Pontifícia Universidade Católica do Rio de Janeiro

Lab 7 - Gerenciamento de Memoria

Sistemas Operacionais

Alunos	Tharik Lourenço
	Suemy Inagaki
Professor	Luiz Fernando Seibel

Rio de Janeiro, 6 de Junho de 2021

Conteúdo

1 Problema 1

Neste trabalho, você deverá implementar um simulador de memória virtual, sim-virtual, em linguagem C. Neste simulador você criará as estruturas de dados e os mecanismos de mapeamento de memória (lógico -¿ físico) necessários para realizar a paginação, e deverá implementar três algoritmos de Substituição de Páginas: o Not-Recently-Used (NRU), o FIFO 2nd. chance e o Least-Frequently-Used (LFU).

O simulador receberá como entrada um arquivo que conterá a sequência de endereços de memória acessados por um programa real. Esses endereços estarão no formato hexadecimal, seguidos por uma letra R ou W, para indicar se o acesso foi de leitura ou escrita. Ao iniciar o simulador, será definido o tamanho da memória (em quadros) para aquele programa e qual o algoritmo de substituição de páginas que será utilizado. O simulador deve, então, processar cada acesso à memória para atualizar os bits de controle de cada página/quadro, detectar falha de páginas (page faults) e simular o processo de substituição de páginas e carga de uma nova página no quadro de página escolhido. Durante a simulação, estatísticas devem ser coletadas, para gerar um relatório curto ao final da execução.

O simulador deverá ter os seguintes quatro argumentos de linha de commando:

- O algoritmo de substituição de página sendo simulado (LRU/2nd. Chance/LFU)
- O arquivo contendo a sequência de endereços de memória acessados (arq.log)
- O tamanho de cada página/quadro de página em KB (tamanhos a serem suportados: 8 a 32 KB)
- O tamanho total de memória física hipoteticamente disponível pode variar de 1 MB a 16MB.

Ou seja, uma invocação do simulador:

```
sim-virtual LRU arquivo.log 8 16
```

indicaria um tamanho de página de 8KB e uma memória fisica de 16 MB.

1.1 Estrutura de Dados

Criamos a estrutura de página da seguinte forma:

```
typedef struct pagina {
    int r; // status referenciada
    int m; // status modificada
    unsigned int endrCod; // endereco da pagina
    int timer; // contador de passagem
} Pagina;
```

E, nos algoritmos LFU e FIFO 2nd chance, usamos uma lista encadeada onde cada elemento tem um ponteiro para Página e um ponteiro para o próximo elemento:

```
struct reg {
    Pagina* conteudo;
    struct reg *prox;
};
typedef struct reg No;
```

Para o algoritmo NRU, utilizamos uma vetor de ponteiros que contém todas as páginas

```
Pagina **allPags = (Pagina**)malloc(numPages*sizeof(Pagina*));
```

E para cada elemento do vetor, alocamos espaço para uma página:

```
for(int i = 0; i < numPages; i++) {
     allPags[i] = (Pagina*) malloc (sizeof(Pagina));
     allPags[i]->timer = 0;
     allPags[i]->endrCod = 0;
 }
```

1.2 Funcionamento

1.2.1 Simulador

O simulador recebe como argumento o nome do algoritmo a ser testado, o arquivo a ser testado, o tamanho das páginas e o tamanho da memória.

Ele chama a função respectiva ao algoritmo passado como argumento:

```
if(strcmp(algoritmo, "NRU") == 0){
    typeNru(filePath, numPages, auxShift, pageSize, memSize);
}
else if(strcmp(algoritmo, "FIFO2") == 0){
    typeFifo2(filePath, numPages, auxShift, pageSize, memSize);
```

```
    }
    else if(strcmp(algoritmo, "LFU") == 0){
        typeLfu(filePath, numPages, auxShift, pageSize, memSize);
    }
```

Os parâmetros passados nas funções são:

- O nome do arquivo
- O número de páginas
- O Shift necessário
- O tamanho das páginas
- O tamanho da memória

Para calcular o número de páginas:

```
numPages = (memSize*1024)/pageSize;
```

Para calcular o shift, chamamos a função calculaShift passando como argumento o tamanho da página.

```
int calculaShift(int size) {
    switch (size){
        case 8:
            return 13;
        case 16:
            return 14;
        case 32:
            return 15;
    }
    return 0;
}

auxShift = calculaShift(pageSize);
```

1.2.2 NRU

Para cada linha do arquivo lido, analisamos os seguintes itens:

- O elemento procurado já está no vetor: nesse caso, incrementamos o R e verificamos e o rw é W ou R, e alterando o m para 1 ou 0 respectivamente.

- O elemento procurado não está no vetor e o vetor ainda tem espaço: adicionamos o elemento ao final do vetor.
- O elemento procurado não está no vetor e o vetor não tem mais espaço (atingiu o limite máximo de páginas): nesse caso, chamamos a função NRU para tratar a retirada de algum elemento.

Na função NRU, verificamos se a página foi referenciada e/ou modificada e adicionamos a nova página na posição i:

```
//trocando a pagina
allPags[i]->endrCod = endrCod >> auxShift;
allPags[i]->r = 1;
allPags[i]->timer = 0;
if(rw == 'W')
    allPags[i]->m = 1;
else
    allPags[i]->m = 0;
```

1.2.3 LFU

Como dito anteriormente, para o LFU utilizamos uma lista encadeada.

Para cada linha lida no arquivo de entrada, percorremos toda a lista e tratamos os seguintes casos:

- Lista vazia: nesse caso, adicionamos o elemento na lista
- O elemento procurado já está na lista: nesse caso, incrementamos o R.
- O elemento procurado não está na lista e a lista ainda tem espaço: adicionamos o elemento ao final da lista.
- O elemento procurado não está na lista e a lista não tem mais espaço (atingiu o limite máximo de páginas): nesse caso, chamamos a função LFU para tratar a retirada de algum elemento.

Para decidir qual elemento será retirado, percorremos a lista inteira e encontramos o menor valor de R, pois o que tiver o menor valor terá sido menos visto (menor frequencia) e por isso deverá ser retirado.

Ao encontrarmos o menor valor de R, percorremos toda a lista novamente para encontrar a primeira ocorrencia de página cujo R é igual ao encontrado previamente (menor).

Após isso, basta retirarmos o elemento encontrado e depois adicionarmos a página que precisávamos adicionar. Podemos ver a implementação nesse trecho de código:

```
No* LFU(Pagina* p, No* le){
    int menor = le->conteudo->r;
    //acha o menor r
    for(No* i = le; i != NULL; i = i->prox){
        if(i->conteudo->r < menor){ //se o r for menor que o menor
            menor = i->conteudo->r;
        }
    }
    le = removeMenor(le, menor);
    inserePag(p, le);
    return le;
}

No* removeMenor(No* le, int menor){

    No* ant = NULL;
    No* aux = le;
    while(aux != NULL)
    {
        if(aux->conteudo->r == menor)
            break;
        ant = aux;
        aux = aux->prox;
    }

    if(ant == NULL){
        le = le->prox;
    }
    else if(aux != NULL){
        ant = aux->prox;
    }
    return le;
}

void inserePag (Pagina* x, No *le)
{
    No *p = le;
    No *nova;
    nova = malloc (sizeof (No));
    while (p->prox != NULL)
        p = p->prox;
```

```
    p->prox = nova;
    nova->conteudo = x;
    nova->conteudo->r = 0;
    nova->prox = NULL;
}
```

1.2.4 FIFO 2nd Chance

Para cada linha lida no arquivo de entrada, percorremos toda a lista e tratamos os seguintes casos:

- Lista vazia: nesse caso, adicionamos o elemento na lista
- O elemento procurado já está na lista: nesse caso, incrementamos o R.
- O elemento procurado não está na lista e a lista ainda tem espaço: adicionamos o elemento ao final da lista.
- O elemento procurado não está na lista e a lista não tem mais espaço (atingiu o limite máximo de páginas): nesse caso, chamamos a função FIFO2 para tratar a retirada de algum elemento.

Na função fifo2, percorremos a lista até que encontremos um elemento com r == 0. Enquanto não encontramos, sempre pegamos o primeiro elemento e o colocamos para o final.

Quando encontramos o elemento cujo r == 0, fazemos a remoção do mesmo (que está na primeira posição da lista) e posteriormente adicionamos a nova página.

1.3 Código Fonte (Simulador)

```
/*
Nomes:
Tharik Lourenço
Suemy Inagaki
Lab 7 - Gerenciamento de Memória
Sistemas Operacionais
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

```c
#include "fifo2.h"
#include "nru.h"
#include "pagina.h"
#include "lfu.h"



int calculaShift(int size) {
    switch (size){
        case 8:
            return 13;
        case 16:
            return 14;
        case 32:
            return 15;
    }
    return 0;
}




int main (int argc, char *argv[]){
    int numPages,auxShift;


    int memSize = atoi(argv[4]);
    int pageSize = atoi(argv[3]);
    char *filePath = argv[2];
    char *algoritmo = argv[1];
    if(memSize < 1 || memSize > 16){
        printf("ERRO: Tamanho da memoria deve estar entre 1 e 16.\n");
        exit(-1);
    }
    if(pageSize < 8 || pageSize > 32){
        printf("ERRO: Tamanho da pagina deve estar entre 8 e 32.\n");
        exit(-1);
    }
    if(strcmp(algoritmo, "LFU") && strcmp(algoritmo, "NRU") && strcmp(algoritmo,
        printf("ERRO: tipo de Algoritmo invalido.\n");
        exit(-1);
```

```
    }
    printf ("Executando o simulador...\n");
    numPages = (memSize*1024)/pageSize;
    auxShift = calculaShift(pageSize);

    /*----------qual tipo e chamar o tipo--------------*/
    double begin = clock();
    if(strcmp(algoritmo, "NRU") == 0){
        typeNru(filePath, numPages, auxShift, pageSize, memSize);
    }
    else if(strcmp(algoritmo, "FIFO2") == 0){
        typeFifo2(filePath, numPages, auxShift, pageSize, memSize);
    }
    else if(strcmp(algoritmo, "LFU") == 0){
        typeLfu(filePath, numPages, auxShift, pageSize, memSize);
    }
    double end = clock();
    printf("Tempo de execução = %f\n", (double)(end-begin)/CLOCKS_PER_SEC);
    return 0;
}
```

1.4 Código Fonte (NRU)

```
/*
Nomes:
Tharik Lourenço
Suemy Inagaki
Lab 7 - Gerenciamento de Memória
Sistemas Operacionais
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pagina.h"
#include "nru.h"
int pageFaultsNRU = 0, pageSujaNRU = 0;

Pagina ** NRU(Pagina ** allPags, int numPages, char rw, unsigned int endrCod,
int auxShift) {
    int i;
```

```
    //escolhendo qual pagina trocar
    for(i = 0; i < numPages; i++){
        //NAO referenciada e NAO modificada
        // (R=0 , M=0)
        if(allPags[i]->r == 0 && allPags[i]->m == 0){
            break;
        }
        //NAO referenciada e modificada
        // (R=0 , M=1)
        else if(allPags[i]->r == 0 && allPags[i]->m > 0){
            pageSujaNRU++;
            break;
        }
        //referenciada e NAO modificada
        // (R=1 , M=0)
        else if(allPags[i]->r > 0 && allPags[i]->m == 0){
            break;
        }
        //referenciada e modificada
        // (R=1 , M=1)
        else if(allPags[i]->r > 0 && allPags[i]->m > 0){
            pageSujaNRU++;
            break;
        }
    }

    //trocando a pagina
    allPags[i]->endrCod = endrCod >> auxShift;
    allPags[i]->r = 1;
    allPags[i]->timer = 0;
    if(rw == 'W')
        allPags[i]->m = 1;
    else
        allPags[i]->m = 0;

    return allPags;
}

void typeNru(char *filePath, int numPages, int auxShift,int
pageSize,int memSize){
```

```
unsigned int endrCod;
char rw;
FILE *arq;
arq = fopen(filePath,"r");
if (arq == NULL){
    printf("Problemas na criacao do arquivo\n");
    exit(-1);
}
// Aloca a matriz de todas as paginas
Pagina **allPags = (Pagina**)malloc(numPages*sizeof(Pagina*));
if(allPags == NULL){
    printf("Erro ao alocar memoria para tabela de páginas.\n");
    exit(-1);
}

for(int i = 0; i < numPages; i++) {
// Aloca as paginas na matriz
    allPags[i] = (Pagina*) malloc (sizeof(Pagina));
    if(allPags[i] == NULL) {
        printf("Erro ao tentar alocar memória para página %d.\n", i);
        exit(-1);
    }
    allPags[i]->timer = 0;
    allPags[i]->endrCod = 0;
}

while((fscanf(arq,"%x %c\n", &endrCod, &rw))!=EOF){
    for(int i=0;i<numPages;i++){
        allPags[i]->timer ++;
        if (allPags[i]->endrCod  == endrCod >> auxShift){
            allPags[i]->timer = 0;
            allPags[i]->r ++;
            if(rw == 'W'){
                allPags[i]->m = 1;
                pageSujaNRU++;
            }
            else
                allPags[i]->m = 0;
            break;
        }
        else if (allPags[i]->endrCod == 0 && i < numPages - 1){
```

```
                //Pagina ainda nao exsite e memoria nao cheia
                allPags[i] -> endrCod = endrCod >> auxShift;
                allPags[i]->r++;
                if(rw == 'W'){
                    allPags[i]->m = 1;
                    pageSujaNRU++;
                }
                else
                    allPags[i]->m = 0;
                break;
            }

            if (i == numPages-1){
                //pagina nao existe na tabela
                //tabela cheia
                pageFaultsNRU++;
                allPags = NRU(allPags, numPages, rw, endrCod,auxShift);
                break;
            }
        }
    }

    printf("Arquivo de entrada: %s\n", filePath);
    printf("Tamanho da memoria fisica: %d MB\n", memSize);
    printf("Tamanho das páginas: %d KB\n", pageSize);
    printf("Algoritmo de substituiçao: NRU\n");
    printf("Numero de Faltas de Paginas: %d\n", pageFaultsNRU);
    printf("Numero de Paginas Escritas: %d\n", pageSujaNRU);

}
```

1.5 Código Fonte (LFU)

```
/*
Nomes:
Tharik Lourenço
Suemy Inagaki
Lab 7 - Gerenciamento de Memória
Sistemas Operacionais
*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pagina.h"
#include "lfu.h"
int pageFaultsLFU = 0, pageSujaLFU = 0;


struct reg {
    Pagina* conteudo;
    struct reg *prox;
};
typedef struct reg No;


//insere no fim
void inserePag (Pagina* x, No *le)
{
    No *p = le;
    No *nova;
    nova = malloc (sizeof (No));
    while (p->prox != NULL)
        p = p->prox;
    p->prox = nova;
    nova->conteudo = x;
    nova->conteudo->r = 0;
    nova->prox = NULL;
}

No* removeMenor(No* le, int menor){

    No* ant = NULL;
    No* aux = le;
    while(aux != NULL)
    {
        if(aux->conteudo->r == menor)
            break;
        ant = aux;
        aux = aux->prox;
    }
```

```
    if(ant == NULL){
        le = le->prox;
    }
    else if(aux != NULL){
        ant = aux->prox;
    }
    return le;
}


No* LFU(Pagina* p, No* le){
    int menor = le->conteudo->r;
    //acha o menor r
    for(No* i = le; i != NULL; i = i->prox){
        if(i->conteudo->r < menor){ //se o r for menor que o menor
            menor = i->conteudo->r;
        }
    }
    le = removeMenor(le, menor);
    inserePag(p, le);
    return le;
}



void typeLfu(char *filePath, int numPages, int auxShift,int
pageSize,int memSize){
    unsigned int endrCod;
    char rw;
    FILE *arq;

    arq = fopen(filePath,"r");
    if (arq == NULL){
        printf("Problemas na criacao do arquivo\n");
        exit(-1);
    }
    No * heap = (No *)malloc(sizeof(No));
    heap->conteudo = NULL;
    heap->prox = NULL;
    No * auxHeap;
    while((fscanf(arq,"%x %c\n", &endrCod, &rw))!=EOF){
```

```
    auxHeap = heap;
    Pagina* auxPagina = (Pagina *)malloc(sizeof(Pagina));
    auxPagina->endrCod = endrCod>> auxShift;
    if(rw == 'W') {
        auxPagina->m = 1;
        pageSujaLFU++;
    }
    else
        auxPagina->m = 0;

    for(int i=0;i<numPages;i++ ){
        if (heap->conteudo == NULL){
            heap->conteudo = auxPagina;
            break;
        }
        heap->conteudo->timer ++;
        if(auxHeap->conteudo->endrCod == auxPagina->endrCod){
            //ja existe
            auxHeap->conteudo->timer=0;
            auxHeap->conteudo->r++;
            break;
        }
        else if (auxHeap->prox == NULL && i < numPages-1){
            //nao existe
            //fila nao cheia
            auxPagina->r++;
            inserePag(auxPagina, heap);
            break;
        }
        else if (i == numPages-1){
            //pagina nao existe
            //tabela cheia -> lfu
            pageFaultsLFU++;
            heap = LFU(auxPagina, heap);
            auxHeap = heap;
            break;
        }
        auxHeap = auxHeap->prox;
    }
}
printf("Arquivo de entrada: %s\n", filePath);
```

```
    printf("Tamanho da memoria fisica: %d MB\n", memSize);
    printf("Tamanho das páginas: %d KB\n", pageSize);
    printf("Algoritmo de substituiçao: LFU\n");
    printf("Numero de Faltas de Paginas: %d\n", pageFaultsLFU);
    printf("Numero de Paginas Escritas: %d\n", pageSujaLFU);
}
```

1.6 Código Fonte (FIFO 2nd Chance)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pagina.h"
#include "fifo2.h"
#include "lfu.h"
int pageFaultsFIFO = 0, pageSujaFIFO = 0;


struct reg {
    Pagina* conteudo;
    struct reg *prox;
};
typedef struct reg No;

void imprime (No *le) {
   No *p;
   for (p = le; p != NULL; p = p->prox)
      printf ("%d -", p->conteudo->endrCod);
    printf("\n");
}
void insere (Pagina* x, No *le)
{
    No *p = le;
    No *nova;
    nova = malloc (sizeof (No));
    while (p->prox != NULL)
        p = p->prox;
    p->prox = nova;
    nova->conteudo = x;
    nova->conteudo->r = 0;
    nova->prox = NULL;
```

```
}

No* removeCelula (No *p)
{
    No *lixo;
    lixo = p;
    p = lixo->prox;
    free (lixo);
    return p;
}

No* arruma(No* le){
    //tira o primeiro elemento e coloca no final
    Pagina* aux = le->conteudo;
    No *lixo = le;
    insere(aux,le);
    le = lixo->prox;
    return le;
}

No* Fifo2(Pagina* p, No* le){
    while (le->conteudo->r > 0)
    {
        le = arruma(le);
    }

    le = removeCelula(le);
    insere(p, le);
    return le;
}


void typeFifo2(char *filePath, int numPages, int auxShift,int
pageSize,int memSize){
    unsigned int endrCod;
    char rw;
    FILE *arq;

    arq = fopen(filePath,"r");
    if (arq == NULL){
        printf("Problemas na criacao do arquivo\n");
```

```
    exit(-1);
}
No * heap = (No *)malloc(sizeof(No));
heap->conteudo = NULL;
heap->prox = NULL;
No * auxHeap;
while((fscanf(arq,"%x %c\n", &endrCod, &rw))!=EOF){
    auxHeap = heap;
    Pagina* auxPagina = (Pagina *)malloc(sizeof(Pagina));
    auxPagina->endrCod = endrCod>> auxShift;
    if(rw == 'W') {
        pageSujaFIFO++;
        auxPagina->m = 1;
        }
    else
        auxPagina->m = 0;

    for(int i=0;i<numPages;i++ ){
        if (heap->conteudo == NULL){
            heap->conteudo = auxPagina;
            break;
        }
        heap->conteudo->timer ++;
        if(auxHeap->conteudo->endrCod == auxPagina->endrCod){
            //ja existe

            auxHeap->conteudo->timer=0;
            auxHeap->conteudo->r++;

            break;
        }
        else if (auxHeap->prox == NULL){
            //nao existe
            //fila nao cheia
            auxPagina->r++;
            insere(auxPagina, heap);
            break;
        }
        else if (i == numPages-1){
            //pagina nao existe
            //tabela cheia -> fifo2 chance
```

```
                pageFaultsFIFO++;
                heap = Fifo2(auxPagina, heap);
                auxHeap = heap;
                break;
            }
            auxHeap = auxHeap->prox;
        }

    }

    printf("Arquivo de entrada: %s\n", filePath);
    printf("Tamanho da memoria fisica: %d MB\n", memSize);
    printf("Tamanho das páginas: %d KB\n", pageSize);
    printf("Algoritmo de substituiçao: FIFO2\n");
    printf("Numero de Faltas de Paginas: %d\n", pageFaultsFIFO);
    printf("Numero de Paginas Escritas: %d\n", pageSujaFIFO);

}
```

1.7 Resultado da Execução

```
$ gcc -c fifo2.c
$ gcc -c nru.c
$ gcc -c lfu.c
$ gcc -o sim-virtual fifo2.o nru.o lfu.o sim-virtual.c
$ ./sim-virtual LFU matriz.log 8 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: LFU
Numero de Faltas de Paginas: 17708
Numero de Paginas Escritas: 67170
Tempo de execu  o = 4.563420

$ ./sim-virtual NRU matriz.log 8 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 33847
Numero de Paginas Escritas: 67176
Tempo de execu  o = 1.195288

$ ./sim-virtual FIFO2 matriz.log 8 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: FIFO2
```

```
Numero de Faltas de Paginas: 221
Numero de Paginas Escritas: 67170
Tempo de execu  o = 3.315142


./sim NRU simulador.log 8 16
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 73301
Numero de Paginas Escritas: 160987
Tempo de execu  o = 3.445997

./sim FIFO2 simulador.log 8 16
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: FIFO2
Numero de Faltas de Paginas: 1599
Numero de Paginas Escritas: 160983
Tempo de execu  o = 4.970342

./sim NRU compilador.log 8 16
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 2156
Numero de Paginas Escritas: 107231
Tempo de execu  o = 2.031088

./sim LFU compilador.log 8 16
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: LFU
Numero de Faltas de Paginas: 7303
Numero de Paginas Escritas: 107184
Tempo de execu  o = 3.679580

./sim FIFO2 compilador.log 8 16
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: FIFO2
Numero de Faltas de Paginas: 422
Numero de Paginas Escritas: 107184
Tempo de execu  o = 3.333899

./sim NRU compressor.log 8 16
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 0
```

```
Numero de Paginas Escritas: 122419
Tempo de execu  o = 0.791401

./sim LFU compressor.log 8 16
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: LFU
Numero de Faltas de Paginas: 0
Numero de Paginas Escritas: 122419
Tempo de execu  o = 0.935386

./sim FIFO2 compressor.log 8 16
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 8 KB
Algoritmo de substitui ao: FIFO2
Numero de Faltas de Paginas: 0
Numero de Paginas Escritas: 122419
Tempo de execu  o = 0.944241

./sim NRU matriz.log 16 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 16 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 173480
Numero de Paginas Escritas: 67239
Tempo de execu  o = 1.117761

./sim FIFO2 matriz.log 16 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 16 KB
Algoritmo de substitui ao: FIFO2
Numero de Faltas de Paginas: 1247
Numero de Paginas Escritas: 67170
Tempo de execu  o = 1.857508

./sim NRU matriz.log 32 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 32 KB
Algoritmo de substitui ao: NRU
Numero de Faltas de Paginas: 289629
Numero de Paginas Escritas: 67288
Tempo de execu  o = 0.800027

./sim FIFO2 matriz.log 32 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16 MB
Tamanho das p ginas: 32 KB
Algoritmo de substitui ao: FIFO2
Numero de Faltas de Paginas: 1805
Numero de Paginas Escritas: 67170
Tempo de execu  o = 1.240008
```

1.8 Comentários

Para 8KB de tamanho de página, o NRU teve o maior número de page faults, seguido do LFU e pelo FIFO2, quando usamos o matriz.log. Quando testamos com os outros, os resultaodos foram um pouco diferentes.

Percebemos que quando temos um tamanho de página maior, temos um maior número de page faults. (podemos ver a comparação da execução para o arquivo matriz.log para 8 KB e 32 KB).

De maneira geral, podemos dizer que o simulador teve o desempenho e resultado esperados!.