Sistemas Operacionais - Trabalho 6 - Gerência de Memória
Pedro Machado Peçanha - 2110535
Eric Ruas Leão - 2110694

Execuções do código:

LRU)
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 2393
Número de Páginas Escritas: 682

Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 4427
Número de Páginas Escritas: 692

Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 254
Número de Páginas Escritas: 49

Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 171
Número de Páginas Escritas: 34

Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 2211
Número de Páginas Escritas: 768

Executando o simulador...

Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 2004
Número de Páginas Escritas: 658

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 3476
Número de Páginas Escritas: 1582

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: LRU
Número de Faltas de Páginas: 3562
Número de Páginas Escritas: 1450

NRU)

Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 3618
Número de Páginas Escritas: 933

Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 28998
Número de Páginas Escritas: 2613

Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB

Algoritmo de substituição: NRU
Número de Faltas de Páginas: 171
Número de Páginas Escritas: 34

Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 24396
Número de Páginas Escritas: 2548

Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 179724
Número de Páginas Escritas: 11210

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 50868
Número de Páginas Escritas: 2873

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: NRU
Número de Faltas de Páginas: 108992
Número de Páginas Escritas: 13868

OPR)

Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 2380
Número de Páginas Escritas: 678

Executando o simulador...

Arquivo de entrada: compilador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 1974
Número de Páginas Escritas: 503

Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 254
Número de Páginas Escritas: 49

Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 171
Número de Páginas Escritas: 34

Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 2205
Número de Páginas Escritas: 766

Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 1606
Número de Páginas Escritas: 55

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 8KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 3371
Número de Páginas Escritas: 1571

Executando o simulador...

Arquivo de entrada: simulador.log
Tamanho da memoria fisica: 16MB
Tamanho das páginas: 32KB
Algoritmo de substituição: OPR
Número de Faltas de Páginas: 2334
Número de Páginas Escritas: 1120

Algoritmo geral:
A função principal calcula a capacidade total da tabela e cria a mesma, armazenando um ponteiro para um vetor (onde cada página será armazenada), a capacidade total e o número de páginas que foram adicionadas na tabela. Após isso, faz-se um loop que percorre por todo log, lendo/escrevendo páginas na tabela implementando os algoritmos que foram pedidos no enunciado.

Algoritmo do LRU:
O algoritmo LRU é de simples implementação, nele basta buscar na tabela qual é a página que está há mais tempo sem ser referenciada, que pode ser acessado através do último acesso que é armazenado na estrutura.

Algoritmo do NRU:
No algoritmo NRU, necessitamos de um bit adicional, o BR, que indica se a página foi usada recentemente. Quando uma página é carregada para a memória, esse bit é alterado para 1 (BR = 1) e, periodicamente, ele é reiniciado (BR = 0). No trabalho, “resetamos” o BR a cada 10 segundos. Além disso, utilizamos em conjunto com o bit de referência, o bit de modificação. Sendo assim, temos 4 classes de páginas, sendo o primeiro número o BR, e o segundo o M. Classe 1, 00, Classe 2, 01, Classe 3, 10, Classe 4, 11, sendo que a Classe 1 é a mais propícia a ser substituída, e a 4 a menos propícia.

Algoritmo do OPR:
O Algoritmo de Substituição de Páginas Ótimo (Optimal Page Replacement) é um algoritmo de substituição de páginas usado para lidar com problemas de falta de memória. O algoritmo funciona mantendo um registro de todas as páginas acessadas nos últimos tempos e, quando uma nova página precisa ser carregada na memória, a página que nunca será acessada novamente, ou que será acessada mais futuramente, caso todas sejam usadas novamente, é removida. Enquanto o LRU e o NRU se baseiam na ideia de que uma página que foi acessada a pouco tempo provavelmente será acessada futuramente no futuro, o OPR analisa se, de fato, tal página não será usada, pois, diferentemente dos outros algoritmos, têm acesso às páginas que serão utilizadas futuramente. Por conta disso, o mesmo se comporta melhor que os algoritmos de NRU e LRU, sempre fazendo a melhor decisão possível dado o quão longe no futuro se pode ver. O OPR acaba tendo menos page faults que os outros dois nos dois formatos, com páginas de 8KB e de 32KB.

Motivo da diferença de page faults:

Como o número de quadros é proporcional à divisão do tamanho da memória física pelo tamanho da página, quando temos páginas maiores, temos menos frames e, assim, temos mais chances de que um page fault ocorra. Em relação ao OPR, o número de page faults é pequeno, e aumenta quando aumentamos o tamanho das páginas nesses casos em específico pois, pelo que entendemos, temos mais daqueles page faults primários, quando a página acaba de entrar no sistema. Sendo assim, o aumento de page faults que ocorre é menor que a diferença dos page faults primários atuais e os page faults primários anteriores.

Código:

```
/*
Pedro Machado Peçanha - 2110535
Eric Ruas Leão - 2110694
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define TRUE 1
#define FALSE 0

typedef struct Quadro {
  int R;
  int M;
  int acesso;
  int BR;
} quadro;

typedef struct Tabela {
  int capacidade;
  int qtd;
  quadro **array;
} tabela;

void (*algoritmo)();

int tempo = 0;
int tam_quadro, tam_mem;
char rw;
int numlogs = 0;
int *listalogs;
int loop = 0;

int modifica(int endereco) {
  int tmp = tam_quadro, c = 0;
  while (tmp > 1) {
```

```
    tmp = tmp >> 1;
    c++;
  }
  return endereco >> (c + 10);
}

void LRU(tabela *vetor, int endereco) {
  unsigned int valor = -1;
  int pos = 0;
  for (int c = 0; c < vetor->capacidade; c++) {
    if (valor > vetor->array[c]->acesso) {
      pos = c;
      valor = vetor->array[c]->acesso;
    }
  }
  vetor->array[pos]->R = modifica(endereco);
  vetor->array[pos]->M = FALSE;
  vetor->array[pos]->acesso = tempo;
}

void NRU(tabela *vetor, int endereco) {

  int bitM = TRUE;
  int bitBR = TRUE;
  /*
  M - TRUE | BR - TRUE -> menos
  M - FALSE | BR - TRUE
  M - TRUE | BR - FALSE
  M - FALSE | BR - FALSE -> mais!!!
  */
  int pos = 0;
  for (int c = 0; c < vetor->capacidade; c++) {

    if (bitBR >= vetor->array[c]->BR && bitM >= vetor->array[c]->M) {
      bitBR = vetor->array[c]->BR;
      bitM = vetor->array[c]->M;
      pos = c;
    }

    if (bitM == FALSE && bitBR == FALSE) {
      break;
    }
  }
  vetor->array[pos]->R = modifica(endereco);
  vetor->array[pos]->M = FALSE;
  vetor->array[pos]->acesso = tempo;
}
```

```
void OPR(tabela *vetor, int endereco) {
  int ultimo = -1;
  int pos = 0;
  for (int i = 0; i < vetor->capacidade; i++) {
    for (int j = loop; j < numlogs; j++) {
      if (vetor->array[i]->R == modifica(listalogs[j])) {
        if (ultimo < j) {
          ultimo = j;
          pos = i;
        }
        break;
      } else if (j == (numlogs - 1)) {
        pos = i;
        vetor->array[pos]->R = modifica(endereco);
        vetor->array[pos]->M = FALSE;
        vetor->array[pos]->acesso = tempo;
        return;
      }
    }
  }
  vetor->array[pos]->R = modifica(endereco);
  vetor->array[pos]->M = FALSE;
  vetor->array[pos]->acesso = tempo;
}

void adiciona(tabela *vetor, int endereco) {
  if (vetor->qtd < vetor->capacidade) {
    vetor->array[vetor->qtd]->R = modifica(endereco);
    vetor->array[vetor->qtd]->M = FALSE;
    vetor->array[vetor->qtd]->BR = FALSE;
    vetor->qtd++;
    return;
  }
  algoritmo(vetor, endereco);
}

tabela *cria_tabela(int c) {
  tabela *vetor;
  vetor = (tabela *)malloc(sizeof(tabela));
  if (vetor == NULL) {
    perror("erro");
    exit(1);
  }
  vetor->qtd = 0;
  vetor->capacidade = c;
  vetor->array = (quadro **)malloc(sizeof(quadro *) * vetor->capacidade);
  if (vetor->array == NULL) {
    perror("erro");
```

```
    exit(1);
  }
  for (int i = 0; i < vetor->capacidade; i++) {
    vetor->array[i] = (quadro *)malloc(sizeof(quadro));
    vetor->array[i]->acesso = -1;
    vetor->array[i]->BR = FALSE;
    vetor->array[i]->M = FALSE;
    vetor->array[i]->R = FALSE;
  }
  return vetor;
}

int procura(tabela *vetor, int endereco) {
  for (int c = 0; c <= vetor->qtd && c < vetor->capacidade; c++) {
    if (vetor->array[c]->R == modifica(endereco)) {
      vetor->array[c]->acesso = tempo;
      vetor->array[c]->BR = TRUE;
      vetor->array[c]->M = FALSE;
      if (rw == 'W') {
        vetor->array[c]->M = TRUE;
      }
      return 0;
    }
  }
  return 1;
}

void libera(tabela *vetor) {
  for (int c = 0; c < vetor->capacidade; c++) {
    free(vetor->array[c]);
  }
  free(vetor->array);
  free(vetor);
}

int main(int argc, char *argv[]) {
  if (argc != 5) {
    perror("Faltam argumentos à serem passados");
    exit(1);
  }
  char *algoritmo_pag = argv[1], *log_nome = argv[2];
  if (strcmp("LRU", algoritmo_pag) && strcmp("NRU", algoritmo_pag) &&
      strcmp("OPR", algoritmo_pag)) {
    perror("Apenas os algoritmos LRU, NRU e OPR são aceitos");
    exit(1);
  }
  if (!strcmp("LRU", algoritmo_pag)) {
    algoritmo = LRU;
```

```
} else if (!strcmp("NRU", algoritmo_pag)) {
  algoritmo = NRU;
} else {
  algoritmo = OPR;
}
FILE *log = fopen(log_nome, "r");
if (!log) {
  perror("Não foi possivel abrir o arquivo log");
  exit(1);
}
tam_quadro = atoi(argv[3]), tam_mem = atoi(argv[4]);
if (tam_quadro < 8 || tam_quadro > 32) {
  perror(
      "O tamanho de cada página/quadro de página deve variar de 8 a 32 KB");
  exit(1);
}
if (tam_mem < 1 || tam_mem > 16) {
  perror("O tamanho total de memória física hipoteticamente disponível pode "
         "variar de 1 MB a 16 MB");
  exit(1);
}
int faltas = 0, escritas = 0, addr;
int capacidade = (tam_mem * 1024) / tam_quadro;
// int capacidade = 7;
tabela *vetor;
vetor = cria_tabela(capacidade);
int a = 0;
while (fscanf(log, "%x %c ", &addr, &rw) == 2)
  numlogs++;
fclose(log);
listalogs = (int *)malloc(sizeof(int) * numlogs);
log = fopen(log_nome, "r");
while (fscanf(log, "%x %c ", &addr, &rw) == 2) {
  listalogs[loop] = addr;
  loop++;
}
fclose(log);
loop = 0;
log = fopen(log_nome, "r");
while (fscanf(log, "%x %c ", &addr, &rw) == 2) {
  if (rw == 'R') {
    // printf("ler\n");
    if (procura(vetor, addr)) {
      // printf("nao achou\n");
      faltas++;
      adiciona(vetor, addr);
    }
    // else
```

```
      // printf("achou\n");
    } else if (rw == 'W') {
      // printf("escrita\n");
      if (procura(vetor, addr)) {
        // printf("nao achou\n");
        faltas++;
        adiciona(vetor, addr);
        escritas++;
      }
      // else
      // printf("achou\n");
    } else {
      perror("O arquivo esta escrito de forma errada");
      exit(1);
    }
    tempo++;
    // sleep(1);
    // resetando o bit de acesso em caso de tempo
    if (!(tempo % 10)) {
      for (int i = 0; i < vetor->capacidade; i++) {
        vetor->array[i]->BR = FALSE;
      }
    }
    loop++;
  }
  libera(vetor);
  free(listalogs);
  fclose(log);
  {
    printf("Executando o simulador...\n");
    printf("Arquivo de entrada: %s\n", log_nome);
    printf("Tamanho da memoria fisica: %dMB\n", tam_mem);
    printf("Tamanho das páginas: %dKB\n", tam_quadro);
    printf("Algoritmo de substituição: %s\n", algoritmo_pag);
    printf("Número de Faltas de Páginas: %d\n", faltas);
    printf("Número de Páginas Escritas: %d\n", escritas);
  }
  return 0;
}
```