Trabalho II - SO

Integrantes: Luan Boschini, Caio Pinho, Vinicius Guazzelli

Instruções de compilação

gcc -o trabalho trabalho.c process_list.c -lm

Relatório Básico

Nosso código representa os processos utilizando a struct Process aglutinado por uma lista encadeada (com nó sentinela) chamada *Processes*.

Cada *process* contém uma *page_table*, a tabela de mapeamento de páginas (*pages*) e quadros (*frames*).

```
struct Process
{
   unsigned pid;
   unsigned size;
   unsigned *page_table;
   byte *content;
};

struct List
{
   struct Node *head;
   struct Node *tail;
};
typedef struct Node node;
typedef struct Process process;
typedef struct List processes;
```

A memória física é representada por um vetor de bytes (*memory*).

Criamos, também, uma estrutura de controle de quadros vazios, que contém um contador de *frames* livres e um bitmap para cada um deles (0 vazio, 1 preenchido).

```
struct FreeFrames
{
   unsigned count;
   byte *frames;
```

```
};
```

Se o valor de *count* é igual ou maior que o tamanho total do processo, é executado o algoritmo de mapeamento e preenchimento da memória física:

```
for (byte page = 0; page < page_c; page++)
   {

       for (byte frame; frame < sizeof(free_frames.frames); frame++)
       {
           if (free_frames.frames[frame] == 0)
           {
               unsigned frame_index = frame * page_size;
               unsigned page_index = page * page_size;
               for (byte offset = 0; offset < page_size; offset++)
               {
                   memory[frame_index + offset] = content[page_index +
offset];
               }

               new_process->page_table[page] = frame;
               free_frames.frames[frame] = 1;
               free_frames.count--;
               break;
           }
       }
   }
```

A memória física é zerada no momento da inicialização e o conteúdo da memória lógica (*content*) dos processos é preenchida com valores randômicos:

```
   memory = (byte *)malloc(memory_size);

   for (char i = 0; i < memory_size; i++)
   {
       memory[i] = 0;
   }
```

```
srand(time(NULL));
```

```
    byte *content = (byte *)malloc(size * sizeof(byte));
    for (int i = 0; i < size; i++)
    {
        content[i] = rand() % 100;
    }
```

Estrutura Auxiliar

```
void insert(processes *list, process *process)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->process = process;
    new_node->next = NULL;

    list->tail->next = new_node;
    list->tail = new_node;
}

process *get_process(processes *list, unsigned pid)
{
    struct Node *current = list->head->next;
    while (current != NULL)
    {
        if (current->process->pid == pid)
        {
            return current->process;
        }
        current = current->next;
    }
    return NULL;
}
```

Para inicializar os processos colocamos eles em uma lista ligada, com duas funções simples para inserção em um lista de processos e conseguir as informações de um dado processo pelo *process identification*(pid).

Teste

./trabalho 8 4 8

```
/home/caiopinho/projects/ufsc/trabalho-sistemas-operacionais-II  caio - ./trabalho 8 4 8
Choose an option
Visualize memory [1]
Create process [2]
Visualize page table [3]
Visualize process list [4]
Exit [5]
1
Memory visualization:

Page/Frame 0
00 00 00 00

Page/Frame 1
00 00 00 00


Porcentagem livre: 100
```

```
2
Process id: 1
Process size: 4
Memory visualization:

Page/Frame 0
47 4e 05 53


Process 1 created with size 4
```

```
3
Enter the PID to visualize the page table: 1
Page table visualization
+----------------+----------------+
| Logical Memory | Physical Memory |
+----------------+----------------+
|          0     |          0     |
+----------------+----------------+
```

```
2
Process id: 2
Process size: 2
Memory visualization:

Page/Frame 0
47 1c
Process 2 created with size 2
```

```
1
Memory visualization:

Page/Frame 0
47 4e 05 53

Page/Frame 1
47 1c 00 00


Porcentagem livre: 0
```

```
3
Enter the PID to visualize the page table: 2
Page table visualization
+----------------+----------------+
| Logical Memory | Physical Memory |
+----------------+----------------+
|          0     |          1     |
+----------------+----------------+
```