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

#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 processo

    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 \* sizeof(char));

    }

    proc->tabelaPaginas = (linhaTabelaDePaginas \*)malloc(NUM\_PAGINAS\_PROC \* sizeof(linhaTabelaDePaginas));

    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\_pagina].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", pagina\_id);  // Indica page fault

        pausa(20);  // pausa para simulação do acesso ao disco

        return -1;

    }

    return indice\_frame;

}

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, endereco\_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

A screen shot of a computer

Description automatically generated