Relatório do Trabalho 2 de INF 1036 - Sistemas Operacionais

Primeiro Trabalho - Gerência de Memória Virtual

Data: 11/12/2023

Alunos:
Leo Lomardo - matrícula 2020201
Lucas Lucena - matrícula 2010796

Objetivo:

Criar um simulador de memória virtual que possui os algoritmos para substituição de páginas Least Recently Used e o Not Recently Used. Para compilar, usamos como parâmetros: a) O algoritmo de substituição de página sendo simulado (NRU/LRU) b) O arquivo contendo a sequência de endereços de memória acessados (arq.log) c) O tamanho de cada página/quadro de página em KB (tamanhos a serem suportados 8 e 16 KB) d) O tamanho total de memória física hipoteticamente disponível pode variar entre 1 MB e 4MB.

Estrutura do programa:

O código fornecido simula o gerenciamento de memória virtual, onde o objetivo é gerenciar a alocação e substituição de páginas em memória, utilizando algoritmos de substituição como o Least Recently Used (LRU) e o Not Recently Used (NRU). Vamos percorrer o funcionamento principal do código:

main.c:

O programa principal recebe argumentos da linha de comando, incluindo o algoritmo de substituição, o arquivo de entrada, o tamanho das páginas e o tamanho total da memória física.
Verifica se a quantidade de argumentos é correta.
Converte os argumentos necessários para valores numéricos.
Imprime informações sobre os parâmetros fornecidos.
Chama a função go_simulator do arquivo sim-virtual.c para iniciar a simulação.

sim-virtual.c:

Estruturas de Dados:

Define duas estruturas: PF representa um quadro de página, e Page representa uma página.

Funções de Inicialização:

createPageFrame: Inicializa e retorna uma estrutura de quadros de página com base no tamanho total da memória e tamanho das páginas.

createPageTable: Inicializa e retorna uma tabela de páginas com base no tamanho das páginas.

Algoritmos de Substituição e Funções Relacionadas:

NRU: Implementa o algoritmo de substituição Not Recently Used (NRU).

swap: Realiza a troca de páginas na tabela de páginas.

LRU: Implementa o algoritmo de substituição Least Recently Used (LRU).

troca_de_paginas: Realiza a troca de páginas com base no critério de substituição (LRU ou NRU).

Função Principal da Simulação (sim-virtual):

Abre o arquivo de entrada fornecido na linha de comando.

Itera sobre as entradas do arquivo, que consistem em endereços de memória em hexadecimal e operações de leitura/escrita (R/W).

Calcula o índice da tabela de páginas com base no endereço de memória.

Verifica se a página está na memória física (tabela de páginas).

Se a página não estiver na memória, realiza a substituição utilizando o algoritmo especificado.

Atualiza estatísticas, como número de faltas de página e número de páginas escritas.

As estruturas de dados (PF e Page) são manipuladas para rastrear informações sobre o estado das páginas na memória física.

O código simula a execução de um programa que acessa diferentes páginas de memória ao longo do tempo e utiliza algoritmos de substituição para gerenciar a memória virtual.

O desempenho do gerenciador é avaliado com base nas estatísticas de faltas de página e páginas escritas.

Solução:

Código do main.c:

```c
#include <stdio.h>
#include <stdlib.h>
#include "sim-virtual.h"

int main (int args, char* argv[]){
   if(args != 5){
       perror("Quantidade de argumentos invalida.");
       exit(1);
   }

   int pageSize = atoi(argv[3]);
   int totalMem = atoi(argv[4]);

   printf("Arquivo de entrada: %s\n",argv[2]);
   printf("Tamanho de memoria fisica: %s MB\n", argv[4]);
   printf("Tamanho das paginas: %s KB\n",argv[3]);
   printf("Algoritmo de substituicao: %s\n",argv[1]);

   simVirtual(totalMem, pageSize,argv);


   return 0;
}
```

Código do sim-virtual.h:

```c
typedef struct page Page;
typedef struct pageframe PF;

int findNextIns(PF *pf, int size);
int NRU(PF *pf, int pfSize);
int swap(PF *pf, Page *pt, int PTIndexOld, int PTIndexNew);
int LRU(PF *pt, int pfSize);
int changePage(PF *pf, Page* pt, int pfSize, char* criterio, int
incertIndex);
void simVirtual(int totalMemory,int pageSize, char **argv);
```

Código do sim-virtual.c:

```c
//Aluno: Lucas Daibes Rachid de Lucena -2010796
//Aluno: Leo Lomardo - 2020201

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <unistd.h>

typedef struct pageframe{
   int idPage;
   int M; //modificado
   int R; //referenciado
   int timeIn;
}PF;

typedef struct page{
   int frameIndex;
}Page;
PF* createPageFrame(int pageSize, int totalMainMem){
   int size = (totalMainMem * 1024) / pageSize;
   printf("total de itens na page frame: %d\n",size);

   PF* memoria = (PF*) malloc(sizeof(PF) * size);

   for(int i = 0; i<size; i++){
       memoria[i].idPage = -1;
       memoria[i].M = 0;
       memoria[i].R = 0;
       memoria[i].timeIn = 0;
   }
   return memoria;
}

Page* createPageTable(int pageSize){
   int totalPages = pow(2, 32 - (int)(ceil(log2(pageSize * 1024))));

   Page* pageTable = (Page*) malloc(sizeof(Page) * totalPages);
```

```
    for(int i = 0; i<totalPages; i++){
        pageTable[i].frameIndex = -1;
    }
    return pageTable;
}

int findNextIns(PF *pf, int size){
    for(int i =0; i< size; i++){
        if(pf[i].idPage == -1){
            return i;
        }
    }
    return -1;
}

int NRU(PF *pf, int pfSize){
    int tempo_ma = -1;
    int target_index;

    for(int i = 0; i<pfSize; i++){
        if(tempo_ma != -1){
            if(pf[i].timeIn < tempo_ma){
              tempo_ma = pf[i].timeIn;
              target_index = i;
            }
        }
        else{
          tempo_ma = pf[i].timeIn;
          target_index = i;
        }
    }
    return target_index;
}

int swap(PF *pf, Page *pt, int PTIndexOld, int PTIndexNew){

    int newIndice = pt[PTIndexOld].frameIndex;
    pt[PTIndexOld].frameIndex = -1;
    pt[PTIndexNew].frameIndex = newIndice;
```

```
    return newIndice;
}

int LRU(PF *pt, int pfSize){

    int menos_referenciada = -1;
    int target_index;

    for(int i = 0; i < pfSize; i++){
        if(menos_referenciada != -1){
            if(pt[i].R < menos_referenciada){
                menos_referenciada = pt[i].R;
                target_index = i;
            }
        }else{
            menos_referenciada = pt[i].R;
            target_index = i;
        }
    }

    return target_index;
}

int changePage(PF *pf, Page* pt, int pfSize, char* criterio, int
incertIndex){
    int target_index;

    if(strcmp(criterio,"LRU") == 0){
        target_index = LRU(pf,pfSize);
    }
    else if(strcmp(criterio,"NRU") == 0){
        target_index = NRU(pf,pfSize);
    }else{
        perror("ALGORITMO INVALIDO");
        exit(1);
    }
    return swap(pf,pt,target_index,incertIndex);
}

void simVirtual(int totalMemory, int pageSize, char **argv){
```

```
    unsigned int pageTableIndex, addr;
    unsigned int next_insert = 0;
    int countPFault = 0;
    int countPageW = 0;
    int pfSize = 0;
    int time = 0;
    char rw;

    PF* pf = createPageFrame(pageSize,totalMemory);
    Page *pt = createPageTable(pageSize);

    FILE *arq = fopen(argv[2],"r");
    if(arq == NULL){
        perror("ARQUIVO NAO ENCONTRADO\n");
        exit(1);
    }

    pfSize = (totalMemory * 1024) / pageSize;

    while(fscanf(arq,"%x %c\n", &addr, &rw) == 2){

        pageTableIndex = addr >> (int)(ceil(log2(pageSize*1024)));

        if(pt[pageTableIndex].frameIndex == -1){
            countPFault++;
            next_insert = findNextIns(pf,pfSize);

            if(next_insert != -1){
                pt[pageTableIndex].frameIndex = next_insert;
                pf[next_insert].idPage = addr;
                pf[next_insert].timeIn = time;

                if(rw == 'W'){
                    pf[next_insert].M = 1;
                    countPageW++;
                }
            }
            else{
                int newIndice =
changePage(pf,pt,pfSize,argv[1],pageTableIndex);
```

```
                pf[newIndice].idPage = addr;
                pf[newIndice].timeIn = time;

                if(rw == 'W'){
                    pf[newIndice].M = 1;
                    countPageW++;
                }
            }
        }else{
            pf[pt[pageTableIndex].frameIndex].R++;
        }
    }
    time++;
    fclose(arq);

    printf("Numero de Faltas de Paginas: %d\n" , countPFault);
    printf("Numero de Paginas Escritas: %d\n", countPageW);
}
```

Resultados:

Testando o código com simulador.log em NRU e LRU, as saídas foram:

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU simulador.log 8 4
Arquivo de entrada: simulador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 862850
Numero de Paginas Escritas: 118980

[lucas@fedora Trabalho2]$ ./sim-virtual LRU simulador.log 8 4
Arquivo de entrada: simulador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
```

```
Algoritmo de substituicao: LRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 862850
Numero de Paginas Escritas: 118980
```

```
[lucas@fedora Trabalho2]$ ./sim-virtual LRU simulador.log 32 4
Arquivo de entrada: simulador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 885848
Numero de Paginas Escritas: 136184

[lucas@fedora Trabalho2]$ ./sim-virtual NRU simulador.log 32 4
Arquivo de entrada: simulador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 885848
Numero de Paginas Escritas: 136184
```

Para o simulador.log, os dois algoritmos tiveram o mesmo resultado, tanto para o número de faltas de página quanto para páginas escritas. Esse resultado se manteve em ambos os testes, o de página de 8kb e de 32kb.
Testando o código com compilador.log em NRU e LRU, as saídas foram:

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU compilador.log 8 4
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 659398
Numero de Paginas Escritas: 20074

[lucas@fedora Trabalho2]$ ./sim-virtual LRU compilador.log 8 4
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: LRU
```

```
total de itens na page frame: 512
Numero de Faltas de Paginas: 660697
Numero de Paginas Escritas: 20051
```

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU compilador.log 32 4
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 759400
Numero de Paginas Escritas: 34563

[lucas@fedora Trabalho2]$ ./sim-virtual LRU compilador.log 32 4
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 759400
Numero de Paginas Escritas: 34563
```

Para compilador.log, usando a página com 8kb, o algoritmo Not Recently Used foi mais eficaz do que o Least Recently Used, já que teve menor número de faltas de página e escreveu menos páginas. Porém, quando usamos o tamanho de página de 32kb, ambos algoritmos tiveram o mesmo desempenho.

Testando o código com matriz.log em NRU e LRU, as saídas foram:

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU matriz.log 8 4
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 188993
Numero de Paginas Escritas: 35006

[lucas@fedora Trabalho2]$ ./sim-virtual LRU matriz.log 8 4
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 188822
Numero de Paginas Escritas: 34978
```

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU matriz.log 32 4
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 235005
Numero de Paginas Escritas: 40938

[lucas@fedora Trabalho2]$ ./sim-virtual LRU matriz.log 32 4
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 226700
Numero de Paginas Escritas: 41195
```

Para o matriz.log, com a página de 16kb, o algoritmo Least Recently Used foi mais eficaz do que o Not Recently Used, mesmo que não muito, já que teve menor número de faltas de página e escreveu menos páginas. Porém, se o tamanho da página for 32kb, a eficiência dos algoritmos trocam, e o Not Recently Used gera menos page faults do que o Least Recently Used, mesmo que escreva mais páginas.
Testando o código com compressor.log em NRU e LRU, as saídas foram:

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU compressor.log 8 4
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 255
Numero de Paginas Escritas: 50

[lucas@fedora Trabalho2]$ ./sim-virtual LRU compressor.log 8 4
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 8 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 512
Numero de Faltas de Paginas: 255
Numero de Paginas Escritas: 50
```

```
[lucas@fedora Trabalho2]$ ./sim-virtual NRU compressor.log 32 4
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: NRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 568206
Numero de Paginas Escritas: 16480

[lucas@fedora Trabalho2]$ ./sim-virtual LRU compressor.log 32 4
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 4 MB
Tamanho das paginas: 32 KB
Algoritmo de substituicao: LRU
total de itens na page frame: 128
Numero de Faltas de Paginas: 567940
Numero de Paginas Escritas: 16391
```

Para o compressor.log, com a página de tamanho de 8kb, os dois algoritmos tiveram o mesmo resultado, tanto para o número de faltas de página quanto para páginas escritas. Porém, com a página de 32kb, o Least Recently Used foi ligeiramente mais eficiente do que o Not Recently Used, tanto para faltas de página quanto para páginas escritas.

Observações e conclusões:

O programa de simulação de memória virtual funciona perfeitamente bem, nos mostrando as diferenças entre utilizar os algoritmos LRU e NRU, juntamente com a quantidade de faltas de página e de páginas escritas, para cada tamanho de página e da memória física.

O número de faltas de páginas varia significativamente entre os diferentes arquivos de entrada e do tamanho da página indicada, demonstrando que o desempenho do gerenciador depende das características do acesso à memória de cada aplicação.

Em geral, o número de faltas de página é mais alto em comparação com o número de páginas escritas, o que é esperado, pois nem toda falta de página resulta em uma escrita.

O algoritmo ideal para substituição de páginas é aquele que escolhe antecipadamente a página que será referenciada mais distante no futuro. Esse algoritmo sempre faz a escolha ótima para minimizar o número de faltas de página. Porém, na prática, é difícil implementar esse algoritmo porque seria necessário ter todas as informações sobre os acessos futuros à memória. Um pseudocódigo desse algoritmo é:

```
função substituicao_otima(fila_referencias, frames_na_memoria):
    páginas_na_memória = array de tamanho frames_na_memoria inicializado com -1
    faltas_de_página = 0

    para cada referência na fila_referencias:
        página_referenciada = referência

        se página_referenciada não está em páginas_na_memória:
            // Encontrar a próxima referência mais distante
            página_a_substituir = -1
            maior_distancia = 0

            para cada página_na_memória:
                se página_na_memória não aparece mais nas referências futuras:
                    faltas_de_página++
                    substituir_página = página_na_memória
                    sair do loop interno

            se substituir_página é -1:
                // Todas as páginas na memória serão referenciadas no futuro
                // Escolher a página mais distante entre as referências futuras
```

```
                para cada página_na_memória:
                    distância = próxima_ocorrência_de_referência(página_na_memória,
fila_referencias)
                    se distância > maior_distancia:
                        maior_distancia = distância
                        página_a_substituir = página_na_memória

            // Substituir a página escolhida
            substituir_página_com(página_a_substituir, página_referenciada)

    retornar faltas_de_página

função próxima_ocorrência_de_referência(página, fila_referencias):
    para cada referência na fila_referencias a partir da posição atual:
        se referência é igual a página:
            retornar posição_da_referência
    retornar infinito
```