Instituto de Informática - UFRGS

Sistemas Operacionais

Implementação de tabelas

Introdução a memória virtual

Aula 15

Introdução

- Sistema operacional deve manter :
  - Paginação:
    - Número de quadros da memória física e a indicação de livre/ocupado
    - Mapeamento de páginas a quadros (por processo ou não, se for invertida)
      - Tabela de páginas
  - Segmentação:
    - Mapa de memória (indicação de áreas livres e ocupadas)
    - Mapeamento de segmentos (por processo)
      - Tabela de segmentos
- Dois problemas:
  - Que tipo de estrutura de dados utilizar ?
  - Onde armazenar essas estruturas ?

Instituto de Informática - UFRGS
A. Carissimi -25-avr.-12

Informação de quadros livres e ocupados: paginação

- Duas formas: listas encadeadas e *bitmap*
  - Bit map é mais natural devido ao tamanho fixo de quadros em memória e o fato ter dois estados (livre ou ocupado)
- Problema: onde armazenar essas estruturas de dados?
  - Memória: desvantagem é o tamanho
  - Disco: desvantagem é o tempo de acesso (desempenho)
- Como o *bitmap* é relativamente pequeno ele pode ser armazenado em memória
  - tam_bitmap = (tam_RAM ÷ tam_quadro ÷ 8) bytes
    - Ex: 1 GB RAM, quadros de 4 KB fornece um bit map de 32 KB

Instituto de Informática - UFRGS
A. Carissimi -25-avr.-12

Informação de segmentos livres e ocupados

- Os segmentos evoluem dinamicamente fornecendo lacunas livres e ocupadas na memória principal (RAM)
  - Lista de lacunas (endereço inicial + tamanho)
  - Método de busca: first fit, best fit, worst fit
- Problema: onde armazenar essa estrutura de dados?
  - Memória: desvantagem pode ser o tamanho
  - Disco: desvantagem é o tempo de acesso (desempenho)
- Lista possui um tamanho variável, pois as lacunas livres surgem em função do uso da memória
  - Podem ser concatenadas
- Normalmente são armazenadas em memória

Instituto de Informática - UFRGS
A. Carissimi -25-avr.-12

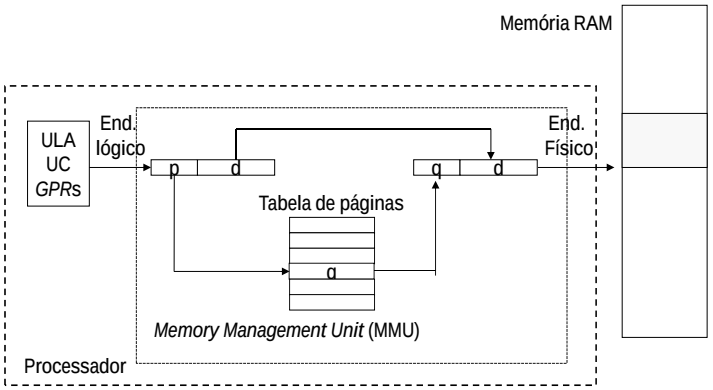
Implementação da tabela de páginas

- Cada processo possui uma tabela de páginas própria
  - Exceção é a tabela de páginas invertida
- Cada entrada da tabela ocupa um certo número de bits
  - log_2 *(tam_ram/tam_quadro)*; bits de acesso, validade, compartilhamento, etc...
- A estrutura de dados natural para implementar a tabela de páginas é um vetor indexado pelo número da página
- Problema: onde armazenar essa estrutura?
  - Registradores no hardware do processador (MMU)
  - Memória RAM
  - Disco
- Compromisso: tamanho versus desempenho

Implementação da tabela de segmentos

- Cada processo possui uma tabela de segmento própria
- Cada entrada da tabela ocupa um certo número de bits
  - Endereço início, limite, bits de acesso, validade, compartilhamento, etc...
- A estrutura de dados natural para implementar a tabela de segmentos é um vetor indexado pelo número da segmento
- Problema: onde armazenar essa estrutura?
  - Registradores no hardware do processador (MMU)
  - Memória RAM
  - Disco

Implementação da tabela de páginas*



*Raciocínio válido para segmentação e segmentação com paginação

Tabela de páginas* em hardware

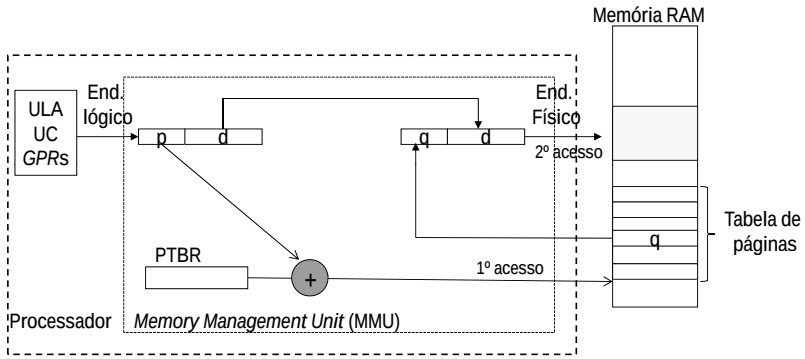
- Implementada como um banco de registradores
- Vantagem:
  - tempo de acesso para tradução de endereço lógico em físico
- Desvantagem:
  - Tamanho da tabela de páginas
    - Número de entradas (páginas) é limitado

*Raciocínio válido para segmentação e segmentação com paginação

Tabela de páginas em memória

- Armazenada na memória RAM
  - Registrador especial para indicar onde ela inicia na memória
    - PTBR (Page Table Base Register)
- Vantagem:
  - Tamanho da tabela de páginas não é limitado por hardware
- Desvantagem:
  - Tempo de acesso
    - Necessário pelo menos dois acessos a RAM para acessar um endereço do espaço do processo

Tabela de páginas* em memória



*Raciocínio válido para segmentação e segmentação com paginação

Translation look-aside buffers (TLBs)

- Solução de compromisso entre implementação via registradores e via memória
- Baseada em uma memória cache especial (TLB) composta por um banco de registradores (memória associativa)
- Idéia é manter a tabela de páginas* em memória com uma cópia parcial da tabela em um banco de registradores (TLB)
  - Página acessada está na TLB (*hit*): similar a solução de registradores
  - Página acessada não está na TLB (*miss*): similar a solução via memória

*Raciocínio válido para segmentação e segmentação com paginação

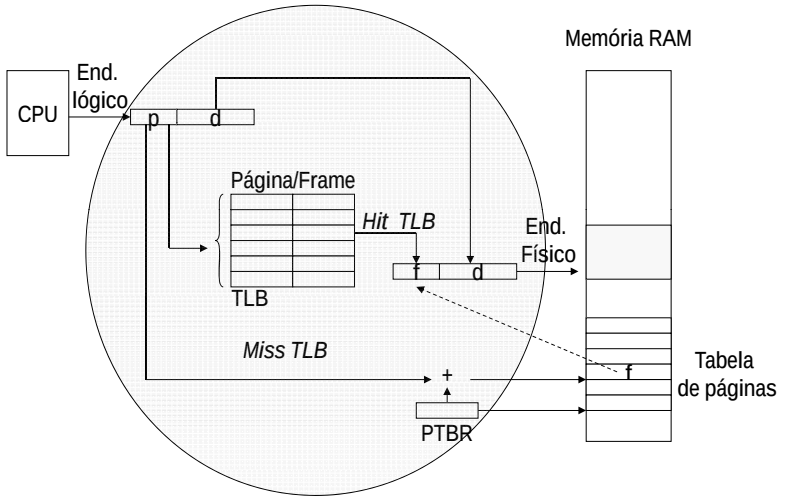
Registradores associativos

- Registradores associativos permitem a busca de valores por conteúdo, não por endereços
  - Pesquisa paralela

Key	value

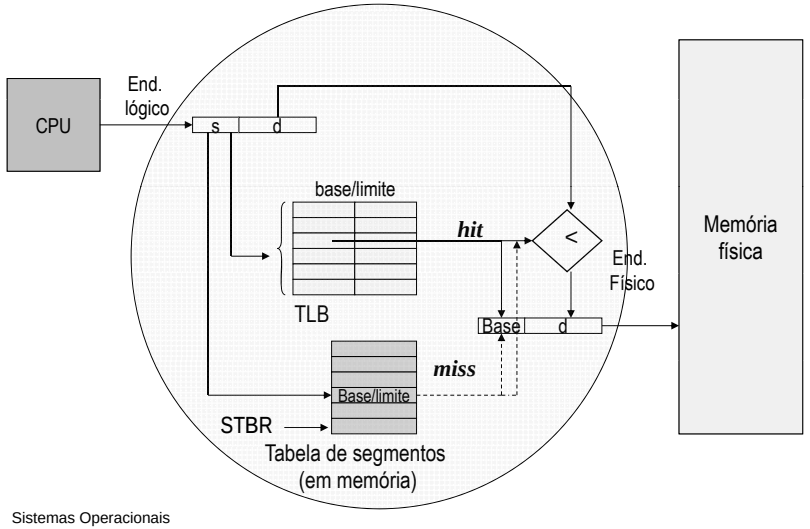
- Funcionamento:
  - Se valor "*key*" está na memória associativa, se obtém valor (*value*).

Implementação da tabela de páginas* via TLB



*Raciocínio válido para segmentação e segmentação com paginação

Implementação da tabela de segmentos via TLB

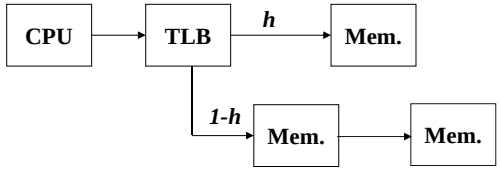


Aspectos relacionados com o uso de TLB: paginação

- Melhora o desempenho no acesso a tabela de páginas
  - Tempo de acesso:1 a 3 ciclos de clock contra 100 a 200 ciclos da memória
- Desvantagem é o seu custo
  - Tamanho limitado (de 8 a 4096 entradas)
    - Processadores atuais tem TLB organizadas em níveis
  - A TLB pertence a MMU e é compartilhada por todos processos
    - Apenas as páginas em uso por um processo necessitam estar na TLB
- Um acesso é feito em duas partes:
  - Se página está presente na TLB (*hit*) a tradução é feita
  - Se página não está presente na TLB (*miss*), consulta a tabela em memória e atualiza entrada na TLB
    - Se paginação multinível acessa os diferentes níveis

Hit ratio (h)

- Probabilidade de qualquer dado referenciado estar na memória, no caso, na TLB
  - Taxa de acerto: *hit ratio*
  - Seu complemento é a taxa de erro: *miss ratio*



$$t_{acesso} = t_{a_{tlb}} + t_{a_{mem}}$$

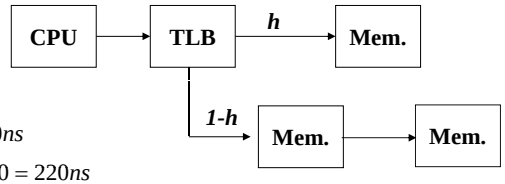
$$t_{acesso} = t_{a_{tlb}} + t_{a_{mem}} + t_{a_{mem}}$$

$$t_{medio} = h \times \left(t_{a_{tlb}} + t_{a_{mem}}\right) + \left(1-h\right) \times \left[t_{a_{tlb}} + t_{a_{mem}} + t_{a_{mem}}\right]$$

Exemplo: influência do *hit ratio* no desempenho

$t_{acesso\_tlb} = 20ns$

$t_{acesso\_mem} = 100ns$



$t_{acesso(hit)} = 20 + 100 = 120ns$

$t_{acesso(miss)} = 20 + 100 + 100 = 220ns$

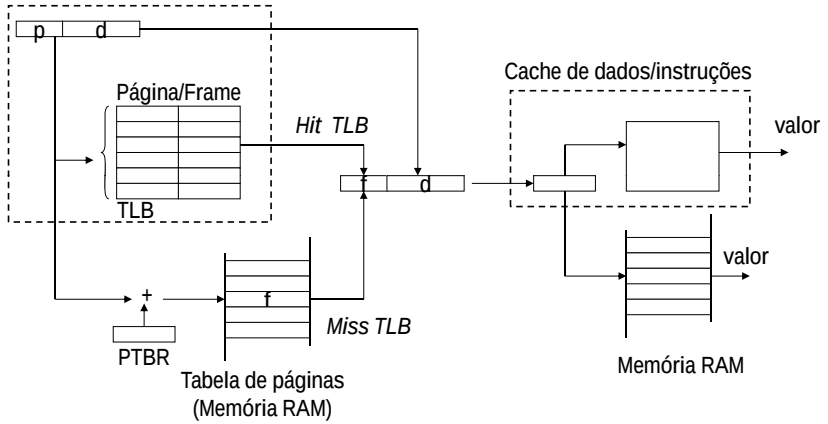
$hit\_ratio = 0.80$

$t_{medio} = 0.80 \times (120) + 0.20 \times (220) = 140ns$

$hit\_ratio = 0.98$

$t_{medio} = 0.98 \times (120) + 0.02 \times (220) = 122ns$

Atenção: TLB não é cache de dados e/ou instruções



Leituras complementares

- A. Tanenbaum. *Sistemas Operacionais Modernos* (3ª edição), Pearson Brasil, 2010.
  - Capítulo 3: seção 3.6
- A. Silberchatz, P. Galvin; *Sistemas Operacionais*. (7ª edição). Campus, 2008.
  - Capítulo 8 (seção 8.4.2)
- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Bookman 4ª edição, 2010
  - Capítulo 6 e capítulo 6 (seções 6.5 e 6.6)

Sistemas Operacionais

Introdução a memória virtual

(Aula 15 – cont.)

Memória real versus memória virtual

- Memória real é a memória fisicamente instalada no sistema (RAM)
- Processos executam na memória real
  - Se não há memória disponível o processo não é criado
- Memória real impõem limitações:
  - Processo não pode ser maior que a memória física
  - O somatório do espaço necessário a *n* processos não pode exceder o tamanho da memória física
    - Limita o grau de multiprogramação
- Memória virtual surge como solução

Memória virtual

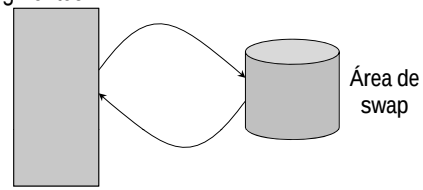
- Expande a memória do sistema através de um espaço em disco
  - Arquivo de swap ou partição de swap
- O sistema operacional tem a ilusão de possuir uma memória maior que a fisicamente instalada
  - Área total disponível = RAM + tamanho do swap
  - Se não há memória (virtual) disponível não se pode criar processo

Vantagens e desvantagens de memória virtual

- Dimensionamento da memória virtual depende do que se deseja do sistema
- Vantagens:
  - Capacidade de executar programas maiores que o tamanho da RAM
  - Aumento do grau de multiprogramação
- Desvantagens:
  - Desempenho: acesso a área de swap é mais lenta que acesso a RAM
- Aumento da memória física é sempre a melhor solução!!

Memória virtual: necessidades para implementação

- Deve haver suporte a hardware para paginação, segmentação ou segmentação com paginação
- Sistema operacional deve controlar o fluxo de páginas/segmentos entre a memória secundária (disco) e a memória principal
- Necessidade de gerenciar:
  - Áreas livres e ocupadas
  - Mapeamento da memória lógica em memória física
  - Substituição de páginas/segmentos



Paginação por demanda

- Objetivo é utilizar memória física de forma eficiente
  - Carregar para a memória apenas os trechos do processo que são necessários a sua execução em determinado momento
    - Beneficia da localidade de referência
- O trecho do programa pode ser um segmento ou uma página
  - Segmentação por demanda ou paginação por demanda
- Paginação é método preferido porque simplifica a alocação
  - Qualquer página pode ser carregada em qualquer frame disponível

Swapping implica em movimentar todo o processo, o termo correto seria *pagging*, porém o termo *swap* "pegou"

Princípio da localidade de referência

- Base de funcionamento da memória virtual
- Na execução de um processo existe uma probabilidade que acessos a instruções e a dados sejam limitados a um trecho
  - Exemplos:
    - Execução da instrução i+1 segue a execução da instrução i
    - Laços `for, while, do-while`
    - Acessos a elementos de vetores
- Em um determinado período de tempo apenas esses trechos do processo necessitam estar em memória
- Possibilidade de "prever" os trechos necessários a sequência de execução

Vantagens da paginação por demanda

- Reduz operações de E/S
  - Não é necessário carregar todo o processo em memória
  - Operações de carga são limitadas a transferência de páginas
- Reduz a quantidade de memória utilizada por processo
  - Aumenta o grau de multiprogramação

Área de swap

- Pode ser:
  - Partição específica em um disco ou um disco físico específico
    - Formato simples: blocos iguais ao tamanho da página
  - Arquivo em um sistema de arquivos (solução Windows)
  - Flexibilidade versus desempenho
- Procedimento simplificado
  - Imagem completa do processo é criado na área de swap
    - Cada página corresponde a um bloco
  - Efetua page-in (swap→memória) e page-out (memória →swap) com base na origem a imagem no swap e o deslocamento é a própria página
    - Desvantagem: tamanho da imagem e desperdício
    - Vantagem: mapeamento direto de página para bloco na área de swap

Área de swap: melhorias (cont.)

- Copiar na área de swap apenas o que é estático (código e globais)
  - Pilha e heap são alocados a medida que são usados
  - Problema: mapeamento não é mais 1:1
    - Tabela para indicar qual bloco do swap corresponde a qual página
- Páginas de códigos podem ser lidas diretamente do executável
- Preferir *page-out* de página não modificadas
- Operação de *swapping* é sobreposta ao processamento
  - E/S realizada através de DMA

Leituras complementares

- A. Tanenbaum. *Sistemas Operacionais Modernos* (3ª edição), Pearson Brasil, 2010.
  - Capítulo 3: (seções 3.4, 3.7)
- A. Silberchatz, P. Galvin; *Sistemas Operacionais*. (7ª edição). Campus, 2008.
  - Capítulo 9 (seções 9.1 e 9.2)
- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Bookman 4ª edição, 2010
  - Capítulo 7 (seção 7.1)