Trabalho Prático
Sistemas Operacionais
Universidade Federal São João Del Rei

Millas Násser, Paulo Henrique, Welton Augusto

21 de Novembro de 2017

1 Simulador

A proposta do simulador consiste em analisar o desempenho de 3 algoritmos de troca de página simulando a memória física e virtual, levando em consideração parâmetros como o tamanho da página e memória.

2 Memória Virtual

Memória virtual é um conceito que permite que programas maiores que a memória física realizem sua operações, através do conceito de páginação, possibilitando que apenas partes de seu binário (páginas) estejam na memória ao serem requisitadas, descocupando a memória ao não serem mais necessárias geralmente cedendo espaço a outras páginas que possam vir a ser requisitadas.

3 Página

Sistemas operacionais dividem a memória em partes denominadas páginas, também conhecidas como moldura, para facilitar sua manipulação, geralmente entre valores de 2KB a 8kB. A divisão de memória em páginas causa um pequeno desperdício de memória para pequenos programas ao ponto de ser irrelevante, em relação a facilidade obtida sobre facilidade para mapear além da redução na quantidade de memória gasta para criar tabelas e listas para mapear páginas logo que a quantidade de páginas a serem mapeadas é inferior a quantidade de entradas na memória.

4 Troca de Página

Um programa ao ser compilado utiliza a capacidade de endereçamento do processador como referência de seu espaço de memória, logo um programa compilado para arquitteruras de 32 bits tendem a utilizar 4GB de memória. Sabe-se

que em um computador vários processos compartilham a memória e através da paginação executam seus códigos. Portanto é normal e com forte tendência em máquinas com menores quantidade de memória física que os programas ocupem toda a memória e a troca de páginas passe a ser necessária forçando que páginas sejam enviadas da memória para o disco rígido sobre o critério estabelecido por algoritmos de troca de página.

5 Heurísticas de troca de página

Apesar da conhecida existência do algoritmo ótimo para troca de páginas, infelizmente não é possível sua implementação, logo a solução é a utilização de heurísticas que maximizem o desempenho da troca de páginas aproximando-as ao máximo possível do algoritmo exato, relacionando parâmetros como tempo na memória, frequência de referências e modificação da página.

5.1 FIFO - First In First Out

Algoritmo de fila como o próprio nome diz, utiliza a ordem de chagada das página na memória como crtério de remoção. Este critério de troca não é considerado eficiente, com enfoque para o pior caso onde páginas frequentemente referênciadas são removidas da memória devido a sua ordem de chegada. Porém é de simples implementação e utiliza estruturas de dados simples para manter sua fila de prioridade, sendo neste trabalho necessário somente a abstração de um vetor circular sobre a memória física, abstração qual é realizada através de um ponteiro do tipo inteiro denominado **ID** marcando o final da lista sofrendo atualizações sempre que uma nova página é enviada para memória. Portanto sempre que surgir a necessidade de realizar uma troca de página o valor contido em **ID** será correspondente ao da página a ser removida, realizando o acesso em O(1).

5.2 LRU - Least Recently Used

O algoritmo LRU *(menos recentemente uasado)*, também conhecido como *aging* é considerado eficiente apesar de não possuir maior complexidade em sua implementação, este é raiz de uma família de algoritmos que derivam sua ideia principal, definida por:

1. Páginas que foram muito utilizadas (referenciadas), provavelmente serão novamente utilizadas.

2. A página que permanceceu em desuso durante o maior tempo deve ser trocada.

Sua implementação como dito anteriormente não é fácil, é necessário uma matriz com **n** entradas por **n** colunas, para **n** igual a quantidade de molduras existente na memória. Sempre que uma moldura é referenciada seu respctivo endereço

é convertido em um inteiro **i**, utilizado como índice, onde todos bits da matriz indexada por **i** são configurados para 1 e todos os bits da coluna indexada por **i** são configurados para 0. No momento da paginação cada linha da matriz possuirá seus bits alterados, que corresponde ao que denomina-se a "idade"da página. Então o algoritmo irá percorrer todas as linhas da matriz e selecionar o índice da linha com soma de bits menos siginificativa, para todo bit a esquerda mais significativo que o de sua direita. O algoritmo LRU com a tabela de **n** x **n** citado anteriormente é geralmente implementado em hardware e possui maior precisão logo que ele "não se esquece de nada"causa do alto custo de sua implementação. A forma usada neste trabalho foi percorrer a memória para descobrir a página com o menor tempo de último acesso.

5.3 Randomic Algorithm

Como o próprio nome diz, este algoritmo utiliza a alieatoriedade como critério para suas decisões. Portano é o algoritmo mais simples de ser implementado, tanto que para o presente cenário é necessário somente o conhecimento sobre a dimensão da memória física em número de páginas e de uma semente para alietoriedade. Ao sortear um id aleatório este será retornado para a função de troca de página e então a página sorteada é removida da memória e outra página ocupará a posição agora livre.

6 Implementação

O simulador como dito anteriormente propõe emular a memória virtual e avaliar o desempenho de algoritmos de paginação, realizando trocas de páginas para testes que aproximam-se de situações diversas do cotidiano.

6.1 Memória Virtual

Para simular a memória virtual foi necessário a criação do tipo **Moldura**, que consiste em uma estrutura composta pelos atributos, **endereco**, **ultimo_acesso** do tipo inteiro e **controle** do tipo char.

- **endereco**: armazena o endereço de uma moldura ná memória física.
- **ultimo_acesso**: registra o tempo da última referência de uma página.
- **controle**: determina o estado da página (referênciada, suja, presente).

6.2 Memória física

Para memória física bastou a alocação de um array do tipo interio onde cada posição deste refere a uma página na memória.

6.3 Comportamento

Para melhor compreensão do comportamento do software, segue abaixo o pseudocódigo relativo a sequênia de operações do simulador.

```
escritas # guarda o número de escritas
leitura # guarda o número de leituras
TEMPO # controla o tempo do sistema
REFERENCIADO # possui o bit de referência
ALTERADO # possui o bit de alteração

para toda página i acessada
   se i não estiver na memória
      se existir moludras livres
         adiciona página na memória física
      se não
         chama o algoritmo de paginação
   memoria_virtual[i].ultimo_acesso := TEMPO
   memoria_virtual[i].controle := REFERENCIADO
   se for uma operação de escrita
      memoria_virtual[i].controle
      memoria_virtual[i].controle := ALTERADO
      escritas := escritas + 1     se nao
      leituras := leituras + 1
TEMPO := TEMPO + 1
```

7 Resultados

7.1 Condições

Os algoritmos serão submetidos a diferentes configurações de memória e páginas com intuito de avaliar suas respectivas estabilidades, considerando como critérios de desempenho a quantidade de erros ao realizar acessos a páginas na memória física e também a quantidade de substituição de páginas sujas por parte dos algoritmos. Para obter maior diversidade nos resultados, foram utilizados 4 arquivos com dados que simulam diferentes aplicações, citadas logo abaixo.

- **compilador.log**: execução de um compilador, que normalmente utiliza um grande número de estruturas de dados internas complexas.

- **matriz.log**: um programa científico que utiliza cálculos matriciais relativamente simples, mas sobre grandes matrizes e vetores.

- **compressor.log**: um programa de compressão de arquivos, que usa estruturas de dados mais simples.

- **simulador.log**: um simulador de partículas, que executa cálculos complexos sobre estruturas relativamente simples.

7.2 FIFO x RANDOM

Através dos testes realizados percebe-se que os algoritmos fifo e random obtiveram resultados bem semelhantes mesmo para a simulação do compilador, onde é esperado maior diferença no desempenho dos algoritmos. Subitamente os dois algoritmos possuiram melhor performance em relação ao algoritmo LRU, com grandes margens de acertos para as diversas aplicações. Para melhor visualização, abaixo seguem os valores obtidos pelos dois algoritmos ao simular o compilador. (Note que as posições preenchidas com '-' significam que a quantidade de memória necessária para armazenar a tabela de páginas é maior do que a memória física disponível).

Hits na memória FIFO (compilador)						
Qtde Mem/Tam Página	2	4	8	16	32	64
128	-	-	-	-	996573	994283
256	-	-	-	996023	995629	993337
512	-	-	996058	996164	996326	996394
1024	-	996441	996836	997196	997570	997971
2048	996056	996752	997205	997601	997993	998358
4096	996248	996978	997383	997781	998145	998480
8192	996380	997051	997441	997780	998138	998460
16384	996380	997149	997562	997950	998271	998603

Figura 1: Hits na memória, Algoritmo FIFO, simulação compilador
Valores em KB

Hits na memória RANDOM (compilador)						
Qtde Mem/Tam Página	2	4	8	16	32	64
128	-	-	-	-	997166	997550
256	-	-	-	996536	996611	996806
512	-	-	996430	996650	996972	997186
1024	-	996583	997007	997352	997738	998140
2048	996109	996813	997273	997674	998065	998406
4096	996279	996987	997429	997846	998214	998537
8192	996380	997086	997462	997827	998184	998509
16384	996380	997149	997606	997964	998297	998626

Figura 2: Hits na memória, Algoritmo RANDOM, simulação compilador
Valores em KB

7.3 Pior caso

Esperava-se que a simulação do compilador gerasse os piores resultados. Porém a simulação da matriz e gerou maiores discrepâncias na taxa de acertos, principalmente para o algoritmo lru, onde foi também foi possível observar notável melhora em seu desempenho conforme a ocorre a expansão da memória e também do tamanho da palavra, visível na tabela abaixo: (Note que as posições preenchidas com '-' significam que a quantidade de memória necessária para armazenar a tabela de páginas é maior do que a memória física disponível).

Qtde de Hits na memória (LRU)						
Mem/Tam Página	2	4	8	16	32	64
128	-	-	-	-	667149	623716
256	-	-	-	834885	729914	715973
512	-	-	980859	960272	846824	765312
1024	-	994328	989072	984097	967718	884115
2048	996628	996369	995256	992125	986771	978071
4096	996940	997198	997051	996173	994027	990038
8192	997007	997425	997603	997621	997040	995673
16384	997007	997458	997787	997990	998051	997670

Figura 3: Hits na memória, Algoritmo LRU, teste matriz
Valores em KB

O número de páginas sujas para o teste **compilador.log** também seguiu o mesmo padrão. O random teve os melores resultados em geral, seguido de perto pelo fifo e o lru gerou resultados muito piores. Porém, para páginas pequenas (2 e 4 KB) e memória suficientemente grande (8 e 16 MB), o núemero de páginas sujas foi de 0 para todos os algorítmos. Este resultado reflete bem a realidade, onde as memórias são bem maiores que 16Mb e as páginas costumam ter 4KB.

7.4 Desempenhos geral

Ao final dos resultados, apesar da diferença existente entre os algoritmos, todos mantiveram altas médias de hits na memória (acertos no acesso a páginas), voltando a atenção para o algoritmo random que manteve leve superioridade sobre o algoritmo de fila e também para o lru, que obteve resultados inferiores ao esperado tanto em hits na memória quanto na substituição de páginas sujas.

média de acertos						
	2	4	8	16	32	64
FIFO	498429	623379	748195	872985	997716	997467
LRU	498306	622058	742473	852536	937947	910996
RANDOM	498442	623402	748246	873083	997945	998193

Figura 4: Média geral de hits na memória.
Valores em KB

media de páginas sujas						
	2	4	8	16	32	64
FIFO	10	21	39	60	77	76
LRU	242	535	1308	4032	11005	17201
RANDOM	7	17	31	49	58	46

Figura 5: Média geral de substituição de páginas sujas.
Valores em KB

7.5 Conclusão

A partir dos resultados obtidos percebe-se superioridade inesperada do algoritmo random em relação aos demais, assim como não só deste mas também do algoritmo de fila, onde ambos demostraram forte estabilidade, algo que não ocorreu com o algoritmo lru, que obteve maior instabilidade nos quesitos: falta de página e substituição de páginas sujas, deste modo obtendo os piores resultados dentre as herísticas então apresentadas.

7.6 Ambiente de Desenvolvimento

Processador: i7 5500
Memoria: 8gb
Sistema Operacional: Archlinux-4.12.3-1

8 Apêndice

8.1 Funções

8.1.1 Herísticas de paginação

sub_fifo: manipula a fila de páginas por ordem ascendente e seleciona páginas para troca.
sub_lru: seleciona página para troca com o menor tempo de último acesso.
sub_random: retorna página para troca aleatoriamente.

8.1.2 Funções de suporte

substituicao: recebe um endereço de uma página para remoção através de uma heurística de paginação e realiza a troca nas memórias com outra página solicitada.
leArquivo: armazena as informações do arquivo de teste em um array de char.
zera_bit: manipula o atributo controle presente nas molduras da memória virtual criando a abstração de página não referênciada.

8.1.3 Mineração e Saida

saida: recebe os dados gerados durante o processamento e formata-os para inserção em planilhas eletrônicas.
print_memoria_virtual: imprimi detalhes da memória virtual, utilizada para fins de debug.

8.2 Constantes

LEN: define a quantidade de caracteres utilizados para o endereço dos testes (core.h).
ZERO_TIME: define o intervalo de operações em que o bit de referência das páginas deve ser zerado (core.h).

9 Bibliografia

- Wikipedia, Cache replacement policies, Ultima atualização: Outubro de 2017, Dísponível em: < https://en.wikipedia.org/wiki/Cache_replacement_policies > Acesso em Novembro de 2017.
- ROMAN, MORANDINI e UEYAM, Gerenciamento de Memória virtual, Algoritmos de paginação, Dísponível em: < http://wiki.icmc.usp.br/images/d/dc/Aula12.pdf > Acesso em Novembro de 2017.
- TNEMBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3 ed. São Paulo: Pearson, 2010. Acesso em Novembro de 2017.