



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Diogo Henrique Fragoso de Oliveira

Trabalho 2: Gerenciamento de Memória com Paginação: INE5611-04238A/B
(20201)

Professor: Eduardo Camilo Inacio, Dr
Florianópolis
2020

RESUMO

POSIX threads é um padrão aberto para criar e manipular *threads*. As bibliotecas que implementam a POSIX threads são chamadas Pthreads, sendo muito difundidas no universo *Unix* e outros sistemas operacionais semelhantes como *Linux* e *Solaris*. Este padrão possui implementação no compilador GCC, respectivamente, através das bibliotecas Pthreads. Neste trabalho, o uso de POSIX threads e mecanismos de sincronização é feito através do experimento realizado em linguagem C, contendo o uso de programação concorrente, programação com memória compartilhada e sincronização entre threads.

Palavras-chave: Threads; POSIX; Programação concorrente; Memória compartilhada;

SUMÁRIO

1	DESCRIÇÃO DO SIMULADOR	3
1.1	CRIAR PROCESSO	4
1.1.1	Algoritmo criar processo	4
2	VISUALIZAR MEMÓRIA	7
3	TABELA DE PÁGINAS	8
3.0.1	Algoritmo criar tabela de paginas	8

1 DESCRIÇÃO DO SIMULADOR

O experimento proposto tem o objetivo implementar o mecanismo de paginação para alocação não contígua de memória para processos. Isto inclui a implementação dos algoritmos e das estruturas de dados necessárias para isso foi feita o arquivo Estrutura de dados a baixo:

```

1 #include <stdbool.h>
2 #ifndef A_H_INCLUDED
3 #define A_H_INCLUDED
4 typedef struct Paginas {
5     int quadro;
6     int numeroPagina;
7     struct Paginas *proximaPagina;
8
9 } pagina_t;
10
11 // Estrutura do processo
12 typedef struct Processo {
13     int identificador;
14     int tamanho_bytes;
15     // Bits de pag na
16     int p;
17     // Bits de deslocamento
18     int d;
19     struct Processo *proximoProcessso;
20     struct Paginas *tabela_paginas;
21     int *enderecos;
22
23 } processo_t;
24
25 // MEMRIA F SICA
26 typedef struct Memoria {
27     // N mero de quadros
28     int numero_quadros;
29     // Bits de frames
30     int f;
31     // Bits de deslocamento
32     int d;
33     // Tamanho tototal de m moria
34     int tamanho_KB;
35     // Tamanho da p gina
36     int tm_pagina;
37     // Tamano maximo processo
38     int tamanho_max_processo;
39     // Array de Bytes de endere os
40     int *enderecos;

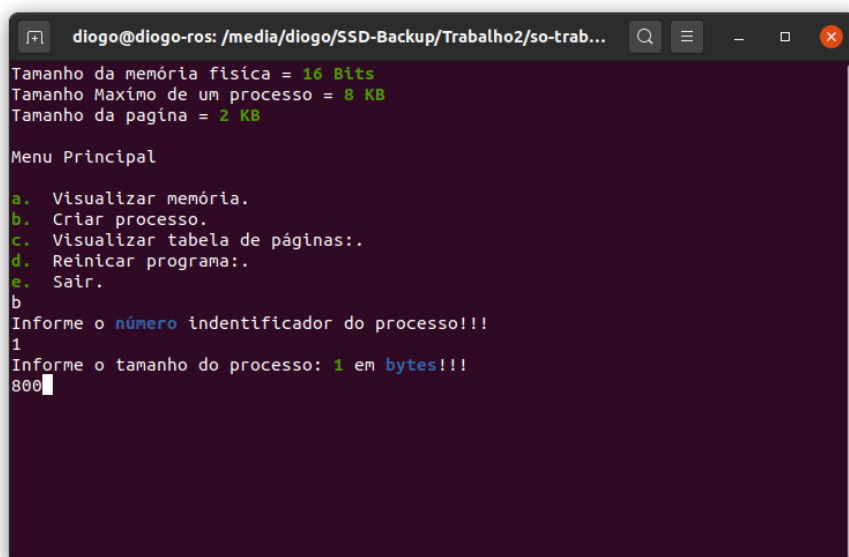
```

```
41 } memoria_t;  
42  
43 bool existeProcesso(int identificador, processo_t *processos);  
44 bool quadroVazio(int quadro, memoria_t *memoria);  
45 int inserirQuadro(int pagina, processo_t *processo, memoria_t *memoria);  
46 processo_t *pegarProcesso(int identificador, processo_t *processos);  
47 #endif
```

1.1 CRIAR PROCESSO

Para criação do processo, o usuário deve informar um número inteiro que identifica processo e o tamanho do processo em bytes conforme 1:

Figura 1 – Criar processo.



Fonte – Autores do trabalho

1.1.1 Algoritmo criar processo

A o algoritmo para criar o processo conforme solicitado no trabalho

```
1 bool adicionarProcesso(unsigned int identificador, unsigned int  
   tamanho_processo) {  
2     bool processo_adicionado = false;  
3     processo_t *tmp_processo = (processo_t *) malloc(sizeof(processo_t));  
4     pagina_t *tmp_paginas = NULL;  
5     tmp_paginas = NULL;  
6  
7     tmp_processo->identificador = identificador;
```

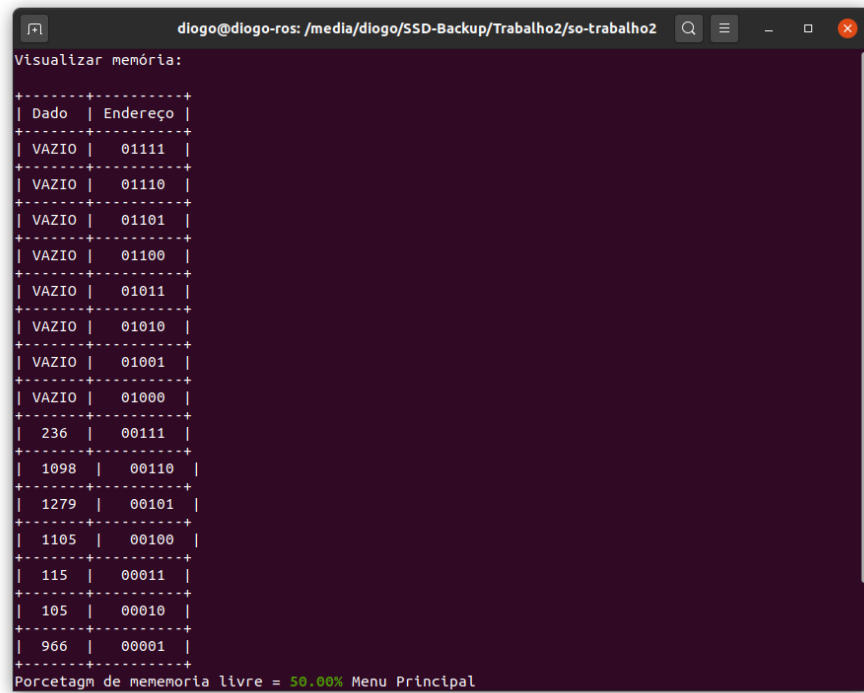
```
8     tmp_processo->tamanho_bytes = tamanho_processo;
9     tmp_processo->p = contadorDeBits(BytesInKB(tamanho_processo) /
    tam_pagina);
10    tmp_processo->d = contadorDeBits(tam_pagina * 1024);
11    tmp_processo->enderecos = malloc(kbInBytes(tamanho_processo) * sizeof(
    int));
12
13    unsigned int t = BytesInKB(tamanho_processo);
14
15    for (unsigned int z = 0; z < t; z++) {
16        tmp_processo->enderecos[z] = rand() % (tam_pagina * 1024);
17    }
18
19    unsigned int paginas = BytesInKB(tmp_processo->tamanho_bytes) /
    tam_pagina;
20
21    for (unsigned int i = 0; i < paginas; i++) {
22        if (tmp_paginas == NULL) {
23            tmp_paginas = (pagina_t *) malloc(sizeof(pagina_t));
24            tmp_paginas->proximaPagina = NULL;
25            tmp_paginas->numeroPagina = i;
26            tmp_paginas->quadro = inserirQuadro(i, tmp_processo,
    memoria_fisica);
27        } else {
28            pagina_t *teste_paginas = (pagina_t *) malloc(sizeof(pagina_t));
29            teste_paginas->proximaPagina = NULL;
30            teste_paginas->numeroPagina = i;
31            teste_paginas->quadro = inserirQuadro(i, tmp_processo,
    memoria_fisica);
32            adicionarPagina(tmp_paginas, teste_paginas);
33        }
34    }
35    tmp_processo->tabela_paginas = tmp_paginas;
36
37    // Verifica se existe processos j criados
38    if (processos == NULL) {
39        processos = (processo_t *) malloc(sizeof(processo_t));
40        processos = tmp_processo;
41    } else {
42        processo_t *current = processos;
43        while (current->proximoProcessso != NULL) {
44            current = current->proximoProcessso;
45        }
46        current->proximoProcessso = tmp_processo;
47        current->proximoProcessso->proximoProcessso = NULL;
48    }
49    return processo_adicionado = true;
```

50 }

2 VISUALIZAR MEMÓRIA

Esta opção exibir o porcentual de memória livre e cada quadro da memória física, com seu respectivo valor 2

Figura 2 – Visualizar memoria.



```
diogo@diogo-ros: /media/diogo/SSD-Backup/Trabalho2/so-trabalho2
Visualizar memória:
+-----+-----+
| Dado | Endereço |
+-----+-----+
| VAZIO | 01111 |
+-----+-----+
| VAZIO | 01110 |
+-----+-----+
| VAZIO | 01101 |
+-----+-----+
| VAZIO | 01100 |
+-----+-----+
| VAZIO | 01011 |
+-----+-----+
| VAZIO | 01010 |
+-----+-----+
| VAZIO | 01001 |
+-----+-----+
| VAZIO | 01000 |
+-----+-----+
| 236 | 00111 |
+-----+-----+
| 1098 | 00110 |
+-----+-----+
| 1279 | 00101 |
+-----+-----+
| 1105 | 00100 |
+-----+-----+
| 115 | 00011 |
+-----+-----+
| 105 | 00010 |
+-----+-----+
| 966 | 00001 |
+-----+-----+
Porcetagn de mememoria livre = 50.00% Menu Principal
```

Fonte – Autores do trabalho

3 TABELA DE PÁGINAS

Esta opção exibir o tamanho do processo e a tabela de páginas para o processo identificado pelo número inteiro informado pelo usuário 3.

Figura 3 – Visualizar memória.

```

diogo@diogo-ros: /media/diogo/SSD-Backup/Trabalho2/so-trabalho2
Informe o número indentificador do processo!!!
1
Tamanho do proceso: 1 é de 8000 bytes!!!

+-----+-----+
| Página | Quadro |
+-----+-----+
| 00     | 000    |
+-----+-----+
| 01     | 001    |
+-----+-----+
| 10     | 010    |
+-----+-----+
| 11     | 011    |
+-----+-----+
Menu Principal

a. Visualizar memória.
b. Criar processo.
c. Visualizar tabela de páginas:.
d. Retniricar programa:.
e. Sair.

```

Fonte – Autores do trabalho

3.0.1 Algoritmo criar tabela de paginas

```

1 int inserirQuadro(int pagina, processo_t *processo, memoria_t *memoria) {
2     int quadro = -1;
3     int paginas = memoria->tamanho_KB / (int)pow(2, memoria->f);
4     int deslocamentoKB = (int)pow(2, processo->d) / 1024;
5     for (int i = 0; i < memoria->tamanho_KB; i += deslocamentoKB) {
6         if (memoria->enderecos[i] == -1) {
7             for (int j = 0; j < deslocamentoKB; j++) {
8                 memoria->enderecos[i + j] = processo->enderecos[(pagina *
deslocamentoKB) + j];
9             }
10            return i / paginas;
11        }
12    }
13    return quadro;
14 }

```