UNIVERSIDADE FEDERAL DE MINAS GERAIS
ALGORITMOS E ESTRUTURAS DE DADOS III

TP 1

Memória Virtual para ÁrvoreB

Aluno: Rafael Ramon de Oliveira Egídio
Email: nomar22@gmail.com
Matricula: 2009052360

1-Introdução

Este documento irá detalhar a solução de um problema relacionado a simulação de um sistema de gerenciamento de memória primária , este problema deverá ser representado na forma de uma árvoreb .

O objetivo deste trabalho prático é implementar ,contabilizar e comparar os resultados de pagefaults para FIFO, LRU e LFU como algoritmos de substituição de páginas.

Os sistemas operacionais utilizados na atualidade abstraem a visão de limitação de memória para desenvolvedores e usuários criando um espaço de memória infinito, para isto utiliza técnicas de paginação para expandir páginas da memória principal para o disco rígido.

Devido ao fato da velocidade de acesso ao disco rígido ser ordens de grandezas mais lento que o acesso a memória primária , são utilizadas políticas para se escolher qual página deverá ficar na memória primária, tentando prever , baseado em algum algoritmo , qual página não será necessária em um curto espaço de tempo.

2-Modelagem e solução proposta

Seguem alguns conceitos diretamente relacionados a solução em questão

2.1*Page Faults*

É o nome dado ao evento de solicitação de uma página que não encontra-se disponível em memória primária.

Memória virtual

Memória virtual é uma abstração que o sistema operacional oferece de forma que o desenvolvedor recebe um espaço ilimitado de memória principal e o sistema operacional fica encarregado de gerenciar as limitações da memória principal na memória secundária .

2.2Políticas de reposição de páginas

Políticas de reposições de páginas , são algoritmos utilizados em gerenciamento de memória para subistituir as páginas da memória secundária para a memória principal neste trabalho prático implementaremos a LRU, LFU e FIFO, estas políticas são acionadas quando a memória atinge o limite de ocupação de páginas e é necessário escolher uma página a ser removida da memória principal.

FIFO – Remove a página mais antiga da memória, tende a prejudicar processos que iniciam junto com o SO e devem ficar na memória em tempo integral.

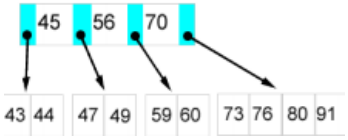
LFU – Remove a página que não está sendo frequentemente usada, ou seja, uma página antiga, mas que esteja sendo usada frequentemente permance na memória, porém prejudica processos novos que ainda possuem pouco acesso a memória.

LRU – Remove a página que não é utilizada a mais tempo, mantém as páginas que foram utilizadas mais recentemente na memória principal

2.3Árvore B

Árvore b é um algoritmo de organização de dados indexados, similar a árvore binária, esta árvore b possui uma ordem m que referencia o número mínimo de registros que um nó deverá ter(com exceção da raiz que pode ter no mínimo 1 registro) e o máximo que é exatamente 2m.

Cada poderá ser uma folha ou um nó pai (possui filhos) , no caso do nó pai existirá em cada registro dois apontadores um para o nó com os registros menores que ele e um para o nó com registros maiores que ele.



-Propriedades da árvore b

Uma árvore b de ordem m terá uma raiz que pode ter até no máximo 2m registros e no mínimo 1 registro. Cada nó poderá ter no mínimo m registros e no máximo 2m registros além de 2m+1 apontadores para os nós filhos.

-Pesquisa

Para realizar uma pesquisa de B em uma estrutura do tipo arvoreb, deve sempre se iniciar pela raiz verificando cada um registro X do nó até encontrar B<X, caso B=X registro encontrado, caso não seja e o nó seja um nó pai, deve-se encaminhar a busca para o nó correspondente ao apontador de registros menores que X, caso seja folha o registro não existe na árvore.

-Inserção

Para inserir registros em uma árvore b, deve-se usar a pesquisa e verificar se o registro existe na árvore, caso não exista inserir o registro no nó. Se número de registros do nó < 2m insere o registro normalmente. Caso não seja deverá ser criado um novo nó e promovido o registro intermediário. Ao ser promovido este registro respeitará a mesma regra de inserção para o nó pai. Caso seja a raiz, será criado uma nova raiz para que o registro intermediário seja promovido.

-Remoção

Para remover registros deve-se antes de tudo localizar o registro na árvore usando a pesquisa, ao encontrar o registro deve-se verificar se ela é uma folha, caso seja remove-se o

registro e verifica se o nó continua a possuir no mínimo m registros, caso sim termina. Caso não verifica se algum irmão possui mais de m registros, caso tenha o registro mais próximo do irmão receptor é promovido para o nó pai e o registro do nó pai é despromovido para o irmão que teve o registro removido.

Caso não possua irmão com mais de m registros, é escolhido um dos 2 irmãos para unificar em um nó só , juntamente com o registro divisor no nó pai , desta forma é removido um registro do nó pai e todas estas regras descritas terão o mesmo comportamento de remoção no nó pai.

3 Ambiente de Teste

Este trabalho foi testado em um computador Intel I3 2.4 GHz 4GB com sistema operacional Linux Ubuntu 12.04 32bits.

4 Requisitos

A solução do trabalho deverá ser implementada na linguagem de programação C. O programa receberá um arquivo de entrada,passado como parametro na chamada do programa e retornará uma saída conforme informada na entrada do programa.

Estrutura do arquivo de entrada:

```
1                   // número de instancias
120 2               // tamanho em bytes da memória , seguido da ordem da árvoreb
18                  // n  número de registros a serem pesquisados
10 5 7 20 9 13 18 32 15 38 40 8 60 27 17 12 37 25          // n números
3         // m número de registros a serem excluídos
32 20 7           // m registros
10                    //b número de registros a  serem consultados
15 25 40 8 60 12 37 8 13 10           //b números
5                   // t números a serem imprimido o caminho
40 37 37 27 12                // t registros
```

Estrutura do arquivo de saída:

O programa deverá imprimir no arquivo de saída seguindo o modelo:

```
7 9 9               //pagefaults para FIFO, LRU e LFU
9 13 18 37 38 40 60             //Caminho para encontrar o registro 40
9 13 18 37                // Caminho para encontrar o registro 37
9 13 18 37                // Caminho para encontrar o registro 37
9 13 18 37 25 27                  //Caminho para encontrar o registro 27
9 13 18 37 10 12        //Caminho para encontrar o registro 12
```

5 Algoritmo da Solução

Para atender o objetivo de simulação proposto no trabalho prático , serão alocadas 3

estruturas de dados para representar a memória primária e uma árvoreb que representará a memória secundária.

Foi sugerido na especificação do trabalho prático a possibilidade de se utilizar a árvoreb implementada pelo professor Nívio Ziviani. A partir da leitura do arquivo de entrada será montada uma árvore b que representará a memória secundária.

Em sequencia é iniciada a etapa de gerenciamento de memória, neste simulação haverão 3 espaços de memórias que deverão ser gerenciados de acordo com cada registro lido no arquivo de entrada . Por ultimo é solicitado que se descrevam as páginas percorridas para se pesquisar n números e por final é imprimido este caminho.

6 Estruturas de dados utilizadas e descrições

Para um melhor entendimento do código e estruturas convencionou-se que neste trabalho prático estruturas de dados recebem um nome e apontadores para estas estruturas recebem o prefixo **Link** que define o tipo como apontador.

```
typedef struct {
  LinkMoldura primeiro;
  LinkMoldura ultimo;
  int len;
  int tipo;
  int maximo;
  int pageMiss;

} Memoria;
```

Trata-se de um lista duplamente encadeada de molduras que detém seu tipo(LFU,LRU,FIFO), um contador de page misses e um inteiro maximo que detém o tamanho da memória em número de páginas.

```
typedef struct Moldura {
        LinkPagina pagina;
        int hits;

        LinkMoldura anterior;
        LinkMoldura prox;
} Moldura;
```

Trata-se de uma célula com um um contador de acesso de página.

```
typedef struct Pagina
{
    short n;
    Registro *r;
    LinkPagina *p;
} Pagina;
```

Trata-se de uma página e possui uma pequena alteração da árvore implementada pelo Nívio que é a transformação da página para tamanho dinâmico.

7 Pseudo-códigos e descrições de funções

arvoreb.c - Partindo-se do príncipio que a árvoreb do Nívio Ziviani foi sugerida, foi abstraída as técnicas que são utilizadas para a criação e gerenciamento deste módulo, foram feitas apenas 2 alterações pontuais:

Transformação da árvore de ordem dinamica:
Uma árvore

Alteração do método de pesquisa :
para mais detalhes verificar na referencia bibliográfica.

8 Módulos

MemóriaPrincipal.c

Tem a função efetiva de gerenciar as instancias da memória principal.

8.1Funções públicas da Biblioteca

São as funções que podem ser utilizadas por qualquer programador que queira utiliza-la sem alterar o funcionamento básico da biblioteca chamadas de funções públicas.

```
void insere Pagina(LinkPagina pagina,LinkMemoria memoria);
```

Função que recebe a página e a memória em que a página deve ser inserida. Verifica na memória qual o seu tipo(LRU,LFU,FIFO) para inserir de acordo com as características da política e remove a página caso a memória já tenha atingido seu máximo.

```
int calculaNumPaginas(int tam, int ordem);
```

Recebe o tamanho da memória solicitada em bytes e a órdem da árvoreb.

```
void removePagina(LinkPagina pagina,int tipo, LinkMemoria memoria);
```

Recebe um apontador de página, o tipo e a memória e de acordo com a política da memória remove a página indicada.
FIFO – removeUltimo
LRU - removeUltimo
LFU - retiraMenosUsado

```
void inserePrimeiro(LinkPagina pagina, LinkMemoria memoria);
```

Insere uma página como primeira ;

```
void removePrimeiro(LinkMemoria lista);
```

Remove a primeira página;

```
void pesquisaMemoriaSecundaria(Registro *x, LinkPagina Ap, LinkMemoria memoria );
```

Função recursiva que olha primeiramente se a página passada já está na memória, se não estiver contabiliza o pagefault e utiliza a **inserePagina**, caso não utiliza a **acessaMemoria**, percorre os registros da página até encontrar um que seja maior que o registro atual se ele for o

pesquisado termina, caso seja menor utiliza o **pesquisaMemoriaSecundaria** no filho da esquerda do registro, caso seja maior utilizao **pesquisaMemoriaSecundaria** no filho da direita.

int ***registroNaMemoria****(Registro x, LinkMemoria memoria,* ***int*** *ordem);*

Dada uma memória paginada verifica se um registro já se encontra na memória, faz o papel da tabela de páginas na memória primária. Evita que se faça uma consulta que gere pagefaults para trazer um registro que já encontra-se na memória.

8.2 Funções internas da biblioteca

Funções de uso restrito a biblioteca, são invisíveis para o usuário e tem a função de manter o funcionamento básico da biblioteca

int ***estaNoTopo****(LinkMoldura moldura, LinkMemoria memoria)*

Verifica se uma moldura de página encontra-se no topo da memória, retorna 1 para TRUE e 0 para FALSE.

int ***estaNoFundo****(LinkMoldura moldura, LinkMemoria memoria)*

Verifica se uma moldura de página encontra-se no final da memória, retorna 1 para TRUE e 0 para FALSE.

void ***sobeNaPilha****(LinkMoldura vaiSubir, LinkMemoria memoria)*

Desloca uma moldura de qualquer posição para o topo da memória, faz a função do heap.

void ***retira****(LinkMoldura moldura, LinkMemoria memoria)*

Faz a emenda , unindo a moldura anterior a moldura posterior e remove a moldura atual liberando o espaço de memória.

void ***retiraMenosUsado****(LinkMemoria memoria)*

Percorre as molduras da memória verificando a moldura que possui menos acessos em caso de empate retira a mais antiga na memória.

void ***acessaMemoria****(LinkMoldura moldura,LinkMemoria memoria)*

Caso a memória for do tipo FIFO não faz nada, caso seja LFU contabiliza um pagehit e caso seja um LRU utiliza a **sobeNaPilha**

LinkMoldura ***paginaNaMemoria****(LinkPagina ap,LinkMemoria memoria)*

Procura uma página na memória principal e retorna um ponteiro para a mesma, caso não estiver retorna nulo.

Pseudo código main

9 Algoritmo principal

Enquanto houverem instancias do problema

Inicia memória FIFO,LRU,LFU

Monta Insere registros arvoreb

Retira registros árvoreb

Enquanto houverem registros para serem pesquisados na memória

se o registro está na memoria FIFO

faz nada

senão

pesquisaMemoriaSecundaria FIFO

se o registro está na memória LRU

sobe a moldura do registro no heap

senão

pesquisaMemoriaSecundaria LRU

se o registro está na memória LRU

contabiliza acesso

senão

pesquisaMemoriaSecundaria LFU

fim Enquanto

Enquanto houverem registros para imprimir caminho

pesquisa

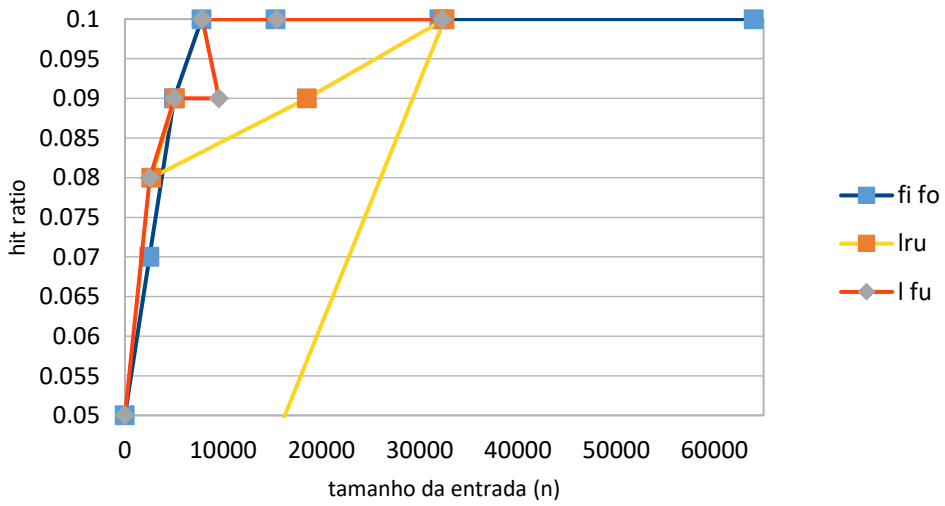
10 Execução

Para a compilar este trabalho deve-se executar o comando make de dentro do diretório raiz do projeto em seguida o comando ./tp1 [arquivodeentrada.txt} [arquivodesaida.txt] o arquivo de entrada deverá seguir a especificação dada em tópico anterior e a saída do programa será impressa no arquivo de saida desde que o mesmo possua permissão de escrita.

11 Avaliação Experimental

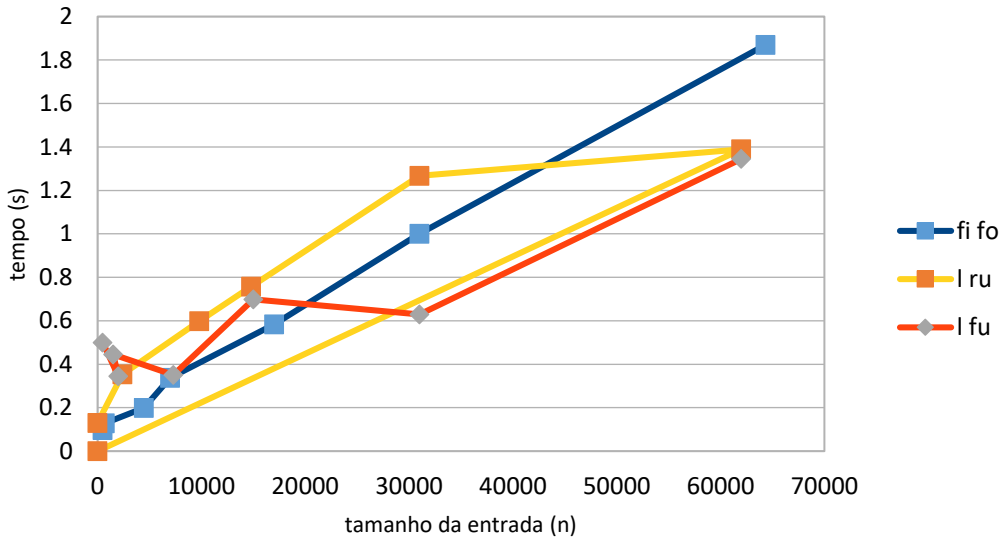
Para a realização dos testes foi utilizada a árvore dada como exemplo na documentação do trabalho prático de ordem 2.

Foi utilizado um script gerador de instancias que disponibiliza dados aleatórios para serem pesquisados de 1 a 10000 registros que obteve os seguintes resultados:



Foi utilizada a entrada a mesma entrada e feito uso da função time do linux para realizar a medida de tempo para finalização do linux

12



Conclusão

O custo assintótico computacional de todas as implementações tenderam para O(n) as diferenças de hit ratios apresentadas nos resultados entre as implementações devem-se unicamente a entrada , pois para cada tipo de informação uma política específica será mais adequada. De uma forma geral podemos verificar que para entradas aleatórias podemos destacar um certo nível de semelhança no resultado de pagefaults entre as políticas de reposição que tende a ocorre devido a aleatoriedade dos registros que foram buscados tendendo a 10% de acerto.

Referências

Ziviani, Nivio (2011) “Projeto de Algoritmos com implementações em Pascal e C”, Cengage Learning, 3ª Edição.