Universidade Presbiteriana Mackenzie
Ciência da Computação
Sistemas Operacionais – Turma 04P11
Alan Meniuk Gleizer – 10416804
Caio Vinicius Corsini Filho – 10342005

Relatório Lab 08 – Paginação

código desenvolvido

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>

#define TAMANHO_FRAME 4096
#define TAMANHO_PAGINA 4096
#define NUM_FRAMES 10
#define NUM_PAGINAS 25
#define NUM_PAGINAS_PROC 5 // por processo

// ---------- FUNÇÕES PARA SIMULAÇÃO DE PAUSA ----------
// PARA LINUX
void pausa(int milisegundos) {
    int microsegundos = milisegundos * 1000;
    usleep(microsegundos);
}

// ---------- ESTRUTURAS ----------
// Frame individual da mem física
typedef struct {
    int id;
    bool ocupado;
    bool alterado;   // indica se o conteúdo foi alterado desde que a página foi carregada no fram
    int processo_id; // ID do processo que está usando o frame (-1 se livre)
    int pagina_id;   // ID da página armazenada no frame (-1 se livre)
    char *dados;     // ponteiro para os dados armazenados no frame
} frame;

// página individual da mem virtual
typedef struct {
    int id;
    int processo_id;
```

```
    char *dados;
} pagina;

// linha individual da tabela de páginas
typedef struct {
    int end_pagina; // endereço / indice da pagina do espaço de endereçamento do pro-
cesso
    int end_frame;  // endereço / indice do frame na mem fisica
} linhaTabelaDePaginas;

// processo individual
typedef struct {
    int pid;
    int *enderecos;
    int num_enderecos;
    int tamanho_processo;
    pagina *espacoEnderecamento;
    linhaTabelaDePaginas *tabelaPaginas;
} processo;

// ---------- INICIALIZAÇÕES ----------
void inicializarMemoFisica(frame memoriaFisica[]) {
    for (int i = 0; i < NUM_FRAMES; i++) {
        memoriaFisica[i].dados = (char *)malloc(TAMANHO_FRAME * sizeof(char));
        memoriaFisica[i].id = i;
        memoriaFisica[i].ocupado = false;
        memoriaFisica[i].alterado = false;
        memoriaFisica[i].pagina_id = -1;
        memoriaFisica[i].processo_id = -1;
    }
}

// aqui, estaríamos simulando a mem. virtual em disco, que o Lucas disse ser opcional
void inicializarMemoVirtual(pagina memoriaVirtual[]) {
    for (int i = 0; i < NUM_PAGINAS; i++) {
        memoriaVirtual[i].dados = (char *)malloc(TAMANHO_PAGINA * sizeof(char));
        memoriaVirtual[i].id = i;
        memoriaVirtual[i].processo_id = -1;
    }
}

// Inicializa a tabela de páginas do processo com espaço de endereçamento
void inicializarTabela(linhaTabelaDePaginas linhas_tabela[]) {
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        linhas_tabela[i].end_pagina = i; // paginas são 0 a max
```

```
        linhas_tabela[i].end_frame = -1; // inicializa com -1 pois ainda não está na
mem. fisica
    }
}

// Inicializa um processo com as suas páginas
void inicializarProcesso(processo *proc, int pid) {
    proc->pid = pid;
    proc->num_enderecos = NUM_PAGINAS_PROC;
    proc->enderecos = (int *)malloc(NUM_PAGINAS_PROC * sizeof(int));

    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        proc->enderecos[i] = i;
    }

    proc->tamanho_processo = NUM_PAGINAS_PROC * TAMANHO_PAGINA;
    proc->espacoEnderecamento = (pagina *)malloc(NUM_PAGINAS_PROC * sizeof(pagina));
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        proc->espacoEnderecamento[i].dados = (char *)malloc(TAMANHO_PAGINA * si-
zeof(char));
    }
    proc->tabelaPaginas = (linhaTabelaDePaginas *)malloc(NUM_PAGINAS_PROC * sizeof(li-
nhaTabelaDePaginas));
    inicializarTabela(proc->tabelaPaginas);
}

// ---------- FUNÇÕES DE MAPEAMENTO ----------

// procura por um frame livre na mem. física, retorna o indíce ou -1
int buscarFrameLivre(frame memoriaFisica[]) {
    for (int i = 0; i < NUM_FRAMES; i++) {
        if (!memoriaFisica[i].ocupado) {
            return i;
        }
    }
    return -1;
}

// Aloca um frame para a página do processo
int alocarFrame(frame memoriaFisica[], processo *proc, int end_pagina) {
    int indice_frame = buscarFrameLivre(memoriaFisica);
    if (indice_frame == -1) {
        printf("LOG: Sem frames livres na memória física.\n");
        return -1;
    }
```

```
    // Atualiza o frame na memória física
    memoriaFisica[indice_frame].ocupado = true;
    memoriaFisica[indice_frame].processo_id = proc->pid;
    memoriaFisica[indice_frame].pagina_id = end_pagina;

    // checra se a página tem dados e copia-los para o frame
    if (proc->espacoEnderecamento[end_pagina].dados != NULL) {
        memcpy(memoriaFisica[indice_frame].dados, proc->espacoEnderecamento[end_pa-
gina].dados, TAMANHO_PAGINA);
    } else {
        memset(memoriaFisica[indice_frame].dados, 0, TAMANHO_PAGINA);
    }

    proc->tabelaPaginas[end_pagina].end_pagina = end_pagina;
    proc->tabelaPaginas[end_pagina].end_frame = indice_frame;

    return indice_frame;
}

// desalocar um frame da memória física de uma página de um processo
void desalocarFrame(frame memoriaFisica[], processo *proc, int end_pagina) {
    int indice_frame = proc->tabelaPaginas[end_pagina].end_frame;
    if (indice_frame != -1) {
        memoriaFisica[indice_frame].ocupado = false;
        memoriaFisica[indice_frame].processo_id = -1;
        memoriaFisica[indice_frame].pagina_id = -1;
        proc->tabelaPaginas[end_pagina].end_frame = -1;
    }
}

// Traduz um endereço virtual para físico
int traduzirEndereco(int endereco_virtual, processo *proc, frame memoriaFisica[]) {
    int pagina_id = endereco_virtual;
    int indice_frame = proc->tabelaPaginas[pagina_id].end_frame;

    if (indice_frame == -1) {
        printf("LOG: Page fault: Página %d não está na memória física.\n", pa-
gina_id);  // Indica page fault
        pausa(20);  // pausa para simulação do acesso ao disco
        return -1;
    }
    return indice_frame;
}
```

```c
void liberarMemoriaProcesso(processo *proc) {
    free(proc->enderecos);
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        free(proc->espacoEnderecamento[i].dados);
    }
    free(proc->espacoEnderecamento);
    free(proc->tabelaPaginas);
}

void liberarMemoriaFisica(frame memoriaFisica[]) {
    for (int i = 0; i < NUM_FRAMES; i++) {
        free(memoriaFisica[i].dados);
    }
}

void main() {
    frame memoriaFisica[NUM_FRAMES];
    inicializarMemoFisica(memoriaFisica);

    processo proc1;
    inicializarProcesso(&proc1, 1);

    printf("Alocando frames para as páginas do processo 1...\n");
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        alocarFrame(memoriaFisica, &proc1, i);
    }

    printf("\nTraduzindo endereços virtuais do processo 1...\n");
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        int endereco_fisico = traduzirEndereco(i, &proc1, memoriaFisica);
        if (endereco_fisico != -1) {
            printf("Endereço virtual %d mapeado para endereço físico %d\n", i, ende-
reco_fisico);
        }
    }

    printf("\nDesalocando frames do processo 1...\n");
    for (int i = 0; i < NUM_PAGINAS_PROC; i++) {
        desalocarFrame(memoriaFisica, &proc1, i);
    }

    // Liberando memoria alocada
    liberarMemoriaProcesso(&proc1);
    liberarMemoriaFisica(memoriaFisica);
}
```

print da execução