**Memória central**

Vários processos devem poder ser executados ao mesmo tempo, e para isso é preciso dividir a memória entre eles. É através da gestão de memória que o SO determina quanta memória é alocada para cada processo, e quanto tempo lhe é dado para utilizar.

À medida que os processos entram no sistema, são inseridos num “input queue” e ficam à espera de que o SO lhes atribua um bocado de memória, de acordo com uma das seguintes estratégias.

**Swapping**

Tipicamente, a quantidade de RAM necessária por todos os processos ativos costuma ser superior à disponível.

Para lidar com isto pode-se recorrer a “swapping”, que consiste em mover um processo inteiro para a memória quando é executado, e depois para o disco, onde é guardado até poder continuar a sua execução. Maior parte dos processos que se encontram num estado “idle” estão guardados em disco para não ocuparem memória. (Existe outra estratégia para lidar com isto: memória virtual).

**Exemplo:** Se um processo de prioridade elevada surge e precisa de ser executado, um ou mais processos de menos prioridade são “swapped” para o disco, libertando memória que vai ser utilizada pelo novo processo. Quando este termina, os processos que foram guardados em disco voltam ao mesmo espaço de RAM que estavam a utilizar antes.

**Alocação Contígua**

Estratégia mais antiga e simples de gestão de memória. SO compara tamanho do próximo processo a ser executado com quantidade de RAM disponível, e se tiver suficiente para esse processo, aloca-a de maneira a que fique contígua à que já estava a ser utilizada por outros processos. Se não existir memória livre suficiente para o processo, ele é adicionado à “input queue”.

Surge outro problema quando o processo aloca memória dinamicamente. Se existir um “buraco” (memória livre) adjacente a este processo, ele pode ser alocado para outro processo. Se não, ele terá de se mover para um buraco maior na memória ou terá de ser “swapped” para o disco até haver memória suficiente para o processo terminar a sua execução.

À medida que os processos terminam, vão libertando a memória para os outros. Como é alocada memória para cada processo, muitas vezes deixam “buracos” na memória que podem ser pequenos demais para o próximo processo da queue. Este terá de esperar até que espaço suficiente fique disponível. Os “buracos” vão ficando cada vez mais pequenos e difíceis de preencher. Isto pode provocar uma situação de fragmentação externa, que ocorre quando existe espaço livre em memória dividido em diferentes “buracos” mais pequenos, tornando impossível a alocação contígua.

**Segmentação**

Consiste na divisão de memória em segmentos/secções. Os seus endereços lógicos incluem um valor que identifica o segmento e o seu offset. **<segment-number, offset>.**

Este tuplo é traduzido para um endereço de memória. Cada segmento tem um comprimento e um conjunto de permissões. Um processo só pode fazer referência a um segmento se o tipo desta for permitido de acordo com as permissões.

Segmentos também podem ser usados para implementar memória virtual, em que ficam associados a uma flag que indica se este está presente na memória física ou não.

A segmentação pode ser implementada de várias formas:

* **Sem paging –** associado a cada segmento existe informação que indica onde o segmento está na memória (“segment base”).
* **Com paging –** em vez de ter associado uma posição na memória, a informação do segmento inclui um endereço de uma tabela de páginas.