Sistemas Operacionais

Gerenciamento de Memória

Prof. Antonio Pires de Almeida Junior

Aula 9

9.1 – Introdução

•Historicamente, a memória principal sempre foi vista como um recurso escasso e caro, obrigando os desenvolvedores a criarem SO's que ocupassem pouco espaço na memória e, ao mesmo tempo, otimizassem seu uso para as demais aplicações.

•Mesmo atualmente, com a redução do custo e consequente aumento da capacidade das memórias, o seu gerenciamento é um dos fatores mais importantes no projeto de um sistema operacional.

•Veremos a partir de agora os principais conceitos relativos ao gerenciamento de memória dentro dos sistemas operacionais.

9.2 – Funções Básicas

•Em geral, programas são armazenados em memórias secundárias (discos, pen drives) por serem dispositivos não-voláteis e de baixo custo.

•Como o processador somente executa instruções presentes na memória principal, o sistema operacional deve sempre transferir programas da memória secundária para memória principal.

•Contudo, o sistema operacional deve otimizar o número de operações de E/S na memória secundária, caso contrário, sérios problemas de desempenho podem ser ocasionados.

•A gerencia de memória deve manter na memória principal o maior número de processos residentes, permitindo maximizar o compartilhamento do processador e demais recursos, além de reduzir o número de operações de E/S no disco.

9.2 – Funções Básicas

•Mesmo na ausência de espaço físico na memoria, o sistema deve permitir que novos processos sejam aceitos e executados. Isso é possível por meio de transferência temporária de processos residentes na memória principal para memória secundária. Este mecanismo é chamado **swapping**.

•Outra preocupação na gerencia de memória é permitir a execução de programas que sejam maiores que a memória física disponível, implementando técnicas como **overlay** e **memória virtual**.

•Em um ambiente de multiprogramação, o SO deve proteger áreas de memória ocupadas por cada processo. Caso algum processo tente acessar uma área que não lhe pertence, o sistema deve impedi-lo.

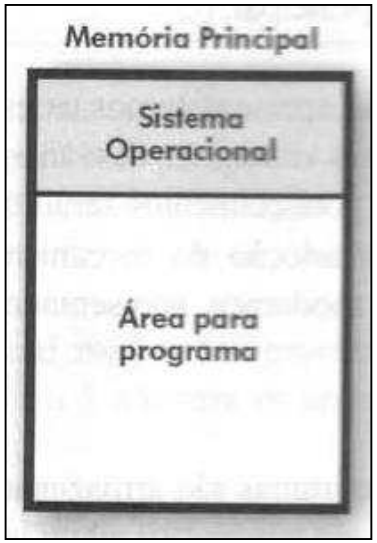
•Apesar da proteção oferecida pela gerencia de memória, mecanismos de compartilhamento deve ser oferecidos, visto que alguns processos necessitam trocar informações.

9.3 – Alocação Contígua Simples

•A alocação contígua foi implementada nos primeiros sistemas operacionais, porém ainda está presente em alguns sistemas monoprogramáveis.

•Nesse tipo de organização, a memória principal é subdividida em duas áreas: uma para o sistema operacional e outra para o programa do usuário.

•Desta forma, o programador deve apenas preocupar-se em não ultrapassar o espaço de memória disponível, ou seja, a diferença entre o tamanho total da memória principal e a área ocupada pelo sistema operacional.

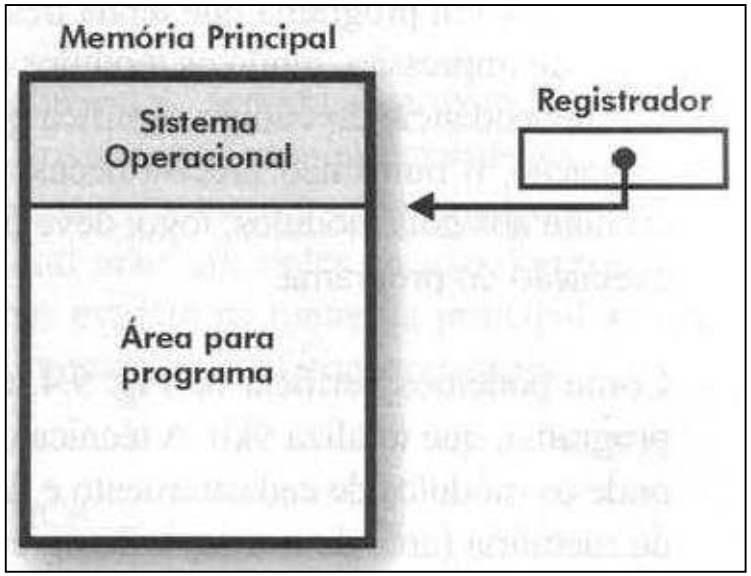


9.3 – Alocação Contígua Simples

•Neste esquema, o usuário tem controle sobre toda a memória principal, podendo ter acesso a qualquer posição de memória, inclusive na área do sistema operacional.

•Para proteger o sistema, alguns sistemas implementam proteção por meio de um registrador que delimita as áreas do SO e do usuário.

•Dessa forma, sempre que o programa faz referência a um endereço da memória, o sistema verifica esta dentro dos limites permitidos. Caso não esteja o programa é cancelado e uma mensagem erro é exibida.



9.3 – Alocação Contígua Simples

•Apesar de fácil implementação, a alocação contígua simples não permite a utilização eficiente dos recursos computacionais, pois apenas um usuário pode dispor destes recursos.

•Caso o programa não utilize toda a memória, existirá um espaço livre e sem uso da memória principal, gerando desperdício.

9.4 – Técnica de Overlay

•Na alocação contígua, todos os programas estão limitados ao tamanho da área de memória principal disponível para o usuário. Programas mais pesados e que necessitam de áreas maiores não podem ser executados com eficiência.

•Uma solução encontrada é dividir o programa em módulos, de forma que seja possível a execução independente de cada módulo, utilizando a mesma área de memória. Essa técnica é chamada de **overlay**.

•Considere uma programa que tenha três módulos: Um principal, uma de cadastramento e outro de impressão, sendo que os dois últimos são independentes.

•A independência dos módulos de cadastramento e de impressão significa que quando um estiver carregado na memória, o outro não precisa necessariamente estar presente. Somente o módulo principal é comum aos dois.

9.4 – Técnica de Overlay

•Neste exemplo, sempre que um dos dois módulos for referenciado pelo módulo principal, este módulo será carregado da memória secundária para memória principal.

•No caso do módulo referenciado já estiver na área de overlay, este não precisará ser carregado novamente.

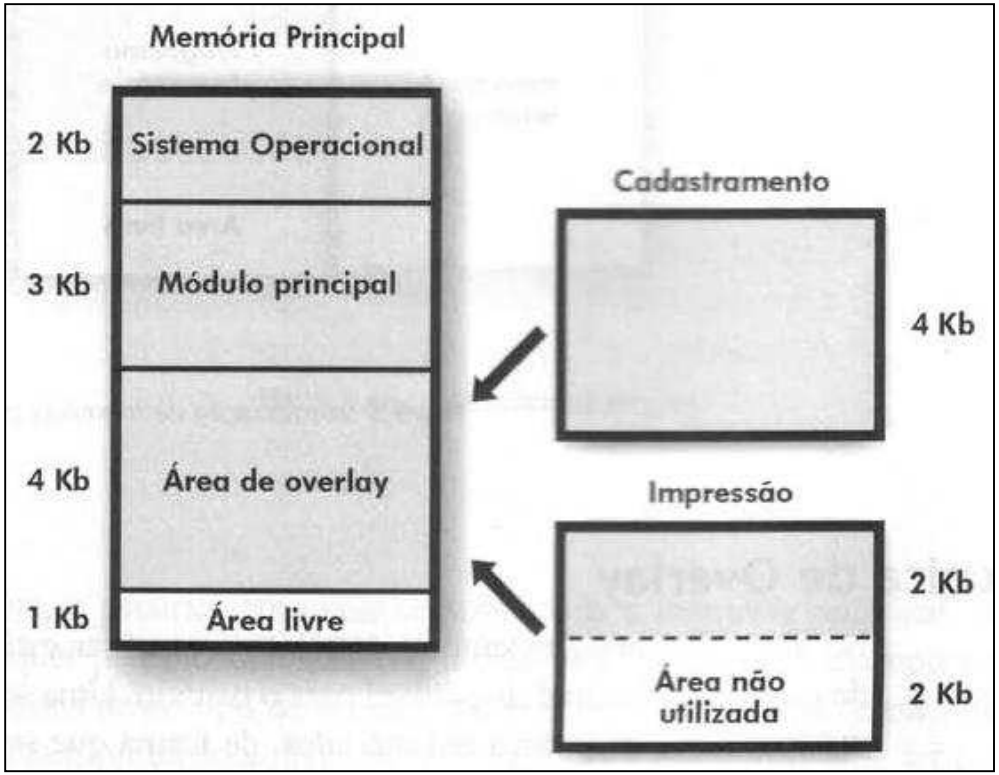
•A definição das áreas de overlay é função do programador, por meio de comandos da linguagem de programação utilizada pelo mesmo.

•A área de overlay é estabelecida a partir do tamanho do maior módulo. Por exemplo, se o maior módulo possuir 5Kb, a área de overlay deve possuir este mesmo tamanho.

9.4 – Técnica de Overlay

•A técnica de overlay, embora permita ao programador expandir o uso da memória, sua utilização deve ser feita com muito cuidado.

•Essa técnica além de dificultar o processo de manutenção dos programas, pode reduzir o seu desempenho, devido a transferência excessiva da memória principal e secundária.



9.5 – Alocação Particionada

•As técnicas apresentadas anteriormente (alocação contígua e overlay) podem ser utilizadas com eficiência somente em sistemas monoprogramáveis.

•Nos sistemas monoprogramáveis, o processador fica a maior parte do tempo ocioso, enquanto a memória principal é subutilizada.

•Os sistemas operacionais evoluíram no sentido de proporcionar melhor aproveitamento dos recursos disponíveis.

•Os sistemas multiprogramáveis já estão muito mais eficientes no uso do processador, necessitando assim, que diversos programas estejam na memória principal ao mesmo tempo e que novas técnicas de gerencia de memória sejam implementadas.

9.5.1 – Alocação Particionada Estática

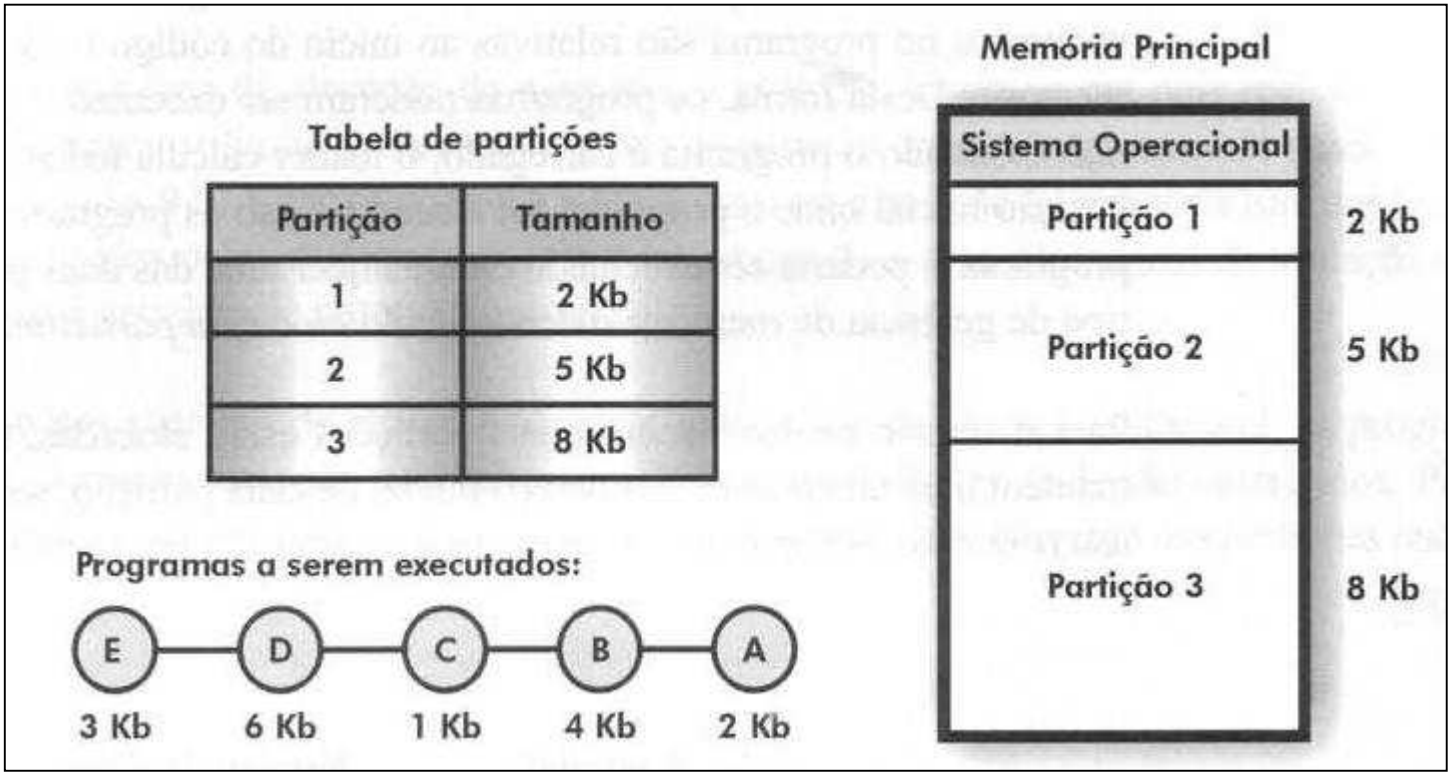
•Nos primeiros sistemas multiprogramáveis, a memória era dividida em pedaços de tamanho fixo, chamados **partições**.

•O tamanho das partições, estabelecidos na fase de inicialização do sistema, era definido em função do tamanho dos programas que executariam no ambiente.

•Sempre que fosse necessário a alteração do tamanho de uma partição, o sistema deveria ser desativado e reinicializado com a nova configuração. Esse tipo de gerencia de memória é conhecida como **alocação particionada estática**.

9.5.1 – Alocação Particionada Estática

•A figura abaixo ilustra o esquema de alocação particionada estática:



9.5.1 – Alocação Particionada Estática

•Inicialmente, os programas só podiam ser carregados e executados em apenas uma partição específica, mesmo se outras estivessem disponíveis.

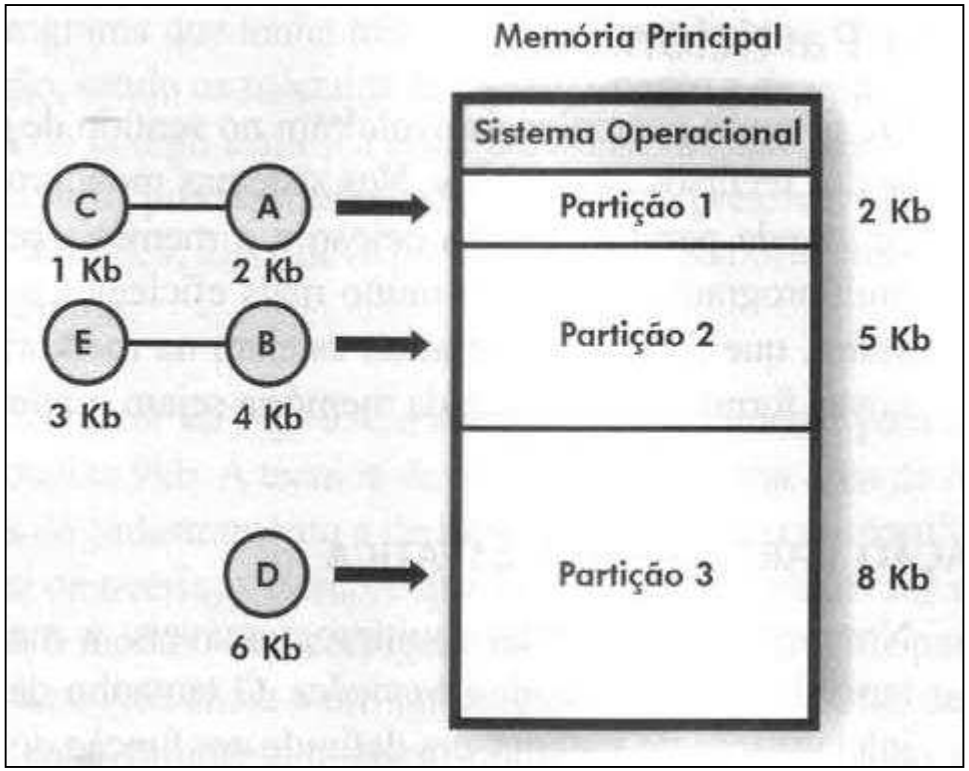
•Essa limitação se devia ao fato dos compiladores e montadores gerar apenas o código absoluto, onde todas as referências a endereços no programa são posições físicas da memória principal.

•A esse tipo de gerenciamento de memória chamou-se de **alocação particionada estática absoluta.**

9.5.1 – Alocação Particionada Estática

•Por exemplo, caso os programas “A” e “B” estejam sendo executados na Partição 1 e 2, respectivamente, e a Partição 3 estivesse livre. Os programas “C” e “E” não poderiam ser processados na Partição 3, pois estes fazem referência a ela.

•A figura abaixo ilustra este exemplo:



9.5.1 – Alocação Particionada Estática

•Com a evolução dos compiladores, o código gerado deixou de ser absoluto e passa a ser **relocável**.

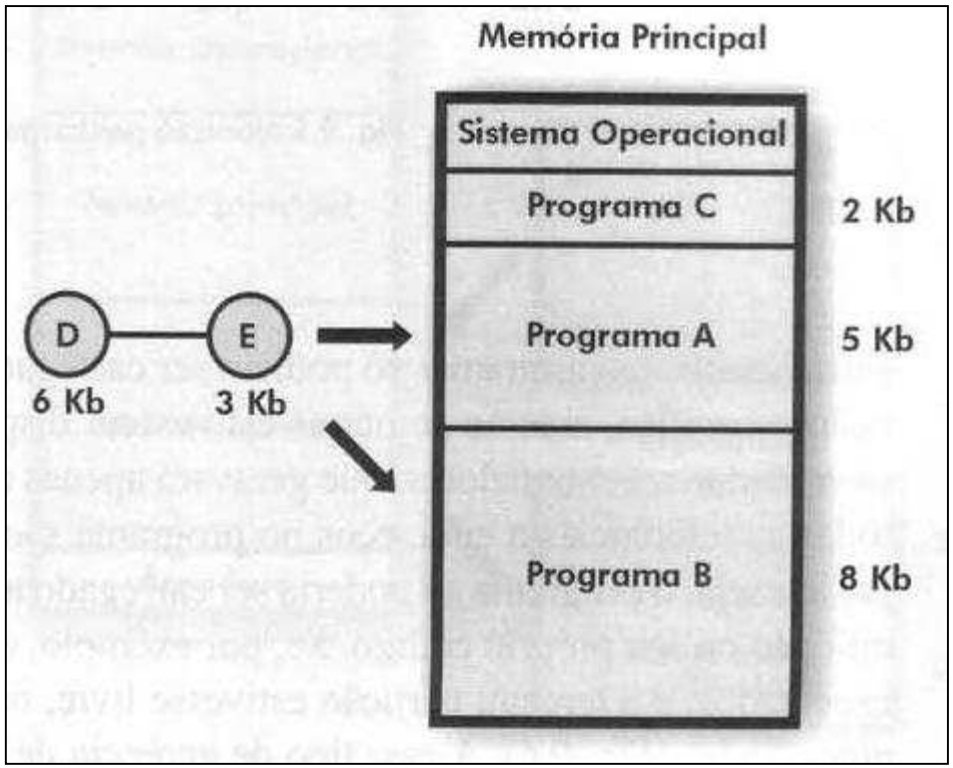
•No código relocável, todas as referências a endereços no programa são relativas ao início do código e não a endereços físicos de memória. Desta forma, os programas puderam ser executados a partir de qualquer partição.

•Quando o programa é carregado, o loader calcula todos os endereços a partir da posição inicial onde o programa foi alocado.

•No exemplo anterior, caso os programas A e B terminassem sua execução, o programa E poderia ser executado em qualquer uma das duas partições livres. Essa técnica é denominada **alocação particionada estática relocável**.

9.5.1 – Alocação Particionada Estática

•A figura abaixo ilustra o esquema da alocação particionada estática relocável:



9.5.2 – Alocação Particionada Dinâmica

•Um grande problema da técnica de Alocação Particionada Estática era a fragmentação dos programas, visto que sobravam espaços de memória em cada partição, gerando desperdício.

•As partições eram de tamanho fixo, caso um programa fosse consideravelmente menor que a partição, um grande espaço de memória seria desperdiçado.

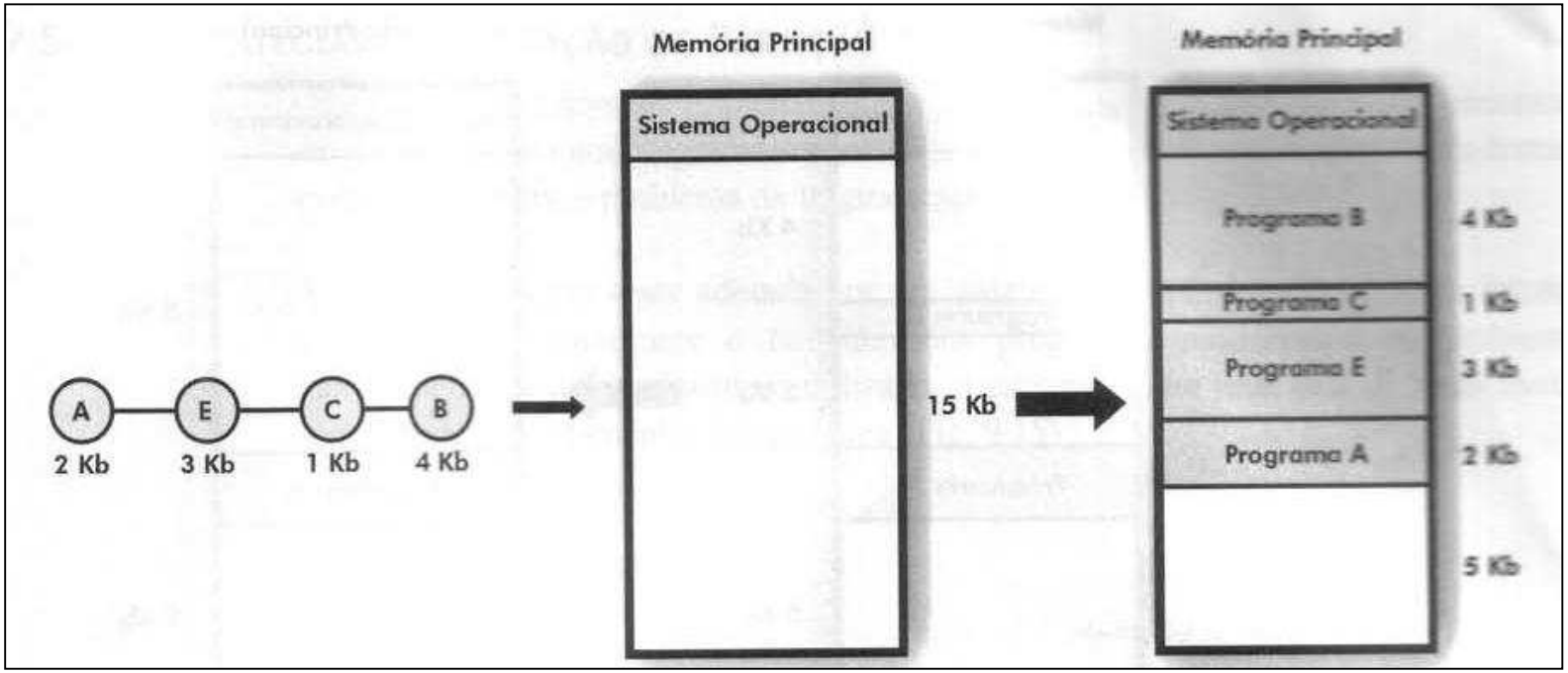
•Na **Alocação Particionada Dinâmica** foi eliminado o conceito de partições de tamanho fixo. Nesse esquema, cada programa utilizaria o espaço necessário, tornando essa área sua partição.

•Como os programas utilizam apenas o espaço que necessitam, no esquema de alocação dinâmica o problema de fragmentação interna não ocorre.

9.5.2 – Alocação Particionada Dinâmica

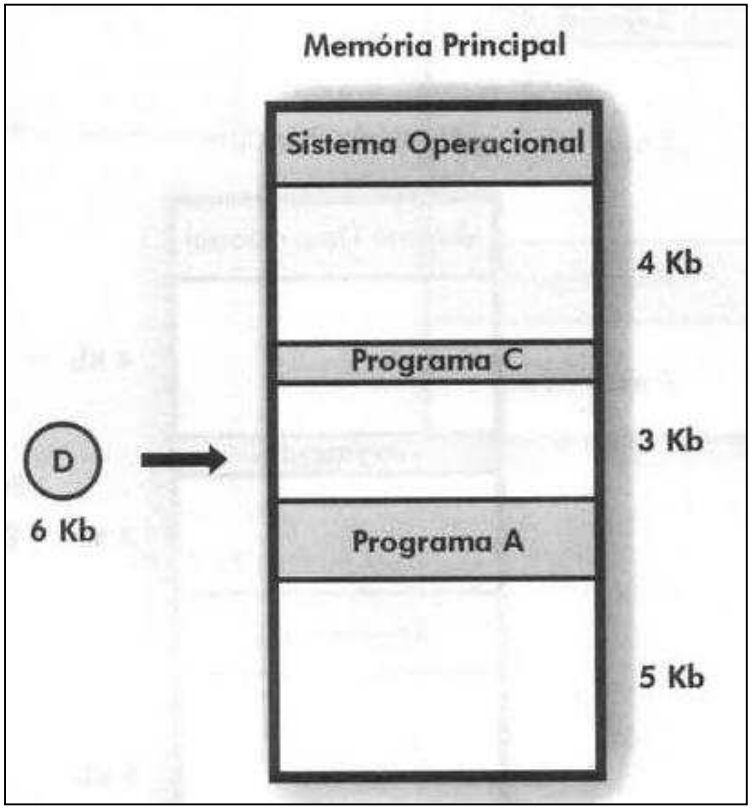
•Entretanto, existe um outro tipo de fragmentação, denominada **fragmentação externa**, que começará a ocorrer quando os programas forem deixando cada vez espaços menores na memória, não permitindo ingresso de novos programas.

•Vejamos o exemplo da figura abaixo, onde existem 4 programas para serem executados:



9.5.2 – Alocação Particionada Dinâmica

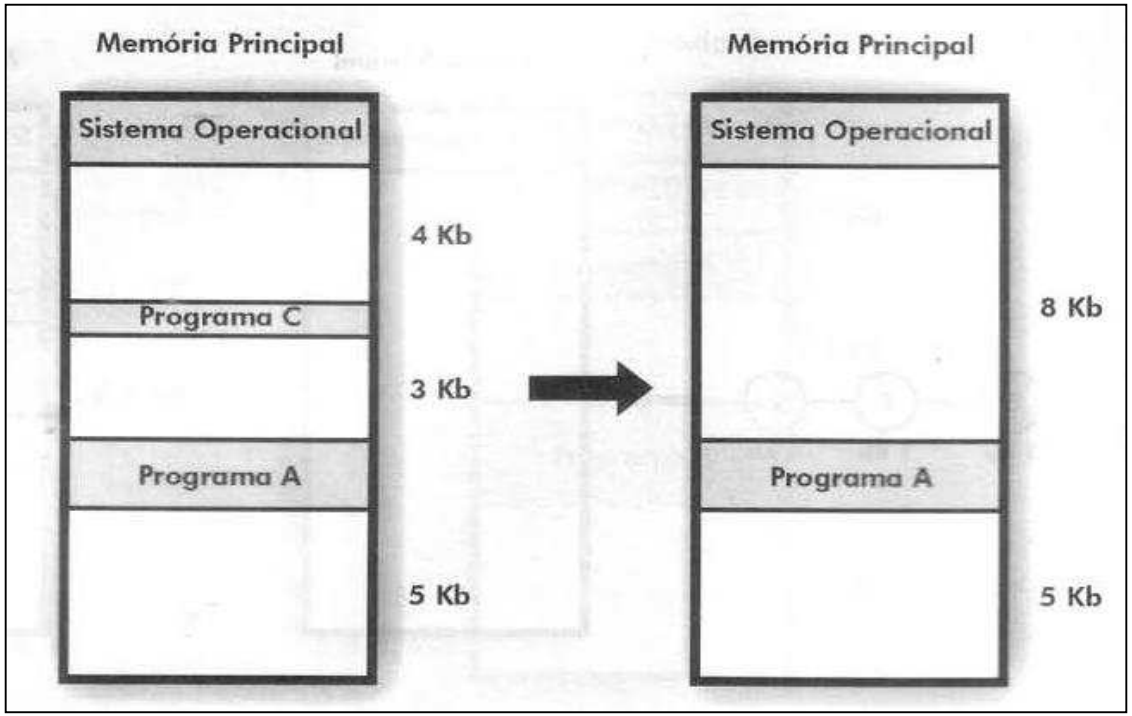
•Embora existam 12Kb de memória disponíveis, e o programa D precise de apenas 6Kb para ser executado, isso não será possível, pois não existem 6Kb contínuos. E a tendência é que os espaços de memória fiquem cada vez menores.



9.5.2 – Alocação Particionada Dinâmica

•Existem duas soluções para o problema da fragmentação externa da memória principal.

1. Conforme os programas terminarem, apenas os espaços adjacentes são reunidos, formando uma única área de memória maior. Como no caso da figura abaixo, quando o programa C terminar, uma área de 8Kb será criada.



9.5.2 – Alocação Particionada Dinâmica

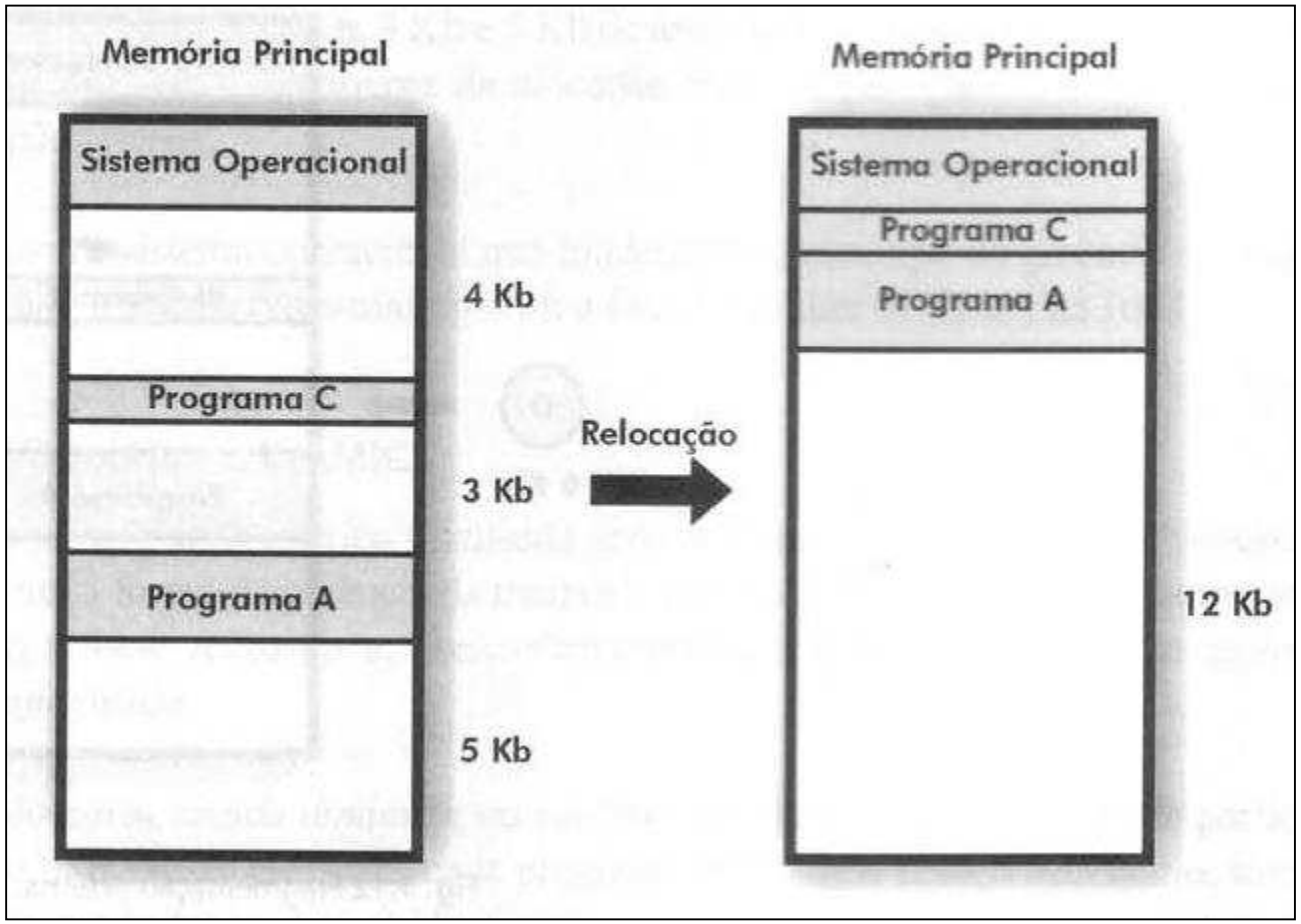
2. A segunda solução consiste na relocação de todas as partições ocupadas, eliminando todos os espaços entre elas e criando uma única memória livre e contínua.

   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**.

   Esse mecanismo é de grande complexidade de implementação, além de consumir muitos recursos computacionais, como processador e áreas de disco.

9.5.2 – Alocação Particionada Dinâmica

•A figura abaixo ilustra um exemplo da técnica de relocação dinâmica:



9.5.3 – Estratégias de Alocação de Partição

•Os sistemas operacionais implementam, basicamente, três estratégias para determinar em qual área livre um programa será carregado para execução.

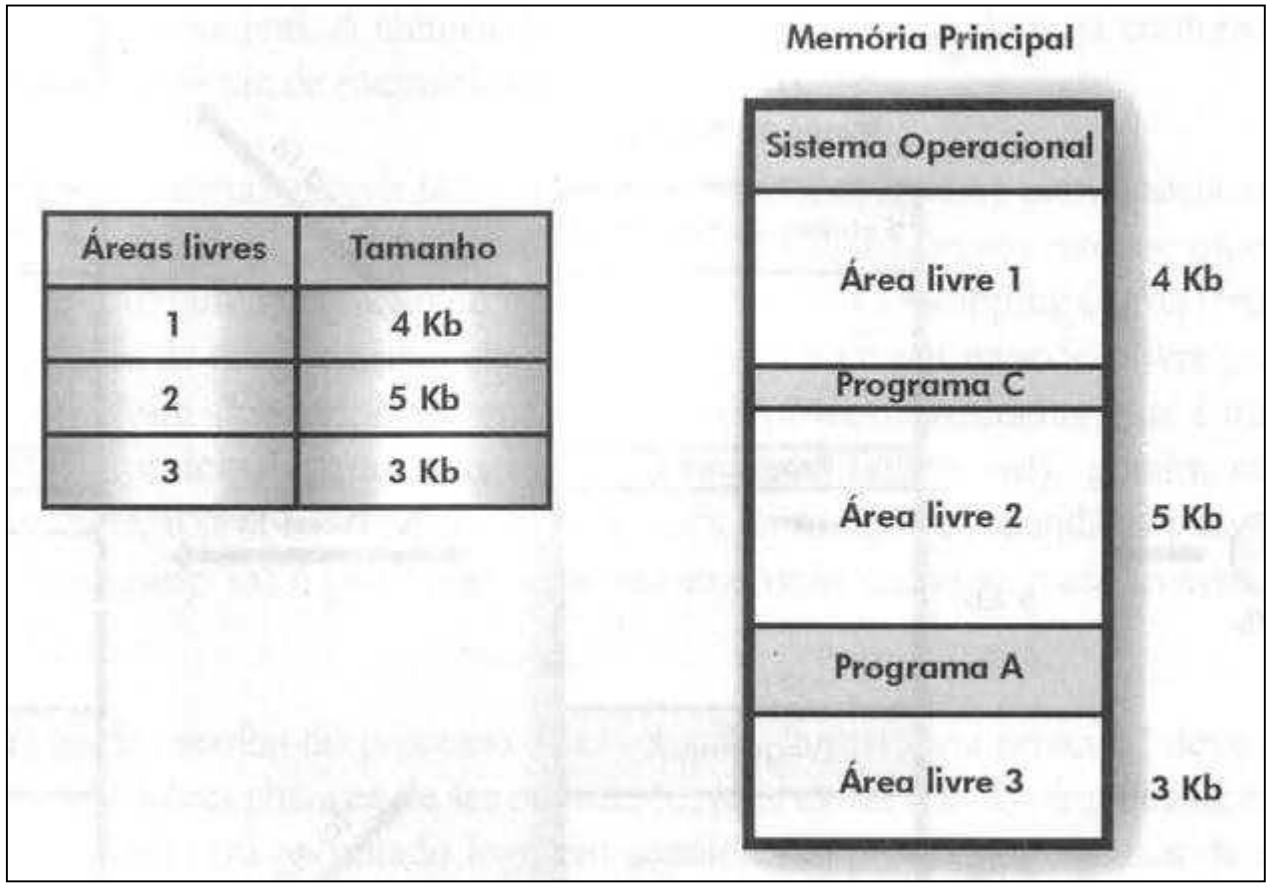
• Essas estratégias tentam amenizar a fragmentação externa.

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

•Independente do algoritmo utilizado, o sistema sempre irá possuir uma lista de áreas livres, como endereço e tamanho de cada área.

9.5.3 – Estratégias de Alocação de Partição

•A figura abaixo ilustra um exemplo de memória com sua lista de respectivas áreas livres:



9.5.3 – Estratégias de Alocação de Partição

•A estratégias para escolha de partição são:

1. **Best-fit**: A melhor partição é escolhida para alocar o programa, ou seja, a que deixará menos espaço vazio. Uma desvantagem é a tendência que cada vez mais a memória fique com pequenos espaços, aumentando a fragmentação.

2. **Worst-fit**: A pior partição é escolhida, ou seja, a que deixa mais espaço vazio para alocar o programa. Essa técnica reduz o problema de fragmentação.

3. **First-fit**: A primeira partição livre de tamanho suficiente é escolhida para carregar o programa. Das três estratégias, a First-fit é a que gasta menos recursos do sistema e também é a mais rápida.

9.6 – 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 da insuficiência de memória principal.

•Em todos os esquemas estudados anteriormente, um processo permanecia na memória principal até o final de sua execução, inclusive nos momentos em que esperava por eventos, como operações de E/S.

•O swapping é uma técnica onde o sistema escolhe um processo residente, que é transferido da memória principal para e memória secundária (swap out), geralmente disco.

•Posteriormente, o processo é carregado de volta da memória secundária para a memória principal (swap in) e pode continuar sua execução como se nada tivesse ocorrido.

9.6 – Swapping

•O algoritmo de escolha do processo a ser retirado da memória principal deve priorizar os processos com menores chances de ser executado, para evitar swap desnecessário.

•O conceito de swapping permite maior compartilhamento da memória principal e, consequentemente, maior utilização dos recursos do sistema computacional.

•Seu maior problema são os elevados custos das operações de entrada e saída, sendo que em computadores com menos recursos disponíveis (pouca memória principalmente), o sistema pode deixar de realizar tarefas importantes para efetuar operações de swapping.

Exercícios

1. Quais as funções básicas da gerência de memória?

2. Qual a diferença entre fragmentação interna e externa da memória principal?

3. Qual a diferença da alocação particionada estática absoluta em relação a alocação estática relocável?

4. Considerando as estratégias para escolha da partição dinamicamente, conceitue as estratégias best-fit e worst-fit especificando prós e contras de cada uma.

5. O que é swapping e para que é utilizada esta técnica?

6. Como funciona a técnica de overlay?

7. Como funciona a Alocação Contígua Simples, quais são suas limitações?