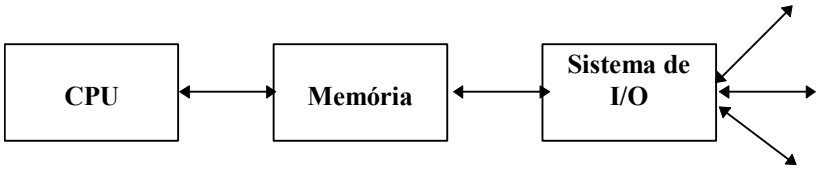
Gerência de Memória Principal

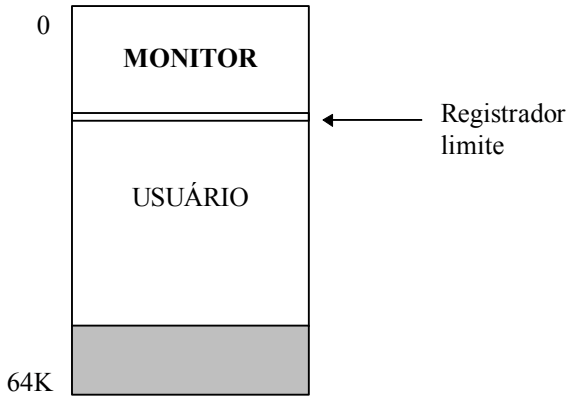
A multiprogramação permite que a CPU possa ser compartilhada por um conjunto de processos, em benefício do desempenho global do computador. Entretanto, para obter esse ganho, é necessário compartilhar a memória entre múltiplos processos.

Introdução

A memória corresponde a um longo *array* de palavras ou bytes, cada um com o seu endereço. A CPU e os controladores dos dispositivos periféricos podem ler e escrever em posições da memória:



Neste esquema a memória é dividida em duas partes: área do usuário e área do sistema operacional. A área do sistema operacional fica geralmente no início da memória porque a tabela de interrupção usualmente ocupa os primeiros endereços da memória.



- **Hardware de proteção**

Se o sistema operacional está na memória baixa e a área de usuário na memória alta, é necessário proteger o código e os dados do SO contra o acesso acidental ou malicioso por parte de um programa de usuário. Esta proteção deve ser fornecida pelo hardware e pode ser implementada através de um registrador de proteção, denominado **limite**. O mecanismo de acesso à memória só "aceita" endereços maiores ou iguais ao valor desse registrador. O valor do registrador é zerado quando a execução vai para o SO e somente o SO pode alterar o valor desse registrador.

- **Relocação**

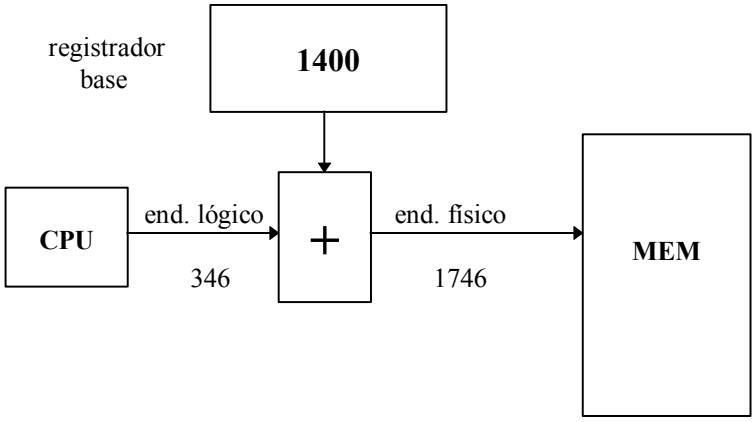
Um problema a ser considerado é a carga de programas. Embora o espaço de endereçamento do computador inicie em 0, o primeiro endereço do programa usuário (**endereço inicial de carga**) é aquele contido no registrador limite. É preciso de alguma forma acertar os endereços referidos no programa do usuário (isto é, no código de máquina correspondente ao programa de usuário).

Se o endereço inicial de carga é conhecido previamente, os endereços absolutos (endereços físicos) podem ser gerados na compilação (mais exatamente, na ligação), durante a geração do código de máquina. Neste caso, uma alteração no endereço de carga implica em recompilar o programa.

Quando o endereço inicial de carga não é conhecido previamente, o código de máquina (**arquivo executável**) é gerado supondo que o programa vai ser carregado e executado a partir do endereço zero. Neste caso, se um programa tem tamanho K, os seus endereços (isto é, os endereços referidos nas instruções de máquina desse programa) certamente são valores entre zero e K-1. Estes endereços entre 0 e K-1 constituem o que se denomina **espaço de endereçamento lógico do programa**. Quando o programa é carregado na memória é necessário acertar os seus endereços lógicos para que eles passem a indicar os **endereços físicos** corretos. O acerto é muito simples, basta somar o endereço inicial de carga à cada endereço lógico referido no programa.

Os endereços lógicos são também referidos como endereços relativos e o processo de acerto dos mesmos é conhecido como **relocação**. A relocação pode ser feita de forma estática ou de forma dinâmica. Na **relocação estática**, os endereços relativos são transformados em absolutos no momento da carga do programa. Para isso, o arquivo executável deve conter também a indicação dos locais do código que referem endereços. Essa indicação é conhecida como **informação de relocação**. É tarefa do **carregador** (**loader**) fazer o acerto dos endereços e isso é feito durante a carga do programa.

No caso da **relocação dinâmica**, os endereços do programa permanecem lógicos o tempo inteiro (isto é, as instruções de máquina não são alteradas para referir endereços físicos). É o mecanismo de acesso à memória que vai fazer a correção (relocação) durante a execução do programa, conforme é explicado a seguir. Para fazer a relocação, é necessário que o endereço inicial de carga fique armazenado em um registrador da UCP. No esquema de proteção descrito anteriormente, esse endereço ficava no registrador limite. Agora, em um computador que faça relocação dinâmica, esse valor fica em um registrador referido normalmente como **registrador base** ou **registrador de relocação**. Quando a UCP requer um acesso ao endereço (lógico) **E**, o mecanismo de acesso a memória entende como sendo um acesso ao endereço **RB**+**E**, onde **RB** é o valor contido no registrador base. A figura abaixo mostra como os endereços relativos são transformados em endereços absolutos durante a execução:



O uso de relocação dinâmica permite movimentar um programa na memória com facilidade, basta alterar o valor do registrador base. Isto permite mudar o tamanho do sistema operacional durante a execução de um programa (como por exemplo para incluir mais *buffers* ou então um controlador de dispositivo pouco utilizado). Outra forma comum de permitir o aumento do tamanho do SO é carregar o programa do usuário sempre no fim da memória. Isto permite que o sistema operacional cresça até o endereço contido no registrador base e não é necessário movimentar o programa de usuário.

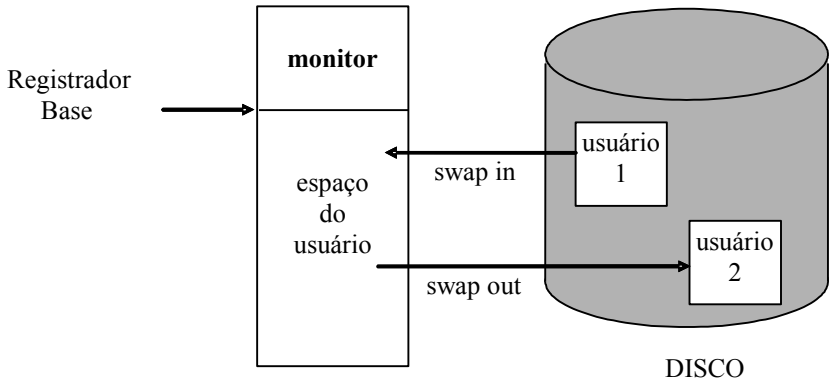
Com o uso de registrador base, passam a haver dois tipos de endereços:

- endereços **lógicos**, entre **0...MAX** (endereços vistos pelo programa);
- endereços **físicos**, entre **RB+0...RB+MAX** (vistos pelo sistema operacional).

Todos os endereços manipulados pelo SO são físicos, pois ele acessa a memória de diversos processos. Cada endereço passado entre um programa de usuário e o SO precisa ser apropriadamente convertido. Este conceito de espaço de endereçamento lógico mapeado para um espaço de endereçamento físico é central quando se trata de esquemas de gerência de memória.

Swapping

O esquema de memória descrito acima é pouco utilizado, uma vez que é monousuário. Entretanto, ele pode ser utilizado para multiprogramação através da adição de *swapping*:



O programa que perde a CPU é copiado para o disco, enquanto o programa que ganha é transferido do disco para a memória principal. Este método foi utilizado nos primeiros sistema *timesharing*. Os programas são mantidos em um disco, que deve ser rápido e grande o bastante para conter diversas imagens de processos (códigos executáveis).

Partições Múltiplas

Na multiprogramação, diversos programas devem estar presentes na memória ao mesmo tempo, para que a CPU possa ser rapidamente alternada entre eles. A solução natural para esta necessidade é dividir a memória em regiões ou partições. Cada partição pode conter um programa a ser executado, limitando então o grau de multiprogramação pelo número de partições. A memória pode ser dividida em partições de duas maneiras distintas:

- **partições fixas**, onde as regiões são estáticas*;*
- **partições variáveis**, onde as regiões são dinâmicas

- **Hardware de proteção**

Passa a ser necessário proteger não apenas o código e dados do sistema operacional, mas também código e dados de outros usuários. É possível implementar proteção para partições múltiplas utilizando duas técnicas diferentes: através de **registradores de limite inferior e superior** ou através de **registrador base e registrador limite**. A primeira técnica é usada quando o computador trabalha com relocação estática. Nesse caso, sendo **LI** e **LS** os endereços contidos nos registradores limite inferior e limite superior, respectivamente, um endereço **E** é consideralo válido quando **LI**≤**E**<**LS**.

A segunda técnica é usada quando o computador trabalha com relocação dinâmica. Nesse caso o registrador base (**RB**) é usado para relocação (todo endereço **E** é mapeado em **RB**+**E**) e o registrador limite (**RL**) é usado para proteção (um endereço **E** é consideralo válido quando **E**<**RL**). Observe que o conteúdo do **RB** é o endereço inicial de carga (início da partição) e o conteúdo do **RL** é o tamanho do programa.

■ Partições fixas

A memória é dividida em partições que não são alteradas durante a execução do sistema. Por exemplo, uma memória de 256K palavras poderia ser dividida da seguinte forma:

- Monitor residente: 64K palavras;
- Espaço para processos pequenos: 16K palavras;
- Espaço para processos médios: 48K palavras;
- Espaço para processos grandes: 128K palavras.

Quando um programa vai ser executado, é necessário levar em conta as suas necessidades de memória, para escolher a partição onde ele vai ser colocado. Tal pode ser especificado pelo usuário ou então automaticamente determinado pelo sistema operacional.

A partir desse momento há duas possibilidades básicas: **(a)** montar uma fila individual para cada tamanho de partição ou **(b)** montar uma fila única que englobe todas as requisições.

Existem dois tipos de perdas de memória por fragmentação. Elas são chamadas de **fragmentação interna** e **fragmentação externa**. Um processo que necessita *m* palavras de memória pode executar em uma região de tamanho *n*, onde *n* >= *m*. A diferença *n* - *m* representa a fragmentação interna, isto é, a memória desperdiçada dentro da região.

Ocorre fragmentação externa quando uma região está disponível para uso, mas é pequena demais para os processos serem executados. Por exemplo, 3 programas estão esperando para serem executados. Eles possuem as seguintes necessidades de memória: 60K, 80K e 90K. Existem 3 partições livres, cada uma com 40K. Apesar da área livre total ser de 120K, a mesma não pode ser aproveitada para processá-los.

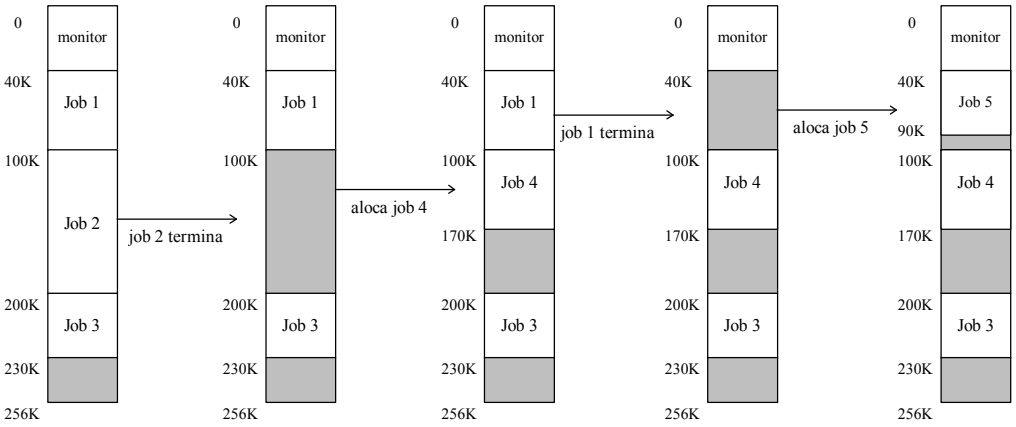
Considere agora uma situação onde existem 4 partições livres, com respectivamente 100K, 40K, 40K e 40K. Neste momento, 4 processos desejam executar, necessitando das seguintes quantidades de memória: 70K, 30K, 60K e 60K. Após fornecer a partição de 100K ao processo que precisa de 70K, e fornecer uma partição de 40K ao processo que precisa de 30K, a perda de memória por fragmentação interna soma 40K (30K + 10K). A perda por fragmentação externa é de 80K (40K + 40K). A perda total nesta situação é de **55% da memória**.

■ Partições Variáveis

O maior problema das partições fixas é determinar a melhor divisão da memória, mesmo porque a melhor divisão varia ao longo do tempo. A solução é permitir que o tamanho das partições varie ao longo do tempo (mecanismo de partições variáveis).

O esquema é bastante simples: o sistema operacional mantém informações sobre quais partes da memória estão disponíveis e quais estão ocupadas. Inicialmente, toda a memória está disponível (um único “buraco”). Quando um processo chega para ser executado, a lista de buracos é pesquisada e o processo é colocado em uma região livre suficientemente grande. Do buraco que representa a região livre é removida a área a ser utilizada pelo processo, sendo o restante mantido no buraco livre. Exemplificando, considere os seguintes processos a serem executados:

JOB	Memória	Tempo
1	60K	10
2	100K	5
3	30K	20
4	70K	8
5	50K	15



Características das partições variáveis

- sempre existe um conjunto de áreas livres espalhadas pela memória;
- este conjunto é pesquisado, sendo necessário uma área maior ou igual à necessidade do programa;
- se a área é maior, a parte restante continua livre;
- quando o processo termina, ele libera a área. Se ela for adjacente a uma outra área livre, ambas são reunidas em uma única área livre. Esta liberação pode fazer com que programas esperando possam ser executados;
- para evitar o trabalho com pequenos espaços, a unidade de alocação não é necessariamente o byte, podendo ser blocos de tamanho 2 Kbytes, por exemplo. Isto introduz uma pequena fragmentação interna.

Algoritmos de alocação para partições variáveis

Existem diversos algoritmos para a escolha de uma área livre. Este problema é conhecido genericamente como alocação dinâmica de memória. Abaixo estão descritos os 3 principais algoritmos:

- *first-fit*: aloca o primeiro espaço livre que seja suficientemente grande;
- *best-fit*: aloca o menor espaço livre que seja suficientemente grande. Produz a menor sobra de espaço livre;
- *worst-fit*: aloca o maior espaço livre. Produz a maior sobra de espaço livre. Este espaço maior que fica poderá ser mais útil do que o pequeno espaço livre deixado pelo best-fit.

Um bom algoritmo, utilizado na prática é conhecido como *circular-fit*. Ele funciona como o *first-fit*, mas inicia a procura na lacuna seguinte à última sobra.

Compactação e *swapping*

A utilização da memória é melhor com partições variáveis do que com partições fixas, mas a fragmentação externa é sempre um problema. Uma solução para isto é a **compactação**. Os programas são deslocados na memória de forma que todo o espaço livre de memória fique reunido em uma só área, no início ou no fim da memória. Se o sistema trabalha com relocação estática, a compactação fica inviável (pois é necessário corrigir os endereços utilizados nos programas). Se o sistema utiliza relocação dinâmica (registradores base e limite) a compactação fica simples, mas mesmo assim, o custo da compactação em termos de tempo de CPU deve ser considerado.

É possível acrescentar *swapping* ao esquema de partições variáveis. Enquanto o processo está suspenso, sua memória é aproveitada por outros processos. Se o hardware dispõe de registradores de base e limite, o programa pode voltar em qualquer ponto da memória. Podemos assim obter compactação através do *swapping.*

Exercícios

1. Cite duas diferenças entre endereços lógicos e físicos.

2. Explique a diferença entre fragmentação interna e externa.

3. Descreva os seguintes algoritmos de alocação: Primeiro-apto; Mais-apto e Menos-apto.

4. Dadas partições de memória com 100K, 500K, 200K, 300K e 600K (em ordem), de que forma cada um dos algoritmos do primeiro-apto, mais-apto e menos-apto alocarão processos com 212K, 417K, 112K e 426K (em ordem)? Qual dos algoritmos faz uso mais eficiente da memória?

5. O que é swapping e como funciona? Resposta...