de troca em disco é gerenciada como uma lista de pedaços disponíveis.

O endereço da área de troca de cada processo é mantido na tabela de processos. O cálculo do endereço onde escrevemos uma página é simples: basta adicionar o deslocamento da página dentro do espaço de endereçamento virtual ao endereço inicial da área de troca. Contudo, antes que um processo possa começar sua execução, a área de troca deve ser inicializada. Para isso, copia-se o processo todo em sua área de troca em disco, trazendo-o de volta à memória quando necessário.

No entanto, esse modelo simples apresenta um problema: os processos podem aumentar de tamanho no decorrer de suas execuções. Consequentemente, talvez seja melhor reservar áreas de troca separadas para código, dados e pilha e permitir que cada uma delas consista em mais pedaço em disco.

O outro extremo consiste em não alocar nada antecipadamente e apenas alocar o espaço em disco para cada página quando ela for enviada para lá e liberar o mesmo espaço quando a página for carregada na memória. A desvantagem é a necessidade de se manter na memória o endereço de disco de cada página armazenada em disco.

Separação da política e do mecanismo: Todos os detalhes de como a MMU trabalha são escondidos do tratador de MMU, o qual possui código dependente de máquina. O tratador de faltas de página é um código independente de maquina e contém a maioria dos mecanismos para paginação. A política é em grande parte determinada pelo paginado externo, o qual executa como um processo do usuário.

Uma vez que o processo inicie a execução, ele pode causar uma falta de página. O tratador de faltas de página calcula qual página virtual é necessária e a envia para o paginador externo, informando qual é o problema.

Essa implementação deixa livre o local onde o algoritmo de substituição de página é colocado. O principal problema dele é que o paginador externo não tem acesso aos bits R e M de todas as páginas. Esses bits ditam regras em muitos dos algoritmos de paginação. Assim, ou algum mecanismo é necessário para passar essa informação para o paginador externo ou o algoritmo de substituição deve estar no núcleo do SO.

A principal desvantagem é a sobrecarga adicional causada pelos diversos chaveamentos entre núcleo e o usuário e a sobrecarga nas trocas de mensagens entre as partes do sistema.

Segmentação:

A memória virtual até agora é unidimensional porque o endereçamento virtual vai de 0 a algum endereço máximo, um após o outro. Para muitos problemas, ter dois ou mais espaços de endereços separados pode ser muito melhor do que ter somente um.

Uma solução extremamente abrangente e direta para o problema da contração das tabela é prover a máquina com muitos espaços de endereçamento completamente independentes, chamados ***segmentos***. Cada segmento é constituído de uma sequência linear de endereços, de 0 a algum máximo. Além disso, os tamanhos dos segmentos podem variar durante a execução. O tamanho de cada segmento de pilha é passível de ser expandido sempre que algo é colocado sobre a pilha e diminuído toda vez que algo é retirado dela. Pelo fato de cada segmento constituir um espaço de endereçamento separado, segmentos diferentes podem crescer ou reduzir independentemente, sem afetarem uns aos outros. Se uma pilha, em certo segmento, precisa de mais espaço de endereçamento para crescer, ela pode tê-lo, pois não exite nada em seu espaço de endereçamento capaz de colidir com ela. É óbvio que um segmento pode ser totalmente preenchido, mas geralmente os segmentos são grandes e esse tipo de ocorrência é raro. Para especificar um endereço nessa memória segmentada e bidimensional, o programa deve fornecer um endereço composto de duas partes: um número do segmento e um endereço dentro do segmento.

<<<<<<<<IMAGEM>>>>>>

É preciso enfatizar que um segmento é uma entidade lógica, a qual o programador conhece e usa como entidade lógica. Um segmento pode conter um procedimento, um vetor, uma pilha ou um conjunto de variáveis escalares, mas em geral ele não apresenta uma mistura de diferentes tipos.

Se cada procedimento ocupa um segmento separado, com o endereço 0 como endereço inicial, a ligação dos procedimentos compilados separadamente é extremamente simplificado. Após todos os procedimentos constituintes de um programa terem sido compilados e ligado, uma chamada para um procedimento no segmento n usará o endereço de duas partes (n, 0) para endereçar a palavra 0 (ponto de entrada).

Com uma memória unidimensional, os procedimentos são fortemente empacotados próximos uns dos outros, sem qualquer espaço de endereçamento entre eles. Consequentemente, a variação do tamanho de um procedimento pode afetar os endereços iniciais dos outros procedimentos não relacionados. Isso, por sua vez, requer a modificação de todos os procedimentos que fazem chamadas aos procedimentos que foram movidos, a fim de atualizar seus novos endereços iniciais.

A segmentação também facilita o compartilhamento de procedimentos ou dados entre vários processos. Um exemplo comum é a biblioteca compartilhada. Muitas vezes, as estações de trabalho modernas, que executam sistemas avançados de janela, tem bibliotecas gráficas extremamente grandes compiladas em quase todos os programas. Em um sistema segmentado, uma biblioteca gráfica pode ser colocada em um segmento e compartilhada entre vário processos, eliminando a necessidade de suas replicações em cada espaço de endereçamento de cada processo.

É preciso compreender por que a proteção faz sentido em uma memória segmentada, mas não em uma memória unidimensional paginada: em uma memória segmentada, o usuário está ciente do que existe em cada segmento. Normalmente, um segmentos não conteria um procedimento e uma pilha.

De certo modo, os conteúdos de uma página são acidentais. O programador desconhece o fato de que a paginação está ocorrendo. Embora fosse possível colocar alguns bits em cada entrada da tabela de páginas a fim de especifica o acesso permitido.