UFSC - Sistemas de Informação - INE5611-04238A/B (20241) - Sistemas Operacionais

Grupo 1

- Cristiane Rodrigues Maragno (22200494)
- Jenifer Sofia Ovejero (22200500)
- Mariana Amaral Steffen (22200511)

Documentação do Código: Gerenciamento de Memória com Paginação

Estrutura do Código

Este programa implementa um sistema básico de gerenciamento de memória com paginação, fornecendo um exemplo prático de como sistemas operacionais modernos gerenciam a memória.

Constantes

- **MAX_PROCESSES**: Define o número máximo de processos que o gerenciador de memória pode lidar simultaneamente.
- **BINARY_SIZE**: Define o tamanho do endereço binário.

Estruturas de Dados

Process

Representa um processo no sistema.

- **process_id**: ID do processo.
- **process_size**: Tamanho do processo.
- **page_table**: Tabela de páginas que mapeia as páginas lógicas para os quadros físicos.
- **num_pages**: Número de páginas do processo.
- **logical_memory**: Memória lógica do processo.

PhysicalMemory

Representa a memória física do sistema.

- **memory**: Vetor que representa a memória física.
- **size**: Tamanho da memória física.
- **page_size**: Tamanho de uma página.
- **free_frames**: Vetor que indica quais quadros estão livres.
- **num_frames**: Número de quadros na memória física.

MemoryManager

Estrutura que gerencia os processos e a memória física.

- **processes**: Vetor de processos.
- **num_processes**: Número de processos no sistema.
- **max_process_size**: Tamanho máximo permitido para um processo.

- **physical_memory**: Instância de PhysicalMemory.

Funções

initialize_memory

Inicializa a memória física e define os quadros como livres(Indicados com o -1).

Parâmetros:

- **MemoryManager *mm**
- **int physical_size**
- **int page_size**
- **int max_process_size**

Funcionalidade:

- Configura o tamanho da memória física;
- Define o tamanho da página;
- Calcula o número de quadros;
- Inicializa a memória física;
- Inicializa uma estrutura auxiliar que nos ajuda a identificar os quadros como livres.

create_process

Cria um novo processo e aloca suas páginas na memória física.

Parâmetros:

- **MemoryManager *mm**
- **int process_id**
- **int process_size**

Funcionalidade:

- Verifica se o processo obedece o tamanho máximo dos processos;
- Calcula o número de páginas necessárias;
- Verifica se há quadros disponíveis o suficiente na memória física;
- Aloca os valores na memória lógica;
- Procura frames vazios;
- Aloca as páginas na memória física;
- Atualiza a tabela de páginas do processo.

A imagem abaixo destaca a parte principal do código, onde iteramos sobre os quadros de memória em busca de um vazio, enquanto ainda houver conteúdo a ser alocado. Quando encontramos um quadro livre, utilizamos uma função que extrai o índice a partir do endereço binário. Em seguida, armazenamos o valor na memória física e atualizamos a tabela de páginas do processo com o número do quadro. Esse processo continua até que todo o conteúdo do processo tenha sido alocado.

```c
int pages_allocated = 0;
int alocated_process = 0;
for(int j = 0; j < mm->physical_memory.num_frames; j++){
    if(pages_allocated < num_pages){
        //procura espaço vazio
        if (mm->physical_memory.free_frames[j]) {
            mm->physical_memory.free_frames[j] = false;
            pages_allocated++;
            //save on page table
            int page_index = getIndex((pages_allocated - 1), mm);
            printf("Alocando quadro %d\n", j);
            new_process->page_table[page_index] = j;

            for(int m = 0; m < mm->physical_memory.page_size; m++){
                if(alocated_process < process_size){
                    alocated_process++;
                    int memory_index = (j * mm->physical_memory.page_size) + m;
                    printf("Espaço no endereço da memória física %d\n", memory_index);

                    //save on memory
                    int logical_index = alocated_process - 1;
                    mm->physical_memory.memory[memory_index] = new_process->logical_memory[logical_index];
                }else{
                    break;
                }
            }
        }
    }else{
        break;
    }
```

view_memory

Exibe o estado atual da memória física.

Parâmetros:

- **MemoryManager *mm**

Funcionalidade:

- Calcula a porcentagem de memória utilizada.
- Exibe o conteúdo da memória física.

view_page_table

Exibe a tabela de páginas de um processo específico.

Parâmetros:

- **MemoryManager *mm**
- **int process_id**

Funcionalidade:

- Exibe a tabela de páginas do processo identificado pelo process_id.

view_logical_memory

Exibe a memória lógica de um processo específico.

Parâmetros:

- **MemoryManager *mm**
- **int process_id**

Funcionalidade:

- Exibe o conteúdo da memória lógica do processo identificado pelo process_id.

display_menu

Exibe o menu de opções para o usuário.

Sem parâmetros.

get_binary

Converte um número inteiro para sua representação binária.

Parâmetros:

- **int num**
- **char *binaryStr**

binaryStringToInt

Converte uma string binária para um número inteiro.

Parâmetros:

- **char *binaryStr**

getIndex

Extrai o índice de um endereço binário com base no tamanho da página. O cálculo determina onde cortar o número binário de acordo com o tamanho especificado da página. Por exemplo, se o tamanho da página for 2, apenas o último bit será necessário, pois um único bit é suficiente para representar o índice.

Parâmetros:

- **int index**

Função Principal

A função main realiza as seguintes operações:

1. Solicita ao usuário os tamanhos da memória física, da página e do processo máximo.
2. Inicializa a memória física.
3. Exibe um menu de opções ao usuário para visualizar a memória física, criar um processo, visualizar a tabela de páginas de um processo ou sair do programa.

Observações

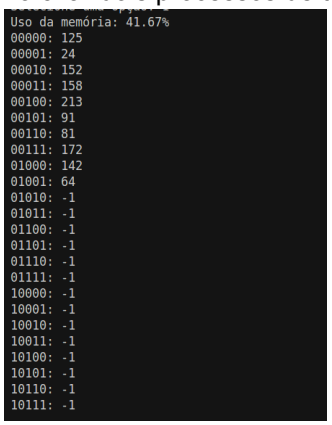
Para realizar as manipulações necessárias com valores binários, como divisão e obtenção de páginas, limitamos o tamanho a 7 bits(#define BINARY_SIZE 7). Essa limitação é importante para garantir a alocação correta de endereços e manipulação da tabela de páginas.

Além disso, nossa implementação pressupõe que os valores inseridos sejam múltiplos de 2. Números ímpares podem causar inconsistências na alocação de memória ou gerar erros no console.

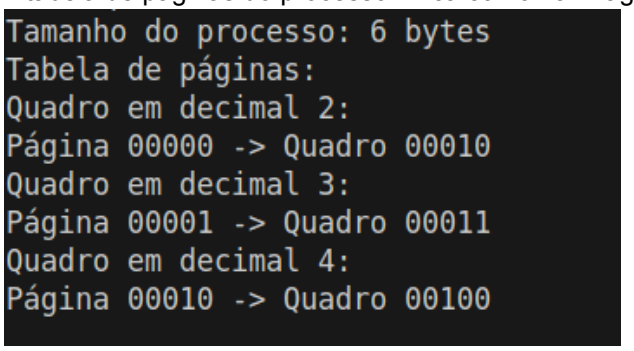
Embora o código não possua mecanismos para finalizar o processo e liberar a memória, nosso algoritmo sempre busca quadros livres, o que permite a alocação dos dados de forma aleatória.

Casos de teste e saídas

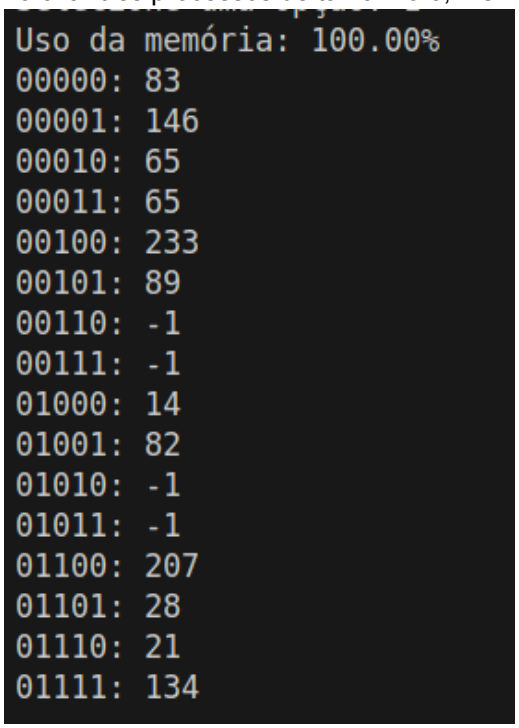
1. Situação a ser testada: Memória física 24, página 2, tamanho máximo do processo 6
    a. Ao criar dois processos de tamanho 4 e 6. A memória física resultante:



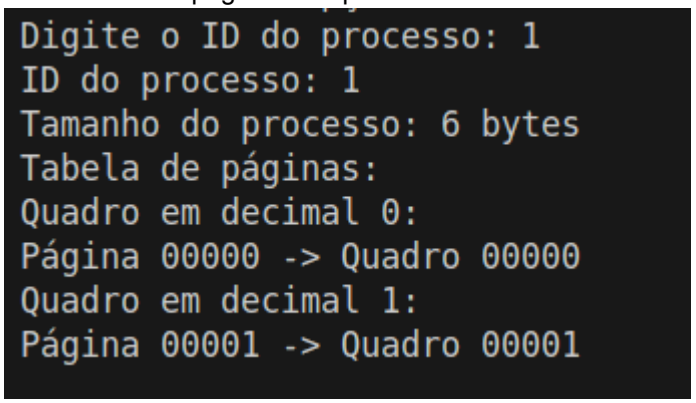
    b. A tabela de páginas do processo 2 fica como na imagem abaixo

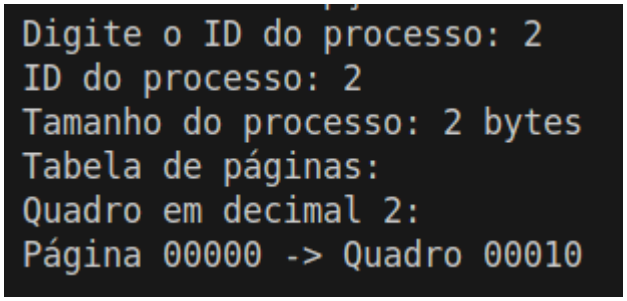


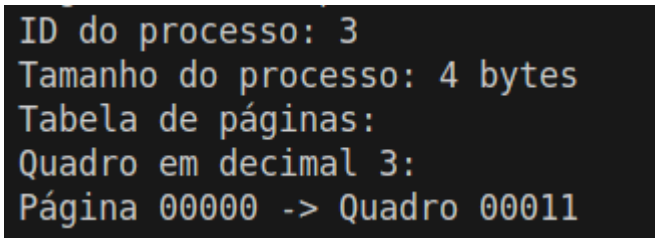
2. Situação a ser testada: Memória física 16, página 4, tamanho máximo do processo 6
    a. Ao criar três processos de tamanho 6, 2 e 4. A memória física resultante:



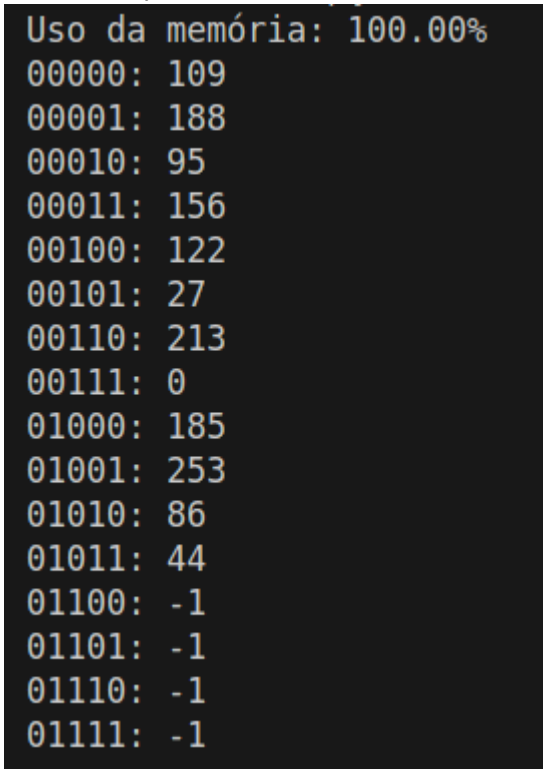
b. E as tabelas de páginas dos processos ficam assim:



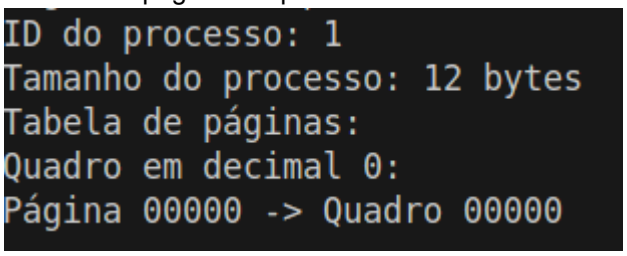




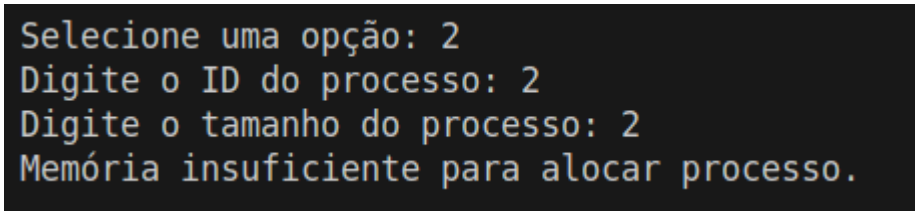
3. Situação a ser testada: Memória física 16, página 16, tamanho máximo do processo 16
   a. Ao criar um processo de tamanho 12, a memória física resultante:



b. A tabela de páginas do processo:



c. Ao tentar adicionar mais um processo, uma mensagem de erro é mostrada



Compilação

Assumindo que o compilador GCC esteja instalado, compile o programa com o seguinte comando:

gcc -o <nome_arquivo_binario> trabalho.c -lm

Execução

Execute o programa com o comando:

./<nome_arquivo_binario>