Sistemas Operacionais

Gerência de Memória

- Nos sistemas monoprogramáveis a gerência de memória não é muito complexa.
- Nos sistemas multiprogramáveis essa gerência se torna crítica, devido à necessidade de se maximizar o número de usuários e aplicações utilizando eficientemente o espaço da memória principal.

- Os programas são, normalmente, armazenados em memórias secundárias, como disco ou fita, por ser um meio não-volátil, abundante e de baixo custo. Como o processador só executa instruções localizadas na memória principal, o SO deve sempre transferir programas da memória secundária para a memória principal antes de serem executados.

- Como o tempo de acesso à memória secundária é muito superior, o SO deve buscar reduzir o número de operações de E/S à memória secundária, caso contrário, sérios problemas no desempenho do sistema podem ser ocasionados.

- A gerência de memória deve tentar manter na memória principal o maior número de processos residentes, permitindo maximizar o compartilhamento do processador e demais recursos computacionais. Mesmo na ausência de espaço livre, o sistema deve permitir que novos processos sejam aceitos e executados.

- Isso é possível através da transferência temporária de processos residentes na memória principal para a memória secundária, liberando espaço para novos processos. Este mecanismo é conhecido como “swapping”.

- Outra preocupação é permitir a execução de programas que sejam maiores que a memória física disponível, implementado através de técnicas de “overlay” e “memória virtual”.

- Em um ambiente de multiprogramação, o SO deve proteger as áreas de memória ocupadas por cada processo, além da área onde reside o próprio sistema. Caso um programa tente realizar algum acesso indevido à memória, o sistema de alguma forma deve impedi-lo.

- Apesar de a gerência de memória garantir a proteção de áreas da memória, mecanismos de compartilhamento devem ser oferecidos para que diferentes processos possam trocar dados de forma protegida.

Alocação Contígua Simples

- Foi implementada nos primeiros SO, porém ainda está presente em alguns sistemas monoprogramáveis. Nesse tipo de organização, a memória principal é subdividida em duas áreas: uma para o SO e outra para o programa do usuário.

Alocação Contígua Simples

- Dessa forma, o programador deve desenvolver suas aplicações, preocupado, apenas, em não ultrapassar o espaço de memória disponível.
- Neste esquema, o usuário tem controle sobre toda a memória principal, podendo ter acesso a qualquer posição da memória, inclusive a área do SO.

Alocação Contígua Simples

- Para proteger o sistema desse tipo de acesso, que pode ser intencional ou não, alguns sistemas implementam proteção através de um registrador que delimita as áreas do SO e do usuário. Dessa forma, sempre que um programa faz referência a um endereço na memória, o sistema verifica se o endereço está dentro dos limites permitidos.

Alocação Contígua Simples

- Caso não esteja, o programa é cancelado e uma mensagem de erro é gerada, indicando que houve uma violação no aceso à memória principal.
- Apesar da fácil implementação e do código reduzido, a alocação contígua simples não permite a utilização eficiente dos recursos computacionais, pois apenas um usuário pode dispor desses recursos.

Alocação Contígua Simples

- Em relação à memória principal, caso o programa do usuário não a preencha totalmente, existirá um espaço de memória livre sem utilização.

Técnica de Overlay

- Na alocação contígua simples, todos os programas estão limitados ao tamanho da área de memória principal disponível para o usuário.
- Uma solução encontrada para o problema é dividir o programa em módulos, de forma que seja possível a execução independente de cada módulo, utilizando uma mesma área de memória. Essa técnica é chamada de “overlay”

Técnica de Overlay

- A definição das áreas de overlay é função do programador, através de comandos específicos da linguagem de programação utilizada.
- A técnica de overlay tem a vantagem de permitir ao programador expandir os limites da memória principal.

Técnica de Overlay

- A utilização dessa técnica exige muito cuidado, pois pode trazer implicações tanto na sua manutenção quanto no desempenho das aplicações, devido a possibilidade de transferência excessiva dos módulos entre a memória principal e a secundária.

Alocação Particionada Estática

- Nos primeiros sistemas multiprogramáveis, a memória era dividida em pedaços de tamanho fixo, chamados “partições”. O tamanho das partições, estabelecidos na fase de inicialização do sistema, era definido em função do tamanho dos programas que executariam no ambiente.

Alocação Particionada Estática

- Sempre que fosse necessária a alteração do tamanho de uma partição, o sistema deveria ser desativado e reinicializado com uma nova configuração. Esse tipo de gerência de memória é conhecido como “alocação particionada estática ou fixa”.

Alocação Particionada Estática

- Inicialmente, os programas só podiam ser carregados e executados em apenas uma partição específica, mesmo se outras estivessem disponíveis. Essa limitação se devia aos compiladores e montadores, que geravam apenas código absoluto.

Alocação Particionada Estática

- No código absoluto, todas as referências a endereços no programa são posições físicas na memória principal, ou seja, o programa só poderia ser carregado a partir do endereço de memória especificado no seu próprio código.

Alocação Particionada Estática

- Com a evolução dos compiladores, montadores, linkers e loaders, o código gerado deixou de ser absoluto e passa a ser relocável. No ”código relocável”, todas as referências a endereços no programa são relativas ao início do código e não a endereços físicos de memória.

Alocação Particionada Estática

- Desta forma, os programas puderam ser executados a partir de qualquer partição.

Alocação Particionada Dinâmica

- Foi eliminado o conceito de partições de tamanho fixo. Nesse esquema, cada programa utilizaria o espaço necessário, tornando essa área sua partição. Como os programas utilizam apenas o espaço de que necessitam, no esquema de alocação particionada dinâmica o problema de fragmentação interna não ocorre.

Alocação Particionada Dinâmica

- A princípio, o problema da fragmentação interna está resolvido, porém, nesse caso, existe um problema.
- Um tipo diferente de fragmentação começará a ocorrer, quando os programas forem terminando e deixando espaços cada vez menores na memória, não permitindo o ingresso de novos programas. Esse tipo de problema é chamado “fragmentação externa”.

Alocação Particionada Dinâmica

- Existem duas soluções para o problema da fragmentação externa da memória principal.
- Na primeira solução, conforme os programas terminam, apenas os espaços livres adjacentes são reunidos, produzindo áreas livres de tamanho maior.

Alocação Particionada Dinâmica

- A segunda solução envolve a relocação de todas as partições ocupadas, eliminando todos os espaços entre elas e criando uma única área livre contígua. Para que esse processo seja possível, é necessário que o sistema tenha a capacidade de mover os diversos programas na memória principal, ou seja, realizar “relocação dinâmica”.

Alocação Particionada Dinâmica

- Esse mecanismo de compactação, também conhecido como “alocação particionada dinâmica com relocação”, reduz em muito o problema da fragmentação, porém a complexidade do seu algoritmo e o consumo de recursos do sistema, como processador e área em disco, podem torná-lo inviável.

Estratégias e Alocação de Partição

- Os SO implementam, basicamente, três estratégias para determinar em qual área livre um programa será carregado para execução. Essas estratégias tentam evitar ou diminuir o problema da fragmentação externa.

Estratégias e Alocação de Partição

- A melhor estratégia a ser adotada por um sistema depende de uma série de fatores, sendo o mais importante o tamanho dos programas processados no ambiente.

Estratégias e Alocação de Partição Best-fit

- Na estratégia best-fit, a melhor partição é escolhida, ou seja, aquela em que o programa deixa o menor espaço sem utilização. Nesse algoritmo, a lista de áreas livres está ordenada por tamanho, diminuindo o tempo de busca por uma área desocupada. Uma grande desvantagem desse método é consequência do próprio algoritmo. Como é alocada a partição que deixa a menor área livre, a tendência é que cada vez mais a memória fique com pequenas áreas não-contíguas, aumentando o problema da fragmentação.

Estratégias e Alocação de Partição Worst-fit

- Nesta estratégia, a pior partição é escolhida, ou seja, aquela em que o programa deixa o maior espaço sem utilização. Apesar de utilizar as maiores partições, a técnica de worst-fit deixa espaços livres maiores que permitem a um maior número de programas utilizar a memória, diminuindo o problema da fragmentação.

Estratégias e Alocação de Partição First-fit

- Na estratégia first-fit, a primeira partição livre de tamanho suficiente para carregar o programa é escolhida. Nesse algoritmo, a lista de áreas livres está ordenada crescentemente por endereços. Como o método tenta primeiro utilizar as áreas livre de endereços mais baixos, existe grande chance de se obter uma grande partição livre nos endereços de memória mais altos.

Estratégias e Alocação de Partição First-fit

- Das três estratégias apresentadas, a first-fit é a mais rápida, consumindo menos recursos do sistema.

Swapping

- Mesmo com o aumento da eficiência da multiprogramação e, particularmente, da gerência de memória, muitas vezes um programa não podia ser executado por falta de uma partição livre disponível. A técnica de swapping foi introduzida para contornar o problema de insuficiência de memória principal.

Swapping

- O conceito de swapping permite maior compartilhamento da memória principal e, consequentemente, maior utilização dos recursos do sistema computacional. Seu maior problema é o elevado custo das operações de entrada/saída. Em situações críticas, quando há pouca memória disponível, o sistema pode ficar quase que dedicado à execução de swapping, deixando de realizar outras tarefas e impedindo a execução dos processos residentes.