Sistemas Operacionais

Gerência de memória (cap. 9)

Sumário

- Conceitos básicos relacionados à memória
- Swapping
- Alocação contígua
- Paginação
- Segmentação

Conceitos básicos

- Programas precisam ser carregados na memória e associados a um processo para poderem ser executados
- Fila de entrada: processos que estejam no disco esperando para serem carregados na mem.
- Há vários passos a serem seguidos antes dos processos poderem executar

Vinculação de endereços (ligação)

- Programas em linguagem de alto nível (quase) nunca referenciam endereços específicos de memória:
  - C define variáveis globais, locais e dinâmicas, mas não define endereços específicos para elas
  - Dependendo da linguagem, do S.O. e do HW, há diversos momentos onde a associação entre os comandos e a memória pode acontecer

Vinculação de endereços (ligação)

- Tempo de compilação
  - Se houver uma definição fixa sobre localização o compilador pode gerar endereços diretamente
  - Se for preciso mudar os endereços, é preciso recompilar o programa
  - Comum para elementos com endereços definidos pelo HW (p.ex., vetor de interrupções)
  - Ocorre também na vinculação de símbolos definidos e acessados em módulos diferentes de um programa

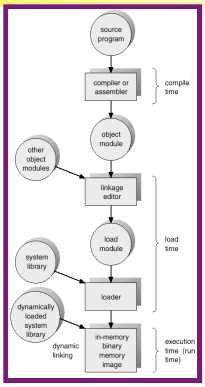
Vinculação de endereços (ligação)

- Tempo de carga:
  - Compilador gera anotações sobre acessos relacionados a certos endereços
  - O carregador (*loader*) lê essas anotações e altera o código do programa carregado para inserir os endereços adequados a cada caso
  - Endereços ganham valor a partir do momento que sabe-se a posição inicial do programa

Vinculação de endereços (ligação)

- Tempo de execução:
  - Em alguns casos a localização de um processo na memória pode mudar em tempo de execução
  - Nesse caso o HW deve prover suporte especial
  - É preciso refazer vínculos eficientemente
    - Mapas de endereços
    - Tabelas de relação
    - Endereçamento relativo

Passos no processamento de um programa até sua execução

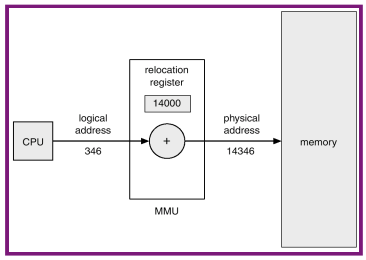


Endereços lógicos x físicos

- End. lógico: gerado pela CPU (virtual)
- End. físico: visto pela unidade de memória (real)
- Mapeamento depende de recursos do HW
  - Unidade de gerência de memória (MMU)
  - Em sistemas simples (sem MMU), virtual=real

Unidade de ger. de mem. (MMU)

- Endereços lógicos são transformados
  - Tabela de tradução
  - Registrador de relocação



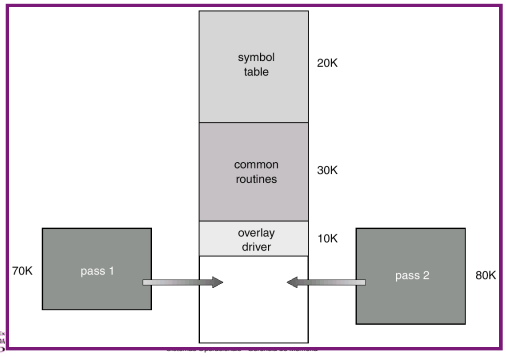
Carga dinâmica

- Permite que partes de um programa só sejam carregados na mem. quando chamadas
  - Melhor utilização da memória
  - Partes não utilizadas podem ser descarregadas
- Não necessariamente exige suporte do S.O., pode ser controlado pelo programa (complexo)
  - Exemplo: *overlays* no DOS

Ligação dinâmica

- Vinculação de endereços postergada até exec.
- Pequeno trecho de código (stub) ocupa o lugar do acesso não vinculado
  - ao ser executado, localiza a rotina correta
  - substitui a si mesmo com o código correto
- Exige suporte do S.O. para controlar o processo
- Particularmente útil para bibliotecas

Exemplo: tradutor de dois passos construído usando *overlays*

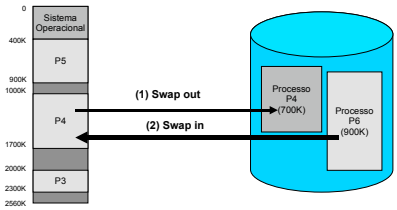


Swapping (permuta de processos)

- Processos inteiros podem ser transferidos temporariamente para memória secundária e posteriormente trazidos de volta
  - P.ex., quando são suspensos pelo escalonador
- Requer uma "memória de retaguarda" (*backing store*) grande o suficiente
- Maior *overhead* é o tempo de transferência
- Presente em diversos S.O. (Unix, Linux, Win*...)

Swapping

- Processos suspensos fora da memória
- Exigem relocação ao serem recarregados

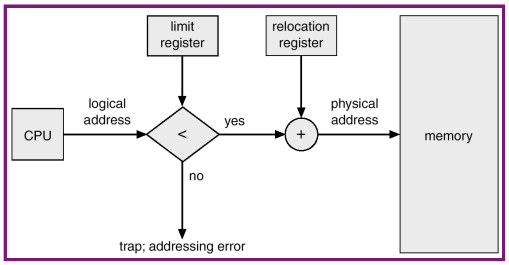


Alocação contígua

- Forma simples de utilização da memória
  - Duas partições:
    - S.O. residente (memória baixa, vetor interrupções)
    - Processo de usuário
  - Para multiprogramação, é desejável que diversos programas estejam na memória
- Alocação contígua: cada processo ocupa um bloco único da memória física

Suporte do hardware para alocação contígua

- É essencial que seja possível fazer relocação dinâmica e proteção entre as áreas de mem.

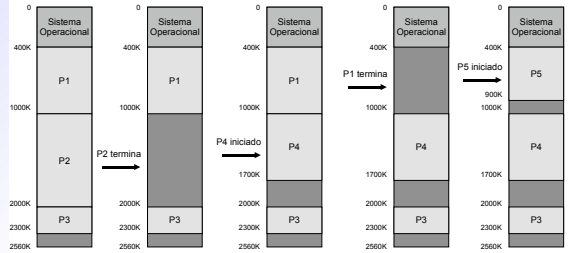


Alocação contígua com múltiplos processos

- Cada processo ocupa um bloco da memória
- Ao terminar, bloco é liberado (buraco)
- Processos que chegam ocupam buracos
- S.O. deve controlar partições e buracos

Alocação contígua com múltiplos processos

- Relocação por tabela ou registrador base



Problema de alocação dinâmica: onde colocar um processo novo?

- *First-fit* (primeiro apto): primeiro que couber
- *Best-fit* (mais apto): o de tamanho mais próximo
- *Worst-fit* (menos apto): sempre o maior buraco

First-fit e *best-fit* se saem melhor que *worst-fit* em termos de velocidade e utilização do espaço.

Fragmentação

- Quebra do espaço em frações não utilizáveis
- Fragmentação externa:
  - Memória não utilizada dividida em muitos buracos pequenos demais para serem úteis
- Fragmentação interna:
  - Limitações na forma como blocos são alocados podem gerar buracos dentro do bloco
  - P.ex., alocação apenas em blocos de 4 KiB

Compactação

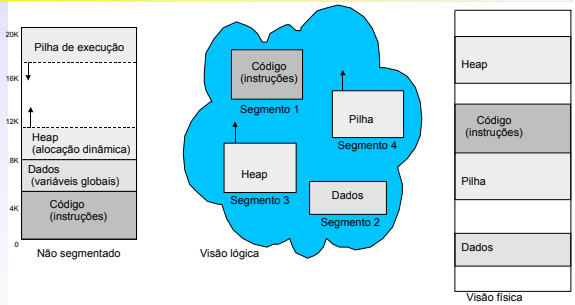
- Solução para a fragmentação externa
  - Mover blocos ocupados para perto uns dos outros
  - Agrupar os buracos em um único bloco maior
  - Só é possível com relocação dinâmica

Outra forma de lidar com fragmentação externa é permitir alocação não contígua (paginação ou segmentação)

Segmentação

- Divisão da memória do processo em unidades
- Baseado na visão lógica do usuário/programador
- Um programa é uma coleção de segmentos de memória independentes (código, dados, pilha, ...)
- Cada um pode ser acessado independentemente
  - Como se fossem diferentes dimensões
- Tornou-se “popular” ao ser adotada pela Intel

Segmentação

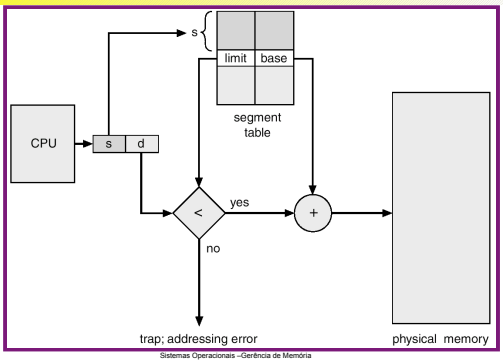


- Mesmos problemas de alocação contígua!

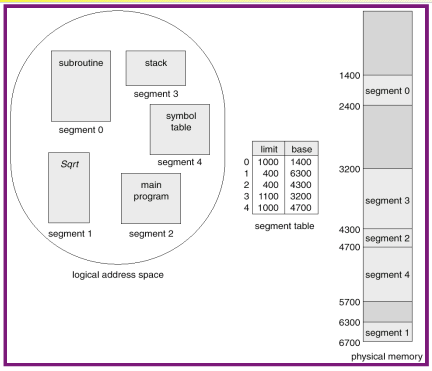
Arquitetura para segmentação

- Endereço lógico é uma dupla: <segm.,offset>
- Tabela de segmentos mapeia segmento em uma área da memória (base + limite)
- Segmentos: pré-definidos (fixos) ou enumeráveis
  - Fixos: *Stack Segment*, *Code Segment*, ... (Intel)
  - Enumeráveis: tabela de segmentos (com limite)
- Segmentos podem ser compartilhados
- Permite controle de acesso refinado
  - p.ex., código != pilha != dados

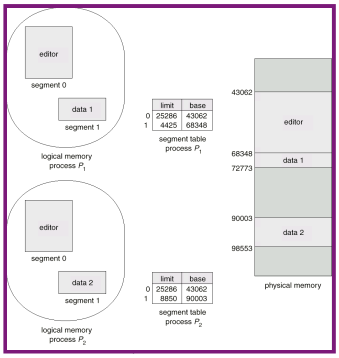
Hardware para segmentação



Segmentação: exemplo



Segmentos compartilhados

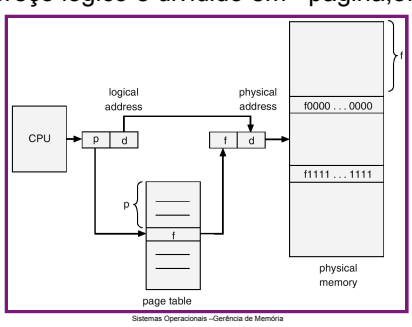


Paginação (*paging*)

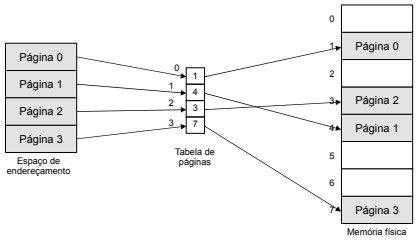
- Memória (lógica/física) é dividida em blocos de tamanho fixo – potências de 2: p.ex., 4 KiB
- Blocos lógicos (páginas) são mapeadas em blocos físicos (quadros) pelo HW
- Endereços lógicos contíguos podem estar em páginas diferentes, em quadros não contíguos
- Quadros vazios são gerenciados
- Programa de n páginas requer n quadros (qq)
- Fragmentação interna (a última página)

Arquitetura de tradução de endereços

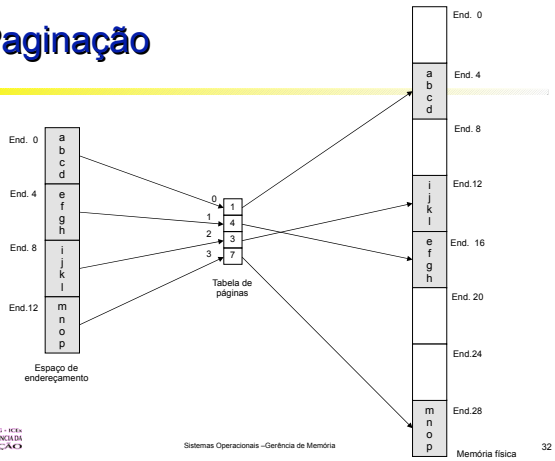
- Endereço lógico é dividido em <página,offset>



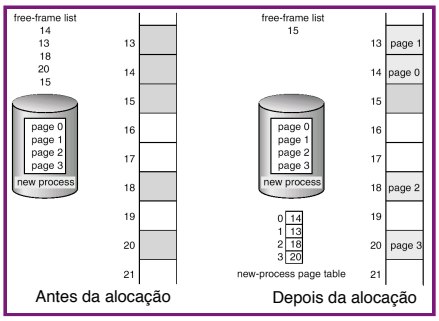
Paginação



Paginação



Gerência de quadros livres



Compartilhamento de páginas

- Compartilhamento de código
  - Deve-se ter atenção com endereços no código
  - Código com endereçamento absoluto
    - Páginas com endereços usados precisam estar nos mesmos endereços lógicos em todos os processos
  - Código com endereçamento relativo
    - Não há restrições de posicionamento

Compartilhamento de páginas

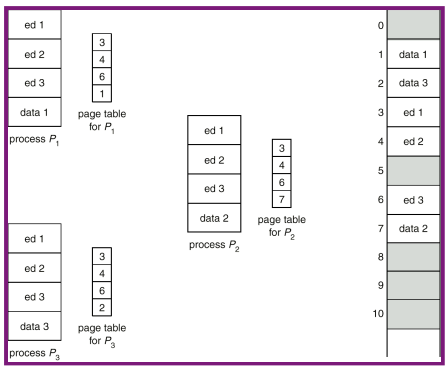
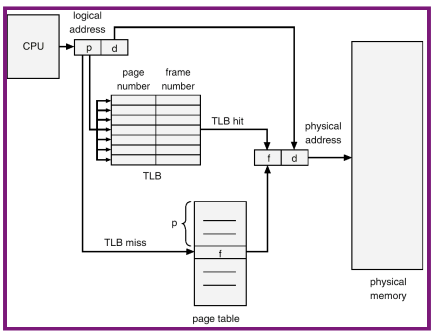


Tabela de páginas

- Tabela fica na memória
- Registrador especial aponta para ela (PTBR)
- Dois acessos à mem. para cada leitura/escrita
  - Um para acessar a tabela
  - Outro para acessar a página
- Hardware especial (cache) evita 1o. acesso
  - *Translation Look-aside Buffer (TLB)*

Hardware de paginação com TLB



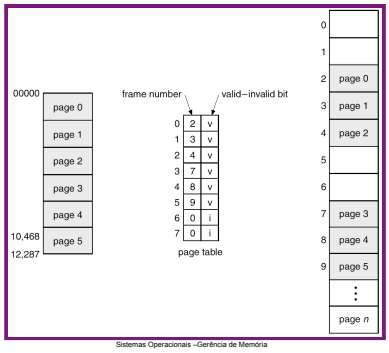
Tempo de acesso efetivo

- Consulta à TLB: ε
- Tempo de um ciclo de memória: t
- Taxa de acerto (hit ratio) na TLB: α
- Tempo de acesso efetivo (TAE)
- TAE = (t + ε) α + (2t + ε) (1 – α) = (2 – α) t + ε

Proteção de memória

- Cada página pode ter bits definindo como cada página pode (ou não) ser usada
  - Apenas leitura/leitura e gravação
  - Executável ou não
  - Página válida/inválida (presente/ausente)

Proteção de memória: válido/inv.



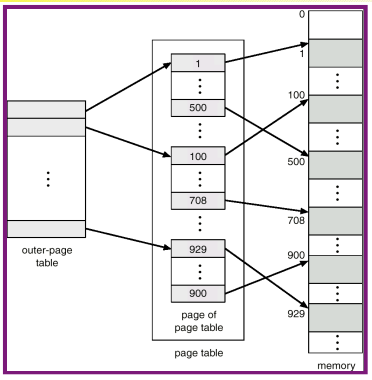
Estruturação da tabela de páginas

- Tabela hierárquica
- Tabela hash (“re-editada” na tradução do livro)
- Tabela invertida

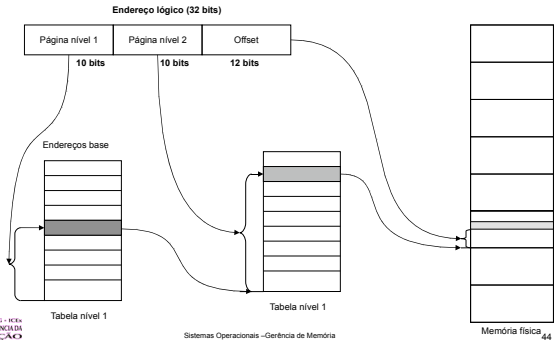
Tabelas hierárquicas (multi-nível)

- Páginas não são muito grandes (4 KiB → 12 bits)
- Se o resto do endereço identificar a página
  - Com endereços de 32 bits, pode haver mais de 1 milhão de páginas (20 Bits)
  - Tabela gigantesca (4 MiB) teria que ser alocada em posições contíguas da memória!
- Solução: quebrar o endereço em várias tabelas
  - P.ex.: para 32 bits, dois níveis de tabelas, cada um com 10 bits

Tabela de páginas de dois níveis



Processo de tradução com 2 níveis



Tabelas de páginas por *hash*

- Comuns em endereços com mais de 32 bits
- Tabelas de mais níveis se tornam proibitivas
- Identificador de página gera um valor de hash
  - Tabela de hash com colisão

Tabelas de páginas por *hash*

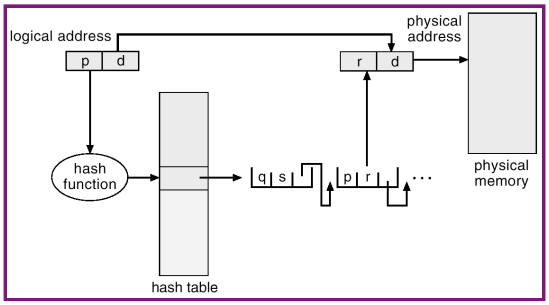
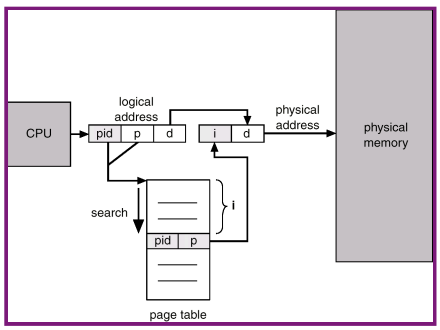


Tabela de páginas invertida

- Uma entrada para cada quadro da mem. física
- Entrada na tabela é o end. lógico que é mapeado para aquele quadro (se não vazio)
- Como vários processos podem ser carregados, entrada deve incluir ident. processo
- Reduz a demanda por memória para a tabela
- Aumenta o tempo de acesso
- Pode-se usar uma tabela de hash para reduzir o custo da busca

Tabela de páginas invertida



Segmentação com paginação

- É possível combinar as duas técnicas e paginar cada segmento
- Usado no Multics e na arquitetura Intel 386
- Flexibilidade x complexidade
- Muitos S.O. para Intel simplesmente ignoram a segmentação

Endereçamento no Intel 80386