

# Gerenciamento de Memória com Paginação

Igor Zimmer Gonçalves  
Higor Abreu Freiburger

## Instruções de execução

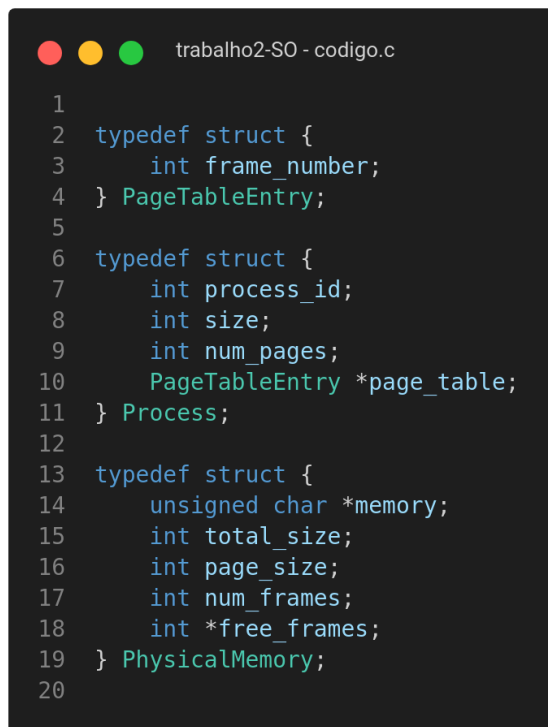
### Compilação:

```
gcc -o codigo codigo.c
```

### Execução:

```
./codigo
```

## Relatório e descrição:



```
trabalho2-S0 - codigo.c
1
2 typedef struct {
3     int frame_number;
4 } PageTableEntry;
5
6 typedef struct {
7     int process_id;
8     int size;
9     int num_pages;
10    PageTableEntry *page_table;
11 } Process;
12
13 typedef struct {
14     unsigned char *memory;
15     int total_size;
16     int page_size;
17     int num_frames;
18     int *free_frames;
19 } PhysicalMemory;
20
```

**PageTableEntry** representa uma entrada na tabela de páginas, indicando o número do frame na memória física.

**Process:** Representa um processo com um PID, tamanho em bytes, número de páginas e uma tabela de páginas (**page\_table**) que mapeia páginas para quadros físicos na memória.

**PhysicalMemory:** Estrutura que simula a memória física, com um bloco de memória (**memory**) e informações sobre o tamanho total da memória, tamanho da página, número de quadros e uma matriz de quadros livres (**free\_frames**).

```

trabalho2-SO - codigo.c

1 void init_physical_memory(PhysicalMemory *pm, int total_size, int page_size) {
2     pm->total_size = total_size;
3     pm->page_size = page_size;
4     pm->num_frames = total_size / page_size;
5     pm->memory = (unsigned char *)calloc(total_size, sizeof(unsigned char));
6     pm->free_frames = (int *)malloc(pm->num_frames * sizeof(int));
7     for (int i = 0; i < pm->num_frames; i++) {
8         pm->free_frames[i] = 1;
9     }
10 }

```

Após o usuário informar os valores de tamanho da memória física, tamanho da página e tamanho máximo de um processo, ocorre a criação e inicialização da memória física com os valores informados. Cada quadro é inicializado como livre, indicado pelo valor 1 (linha 8).

```

trabalho2-SO - codigo.c

1
2 Process create_process(int process_id, int size, int page_size, PhysicalMemory *pm) {
3     Process p;
4     p.process_id = process_id;
5     p.size = size;
6     p.num_pages = (size + page_size - 1) / page_size; // Round up
7     p.page_table = (PageTableEntry *)malloc(p.num_pages * sizeof(PageTableEntry));
8
9     for (int i = 0; i < p.num_pages; i++) {
10         int frame_found = 0;
11         for (int j = 0; j < pm->num_frames; j++) {
12             if (pm->free_frames[j] == 1) {
13                 p.page_table[i].frame_number = j;
14                 pm->free_frames[j] = 0;
15                 frame_found = 1;
16                 break;
17             }
18         }
19         if (!frame_found) {
20             printf("Not enough memory to allocate process %d\n", process_id);
21             exit(1);
22         }
23     }
24
25     for (int i = 0; i < size; i++) {
26         pm->memory[p.page_table[i / page_size].frame_number * page_size + (i % page_size)] = rand() % 256;
27     }
28
29     return p;
30 }
31

```

Caso o usuário selecione a opção de criação de um processo ele deve fornecer um PID e tamanho em bytes. Então, o programa calcula a quantidade necessária de páginas para o processo, aloca espaço na tabela de páginas e preenche os dados na memória com informações aleatórias para simular o conteúdo de um processo.

```

void display_physical_memory(PhysicalMemory *pm) {
    int free_frames_count = 0;

    printf("Physical Memory:\n");
    for (int i = 0; i < pm->num_frames; i++) {
        printf("Frame %d: ", i);
        if (pm->free_frames[i] == 1) {
            printf("Free\n");
            free_frames_count++;
        } else {
            for (int j = 0; j < pm->page_size; j++) {
                printf("%02X ", pm->memory[i * pm->page_size + j]);
            }
            printf("\n");
        }
    }

    double percent_free = (double)free_frames_count / pm->num_frames * 100.0;
    printf("Memória livre: %.2f%%\n", percent_free);
}

```

Caso o usuário selecione a opção de visualização da memória, exibe o estado da memória física e a porcentagem de memória livre.

```

trabalho2-SO - codigo.c

1 void display_page_table(Process *p) {
2     printf("\n");
3     printf("Process %d Page Table:\n", p->process_id);
4     printf("Tamanho do processo: %d bytes\n", p->size);
5     for (int i = 0; i < p->num_pages; i++) {
6         printf("Page %d -> Frame %d\n", i, p->page_table[i].frame_number);
7     }
8 }

```

Por fim, caso o usuário selecione a opção de visualização da tabela de páginas, o programa exibirá como as páginas estão mapeadas para quadros físicos na memória.

## Exemplo/caso de teste:

```
o igrzi@igrsys:~/code/ufsc/trabalho2-S0$ ./codigo
Informe o tamanho da memória física: 128
Informe o tamanho da página (quadro): 32
Informe o tamanho máximo de um processo: 64

1. Visualizar memória
2. Criar processo
3. Visualizar tabela de páginas
4. Sair
Escolha:
```

Ao executar, o usuário é requisitado a informar o tamanho da memória física, tamanho da página e tamanho máximo do processo, respectivamente. Após isso é apresentado com as opções e espera um input da opção escolhida pelo usuário

```
Escolha: 1

Physical Memory:
Frame 0: Free
Frame 1: Free
Frame 2: Free
Frame 3: Free
Memória livre: 100.00%
```

Após escolher a opção 1, caso o usuário não tenha criado nenhum processo, a memória indicará que todas as páginas se encontram livres, ou seja, sem nenhuma informação preenchida.

```
Escolha: 2
```

```
Informe o ID do processo: 1
```

```
Informe o tamanho do processo: 64
```

```
Physical Memory:
```

```
Frame 0: D4 C7 03 73 D0 66 8F 79 7D B7 53 38 90 B8 41 20 A0 2B 07 3F EB D3 D7 5D 50 7E 1C 66 A8 D0 28 7D
```

```
Frame 1: 97 2B F0 67 92 7F E0 0F 37 33 47 C7 EB 88 E7 8C B3 EE CB 9E C2 A2 FB 12 20 17 78 C9 E7 A1 46 7E
```

```
Frame 2: Free
```

```
Frame 3: Free
```

```
Memória livre: 50.00%
```

Ao criar um processo de tamanho 64 bytes e, visualizar a memória, podemos observar que foram necessárias 2 páginas para mapeamento de um processo, pois o processo de tamanho 64 bytes não caberia em apenas uma página de tamanho 32 bytes. Vemos também que atualmente a memória se encontra 50% preenchida.

```
Escolha: 3
```

```
Informe o ID do processo: 1
```

```
Process 1 Page Table:
```

```
Tamanho do processo: 64 bytes
```

```
Page 0 -> Frame 0
```

```
Page 1 -> Frame 1
```

Após escolher a opção 3, podemos ver que como são necessárias 2 páginas para armazenar o processo de PID 1, o programa mapeia cada uma das páginas para os quadros físicos na memória.