Sistemas Operacionais

Gerenciamento de Memória

Alocação

SO - Gerenciamento de Memória

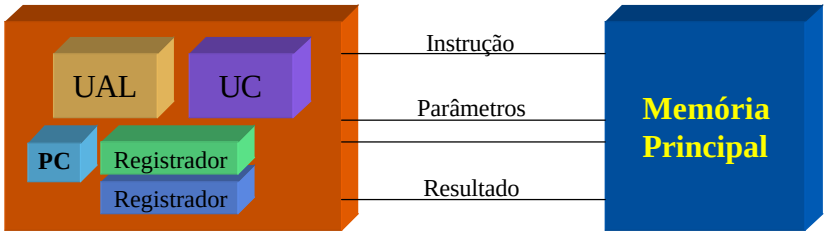
- Introdução
- Atribuição de Endereços, Carregamento Dinâmico, Ligação Dinâmica
- Espaço de Endereçamento Físico *versus* Lógico
- Alocação Contígua Simples
- Alocação Particionada
- Fragmentação Externa e Interna
- Gerência de Memória com Mapeamento de Bits
- Gerência de Memória com Listas Ligadas
- Algoritmos de Alocação de Partições
- *Swapping*

Introdução

- Memória ➔ um dos mais importantes recursos de um computador
  - necessidade de ser gerenciada
- Funções de um Gerente de Memória
  - controlar quais partes da memória estão em uso
  - controlar quais partes estão livres
  - alocar memória a processos quando estes precisarem
  - desalocar quando eles não mais necessitarem
  - gerenciar a troca (*swapping*) dos processos entre memória principal e disco, quando a memória não é grande o suficiente para guardar todos os processos

Introdução

- A CPU busca instruções da memória de acordo com o valor do contador de instruções (PC) do programa em execução. Essas instruções podem:
  - buscar novos valores de endereços específicos da memória
  - armazenar novos valores em posições específicas da memória

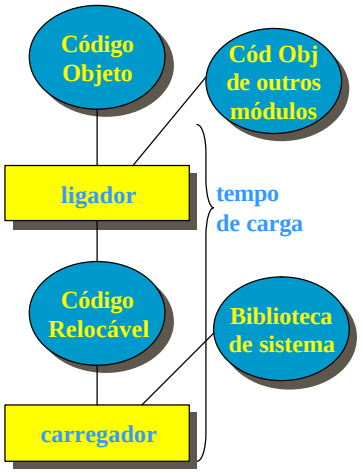


Atribuição de Endereços

- A maioria dos sistemas permite que os processos de usuários sejam colocados em qualquer parte da memória
  - assim, embora o espaço de endereçamento comece em 0000, o endereço inicial de um processo não precisa ser este.
- Em tempo de compilação
  - código absoluto: se o programa vai ser carregado no endereço E, o código gerado pelo compilador inicia-se neste endereço (se for alterado, deve ser recompilado). Ex.: ".COM" do DOS

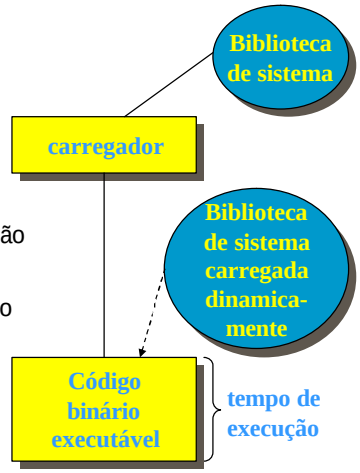
(cont.) Atribuição de Endereços

- Em tempo de carga
  - código relocável: quando não se conhece em tempo de compilação a posição em que o programa será armazenado. Ex.:
    - Compilador: 14 bytes do início
    - Carregador: 74014

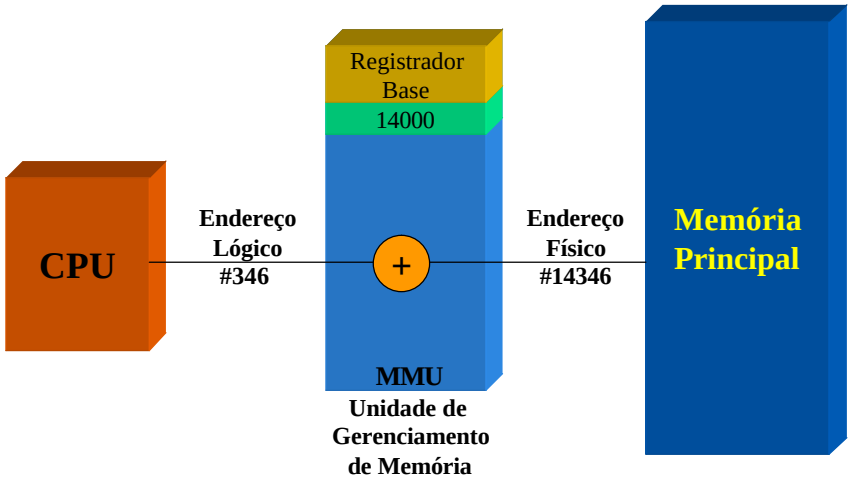


Atrib. de Endereços, Carregamento e Ligação Dinâmicos

- Em tempo de execução
  - se o processo se move durante a execução, a determinação do endereço deve ser atrasada para o tempo de execução
- Carregamento Dinâmico (usuário)
  - rotinas carregadas em tempo de execução
- Ligação Dinâmica (sist. operacional)
  - *stub*: pedaço de código para a localização da biblioteca dinâmica (se ela está presente ou não)
    - Economia de espaço
    - Atualização
    - Controle de concorrência



Espaço de Endereçamento Físico *versus* Lógico



Organização e Gerência da Memória

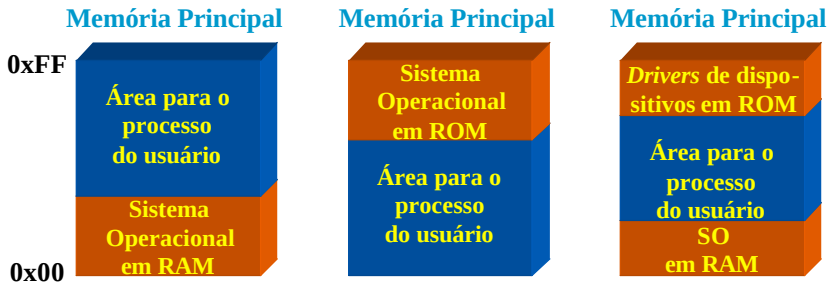
- Toda vez que desejarmos executar um programa (residente em memória secundária), deveremos, de alguma forma, carregá-lo para a memória principal, para que o processador possa referenciar suas instruções e seus dados
- Nos sistemas monoprogramados a gerência da memória não é muito complexa, mas nos sistemas multiprogramados ela se torna crítica
- A seguir apresentaremos os principais esquemas de organização e gerência da memória principal

Alocação Contígua Simples

- Implementada nos primeiros SOs e ainda está presente em alguns sistemas monoprogramados
- Esquema mais simples para gerência de memória
- A memória principal é dividida em duas partes:
  - uma para o Sistema Operacional
  - outra para o(s) programa(s) do(s) usuário(s)
- SO pode ocupar tanto a parte baixa quanto a parte mais alta da memória, dependendo da localização do vetor de interrupções

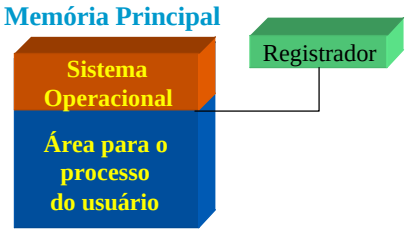
(cont.) Alocação Contígua Simples

- No IBM-PC é utilizado o terceiro modelo: nos 8KB mais altos dos 1 MB de endereçamento existe uma ROM com um programa chamado BIOS (Basic Input Output System), correspondente a todos os *drivers*



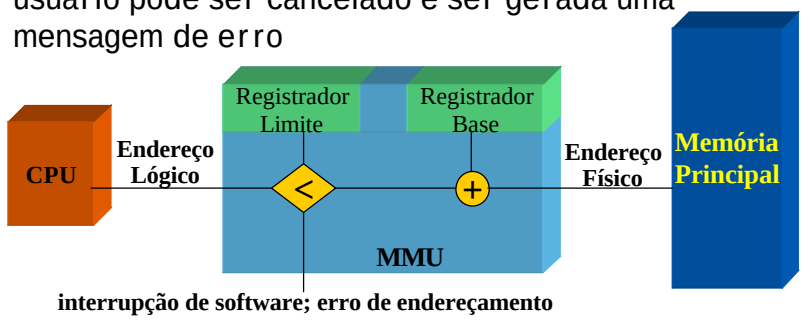
(cont.) Alocação Contígua Simples

- Usuário tem controle sobre toda a memória principal, podendo acessar até a área do SO (caso do DOS)
- Pode-se implementar um esquema de proteção usando um registrador que delimite as áreas do SO e usuário



(cont.) Alocação Contígua Simples

- Sempre que um programa do usuário faz referência a um endereço de memória, o sistema verifica se o endereço está nos seus limites (registrador base e registrador limite). Caso não esteja, o programa de usuário pode ser cancelado e ser gerada uma mensagem de erro



(cont.) Alocação Contígua Simples

- Problema: não permite a utilização eficiente do processador e nem da memória principal
- Em SOs que suportam o conceito de MULTIPROGRAMAÇÃO, é preciso que existam outras formas de organização da memória principal, a fim de suportar vários processos na memória simultaneamente

Multiprogramação

- Motivação:
  - tornar mais fácil programar uma aplicação, dividindo-a em dois ou mais processos
  - existência de serviço interativo para vários usuários simultaneamente (tempo de resposta)
  - utilização da CPU (existência de processos I/O *bound*)
- Modelos de Multiprogramação
  - Modelos Simplista
    - Se cada processo gasta 20% de seu tempo na memória usando o processador, então 5 processos alcançam um fator de utilização da CPU de 100%
  - E se os 5 processos esperam por E/S simultaneamente?

(cont.) Multiprogramação

- Modelo Probabilistico
  - Cada processo gasta p% de seu tempo com E/S
  - n processos ➔ probabilidade de estarem aguardando E/S simultaneamente: p^n

$$F.U._{CPU} = 1 - p^n$$

  - Exemplo:
    - 1 MB de memória, SO de 200K, processos de 200K e cada um gasta 80% do tempo com E/S ➔ 4 processos na memória ➔ $FU_{CPU} \cong 60\%$
    - Adição de 1 MB de RAM ➔ grau de 4 p/ 9 ➔ FU ≅87%
    - Adição de mais 1 MB de RAM ➔ grau de multiprogramação de 9 para 14 ➔ FU ≅ 96%

Alocação Particionada Estática

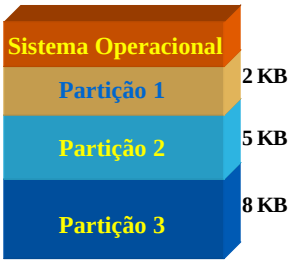
- Nos primeiros SOs multiprogramados a memória principal era dividida em pedaços de tamanho fixo (possivelmente distintos), chamados partições
- O tamanho das partições era estabelecido na fase de inicialização do sistema (*boot*), em função dos tamanhos dos programas que iriam ser executados
  - alteração de tamanho de partição
    - ➔ reinicialização

Tabela de partições

Partição	Tamanho
1	2 KB
2	5 KB
3	8 KB

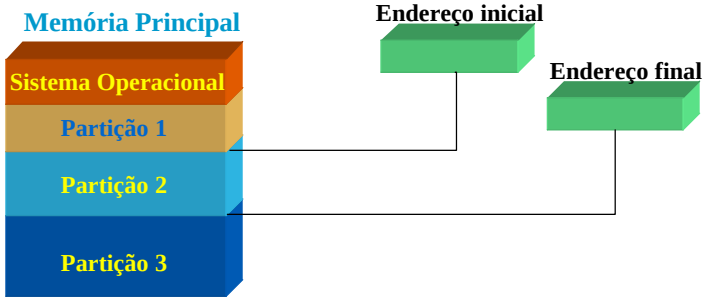
Programas a serem executados

Memória Principal



Alocação Particionada Estática - Proteção

- A proteção baseia-se em dois registradores, que indicam os limites inferior e superior da partição onde o programa está sendo executado
- Acessos do processo usuário provocam a comparação com o registrador limite, caso haja invasão, o processo é terminado com uma mensagem de erro.



Alocação Particionada Estática - Proteção

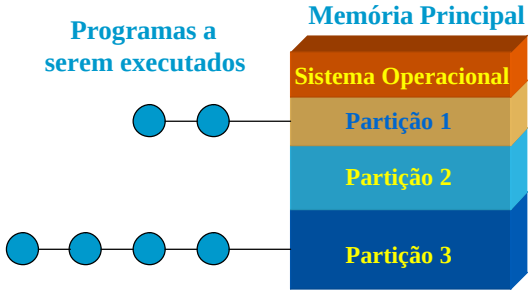
- E as chamadas aos serviços do SO ?
  - Isto é resolvido através da instrução chamada ao supervisor (*supervisor call*)

Modo usuário *versus* Modo supervisor

- Modo Usuário:
  - Execução das instruções disponíveis aos usuários
- Modo Supervisor:
  - Todas as instruções permitidas (não é feita comparação com o registrador limite)
- A passagem do modo usuário para o modo supervisor ocorre quando da solicitação de um serviço

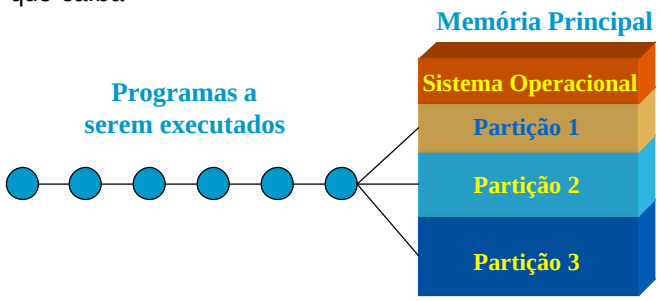
Alocação Particionada Estática

- Partições Fixas com Filas Múltiplas:
  - processo é colocado em uma partição determinada. Se estiver ocupada, aguarda na fila;
  - ou, processo é colocado na fila da menor partição em que caiba
- Problema: partições não utilizadas



Alocação Particionada Estática

- Partições Fixas com Fila Única:
  - fila única para todas as partições;
  - processo é colocado na fila da menor partição disponível em que caiba

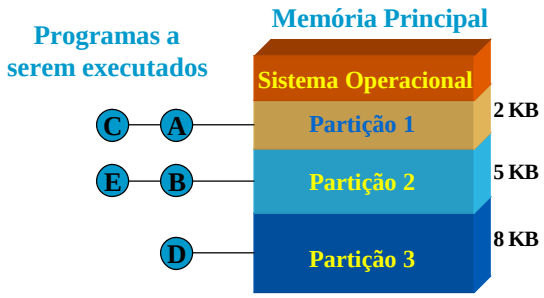


Alocação Particionada Estática

- Problemas:
  - partições vazias por falta de “clientes”
  - colocação de um programa menor em uma partição grande
- Uma solução:
  - correr toda a fila antes de escolher o “cliente”
  - Problema:
    - discrimina pequenos processos

Aloc. Particionada Estática Absoluta

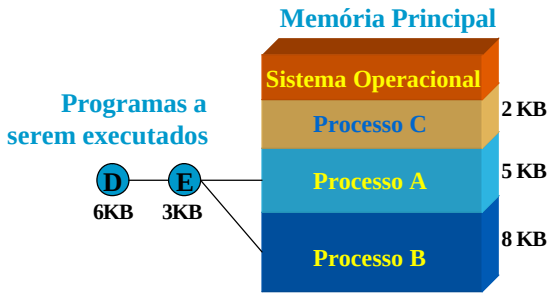
- Em sistemas em que os compiladores e montadores geram apenas código absoluto, os processos podem executar em apenas uma das partições, mesmo que outras estejam livres



- Se os processos A e B estivessem executando e a partição 3 estivesse livre, os processos C e E não poderiam ser executados

Aloc. Particionada Estática Relocável

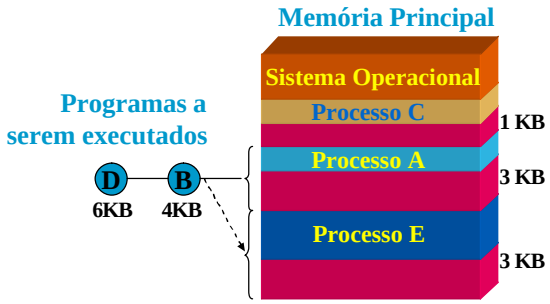
- Com a evolução dos compiladores, *linkers* e *loaders*, tornou-se possível a geração de código relocável, permitindo que os processos sejam carregados em qualquer uma das partições



- Se os processos A e B terminarem de executar, o processo E pode ser carregado e executado em qualquer uma das duas partições

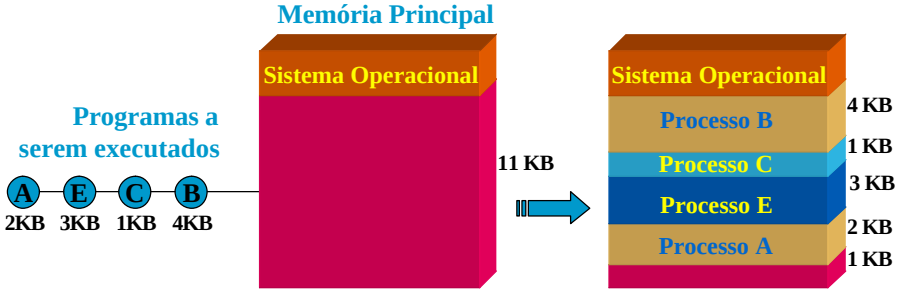
Alocação Particionada Estática

- Problema da APE: fragmentação excessiva
  - Fragmentação é o problema que surge quando pedaços de memória ficam impedidos de serem utilizados por outros processos.

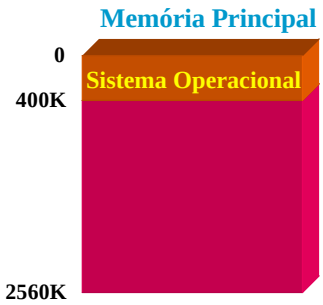


Alocação Particionada Dinâmica

- Visa reduzir o problema da fragmentação e, consequentemente, aumentar o grau de compartilhamento da memória
- É eliminado o conceito de partições de tamanho fixo
- Na APD, cada processo utiliza o espaço necessário, passando esse pedaço a ser a sua partição

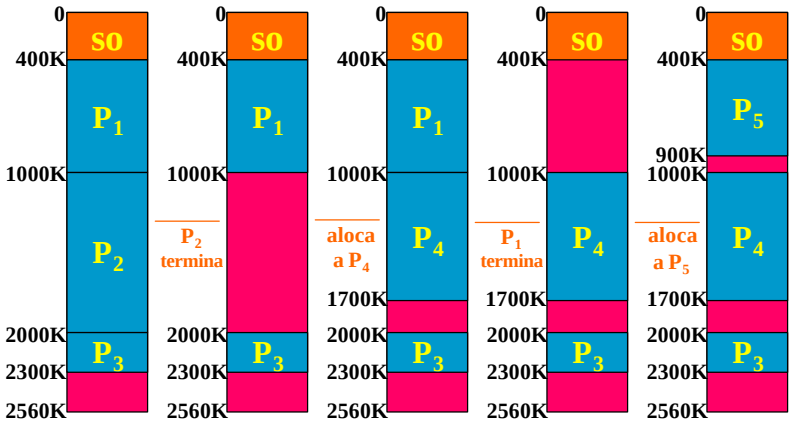


Alocação Particionada Dinâmica - Exemplo



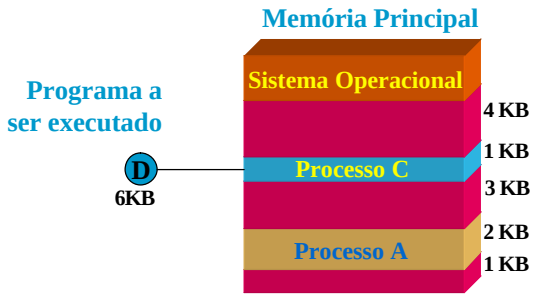
Fila de processos		
Processo	Tamanho	Tempo
P_1	600K	10
P_2	1000K	5
P_3	300K	20
P_4	700K	8
P_5	500K	15

Alocação Particionada Dinâmica - Exemplo



Alocação Particionada Dinâmica

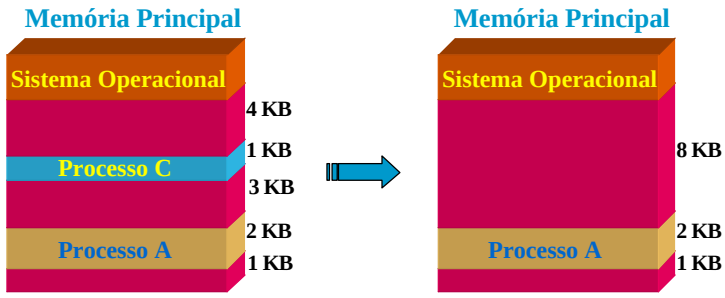
- Neste esquema, a fragmentação começa a ocorrer quando os processos forem terminando e deixando espaços cada vez menores na memória, não permitindo o ingresso de novos processos



- Apesar da existência de 8 Kbytes de memória livre, o processo D não poderá ser carregado

Alocação Particionada Dinâmica

- Soluções para a fragmentação:
  - Primeira: aglutinar espaços livres adjacentes em um único espaço de tamanho maior. No exemplo anterior, caso o processo C termine, teríamos:

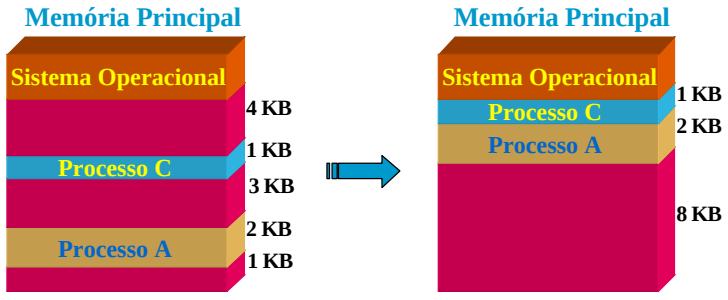


Alocação Particionada Dinâmica

- Como gerenciar?
  - Lista de partições ocupadas
  - Lista de partições livres
  - Ao término do processo, um “buraco” pode ser:
    - adicionado à lista de partições livres (caso a nova partição livre esteja cercada de partições ocupadas)
    - incorporada a uma das partições da lista de partições livres (caso o “buraco” seja adjacente a uma partição livre)
- Não resolve completamente o problema (pode haver buracos pequenos...)

Alocação Particionada Dinâmica

- Segunda: fazer uma relocação de todas as regiões ocupadas (movimento de todos os processos), eliminando todos os espaços entre elas e criando-se uma única área livre contígua. Este método é conhecido como compactação.

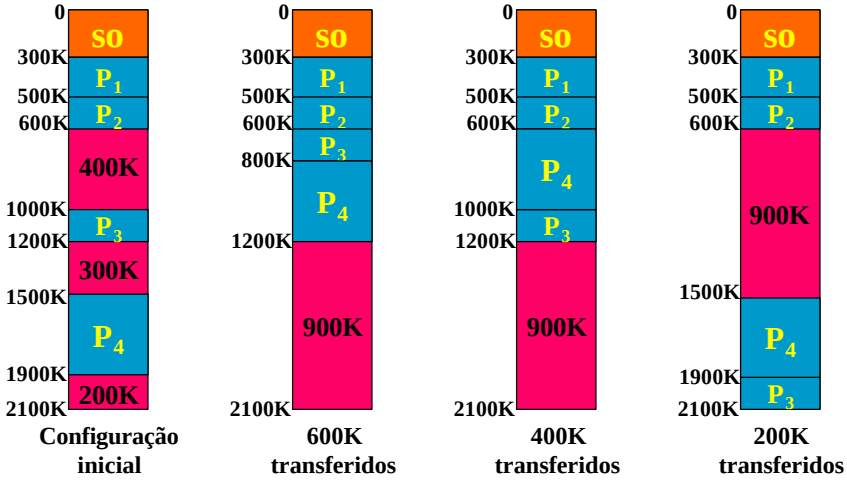


Alocação Particionada Dinâmica

- Problemas:
  - Consumo de recursos (grande quantidade de área em disco e de tempo de processador consumido em sua implementação)
  - Sistema necessita parar (pode ser desastroso para sistemas de tempo real)

Alocação Particionada Dinâmica - Exemplo

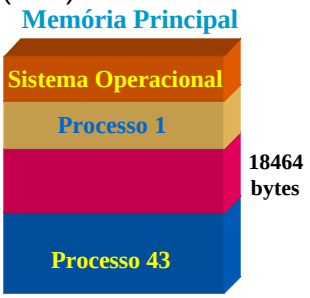
Modos de Compactação



Fragmentação Externa e Interna

- Fragmentação Externa:
  - existe memória suficiente para satisfazer uma requisição, mas ela não é contínua
    - Estatisticamente, de cada N blocos alocados, N/2 blocos são perdidos devido à fragmentação (33%)
- Fragmentação Interna:
  - memória interna a uma partição, mas que não é usada. Ex.:

requisição de 18462 bytes

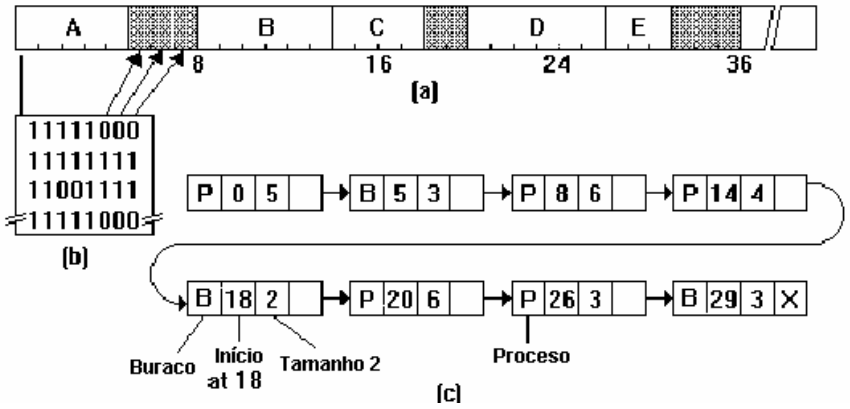


- Uma solução para fragmentação externa é compactação

Gerência de Memória com Mapeamento de Bits

- Memória subdividida em unidades de alocação
- A cada unidade de alocação ➔ 1 bit no mapa
  - Se bit é 0, esta parte da memória está livre
  - Se bit é 1, está ocupada
- Tamanho da unidade de alocação *versus* tamanho do mapa
  - Se a unidade de alocação for de 4 bytes ➔ a parte da memória gasta com este mapa é de apenas 3%
  - Se a unidade for maior ➔ tamanho do mapa é menor, mas aumenta o desperdício de memória (fragmentação interna)
- Desvantagem: para encontrar k unidades de alocação livres ➔ percorrer mapa inteiro procurando k bits iguais a zero ➔ muito lento!

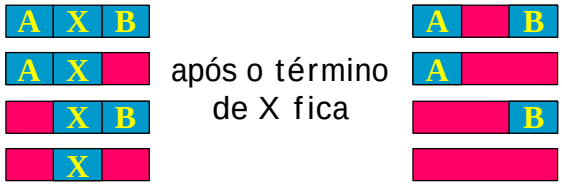
Gerência de Memória



(a) Parte da memória (b) Mapa de bits correspondente
(c) Lista ligada correspondente

Gerência de Memória com Listas Ligadas

- Lista ligada de segmentos alocados e de segmentos livres
- Ordenação por endereços
  - rápida atualização quando do término/remoção da memória
  - Possíveis combinações para a conclusão de um processo X:



  - Com listas duplamente encadeadas a tarefa é ainda mais simples!

Algoritmos de Alocação de Partições

- Estratégias de alocação de memória para escolher o ponto em que deve ser carregado um processo recém criado ou que veio do disco por troca (*swapped in*)
- Listas separadas para processos e buracos:
  - mais rápido na busca por buracos
  - mais lento na liberação de memória
  - variante: os próprios buracos podem ser usados para implementar a lista de partições livres
- *First-fit:*
  - A lista de partições livres é percorrida e a primeira partição suficientemente grande para o pedido é escolhida
  - A lista pode ser mantida em ordem aleatória ou ordenada em ordem crescente de endereço
    - rápido

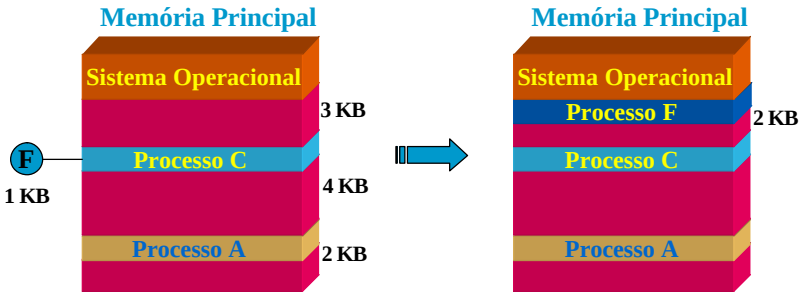
Algoritmos de Alocação de Partições

- *Next-fit:*
  - Idêntico ao *First-fit* com a diferença que a busca se inicia a partir do último ponto em que encontrou um buraco
    - desempenho ligeiramente mais lento que o *First-fit*
- *Best-fit:*
  - A lista de partições livres é percorrida e a menor partição suficientemente grande para o pedido é escolhida
  - A lista é ordenada em ordem crescente de tamanho
    - lento se a lista não estiver ordenada (teria que percorrer a lista inteira)
    - desempenho ruim devido a muitos buracos pequenos (fragmentação externa)

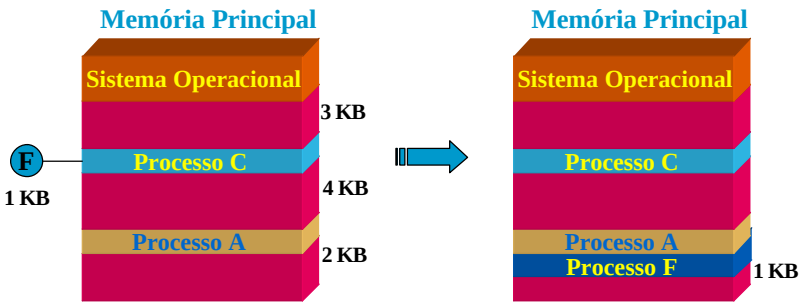
Algoritmos de Alocação de Partições

- *Worst-fit:*
  - A lista de partições livres é percorrida e a maior partição suficientemente grande para o pedido é escolhida
  - A lista é ordenada em ordem decrescente de tamanho
    - diminui um pouco o problema da fragmentação
- *Quick-fit:*
  - Listas separadas para alguns tamanhos mais comuns especificados (por exemplo, uma fila para 2K, outra para 4K, outra para 8K, etc)
    - busca por buraco rápida
    - entretanto, a liberação de memória é lenta (reagrupar buracos e modificá-los de fila)

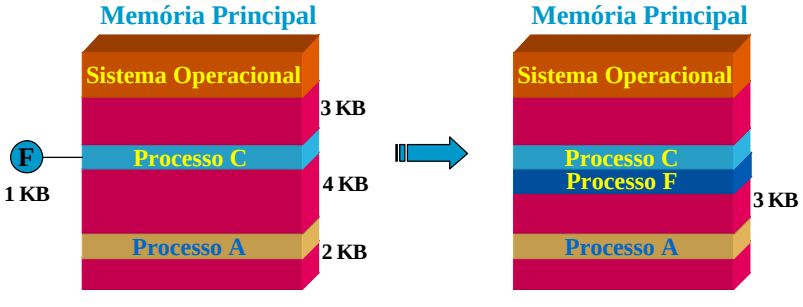
First - fit



Best - fit



Worst - fit

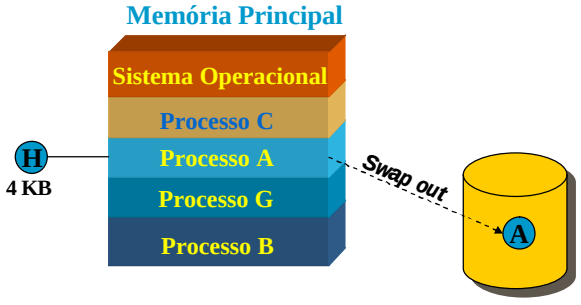


Swapping

- Mesmo com o aumento da eficiência da multiprogramação e da gerência de memória, muitas vezes um programa não podia ser executado por falta de uma partição livre disponível
- Em todos os esquemas apresentados anteriormente, um programa permanecia na memória principal até o final de sua execução, inclusive nos momentos em que esperava por um evento, como uma operação de E/S
- Uma solução é a técnica de *swapping,* cujo objetivo é o de liberar espaço na RAM para que outros processos possam ser carregados e executados

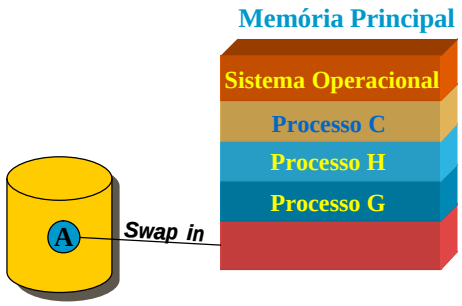
Swapping

- Neste esquema, o SO escolhe um programa residente na memória principal, levando-o da memória para o disco (*swap out*)



Swapping

- Posteriormente, o processo retorna para a memória principal (*swap in*), como se nada tivesse ocorrido



Considerações: Relocação

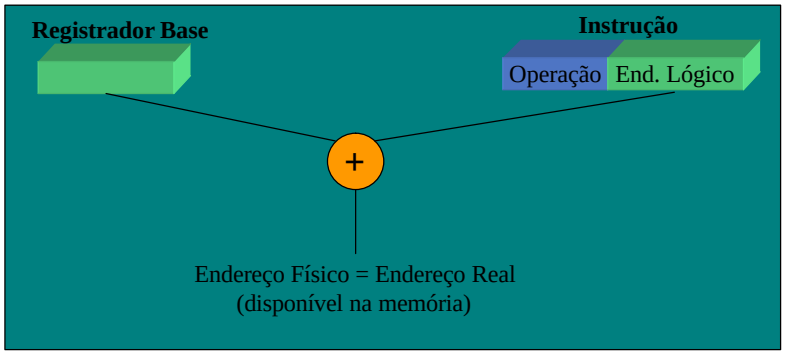
- Mapeamento de espaço lógico em espaço físico de endereçamento, em tempo de carregamento, torna ineficiente recarregar o processo em outra região física diferente da do primeiro carregamento
- Solução:
  - Introdução de *hardware* que faça o mapeamento durante a execução do processo: interceptador de endereços lógicos que soma um valor base aos mesmos, produzindo o endereço físico
  - Não há mais necessidade da tabela de endereços relocáveis

Considerações: Relocação

- Relocação Estática:
  - Mapeamento em tempo de carregamento
- Relocação Dinâmica:
  - Mapeamento durante a execução
- Endereço Lógico = Endereço Virtual
  - É o valor de um deslocamento em relação ao início do programa

Considerações: Relocação

- Exemplo de *hardware* para mapeamento
  - Contém o endereço físico onde foi carregado o início do programa



Considerações: Proteção

- Proteção em relocação dinâmica
  - Consiste em comparar o endereço físico com o comprimento do programa
  - Instrução de chamada ao supervisor
- Separação entre dados e instruções
  - Permite o compartilhamento de instruções por dois ou mais processos
  - Permite duplicar o tamanho máximo do espaço lógico
  - Exige a duplicação do *hardware* necessário para relocação e proteção

Alocação de Espaço de *Swapp*

- Garantir espaço em disco sempre que algum processo é *swapped out*
- Os algoritmos para gerenciar o espaço alocado em disco para *swapping* são os mesmos de gerência da memória principal
- Em alguns sistemas, isto é feito somente no momento da criação do processo e tal espaço permanece reservado
- A única diferença é que a quantidade de espaço reservado no disco deve ser um múltiplo do tamanho dos blocos do disco