

INF1316 - 3WB

T5 - Gerenciamento de Mem[oria

João Victor Godinho Woitschach - 2011401

Para esse trabalho, foi desenvolvido um gerenciador de memória, com os algoritmos de substituição de páginas LRU e NRU

Código-fonte e suas explicações:

→ PARTE 1: ESTRUTURAS DE DADOS UTILIZADAS

Para este programa, foram criados apenas 2 TADs Page e PageFrame, visto que, na prática, não precisamos utilizar de outro para representar memória virtual.

```
struct page{  
  
    int frame_index; //indice do quadro na "memoria fisica"  
  
} typedef Page;
```

A estrutura Page tem o único campo de “frame_index”, que representa qual page frame aquela página contém. Caso o valor seja -1, aquela página não está armazenada na memória física.

```
struct pageframe{  
  
    int page_id;  
  
    int foi_modificada;  
  
    int foi_referenciada;  
  
    int time; //em que momento de tempo essa pagina foi inserida neste  
pageframe  
  
}typedef PageFrame;
```

Já a estrutura PageFrame possui algumas características, como por exemplo o conteúdo daquele page frame, se ela foi modificada alguma vez ao longo do programa, quantas vezes ela foi referenciada, e em qual iteração do programa ela foi inserida na PageFrame array.

→ PARTE 2: FUNCIONAMENTO DO CÓDIGO

```
int main (int args, char* argv){

    printf("Executando o simulador...\n");

    if(args != PARAMETERS){ //quantidade invalida de parametros
        raise(ERROR_PARAMETERS);
    }

    /*
    argv[1] --> LRU | NRU
    argv[2] --> arquivo.log
    argv[3] --> tamanho da pagina
    argv[4] --> total de memoria

    */

    int tam_pagina = char_to_int(argv[3]);
    int total_mem = char_to_int(argv[4]);

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

    go_simulator(total_mem,tam_pagina,argv);
```

```
    return 0;
}
```

Inicialmente, resgatamos os valores passados por parâmetro em argv na main, e inserimos eles na função principal do programa: “go_simulator”.

→PARTE 3: MAIN LOOP DO PROGRAMA

```
void go_simulator(int memoria_total,int tam_pagina, char * argv[]){
//vm == virtual_memory array ; pt == page_table array ; criteria ==
NRU | LRU

    unsigned int pt_index; //indice da tabela de paginas daquele hex

    unsigned int next_insert = 0; //proxima posicao vazia da tabela de
frames

    int page_fault = 0; //contador de page faults

    int time = 0; //contador de tempo

    int paginas_escritas = 0;

    int tam_pf = (memoria_total * 1000) / tam_pagina;

    //printf("tamanho total da page frame: %d\n",tam_pf);

    PageFrame* pf = create_pf(tam_pagina,memoria_total);

    Page *pt = create_page_table(tam_pagina);

    FILE *f = fopen(argv[2],"r"); //abrindo o arquivo

    unsigned int add; //endereço lido no .log

    char action; //acao a ser feita com o endereço a ser lido

    if(f == NULL) raise(NO_FILE);
```

Inicialmente, criamos a “PageFrame *pf” baseado no tamanho de cada página e memória total. O tamanho de cada página pode ser calculado a partir da divisão entre a memória total dada em MB sendo convertida em KB divididos pelo tamanho de cada página. A partir desse resultado também conseguimos criar a tabela de páginas.

```
while(!feof(f)) {

    fscanf(f,"%x %c\n", &add, &action);

    pt_index = add >> (int)(ceil(log2(tam_pagina*1000)));

    ;

    if(pt[pt_index].frame_index == -1){ //aquele elemento da tabela
de paginas nao esteja associada a nenhum pageframe

        //printf("pagina nao inserida na page frame, procurando
espaco vazio...\n");

        page_fault++;

        next_insert = find_next_insert(pf,tam_pf); //procura espaco
vazio

        if(next_insert != -1){ //encontrei um espaco vazio

            //printf("encontrei indice vazio na page frame!:
%d\n",next_insert);

            pt[pt_index].frame_index = next_insert;

            pf[next_insert].page_id = add;

            pf[next_insert].time = time;

            if(action == 'W'){

                pf[next_insert].foi_modificada = 1;

                paginas_escritas++;
```

```

    }

    }

    else{// nao encontrei um espaco vazio

        //printf("nao encontrei enspaco vazio, efetuando troca
de pagina");

        int qg =
troca_de_paginas(pf,pt,tam_pf,argv[1],pt_index);

        pf[qg].page_id = add;

        pf[qg].time = time;

        if(action == 'W'){

            pf[qg].foi_modificada = 1;

            paginas_escritas++;

        }

    }

    }else{ //aquele elemento esta relacionado a algum page frame

        pf[pt[pt_index].frame_index].foi_referenciada++;

    }

}

time++;

fclose(f);

printf("Numero de Faltas de Páginas: %d\n" ,page_fault);

printf("Numero de Páginas Escritas: %d\n",paginas_escritas);

```

O loop principal do programa acontece efetivamente neste while. Primeiramente é lido o hexadecimal do arquivo .log e seu char 'R' ou 'W'. Depois disso, é calculado qual o índice da página que representa aquele hex através de uma operação que envolve o bitshift com o endereço lido.

Com o índice da página em mãos, é possível reconhecer se aquela página foi alocada na “memória” ou não: vendo se o índice para page frame é == a -1. Caso seja, então significa que é necessário alocar este na memória física, contabilizando um *page fault*, e procurando o próximo espaço vazio de forma linear na page frame.

Caso tenha encontrado um espaço vazio, insere seus respectivos valores naquele page frame e atualiza o conteúdo daquele índice de page table, para referenciar aquele elemento de page frame, formalizando a dupla indexação.

Caso não tenha encontrado nenhum espaço vazio, então isso significa que é necessário efetuar uma troca de páginas baseada no critério passado no início do programa (LRU ou NRU)

→PARTE 4: TROCA DE PAGINAS

```
int troca_de_paginas(PageFrame *pf, Page* pt, int tam_pf, char*
criterio, int to_insert_index){ //to_insert e aquele que queremos
inserir em PageFrame

    int target_index = -1;

    int comp1, comp2;

    comp1 = strcmp(criterio, "LRU");
    comp2 = strcmp(criterio, "NRU");

    //printf("criterio : %s\n", criterio);
    //printf("valor de strcmp com LRU : %d\n", strcmp(criterio, "LRU"));
    //printf("valor de strcmp com NRU : %d\n", strcmp(criterio, "NRU"));

    if(comp1 == 0){
```

```

        target_index = lru(pf,tam_pf);

    }

    else if(comp2 == 0){

        target_index = nru(pf,tam_pf);

    }

    else raise(NO_ALGO);

    return swap(pf,pt,target_index,to_insert_index); //to insert index
--> indice que vamos inserir na pf ; target --> indice do elemento a
ser trocado
}

```

Para que haja a substituição de páginas, temos a função “troca_de_paginas”, que retorna o índice do frame que será trocado de uma página para outra.

Durante essa função para que esse índice seja revelado, é necessário saber qual dos mecanismos de substituição de páginas foram escolhidos como LRU ou NRU. Caso não seja nenhum dos dois, é levantado uma exceção.

```

int nru(PageFrame *pf, int tam_pf){

    int tempo_ma = -1; //tempo mais antigo = numero da iteracao que
aquela pagina foi inserida

    int target_index;

    for(int i = 0; i<tam_pf; i++){ //percorre page frame indice a indice
procurando o menor tempo

        if(tempo_ma != -1){

            if(pf[i].time < tempo_ma){ //tempo da interação anterior

                tempo_ma = pf[i].time;

                target_index = i;

            }

        }

    }
}

```

```

    }

    else{//primeiro menor tempo!

        tempo_ma = pf[i].time;

        target_index = i;

    }

}

return target_index;
}

```

Nesta implementação do NRU, estamos procurando a página mais antiga dentro da page frame. Quando encontrada, retornamos seu índice.

```

int lru(PageFrame *pt, int tam_pf){

    int menos_referenciada = -1;

    int target_index;

    for(int i = 0; i<tam_pf;i++){

        if(menos_referenciada != -1){

            if(pt[i].foi_referenciada < menos_referenciada){

                menos_referenciada = pt[i].foi_referenciada;

                target_index = i;

            }

        }else{

            menos_referenciada = pt[i].foi_referenciada;

            target_index = i;

        }

    }

}

```



```
    }  
  
    }  
  
    return target_index;  
}
```

Já na implementação de LRU, estamos procurando a página que foi menos referenciada dentro da page frame.

Para cada busca, é lido linearmente o vetor e todas suas posições.

→PARTE 5: RESULTADOS

ENTRADA E SAÍDA DO PROGRAMA:

Utilizando o exemplo de entrada:

```
exec p1, prioridade=3, inicio_tempo_execucao=0, tempo_total_execucao=8  
exec p2, prioridade=2, inicio_tempo_execucao=2, tempo_total_execucao=9  
exec p3, prioridade=1, inicio_tempo_execucao=3, tempo_total_execucao=11  
exec p4, prioridade=2, inicio_tempo_execucao=4, tempo_total_execucao=13  
exec p5, prioridade=1, inicio_tempo_execucao=5, tempo_total_execucao=14
```

Saída disponível no arquivo “output.txt”

EXECUÇÃO DO PROGRAMA:

Para compilar o programa, podemos utilizar do comando:

```
gcc -Wall -lm -o teste main.c mmu.c
```

Dois exemplos de comandos para rodar o código podem ser, por exemplo:

```
./teste LRU matriz.log 8 16
```

```
./teste NRU matriz.log 8 16
```

Como resultados, tivemos:

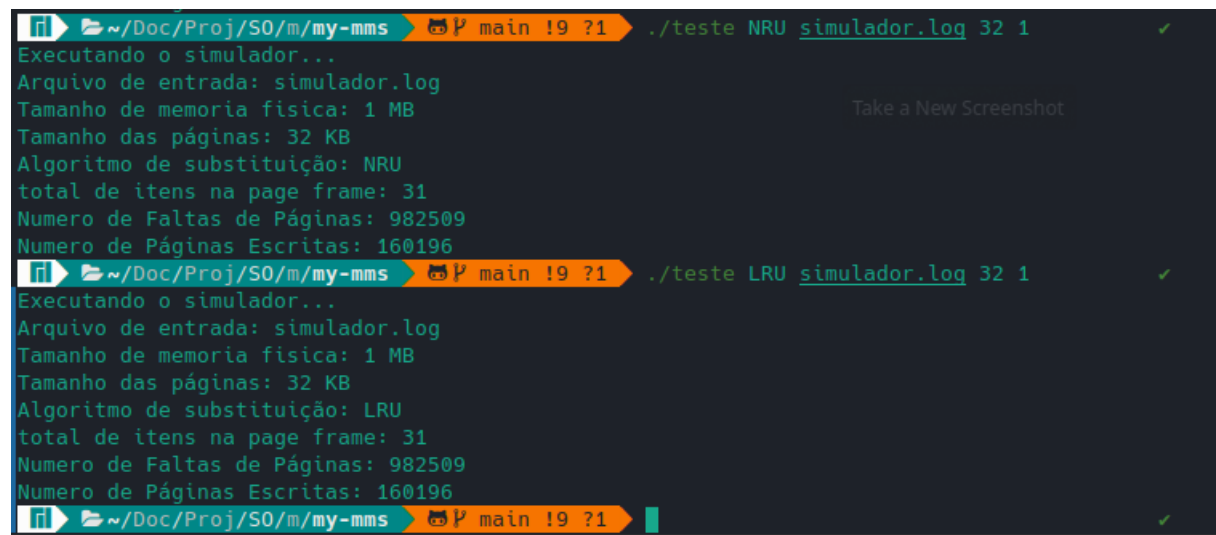


```
~/Doc/Proj/SO/m/my-mms main !9 ?1 ./teste LRU simulador.log 8 16 ✓
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho de memória física: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: LRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 133218
Numero de Páginas Escritas: 22482

~/Doc/Proj/SO/m/my-mms main !9 ?1 ./teste NRU simulador.log 8 16 ✓
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho de memória física: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: NRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 133218
Numero de Páginas Escritas: 22482

~/Doc/Proj/SO/m/my-mms main !9 ?1 ✓
```

Algo curioso que aconteceu foi que o número de page faults foi idêntico tanto do LRU quanto NRU. Isso provavelmente se deve ao caso que os critérios para substituição de página não foram muito simples.



```
~/Doc/Proj/SO/m/my-mms main !9 ?1 ./teste NRU simulador.log 32 1 ✓
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho de memória física: 1 MB
Tamanho das páginas: 32 KB
Algoritmo de substituição: NRU
total de itens na page frame: 31
Numero de Faltas de Páginas: 982509
Numero de Páginas Escritas: 160196

~/Doc/Proj/SO/m/my-mms main !9 ?1 ./teste LRU simulador.log 32 1 ✓
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho de memória física: 1 MB
Tamanho das páginas: 32 KB
Algoritmo de substituição: LRU
total de itens na page frame: 31
Numero de Faltas de Páginas: 982509
Numero de Páginas Escritas: 160196

~/Doc/Proj/SO/m/my-mms main !9 ?1 ✓
```

No caso de páginas de 32 KB e memória total de 1MB, como é possível observar, obtivemos resultados análogos. A diferença é que o número de *page faults* e páginas escritas aumentou drasticamente devido a uma diminuição drástica na memória total (anteriormente 8 MB)

```

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste LRU matriz.log 8 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: LRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 28977
Numero de Páginas Escritas: 7033

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste NRU matriz.log 8 16
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: NRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 28980
Numero de Páginas Escritas: 7004

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1

```

Para o programa “matriz.log”, não obtivemos um número idêntico de *page faults* ou de páginas escritas, mas ainda sim foram muito próximos um do outro.

```

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste LRU compilador.log 8 16
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: LRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 10015
Numero de Páginas Escritas: 3681

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste NRU compilador.log 8 16
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: NRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 10047
Numero de Páginas Escritas: 3685

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1

```

O mesmo vale para “compilador.log”

```

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste NRU compressor.log 8 16
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: NRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 255
Numero de Páginas Escritas: 50

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1  ./teste LRU compressor.log 8 16
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho de memoria fisica: 16 MB
Tamanho das páginas: 8 KB
Algoritmo de substituição: LRU
total de itens na page frame: 2000
Numero de Faltas de Páginas: 255
Numero de Páginas Escritas: 50

~/Documents/Projects/SO/my-mms/my-mms  P main !9 ?1

```

Já no caso do “compressor.log”, foram obtidos resultados mais próximos do “simulador.log

Com isso, podemos concluir que a eficiência de ambos, na maneira em que foram implementadas, são muito próximas uma das outras, e alguns casos idêntica.