

Escola de Ciências e Tecnologias

Engenharia Informática

Programação I

2018/2019

*Trabalho prático*

- Modelo 5 estados -

Luís Antunes – 34160

Sarah Simon Luz – 38116

Ana Ferro – 39872

Docente: Luís Rato

Nuno Miranda

**Índice**

Introdução……………………………………………………………………………………………………………………2

Modelo de 5 estados.………………………………….……………………………………………………………….3

Conclusão…………………………………………………………………………………………………...…..……….….6

**Introdução**

O presente trabalho foi desenvolvido no âmbito da disciplina de Sistemas Operativos I e consiste na recriação de um modelo de 5 estados que consome programas constituídos por um conjunto de instruções com escalonamento Round Robin, Quantum 4,implementada na linguagem C.

**Modelo de 5 estados**

**void start\_queues()**

**-Descrição:** Inicializa as queues de cada estado (New, Ready, Run, Block, Exit).

**void output\_files()**

**-Descrição:** Escreve em ficheiro 2 modos de output. No scheduler\_simples.out escreve os estados dos processos a cada instante. No scheduler\_complexo.out adiciona ao output scheduler\_simples.out o print da memória sempre que um novo processo é colocado ou removido da memória.

**void removerMEM(int pid);**

**-Descrição:** Liberta espaço na memória á medida que os processos terminam.

**-Argumentos:**

**int pid-** Process ID (PID) do processo terminado.

**void inserirMEM\_fork(int blocosNecessarios, int inicio\_bloco, int fim\_bloco, int pid\_original)**

**-Descrição**: Insere na memória as instruções do processo filho (criado na função fork).

**-Argumentos:**

**int blocosNecessarios-** Número de blocos necessários para inserir as instruções do processo em páginas de dimensão 10;

**int inicio\_bloco-** Posição na MEM[] onde se vai começar a introduzir as instruções;

**int fim\_bloco-** Posição na MEM[] até onde se pode meter instruções, de acordo com os blocos necessários

**int pid\_original-** Process ID do processo pai

**void inserirMEM(int blocosNecessarios,int inicio\_bloco, int fim\_bloco, int pid, int numInst)**

**-Descrição**: Insere na memória as instruções do processo filho (criado na função fork).

**-Argumentos:**

**int blocosNecessarios-** Número de blocos necessários para inserir as instruções do processo em páginas de dimensão 10;

**int inicio\_bloco-** Posição na MEM[] onde se vai começar a introduzir as instruções;

**int fim\_bloco-** Posição na MEM[] até onde se pode meter instruções, de acordo com os blocos necessários;

**int pid-** Process ID do processo a inserir na MEM[];

**int numInst-** Número de instruções (conjuntos de 3 números inteiros).

**int bestFit(int blocosNecessarios, int pid, int count)**

**-Descrição:** Esta função procura o menor espaço livre na MEM[] capaz de guardar um determinado número de instruções de um certo processo.

**-Argumentos:**

**int blocosNecessarios-** Número de blocos necessários para inserir as instruções do processo em páginas de dimensão 10;

**int pid-** Process ID;

**int count-** Número de instruções (conjuntos de 3 números inteiros).

**-Valor de retorno:** Retorna a posição inicial do bloco(int)

**void estados\_NEW(char \*processo[], int n\_processos, int tempo)**

**-Descrição:** Cria um processo novo e insere-o na queue do NEW. Esta função começa por guardar as instruções do processo lido do ficheiro de input num array auxiliar, de seguida, é chamada a função bestFit() que retorna a posição onde se vai começar a inserir as instruções e com esses dados guarda-os no pcb do novo processo.

**-Argumentos:**

**char \*processo[]-** Array auxiliar onde é guardada a leitura do input do processo

**int n\_processos-** Número de processos já existentes;

**int tempo-** Instante correspondente á entrada do processo no estado NEW

**void avancarInst()**

**-Descrição:** Avança para a instrução seguinte guardando essa posição no pcb do processo que se encontra no RUN.

**int fork()**

**-Descrição:** Duplica o processo que está no processo RUN (processo pai) e coloca o duplicado (processo filho) no estado READY. O processo pai continua no RUN caso ainda não tenha esgotado o seu Quantum.

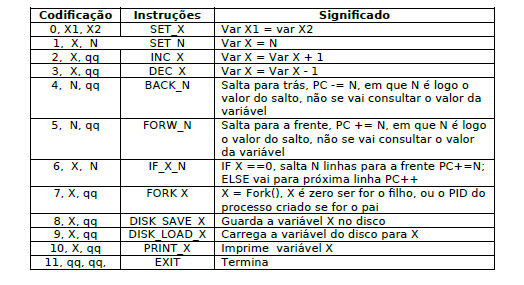
**-Valor de retorno**

**return n\_processo-** Process ID do processo filho

**return -1 –** Caso não haja espaço suficiente para criar o processo duplicado

**void instruções()**

**-Descrição:** Realiza as seguintes instruções:



**void estados()**

**-Descrição:** Faz a mudança de estados, tendo em conta a prioridade de entrada no READY (BLOCK > RUN > NEW) e os tempos consumidos em cada estado.

**int main()**

**-Descrição:** Enquanto existir processos no sistema são continuamente chamadas as funções acima descritas. Quando já não existir mais nenhum processo o programa dá-se como terminado.

**Conclusão**

Dada a conclusão do trabalho, podemos dizer que foi um trabalho bastante interessante, onde pudemos consolidar a matéria lecionada. A recriação de um modelo de 5 estados foi rica em aprendizagem.

Durante a realização do trabalho encontrámos diversos obstáculos, nomeadamente na gestão de memória e na instrução fork, mas depois de ultrapassados foi possível aperfeiçoar o programa. Apresenta falhas relativamente à impressão da instrução 10, fazendo-o no instante errado, e não foi implementado o gestor de memória Next Fit.

Após testar os vários testes que se encontram no Moodle, concluímos que são executados com sucesso à exceção do último.