Capítulo 6

Gerenciamento de Memória

A memória é um recurso importante que deve ser gerenciado com cuidado. Enquanto o computador doméstico médio hoje em dia tem 50 vezes mais memória do que o IBM 7034 (segundo Tanembaum), o maior computador no mundo no início da década de 60, os programas estão crescendo tão rapidamente quanto as memórias. Poderíamos dizer que os programas tendem a expandir até ocupar toda a memória disponível para armazená-los.

Idealmente, o que todo programador gostaria de ter à disposição é uma memória rápida e infinitamente grande que também fosse não-volátil, isto é, que não perdesse seu conteúdo quando a energia elétrica cai. Infelizmente, a tecnologia ainda não oferece esse tipo de memória. A maioria dos computadores, portanto, tem uma hierarquia de memória, com uma pequena quantidade de memória cache volátil, muito rápida e cara, alguns megabytes de memória principal volátil (RAM), de velocidade e preço médios, além de centenas ou milhares de megabytes de armazenamento em disco, não-volátil, lento e barato. O trabalho do sistema operacional é coordenar como essas memórias são utilizadas.

A parte do sistema operacional que gerencia a hierarquia de memória é chamada gerenciador de memória. Seu trabalho é controlar que partes da memória estão em uso e que partes não estão, alocar memória para processos quando eles necessitam e desalocar quando eles terminarem, e gerenciar a troca entre a memória principal e o disco quando a memória principal é muito pequena para armazenar todos os processos.

Veremos agora como funciona o esquema de um dos mais simples gerenciamentos de memória.

6.1 Gerenciamento Básico de Memória

Os sistemas de gerenciamento de memória podem ser divididos em duas classes: aqueles que movem processos de um lado para outro entre a memória principal e o disco durante execução (fazendo troca e paginação) e aqueles que não o fazem. Os últimos são serão os estudados agora.

6.2 Monoprogramação sem Troca ou Paginação

O esquema mais simples possível de gerenciamento de memória é executar somente um programa por vez, compartilhando a memória entre esses programas e o sistema operacional. Três variações desse tema são possíveis. O sistema operacional pode estar na parte inferior da memória RAM (Random Access Memory, Memória de Acesso Aleatório), ou pode estar em ROM (Read Only Memory, Memória Apenas de Leitura) na parte superior de memória, ou os drivers de dispositivo podem estar na parte superior da memória em uma ROM e o restante do sistema de RAM na parte inferior. O último modelo é utilizado por pequenos sistemas MS-DOS, por exemplo. No IBM PC, a parte do sistema na ROM é chamada BIOS (Basic Input Output System, Sistema Básico de Entrada e Saída).

Quando o sistema está organizado dessa maneira, somente um processo por vez pode estar executando. Logo que o usuário digita um comando, o sistema operacional copia o programa solicitado do disco para a memória e executa-o. Quando o processo termina, o sistema operacional exibe um aviso de comando e espera um novo comando. Quando recebe o comando, ele carrega um novo programa na memória, sobrepondo o primeiro.

6.3 Multiprogramação com Partições Fixas

Embora a monoprogramação seja, às vezes, utilizada em computadores pequenos com sistema operacional simples, com freqüência é desejável permitir que múltiplos processos executem simultaneamente. Em sistemas de compartilhamento de tempo, Ter múltiplos processos na memória simultaneamente significa que quando um processo é bloqueado esperando a E/S acabar, outro pode utilizar a UCP. Assim, a multiprogramação aumenta a utilização da UCP. Entretanto, mesmo em computadores pessoais é freqüentemente útil ser capaz de executar dois ou mais programas ao mesmo tempo.

A maneira mais fácil de alcançar a multiprogramação é simplesmente dividir a memória em n partições (possivelmente desiguais). Esse particionamento pode ser feito, por exemplo, manualmente quando o sistema é iniciado.

Quando um job chega, pode ser colocado na fila de entrada da menor partição capaz de armazená-lo. Uma vez que as partições são fixas nesse esquema, qualquer espaço em uma partição não utilizada por um job é perdido.

A desvantagem de classificar os jobs de entrada em filas separadas torna-se aparente quando a fila para uma partição grande está vazia, mas a fila para uma partição pequena está cheia. Uma organização alternativa é manter uma fila única. Sempre que uma partição torna-se livre, o job mais próximo do início da fila que se ajusta na partição vazia poderia ser carregado nela e executado. Uma vez que é indesejável desperdiçar uma partição grande com um job pequeno, uma estratégia diferente é pesquisar a fila de entrada inteira sempre que uma partição torna-se livre e selecionar o maior job que se ajusta. Note que o último algoritmo discrimina jobs pequenos porque estes não merecem ter uma partição inteira, ao

passo que normalmente é desejável dar o melhor serviço aos jobs pequenos (que supostamente são interativos), não o pior.

Uma saída é dispor de pelo menos uma partição pequena, a qual permitirá que jobs pequenos sejam executados sem alocar uma partição grande para eles.

Outra abordagem é estabelecer uma regra determinando que um job elegível para executar não pode ser ignorado mais que k vezes. Cada vez que é ignorado, ele ganha um ponto. Quando adquiriu k pontos, ele não pode ser ignorado novamente.

Esse sistema, com partições fixas definidas pela manhã pelo operador e não alteradas depois, foi utilizado por mainframes IBM OS/360 e grande porte durante muitos anos. Ele foi chamado MFT (Multiprogramação com um número Fixo de Tarefas ou OS/MFT). Ele é simples de entender e igualmente simples de implementar: os jobs que chegam são enfileirados até que uma partição adequada esteja disponível, momento em que o job é carregado nessa partição e executa até a sua conclusão. Hoje em dia, poucos sistemas operacionais suportam esse modelo, se é que algum suporta.

6.4 Realocação e Proteção

A multiprogramação introduz dois problemas essenciais que devem ser resolvidos – realocação e proteção. Imaginando-se jobs diferentes que serão executados em endereços também diferentes. Quando um programa é vinculado (i.e., o programa principal, procedimentos de usuário gravado, e procedimentos de biblioteca são combinados em um único espaço de endereço), o linkeditor deve saber em que endereço o programa deve começar na memória.

Por exemplo, suponha que a primeira instrução seja uma chamada para um procedimento no endereço absoluto 100 dentro do arquivo binário, produzido pelo linkeditor. Se esse programa for carregado na partição 1, essa instrução saltará para o endereço absoluto 100, que está dentro do sistema operacional. O que é necessário é uma chamada para 100K + 100. Se o programa for carregado na partição 2, ele executará como uma chamada para 200K + 100 e assim por diante. Esse problema é conhecido como problema da realocação.

Uma possível solução é realmente modificar as instruções enquanto o programa é carregado na memória. O programa carregado na partição 1 tem 100K adicionado a cada endereço, o programa carregado na partição 2 tem 200K adicionado aos endereços e assim por diante. Para realizar uma realocação como essa durante o carregado, o linkeditor deve incluir no programa binário uma lista ou um mapa de bits, informando quais palavras do programa são endereços a serem realocados e quais são códigos de instruções, constantes ou outros itens que não devem ser realocados. O OS/MFT funcionava dessa maneira. Alguns microcomputadores também trabalham assim.

A realocação durante o carregamento não resolve o problema da proteção. Um programa malicioso sempre pode construir uma nova instrução e saltar para ela. Como os programas nesse sistema utilizam endereços absolutos de memória em vez de endereços

relativos a um registrador, não há como impedir que um programa crie uma instrução que lê ou grava qualquer palavra na memória. Em sistema multiusuários, é indesejável deixar um processo ler ou gravar memória pertencente a outros usuários.

A solução que a IBM escolheu para proteger os 360 foi dividir a memória em blocos de 2KB e atribuir um código de proteção de 4 bits a cada bloco. O PSW continha uma chave de 4 bits. O hardware do 360 interrompia qualquer tentativa, por parte de um processo em execução, de acessar a memória cujo código de proteção diferia da chave PSW. Uma vez que somente o sistema operacional podia alterar os códigos e a chave de proteção, os processos de usuário eram impedidos de interferir um no outro e no próprio sistema operacional.

Uma solução alternativa para ambos os problemas, realocação e proteção, é equipar a máquina com dois registradores especiais de hardware, chamados de registrador de base e registrador de limite. Quando um processo é agendado, o registrador base é carregado com o endereço do início de sua partição, e o registrador de limite é carregado com o comprimento de sua partição. Todo endereço de memória gerado tem o registrador de base automaticamente adicionado a ele próprio antes de ser enviado para memória. Assim, se o registrador de base for 100K, uma instrução CALL 100 efetivamente transforma-se em uma instrução CALL 100K + 100, sem que a instrução em si seja modificada. Os endereços também são verificados em relação ao registrador de limite para certificar-se de que eles não tentarão endereçar memória fora da partição atual. O hardware protege os registradores de base e de limite para impedir que os programas de usuários os modifiquem.
O CDC 6600 – o primeiro supercomputador do mundo – utilizava esse esquema. A UCP Intel 8088 utilizada no IBM PC original empregava uma versão mais fraca desse esquema – os registradores de base, mas não os registradores de limite. Desde os 286, um esquema melhor foi adotado.