Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 11 - TLB e Paginação Multi-nível

Aula de Hoje

Na aula anterior

Vimos os conceitos básicos de paginação.

- Espaço de endereçamento é quebrado em tamanho fixo.
- Localização das tabelas na memória física necessita de uma tabela.
- Tabela gasta memória e adiciona custos de processamento:
  - Novas instruções para calcular a posição.
  - **Fetch da própria tabela na memória**.

Na aula de hoje, veremos como melhorar o processo de acesso à tabela de páginas.

Aula de Hoje

Como agilizar a paginação

S.O. sozinho tem limites sobre o que ele pode fazer para melhorar o desempenho.

Por isso, é necessário hardware novo: O Translation Lookaside Buffer, ou **TLB**.

O TLB é basicamente um cache de traduções de endereços comuns. Vocês podem já ter visto algo a respeito em OC/AC.

A cada referência a um endereço virtual, primeiro o sistema checa no TLB para procurar a tradução, sem precisar ir até a RAM buscar a tabela de páginas.

Algoritmo Básico do TLB

Assumindo uma tabela de páginas linear (vetor) e TLB via hardware:

1. Extrair o número da página do endereço e checar se a sua tradução está na TLB.
2. Se está, temos um TLB hit, e podemos extrair o número do quadro onde está a página e concatenar com o offset.
3. Se não está, ocorre um TLB miss, e as informações de tradução devem ser carregadas.
    1. Acessar a tabela de páginas na RAM para encontrar a tradução.
    2. Verificar se a página é válida e acessível.
    3. Atualizar o TLB.
    4. Repetir a instrução.

TLB

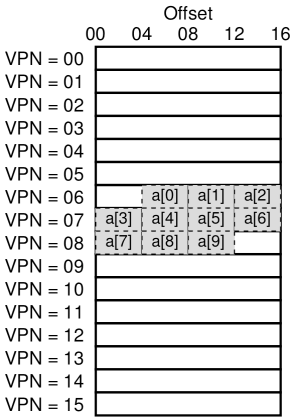
O TLB, como qualquer cache, é baseado na idéia que os acertos de cache serão mais comuns que os erros.

Cada miss tem um alto custo, adicionando no mínimo um acesso extra à RAM para recuperar a entrada da tabela de páginas.

Como toda cache, o objetivo é minimizar os miss.

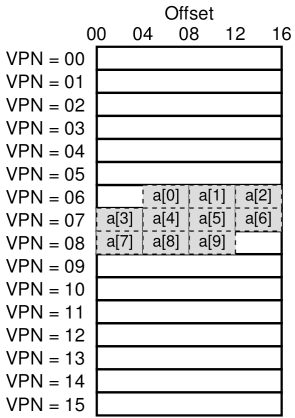
Como? Com os mesmos princípios que já conhecemos ;-)

Exemplo: Acessar um vetor



- Endereçamento virtual de 8-bit, com páginas de 16 bytes (4 bits de VPN e 4 de offset).
- Vetor `a` com 10 Inteiros de 4 bytes, começando no endereço virtual 100.
- Primeiro elemento começa em VPN=6,offset=4. Vetor continua nas duas páginas seguintes.

Exemplo: Acessar um vetor

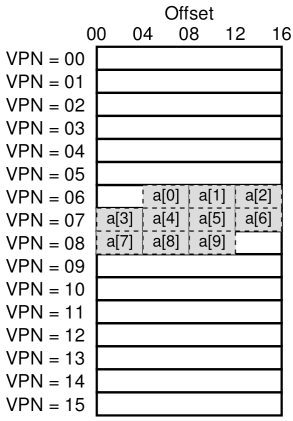



```
int soma;
for(i=0;i<10;i++)
        soma+=a[i];
```

(ignorem por enquanto os acessos à `i` e a `soma`).

Primeiro acesso à `a[0]` será um TLB miss. E o a `a[1]`?

Exemplo: Acessar um vetor

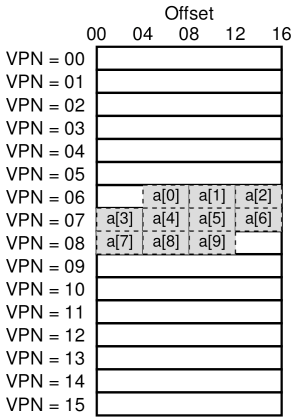



```
int soma;
for(i=0;i<10;i++)
        soma+=a[i];
```

(ignorem por enquanto os acessos à `i` e a `soma`).

Primeiro acesso à `a[0]` será um TLB miss. E o a `a[1]`?E os próximos miss?

Exemplo: Acessar um vetor




```
int soma;
for(i=0;i<10;i++)
        soma+=a[i];
```

(ignorem por enquanto os acessos à `i` e a `soma`).

Primeiro acesso à `a[0]` será um TLB miss. E o a `a[1]`?E os próximos miss?

Nos dez acessos: **miss**, hit, hit, **miss**, hit, hit, hit, **miss**, hit, hit.
hit rate: 70%

Exemplo

Esta taxa de hit se dá devido à localidade espacial.

Tamanho da página também afeta. Por que?

Exemplo

Esta taxa de hit se dá devido à localidade espacial.

Tamanho da página também afeta. Por que?

Além disso, se o programa logo após este for acessar este vetor de novo, podemos ter uma taxa de hit ainda maior.

Como as três páginas já estão na TLB, não teremos os miss iniciais.

Conceito de localidade temporal.

O que acontece durante um miss?

Em tempos de arquiteturas **CISC**, o hardware cuidava dos miss. Isso foi até o x86

Para isto, o hardware é obrigado a saber exatamente onde a tabela de páginas esta na memória, além do *formato* exato destas informações.

Em arquiteturas **RISC**, o hardware simplesmente levanta uma excessão para colocar as informações no TLB.

Miss em RISC

O handler de interrupções (em modo kernel) vai então procurar pela tradução e adicionar ela no TLB via instrução privilegiada.

Alguns detalhes:

- O retorno desta excessão deve ser diferente: voltar para a mesma instrução ao invés da anterior.
- Tomar cuidado com cadeias de miss (se memória do handler não estiver na TLB por exemplo).

Miss por software tem duas vantagens: flexibilidade e simplicidade.

Conteúdo da TLB

TLB normal tem entre 32 a 128 entradas totalmente associativas. Lembram?

Conteúdo da TLB

TLB normal tem entre 32 a 128 entradas totalmente associativas. Lembram?Significa que as informações de uma página podem estar em qualquer lugar da cache, e o hardware busca nela inteira em paralelo. Uma entrada de TLB pode ser assim:

VPN | PFN | Outros Bits

VPN é o número da página virtual, PFN é o número do quadro físico da memória onde a página está, e os outros bits são flags, como um bit de validade na cache e proteção.

Aula de Hoje

Trocas de Contexto

Temos um problema: a TLB contém traduções de páginas do processo atual, sendo inúteis para outros processos.

S.O. tem que tomar cuidado para que um processo não use memória dos outros.

O que fazer?

Trocas de Contexto

Temos um problema: a TLB contém traduções de páginas do processo atual, sendo inúteis para outros processos.

S.O. tem que tomar cuidado para que um processo não use memória dos outros.

O que fazer?

Flush: limpar a cache.

Trocas de Contexto

Temos um problema: a TLB contém traduções de páginas do processo atual, sendo inúteis para outros processos.

S.O. tem que tomar cuidado para que um processo não use memória dos outros.

O que fazer?

Flush: limpar a cache.

Contudo, tem um custo: todos os primeiros acessos do novo processo serão miss.

Trocas de Contexto

Uma saída de alguns sistemas é o compartilhamento de TLBs entre trocas de contexto.

Ou seja, não limpar o TLB, mas sim guardar um identificador do processo dono da página (com uma versão resumida do PID).

Dessa forma, a TLB pode guardar informações de vários processos ao mesmo tempo, podendo haver a possibilidade de quando o processo voltar algumas páginas dele ainda estarem presentes.

Políticas de substituição de TLB

TLB é crítico na performance, então a política tem que ser extremamente simples para evitar perda de desempenho.

Contudo, o custo de um miss é alto.

Veremos melhor políticas de substituição em Swap, daqui a duas aulas.

Dois exemplos: LRU e Random.

Ué? Random? Por que?

Aula de Hoje

Relembrando

Se a tabela de paginação for linear, num espaço de 32 bits, com páginas de 4K e 4bytes por entrada na tabela, a tabela de cada processo ficaria em torno de **4 MEGAS**.

Por processo!

Como fazer para ter tabelas menores?

Solução simples: Aumentar o tamanho das páginas

Um jeito simples de reduzir a tabela é aumentar o tamanho das páginas.

No espaço de 32bits, se tivermos páginas de 16KB, isso daria um offset maior (**14bits**) e uma VPN menor (**18**), que levaria a 2^{18} entradas na tabela, o que x4bytes por entrada daria 1MB por processo.

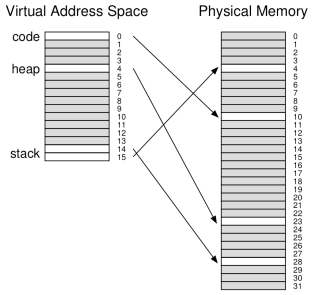
Problema: Fragmentação interna.

Maior parte dos sistemas usam páginas menores, como 4K e 8K.

Solução híbrida: Segmentação Paginada

Combinação entre paginação e segmentação.

Imaginem um pequeno processo com 16K de endereçamento virtual e páginas de 1K, que tem pouca memória usada no heap/stack:



PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
23	1	rw-	1	1
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
28	1	rw-	1	1
4	1	rw-	1	1

Segmentação paginada

Ao invés de uma única tabela para o espaço todo, por que não uma para cada segmento?

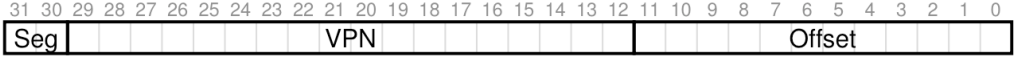
Lembrem que na segmentação haviam registradores **base** e **bound** para cada segmento. Eles agora guardarão os endereços das tabelas de página dos segmentos.

Base indica o começo da tabela do segmento, bound quantas páginas tem.

Exemplo

Num espaço virtual de 32bits com páginas de 4KB e espaço quebrado em quatro segmentos (só usaremos 3 aqui): code, heap, stack.

Para determinar o segmento, podemos usar os dois primeiros bits do endereço:



A partir daí, a tradução é muito similar à da paginação, com a adição do registrador base às contas.

Vantagens e desvantagens

Vantagem: Devido ao uso do registrador Bound, não é necessário alocar a tabela toda de uma vez.

Desvantagens: Segmentação obriga o crescimento dos endereços a seguir uma ordem pre estabelecida.

Também pode trazer de volta a fragmentação externa, especialmente com as tabelas em memória tendo tamanhos arbitrários.

Paginação Multi-Nível

Uma das vantagens da segmentação paginada é que ela não gasta memória com tabela de páginas não alocadas.

Outra abordagem que resolve este problema é a paginação com tabela Multi-Nível.

Nela, a tabela linear é transformada em uma espécie de árvore.

Abordagem mais comum em sistemas modernos.

Paginação multi-nível

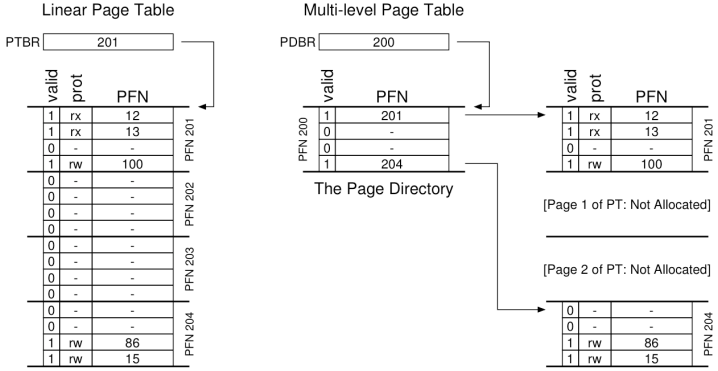
Abordagem: Dividir a própria tabela de tradução páginas em pedaços do tamanho de uma página. Se nenhuma das páginas de um pedaço é valida, então este pedaço também não é alocado.

Ou seja, só aloca os pedaços da tabela que estão sendo usados.

E aloca via paginação, resolvendo fragmentação externa.

Paginação Multi-Nível

Para fazer a PMN, se usa uma nova estrutura, o diretório de páginas:



Vantagens

- Só aloca tabela proporcionalmente ao consumo de memória do processo.
- Usar páginas (novamente) facilita o gerenciamento de espaço, pois basata escolher uma página livre qualquer quando precisa crescer, devido à camada de indireçãoprovida pelo diretório.

Contudo, nem tudo é vantagem: cada miss significa dois acessos à memória: um no diretório e outro pra parte da tabela.

É um exemplo de trade-off entre tempo e espaço.

Outro problema é o aumento da complexidade.

Finalizando

Existem outras abordagens, como Paginação com mais de dois níveis.

Contudo, deixaremos isto para o interesse de cada um ir atrás.

E Não percam na próxima aula: SWAP (ou o que antigamente chamavam de memória virtual).