Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 12 - Memória Virtual em disco

Aula de Hoje

Introdução

Um dos problemas que sistemas de computação sempre tiveram é não ter memória o suficiente para tudo que se quer fazer.

Conforme a RAM cresce, cresce também o escopo do que era feito pelos processos.

Espaço de endereçamento muito grande, diversos processos. Como fazer tudo caber na memória?

Hierarquia de memória.

Introdução

Sistemas Operacionais modernos adicionam mais uma camada de virtualização da memória física através do **Swap** (troca).

Ilusão de que a memória RAM é maior do que é, copiando pedaços de memória "não necessários"para armazenamento secundário.

Esse processo é essencial para sistemas multiprogramados.

Aula de Hoje

Aula de Hoje

Espaço de Swap

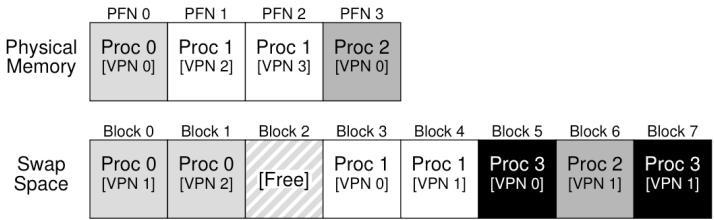
Primeiro passo para Swap é reservar espaço em armazenamento secundário, o Swap Space ou espaço de swap.

Neste espaço serão trocadas páginas da memória principal para o disco e do disco para a memória.

Logo, este espaço é formado por N páginas no disco, cada uma com um endereço próprio. O tamanho deste espaço determina quanta memória extra o sistema dispõe

No Linux, se usa uma partição especial para Swap, enquanto no Windows é gerado um arquivo especial chamado pagefile.sys.

Espaço de Swap



Em alguns sistemas, além da área de Swap, executáveis também podem ser virtualizados, com o sistema recarregando código diretamente do arquivo executável do processo rodando.

Aula de Hoje

Bit de Presença

Além do espaço no disco, outros mecanismos devem ser implementados para termos um Swap funcional.

Vamos relembrar o que ocorre num acesso a memória:

- O processo utiliza um endereço virtual, e o Hardware é encarregado de transformar este endereço em real.
- Para isto, ele procura pela página virtual na TLB, e se foi **hit**, já era.
- Se não está na TLB (miss), o hardware procura pela entrada correspondente na tabela de páginas do processo, verifica se ela é válida e adiciona ela ao TLB, para então repetir o acesso.

Contudo, para utilizar swap (extrair/colocar páginas da mémoria no disco), precisaremos adicionar mais mecanismos nesse processo.

Bit de Presença

Agora, ao procurar pela página correspondente na memória, pode ser que aquela página não esteja mais presente na memória física.

- Ou seja, foi realocada para o disco (swap).

É necessária uma nova informação: O bit de presença. Se ele for 1, a página está na memória e está tudo bem.

Se for 0, então a página não está na memória (pode estar no disco), e ocorre um **page fault**.

Aula de Hoje

Page Fault

Normalmente, mesmo em sistemas com TLB totalmente via Hardware, as page faults são resolvidas pelo sistema operacional.

- Page Faults são lentas (pelo acesso ao disco), então overhead do S.O. é desprezível no tempo total do processo.
- Para ser via hardware, a TLB teria que ter acesso ao hardware de E/S e conhecer o espaço de endereçamento do disco, e isso não é trivial.

Após um page fault, o page-fault handler fica responsável por trazer a página referenciada de volta para a memória principal e atualizar a tabela de páginas do processo.

Page Fault

Informação sobre onde a página procurada está no disco deve estar disponível para o S.O. poder achar a mesma.

Onde? Na própria tabela de páginas. Ao invés de guardar um endereço na memória RAM, quando o bit de presença é zero, ela guarda o endereço da página no disco.

Após a página ser localizada na memória e trazida para um quadro livre na RAM, o S.O. atualiza o endereço da página na tabela e seta o bit de presença pra 1.

Ele pode (ou não) atualizar a TLB com o novo endereço, ou simplesmente deixar acontecer outro TLB miss e ele se atualizar sozinho.

Importante: Durante o acesso da página no disco, o processo fica **bloqueado**. O que isto significa para o sistema?

Page Fault

E se a memória está cheia?

Page Fault

E se a memória está cheia?

S.O. precisa liberar um quadro na memória, tirando uma página da memória e gravando-a no disco.

Como escolher? Essa é uma política de swap, e veremos logo mais ;-)

No linux, `/usr/bin/time -v <processo>` mostra varias estatisticas, incluindo as pagefaults

Fluxo de um page fault

```
1  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2  (Success, TlbEntry) = TLB_Lookup(VPN)
3  if (Success == True)   // TLB Hit
4      if (CanAccess(TlbEntry.ProtectBits) == True)
5          Offset   = VirtualAddress & OFFSET_MASK
6          PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7          Register = AccessMemory(PhysAddr)
8      else
9          RaiseException(PROTECTION_FAULT)
10 else                  // TLB Miss
11     PTEAddr = PTBR + (VPN * sizeof(PTE))
12     PTE = AccessMemory(PTEAddr)
13     if (PTE.Valid == False)
14         RaiseException(SEGMENTATION_FAULT)
15     else
16         if (CanAccess(PTE.ProtectBits) == False)
17             RaiseException(PROTECTION_FAULT)
18         else if (PTE.Present == True)
19             // assuming hardware-managed TLB
20             TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
21             RetryInstruction()
22         else if (PTE.Present == False)
23             RaiseException(PAGE_FAULT)
```

```
1  PFN = FindFreePhysicalPage()
2  if (PFN == -1)              // no free page found
3      PFN = EvictPage()       // run replacement algorithm
4  DiskRead(PTE.DiskAddr, pfn) // sleep (waiting for I/O)
5  PTE.present = True          // update page table with present
6  PTE.PFN     = PFN           // bit and translation (PFN)
7  RetryInstruction()          // retry instruction
```

Quando trocar páginas

Na prática, o S.O. não espera a memória ficar lotada para fazer as trocas.

Quando trocar páginas

Na prática, o S.O. não espera a memória ficar lotada para fazer as trocas.

É bom sempre ter um conjunto de quadros livre para não atrasar futuras novas alocações.

High Watermark(HW) e Low Watermark(LW): Se o número de quadros livres é menor que o LW, o S.O. roda um processo em background para liberar quadros na RAM via swap até alcançar HW quadros livres.

Este processo, chamado de swap daemon ou page daemon, volta a dormir após liberar os quadros.

As trocas do swap daemon são feitas em grupos, o que ainda ajuda a aumentar a velocidade desta gravação.

Aula de Hoje

Políticas de Swap

Assim que a memória começa a ficar escassa, o S.O. tem uma série de decisões importantes a tomar.

A mais importante é decidir quais páginas tirar da memóriapara liberar quadros.

Política de substituição.

Cache

Na prática, o problema de substituição de páginas pode ser visto apenas como mais uma camada de cache no sistema de memória.

Assim, nosso objetivo volta a ser minimizar os cache miss .

Tempo médio de acesso a memória:

$$TMAM = (P_{Hit}.T_M) + (P_{Miss}.T_D)$$

Se em cada dez acessos, nove são hit, o tempo de acessar memória for $100ns$ (ou 0.0001ms), e o tempo de miss é $10ms$, quanto é o tempo médio de acesso à memória?

Cache

Na prática, o problema de substituição de páginas pode ser visto apenas como mais uma camada de cache no sistema de memória.

Assim, nosso objetivo volta a ser minimizar os cache miss .

Tempo médio de acesso a memória:

$$TMAM = (P_{Hit}.T_M) + (P_{Miss}.T_D)$$

Se em cada dez acessos, nove são hit, o tempo de acessar memória for $100ns$ (ou 0.0001ms), e o tempo de miss é $10ms$, quanto é o tempo médio de acesso à memória?

E se fosse um miss a cada cem acessos?

Aula de Hoje

Política Ótima

Quando se trabalha com políticas, uma estratégia comum é propor uma solução ótima (e muitas vezes impraticável), e comparar as políticas propostas com ela.

- Padrão ouro.

Política Ótima para substituição: Escolher a página que vai demorar mais para ser acessada no futuro.

Se é para tirar uma página da memória, é melhor escolher uma que não será utilizada tão cedo.

Política Ótima

Vamos nos próximos exemplos assumir um processo de quatro páginas que acesse as seguintes páginas numa RAM de 3 quadros: 0,1,2,0,1,3,0,3,1,2,1:

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	2	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	3	0, 1, 2
1	Hit		0, 1, 2

Figure 22.1: **Tracing The Optimal Policy**

Tipos de miss

3 C's:

- Cold Start (ou compulsório).
- Miss de Capacidade, que ocorre quando o cache está cheio.
- Miss de Conflito, quando um item tem que ser retirado da memória mesmo com espaço sobrando (como o que acontece em caches não totalmente associativas.

Não é incomum calcular a Hit rate ignorando os miss compulsórios (desde que isso seja mostrado claramente).

Aula de Hoje

FIFO

Primeiro a Entrar, primeiro a sair.

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		First-in→	0
1	Miss		First-in→	0, 1
2	Miss		First-in→	0, 1, 2
0	Hit		First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2
3	Miss	0	First-in→	1, 2, 3
0	Miss	1	First-in→	2, 3, 0
3	Hit		First-in→	2, 3, 0
1	Miss	2	First-in→	3, 0, 1
2	Miss	3	First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2

Anomalia de Belady

Um efeito interessante no FIFO foi visto em 1969. Os acessos 1,2,3,4,1,2,5,1,2,3,4,5 geravam uma mudança no hit rate quando o cache sobe de 3 para 4 páginas.

Contudo, a hit rate diminui ao invés de aumentar!

Aula de Hoje

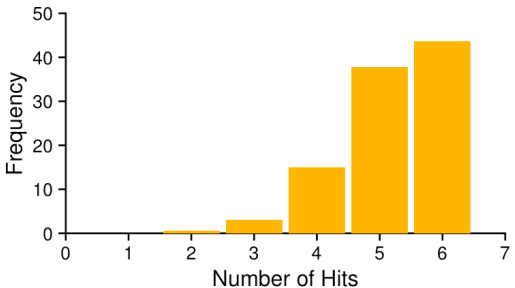
Random

Como sempre ocorre com políticas, também é proposto simplesmente escolher aleatoriamente uma página para tirar da memória:

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	0	1, 2, 3
0	Miss	1	2, 3, 0
3	Hit		2, 3, 0
1	Miss	3	2, 0, 1
2	Hit		2, 0, 1
1	Hit		2, 0, 1

Random

Random depende completamente da sorte. Neste exemplo foi melhor que o FIFO, mas tudo depende da sorte na escolha da semente:



Aula de Hoje

LRU

FIFO e Random tem o problema de potencialmente retirar páginas que podem ser referenciadas logo a seguir.

Por exemplo, páginas com código rodando ou estruturas de dados importantes.

Por isso, foram criadas políticas que utilizam o histórico de acessos para decidir qual página retirar.

Critérios como **frequência** de acessos e quão **recente** foi o último acesso.

Princípio da localidade.

LRU

LFU (Least Frequently Used) e LRU (Least Recently Used).

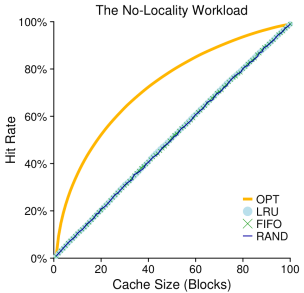
Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1	Hit		LRU→ 2, 0, 1
3	Miss	2	LRU→ 0, 1, 3
0	Hit		LRU→ 1, 3, 0
3	Hit		LRU→ 1, 0, 3
1	Hit		LRU→ 0, 3, 1
2	Miss	0	LRU→ 3, 1, 2
1	Hit		LRU→ 3, 2, 1

Aula de Hoje

Workloads:Sem Localidade

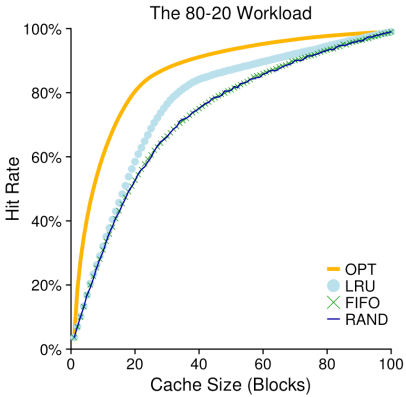
Para entender melhor o funcionamento dos algoritmos, mostrarei resultados dos mesmos com 3 workloads:

Sem localidade: 100 páginas, 10,000 acessos completamente aleatórios (sem localidade), tamanho da ram de 1 a 100 quadros.



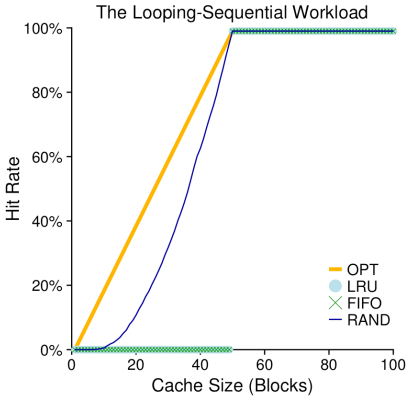
80-20

80-20: 100 páginas, 10,000 acessos, com 80% dos acessos sendo feitos a 20% das páginas, as páginas "quentes", tamanho da ram de 1 a 100 quadros.



Looping

Looping: 50 páginas, 10,000 acessos, as páginas são acessadas na sequência, dando a volta quando termina tamanho da ram de 1 a 100 quadros.



Aula de Hoje

Implementação de políticas

LRU: Cada acesso a uma página implica em atualizar uma estrutura de dados que contenha as páginas em ordem de acesso.

FIFO: Basta manter uma fila.

Manter lista de referencias recentes é complicado, e precisa de uma atualização **a cada acesso à memória**.

Implementação de políticas

Exemplo de implementação: Hardware poderia atualizar um campo de último acesso com o horário do último acesso.

Procurar uma página seria basicamente percorrer esta estrutura para achar o mais antigo.

Ainda assim, conforme a memória cresce, esse tempo vai ficando proibitivo.

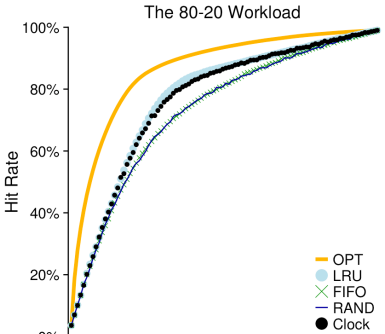
Por isso, LRU não é factível, e foram propostas aproximações do mesmo.

Aula de Hoje

LRU

Bit de referência: a cada acesso à memória, um bit correspondente àquela memória é setado para 1.

Algoritmo do relógio: Fila circular, onde o elemento da vez só é escolhido se o bit de referência é 0. Se for 1, zera e parte para o próximo (que pode inclusive ser aleatório).



Dirty Bit

Dirty bit representa se a página foi modificada desde a última vez que foi colocada na memória.

Se uma página foi apenas lida (e tem uma cópia dela no disco) o custo de fazer swap nela é nulo.

Desta forma, o algoritmo pode dar prioridade à páginas "limpas".

Thrashing

Thrashing é o que ocorre quando a demanda por memória excede a memória física.

Excesso de Swap pode deixar o sistema inutilizável.

S.O.s podem escolher alguns processos para deixar de quarentena, tentando manter apenas processos com working sets mais gerenciáveis.

Também pode rolar um out-of-memory killer, mas esta opção é mais radical.