Universidade Federal de Santa Catarina

Relatório do trabalho 2
Sistemas Operacionais (INE5611)

Giovane Pimentel de Sousa
Guilherme Henriques do Carmo
Isabela Vill de Aquino

Florianópolis
2024

Instruções de compilação:

Na raíz do projeto, execute:

```
gcc -o main main.c system/system.c memory/memory.c process/process.c
utils/utils.c
```

Para executar:

```
./main
```

Ou
Caso você tenha cmake, basta executar ***cbuild.sh***, no qual compilará e executará o projeto automaticamente.

cbuild.sh:

```
#!/bin/bash
mkdir -p build
cd build
cmake --build . --config Debug --target all --
./main
```

Relatório:

O código inicia chamando *init_system()* de *system.c*, que contém toda a interface do sistema, validações de inputs e a inicialização da memória física.

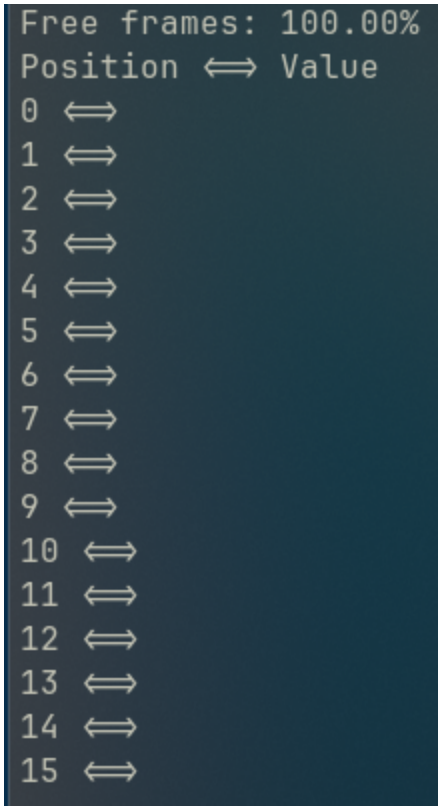
Então, depois de toda configuração inicial feita (tamanho da memória física, do quadro e da página e tamanho máximo da memória lógica), é chamada a função de inicialização da memória física:

```
void init_physical_memory() {
  head_free_frames = (Node *)malloc(size: sizeof(Node));
  head_free_frames→id_serial = 0;
  total_frames = PHYSICAL_MEMORY_SIZE / FRAME_SIZE;
  free_frames = total_frames;
  physical_memory = (char *)malloc(size: PHYSICAL_MEMORY_SIZE * sizeof(char));

  Node *cursor = head_free_frames;
  for (int i = 1; i < total_frames; i++) {
    Node *next_node = (Node *)malloc(size: sizeof(Node));
    next_node→id_serial = i;
    cursor→next = next_node;
    cursor = cursor→next;
  }
}
```

O *head_free_frames* é do tipo Node, que possui os atributos *id_serial* e *next*, que é um ponteiro para outro Node. id_serial é apenas um identificador sequencial de cada posição da memória física. Essa estrutura é apenas para mapear a memória física de fato, que é um array de *char*.

A partir daí, o sistema segue e dá a livre escolha do usuário para o que fazer. Caso o usuário escolha ver a memória física essa será a saída:

Para:
Memória física: 128 bytes.
Tamanho do quadro e da página: 8 bytes.



Ou seja, 128 / 8 = 16 frames de 8 bytes cada. Todos estão vazios, inicialmente.

Caso o usuário deseje criar um processo, será pedido um valor arbitrário de identificador para o processo, o PID, além de pedir o tamanho em bytes do processo. Se o usuário colocar um tamanho maior do que o máximo possível, ele será avisado que o processo excede o limite e o processo não será criado:

Para tamanho máximo da memória lógica: 32 bytes.

```
[1] Show memory.
[2] Create process.
[3] Show page table.
[0] Exit.
Option: 2
Enter a PID number: 1
Enter a size to the process: 64
The process size entered exceed memory limit! Enter a size lower than 32 bytes.
```

Caso um tamanho válido seja inserido, a lógica de criação de processo será chamada:

```c
int create_process(int pid, int size) {
    if (find_process(pid) != NULL) {
        printf(format: "Process with PID %d already created.\n", pid);
        return 0;
    }

    int total_pages = size / PAGE_SIZE;
    Process *new_process = (Process *)malloc(size: sizeof(Process));
    new_process->pid = pid;
    new_process->size = size;
    new_process->next_process = NULL;
    init_logical_memory(process: new_process);
    init_table_page(process: new_process);
    include_process(process: new_process);

    return 1;
}
```

Process é uma struct que contém os seguintes atributos:

```c
typedef struct Process {
    int pid;
    int size;
    char *logical_memory;
    PageTableEntry *page_table;
    struct Process *next_process;
} Process;
```

PageTableEntry é apenas uma struct para mapear as páginas com os frames. Os processos são uma lista encadeada, então cada novo processo é incluído na chamada de *include_process()*.

init_logical_memory() inicializa a memória lógica, que assim como a memória física é um array de *char*. São postos apenas letras maiusculas, para fim de visualização:

```c
static void init_logical_memory(Process *process) {
    process->logical_memory = (char *)malloc(size: process->size * sizeof(char));
    for (int i = 0; i < process->size; i++) {
        process->logical_memory[i] =
            rand() % 25 + 65; // Randomly set only printable chars without space
    }
}
```

init_table_page() também aloca na memória física:

```c
static void init_table_page(Process *process) {
  int num_pages = process->size / PAGE_SIZE;
  PageTableEntry *new_page_table =
      (PageTableEntry *)malloc(size: num_pages * sizeof(PageTableEntry));

  char page_auxiliary[PAGE_SIZE];
  int frame_allocated;

  // Memory allocation
  for (int i = 0; i < num_pages; i++) {
    for (int offset = 0; offset < PAGE_SIZE; offset++) {
      page_auxiliary[offset] = process->logical_memory[i * PAGE_SIZE + offset];
    }
    frame_allocated = allocate_frame(page: page_auxiliary);
    PageTableEntry *entry = (PageTableEntry *)malloc(size: sizeof(PageTableEntry));
    entry->page_position = i;
    entry->frame_position = frame_allocated;
    new_page_table[i] = *entry;
  }

  // Table page included
  process->page_table = new_page_table;
}
```

A alocação ocorre com a chamada de *allocate_frame()*, que passa como argumento um array do tamanho da página, para alocar um frame. A alocação ocorre até todas as páginas da memória lógica do processo terem sido alocadas.

allocate_frame() aloca um espaço do array de *char* e remove um Node da lista encadeada usada de mapeamento:

```c
int allocate_frame(char page[FRAME_SIZE]) {
  Node *anterior = get_some_frame_to_allocate();
  Node *cursor;
  if (anterior->next == NULL) {
    cursor = anterior;
  } else {
    cursor = anterior->next;
  }
  int frame_start = cursor->id_serial;

  frame_start = frame_start * FRAME_SIZE;
  for (int offset = 0; offset < FRAME_SIZE; offset++) {
    physical_memory[frame_start + offset] = page[offset];
  }

  free_frames--;
  int frame_allocated = cursor->id_serial;
  if (cursor->next != NULL) {
    anterior->next = cursor->next;
    free(ptr: cursor);
  } else {
    free(ptr: cursor);
  }

  return frame_allocated;
}
```

Para escolher um frame de alocação, foi utilizada a função *get_some_frame_to_allocate()*:

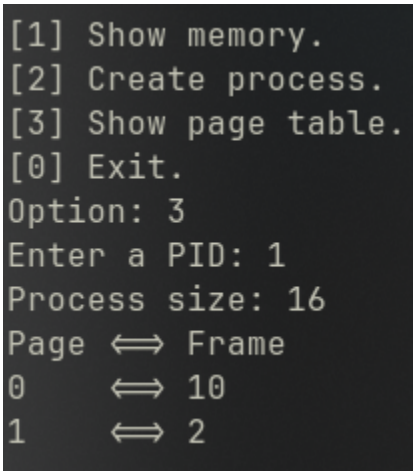
```c
static Node *get_some_frame_to_allocate() {
  Node *cursor;
  int frame;
  // If the amount of free frames are greater than 50%, they are randonmly
  // choosed
  if ((double)free_frames / total_frames >= 0.5) {

    // While runs until draw a free frame
    while (1) {
      frame = rand() % free_frames;
      cursor = head_free_frames;
      if (cursor->id_serial == frame) {
        head_free_frames = head_free_frames->next;
        cursor->next = NULL;
        return cursor;
      }

      while (cursor->next != NULL) {
        if (cursor->next->id_serial == frame) {
          // Returning the previous frame, to delete the choosed frame
          return cursor;
        }

        if (frame < cursor->id_serial) {
          break;
        }

        cursor = cursor->next;
      }
    }
    // Else, return the first free frame
  } else {
    cursor = head_free_frames;
    head_free_frames = head_free_frames->next;
    cursor->next = NULL;
    return cursor;
  }
}
```

A lógica é a seguinte: é escolhido um frame aleatoriamente entre os disponíveis, caso a quantidade de frames disponíveis seja maior ou igual a 50% do total de frames. Caso seja menor, a alocação é sequencial, sendo escolhido o primeiro frame livre. Foi escolhida essa lógica para, no geral, não tornar a alocação contígua, mas também para evitar que muitas execuções de sorteios de frames acontecessem sem que o frame sorteado estivesse livre. Até 50% foi escolhido arbitrariamente e considerado pela equipe um valor limite justo.

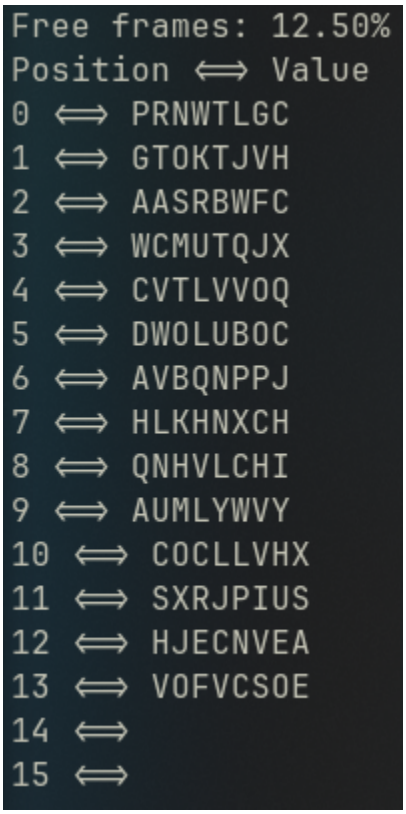
Tendo o processo sido criado, uma resposta é dada para o usuário que o processo foi criado e agora, caso o usuário deseje ver a memória física, a saída será:

Para:
Memória física: 128 bytes.
Tamanho do quadro e da página: 8 bytes.
Tamanho do processo: 16 bytes.

Caso o usuário deseje ver a tabela de páginas do processo de PID 1:



Conforme contínuas criações de processos acontecerem a memória uma hora ficará cheia ou perto disso:



Caso o usuário escolha agora no nosso exemplo alocar um processo de 32 bytes, ele receberá um aviso de erro e o processo não será criado, pois somente 16 bytes ainda podem ser alocados na memória física:

```
[1] Show memory.
[2] Create process.
[3] Show page table.
[0] Exit.
Option: 2
Enter a PID number: 5
Enter a size to the process: 32
There is not enough memory to allocate a process with that size.
```

Caso a memória esteja cheia, não será possível alocar mais nenhum processo:

```
Free frames: 0.00%
Position ⟺ Value
0 ⟺ PRNWTLGC
1 ⟺ GTOKTJVH
2 ⟺ AASRBWFC
3 ⟺ WCMUTQJX
4 ⟺ CVTLVVOQ
5 ⟺ DWOLUBOC
6 ⟺ AVBQNPPJ
7 ⟺ HLKHNXCH
8 ⟺ QNHVLCHI
9 ⟺ AUMLYWVY
10 ⟺ COCLLVHX
11 ⟺ SXRJPIUS
12 ⟺ HJECNVEA
13 ⟺ VOFVCSOE
14 ⟺ PESOBONT
15 ⟺ OGFGQBAX

[1] Show memory.
[2] Create process.
[3] Show page table.
[0] Exit.
Option: 2
Enter a PID number: 7
Enter a size to the process: 8
There is not enough memory to allocate a process with that size.
```