GERÊNCIA DE MEMÓRIA

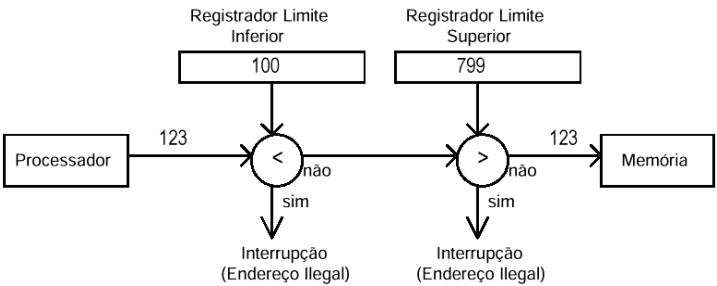
- memória = vetor de palavras (ou *bytes*), cada uma com endereço próprio
- a memória é usada para armazenar os diversos programas em execução, bem como os dados sobre a execução dos programas
- na multiprogramação diversos processos são colocados na memória ao mesmo tempo para que o chaveamento entre eles seja o mais rápido possível
- o SO deve permitir que os processos compartilhem a memória de forma segura e eficiente, usando os recursos disponíveis no *hardware*

MEMÓRIA LÓGICA E FÍSICA (1)

- **memória lógica** é aquela que o processo enxerga, ou seja, aquela que o processo é capaz de acessar
  - os endereços manipulados pelo programa são lógicos
  - em geral capa processo possui uma memória lógica independente da memória lógica dos outros processos
- **memória física** é aquela que é efetivamente acessada pelo circuito integrado de memória
  - dois processos podem ter espaços de endereçamento iguais que correspondem a áreas diferentes do espaço de endereçamento físico
- a unidade de gerência de memória (MMU):
  - provê mecanismos de gerência de memória para o SO
  - converte/mapeia endereços lógicos em físicos

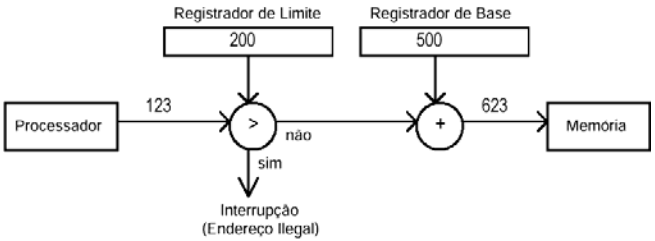
MEMÓRIA LÓGICA E FÍSICA (2)

- quando os endereços lógicos coincidem com os endereços físicos, pode-se implementar proteção de memória usando **dois registradores de limite**
  - os endereços do programa são gerados a partir de 0 e devem ser ajustados por um carregador realocador durante a carga
  - relocação em tempo de carga



MEMÓRIA LÓGICA E FÍSICA (3)

- quando os endereços lógicos não coincidem com os físicos, pode-se trabalhar com um **registrador de limite e outro de base**
  - os endereços são gerados a partir de 0, mas não é necessária nenhuma alteração de endereço
  - o carregador é chamado de absoluto
  - relocação em tempo de execução



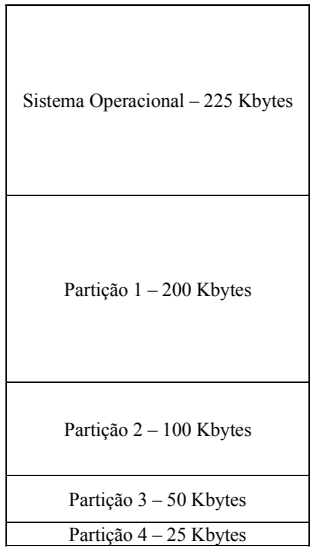
MEMÓRIA LÓGICA E FÍSICA (4)

- o acesso aos registradores de limite e de base é feito através de instruções privilegiadas
- o conteúdo desses registradores faz parte do contexto do processo
- técnicas de gerência de memória
  - partições fixas
  - partições variáveis
  - *swapping*
  - paginação
  - segmentação
  - segmentação paginada

PARTIÇÕES FIXAS (1)

- é simples
- reserva-se uma área de memória para o SO
- o resto da memória é dividido em partições fixas de tamanhos diferentes
- o número de partições/processos em execução é fixo
- novos processos devem aguardar que uma partição de tamanho suficiente fique livre
- há desperdício de memória com:
  - fragmentação interna
  - fragmentação externa

PARTIÇÕES FIXAS (2)



PARTIÇÕES VARIÁVEIS (1)

- as partições são ajustadas dinamicamente às necessidades dos processos
- o tamanho e o número de partições é variável
- o SO mantém uma lista de áreas livres de memória
- quando for necessário alocar memória, percorre-se a lista usando uma das seguintes técnicas:
  - first-fit: a primeira área suficiente
  - best-fit: a que deixa a menor sobra
  - worst-fit: a que deixa a maior sobra
  - circular-fit (ou next-fit): a próxima a partir da última alocação
  - last-fit: a última área

PARTIÇÕES VARIÁVEIS (2)

- a área necessária é alocada e o restante continua livre
- quando um processo termina, a memória que ele ocupava é liberada
- áreas livres adjacentes devem ser agrupadas
- a alocação pode ser:
  - exata:
    - não ocorre fragmentação interna
    - podem surgir áreas de memória pequenas, difíceis de serem alocadas
  - em parágrafos (ou blocos):
    - há uma pequena fragmentação interna
    - facilita o alinhamento de variáveis na memória
    - são necessários menos bits para endereçar uma área de memória

PARTIÇÕES VARIÁVEIS (3)

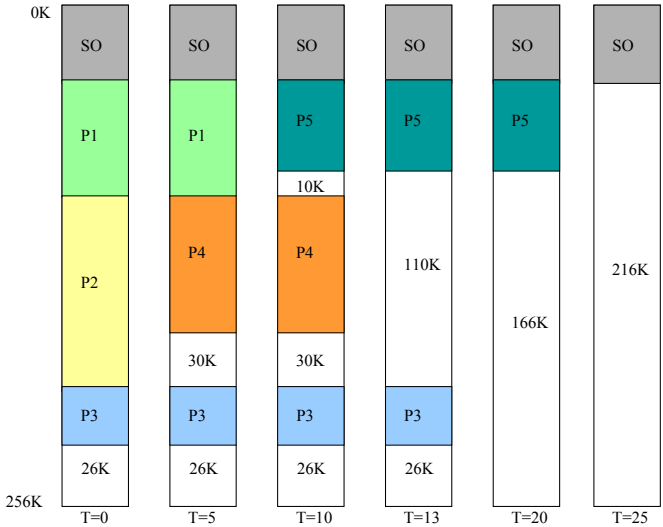
- a lista de áreas livres contém um descritor para cada área com: o endereço, o tamanho e apontadores para lacunas adjacentes
- este descritor pode ser armazenado no início do próprio parágrafo, evitando a necessidade de alocar memória para a lista de áreas livres
- a fragmentação externa é um problema grave
- pode-se compactar a memória, agrupando as áreas livres de memória, o que exige:
  - certo tempo de processamento e
  - algum mecanismo de relocação dinâmica

PARTIÇÕES VARIÁVEIS (4)

- exemplo: os seguintes processos devem ser executados em um sistema com 256Kbytes de memória física (o SO ocupa 40Kbytes da memória física)

Processo	Mem. (K)	Tempo
P1	60	10
P2	100	5
P3	30	20
P4	70	8
P5	50	15

PARTIÇÕES VARIÁVEIS (5)



Processo	Mem. (K)	Tempo
P1	60	10
P2	100	5
P3	30	20
P4	70	8
P5	50	15

EXERCÍCIO (1)

- Considere um sistema cuja gerência de memória é feita através de partições variáveis. Nesse momento, existem as seguintes lacunas (áreas livres): 10K, 4K, 20K, 18K, 7K, 9K, 12K e 13K, nessa ordem. Quais espaços serão ocupados pelas solicitações: 5K, 10K e 6K, nessa ordem, se:
  - first-fit for utilizado?
  - best-fit for utilizado?
  - worst-fit for utilizado?
  - circular-fit for utilizado?
  - last-fit for utilizado?

EXERCÍCIO (2)

- Gerencie uma área de memória de usuários de 640 KB através da utilização de múltiplas partições variáveis para as seguintes requisições:

Processo:	**A**	**B**	**C**	**D**	**E**
Chegada:	0	0	0	11	12
Memória (KB):	320	192	128	224	64
Duração:	10	15	8	10	9

  - use a política first-fit
  - use a política best-fit
  - use a política worst-fit

SWAPPING (1)

- é utilizado quando não é possível manter todos os processos simultaneamente na memória
- um processo sofre *swap-out* quando ele é retirado da lista de prontos, inserido na lista de suspensos e copiado da memória para o disco
- um processo sofre *swap-in* quando ele retorna para a memória
- o *swapping* permite que o SO execute mais processos do que a memória normalmente suportaria

SWAPPING (2)

- o custo para os processos é alto
- o processo deve ficar no disco um um tempo razoável para justificar o *swapping*
- é mais aceitável para processos que executam em segundo plano do que para processos interativos
- pode ser usado tanto com partições fixas quanto com partições variáveis

PAGINAÇÃO (1)

- acaba com a necessidade de colocar o programa em uma área contígua de memória
- elimina a fragmentação externa
- o espaço de endereçamento lógico (memória lógica) é dividido em **páginas** de tamanho fixo
  - cada endereço lógico pode ser dividido em duas partes: número de página e deslocamento
  - todos os bytes de uma mesma página possuem o mesmo número de página, porém, deslocamentos diferentes

PAGINAÇÃO (2)

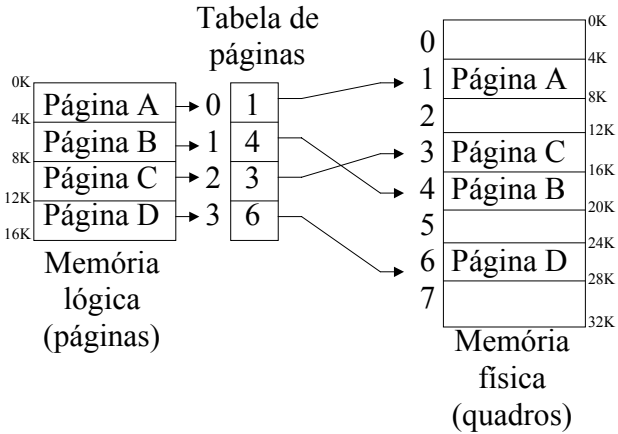
- o espaço de endereçamento físico (memória física) é dividido em **quadros** (*frames* ou páginas físicas), de tamanho igual ao tamanho da página
  - cada endereço físico pode ser dividido em duas partes: número de quadro e deslocamento
- as páginas de um programa podem ser colocadas em quaisquer quadros disponíveis da memória física
- o SO gerencia a carga do programa montando uma **tabela de páginas** que contém o número do quadro onde cada página foi carregada

PAGINAÇÃO (3)

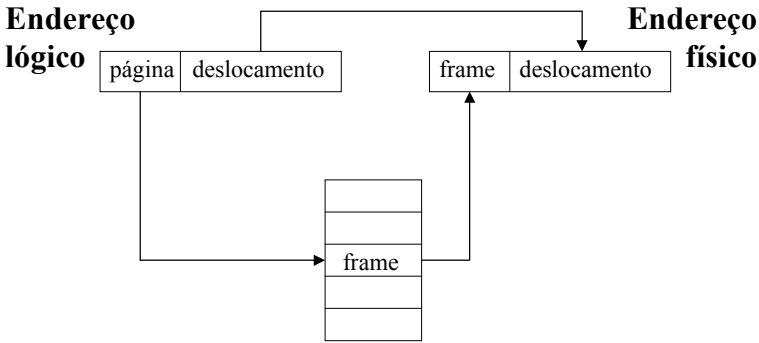
- visão geral:
  - o compilador gera código para os programas como se eles fossem executar de forma contígua na memória
  - o SO carrega as páginas do processo em quaisquer quadros livres e cria a tabela de páginas
  - o hardware converte endereços lógicos em físicos durante a execução com a ajuda da tabela de páginas

PAGINAÇÃO (4)

- exemplo: (páginas de 4Kbytes)
  - qual o endereço físico correspondente ao endereço lógico 7K?



PAGINAÇÃO (5)


Endereço lógico
página
deslocamento
Endereço físico
frame
deslocamento
frame
Tabela de páginas

PAGINAÇÃO (6) - EXERCÍCIOS

- considerando a tabela de páginas apresentada anteriormente, quais os endereços físicos correspondentes aos seguintes endereços lógicos?
  - 1K?
  - 14K?
  - 17K?
- calcule o número de bits para página, quadro e deslocamento.
  - Memória lógica = 512K, Memória física = 512K, Tamanho de página = 8K
  - Tamanho de página = 4K, Número de páginas = 8, Número de quadros = 16

PAGINAÇÃO (7)

- a paginação não apresenta fragmentação externa, apenas uma pequena fragmentação interna, referente à última página do processo
- dimensões típicas:
  - tamanho das páginas: 1Kbyte até 8Kbytes
  - endereçamento lógico: 64Kbytes até vários Gbytes
  - endereçamento físico: geralmente menor do que o lógico
- se as páginas forem maiores:
  - um processo terá menos páginas e a tabela de páginas será menor
  - a leitura do disco será mais eficiente
  - maior fragmentação interna

PAGINAÇÃO (8) - IMPLEMENTAÇÃO

- o controle dos quadros livres pode ser implementado através de:
  - um mapa de bits (pode ser lento com tabelas grandes com alta ocupação)
  - lista encadeada de quadros livres

- a tabela de páginas pode ser implementada:
  - através de registradores dedicados de acesso rápido
  - na própria memória principal
  - usando *Translation Lookaside Buffer* (TLB)

PAGINAÇÃO (9) - IMPLEMENTAÇÃO

- **TP em registradores dedicados de acesso rápido**:
  - só pode ser usado com tabela pequenas
  - é rápido
  - na troca de contexto, os registradores devem ser salvos no descritor de processo
- **TP na memória principal**:
  - usa dois registradores: PTBR (*Page Table Base Register*) e PTLR (*Page Table Limit Register*)
  - suporta tabelas grandes
  - o tempo de acesso à memória é duplicado
  - na troca de contexto, o PTBR e o PTLR devem ser salvos no descritor de processo

PAGINAÇÃO (10) - IMPLEMENTAÇÃO

- **TP usando *Translation Lookaside Buffer* (TLB):**
  - as entradas mais acessadas ficam em uma memória cache
  - quando uma entrada é pesquisada pode ocorrer um acerto (um único acesso à memória) ou uma falha (dois acessos à memória e a informação é atualizada no TLB)
  - o TLB geralmente é implementado usando memória associativa
  - na troca de contexto, o PTBR e o PTLR devem ser salvos no descritor de processo e a cache deve ser esvaziada
  - esquemas alternativos salvam o número do processo nas entradas do TLB

PAGINAÇÃO (11) - EXERCÍCIO

- Qual a taxa efetiva de acesso à memória, quando se tem tempo de acesso à memória associativa de 5ns, tempo de acesso à memória principal de 60ns e taxa de acerto de 80%?

PAGINAÇÃO (12) - PROTEÇÃO

- além do n. do quadro, cada entrada da TP pode conter:
  - bit de válido/inválido
  - bit de apenas leitura (read-only, RO)
  - bit de apenas execução (execute-only, XO)
  - bit de leitura e escrita (read-write, RW)
- o acesso a uma página de forma diferente do que foi definido nos seus bits gera uma interrupção de erro
- a proteção de memória é garantida pela MMU:
  - o SO deve atualizar a tabela de páginas e os registradores PTBR e PTLR em modo monitor
  - a partir daí, qualquer processo que tentar acessar uma posição de memória que não seja sua será abortado

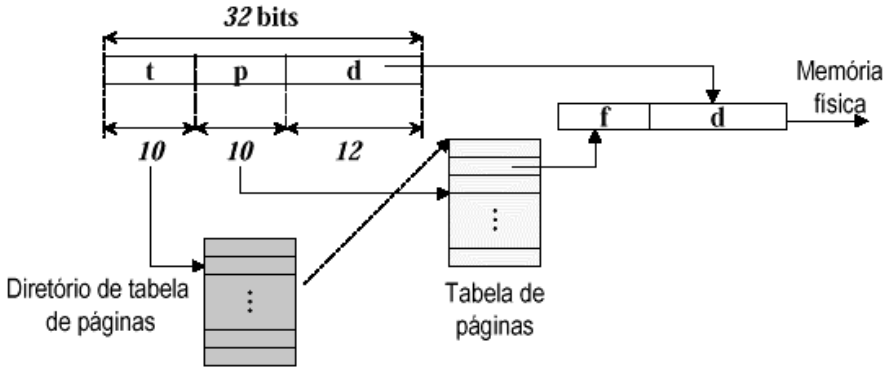
PAGINAÇÃO (13)

- a grande capacidade de endereçamento dos processadores atuais exige uma TP muito grande

- solução: dividir a TP em dois níveis: diretório de TP e tabela de páginas

- o endereço lógico de 32 bits é formado por:
  - 10 bits para definir a entrada do diretório da TP
  - 10 bits para definir o número da página na TP
  - 12 bits de deslocamento.

PAGINAÇÃO (14)



PAGINAÇÃO (15) - EXERCÍCIO

- Considere que os processos da tabela abaixo devem ser executados em um SO com paginação. A memória total é de 64K, o tamanho das páginas é de 4K e o SO ocupa 8K. Mostre como seria a alocação de quadros para cada processo.

Processo	Mem. (K)	T. Cheg	T. Exec
P1	8	0	10
P2	6	1	15
P3	15	2	10
P4	5	3	14
P5	10	15	5

SEGMENTAÇÃO (1)

- a memória lógica geralmente é vista como uma coleção de segmentos de tamanho variável
- quatro segmentos típicos são código, dados estáticos, dados dinâmicos e pilha
- o programador atribui nome aos segmentos
- o compilador transforma esses segmentos em números
- um endereço lógico é formado por um número de **segmento** e por um **deslocamento** dentro do segmento
- no momento da carga, o SO cria uma tabela de segmentos, guardando a posição da memória física e o tamanho de cada segmento

SEGMENTAÇÃO (2)

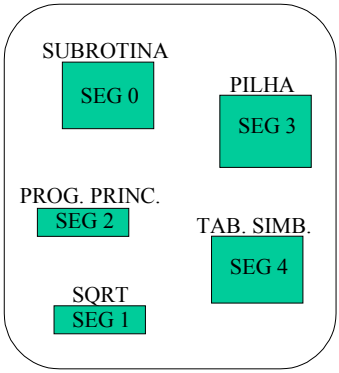
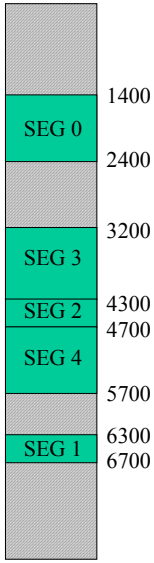
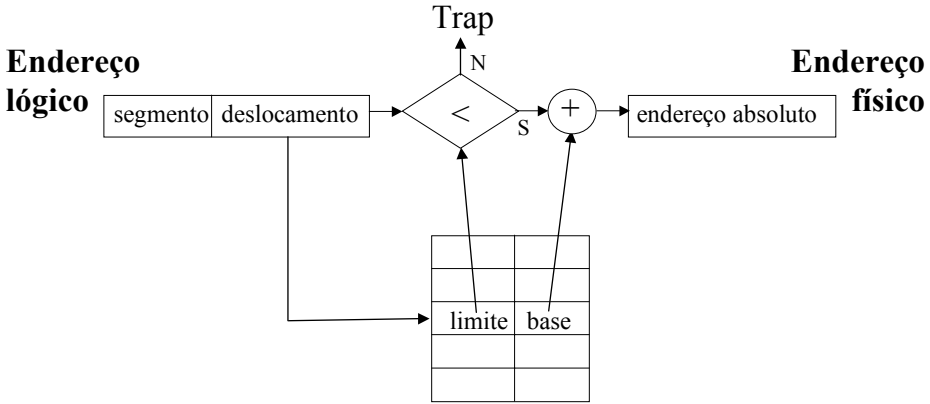


Tabela de Segmentos

	Limite	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



SEGMENTAÇÃO (3)

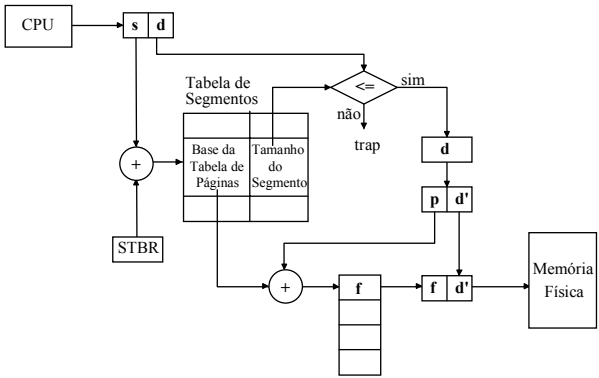


SEGMENTAÇÃO (4)

- a TS é usada pelo hardware para converter endereços lógicos em físicos (implementação igual à da TP)
- também são usados os mesmos bits de proteção da TP
- não apresenta fragmentação interna, mas pode ocorrer fragmentação externa
- a grande vantagem da segmentação está na facilidade de compartilhar memória
- vários processos podem, por exemplo, compartilhar os segmentos de uma biblioteca (marcados como de apenas leitura), evitando-se a necessidade de haver várias cópias do mesmo código na memória

SEGMENTAÇÃO PAGINADA

- atenua a fragmentação externa da segmentação
- os programas são formados por segmentos que por sua vez são divididos em páginas
- apresenta uma pequena fragmentação interna



BIBLIOGRAFIA

- Material elaborado a partir do livro-texto da disciplina:

  OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas Operacionais**. Porto Alegre: Instituto de Informática da UFRGS/Editora Sagra Luzzatto, 2000.