



UNIVERSIDADE DE ÉVORA

Simulador de Escalonamento de Processos

Sistemas Operativos I

Professor: Luís Rato

Realizado por: Miguel de Carvalho 43108

3 de Abril de 2020

1 Introdução

Neste trabalho foi solicitado a realização de um programa que simule o **Escalonamento de Processos** num modelo de 3 estados. Na figura abaixo está representado o diagrama que descreve o modelo de 3 estados:

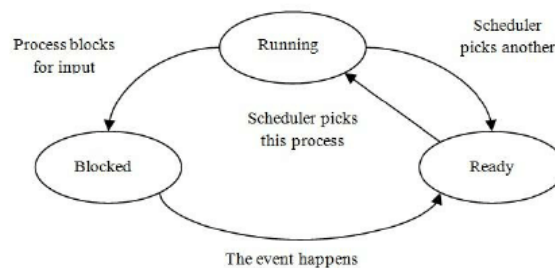


Figura 1: Diagrama de 3 Estados

O **Escalonador de Processos** faz parte do **Sistema Operativo** e é responsável por decidir em que momento cada processo estará no CPU. Existem muitos algoritmos de escalonamento para realizar essa decisão.

Neste trabalho serão utilizados o algoritmo **FCFS** e o **Round Robin (RR)**:

- O **FCFS** é um algoritmo de escalonamento não preemptivo que prioriza os processos pela ordem de chegada. Executa o processo todo do início ao fim sem o interromper, até estar concluído. Quando aparece um novo processo e ainda existe um em execução, esse novo irá para a fila de espera.
- O **Round Robin (RR)** é um algoritmo de escalonamento preemptivo que apresenta um funcionamento igual ao do **FCFS**, mas com tempo limite de execução, o **Quantum**. Ou seja, quando o processo se encontra em execução este irá ser interrompido quando o tempo de execução for igual ao `Quantum` e irá para a fila de espera (`READY`).

2 Implementação

Primeiramente comecei por pensar como deveria proceder para realizar o trabalho, e na primeira tentativa comecei por desenvolver o **FCFS**, mas na segunda tentativa cheguei a conclusão de que não seria necessário realizar um programa para a implementação do **FCFS** e outro para o **Round Robin**, pois ambos são iguais, exacto existir um limite de execução no **RR**.

Poderei então meter um limite muito grande na execução, no **Quantum**, e estarei perante o FCFS.

Comecei por proceder há criação das queues que iriam ser utilizados para o **READY**, **RUN** e **BLOCKED** e das respetivas funções essenciais para a sua manipulação.

O próximo desafio foi proceder a leitura do input (ficheiro com os processos e os tempos), para isso foi necessário proceder a criação de uma **Struct** para os processos.

Por conseguinte, comecei por desenvolver a função que iria realizar as trocas dos processos entre os 3 estados. Esta função foi a mais difícil de realizar devido a complexidade dos requisitos para os processos mudarem de estados. Foi necessário também proceder a criação de novas funções para facilitar alguns procedimentos que eram realizados de forma repetitiva.

3 Funções

- Funções pertencentes há manipulação das queues:

- A função "queue_t *create_queue (int sz)" cria a estrutura e aloca espaço para a queue;
- A função "void insert (queue_t *queue, int element)" adiciona elementos na stack;
- A função "bool full (queue_t *queue)" verifica se a stack se encontra **cheia**. Caso esteja, devolve True;
- A função "bool empty (queue_t *queue)" verifica se a stack se encontra **vazia**. Caso esteja, devolve True;
- A função "int get (queue_t *queue)" remove o primeiro elemento da stack, dando **return** desse valor e move os elementos para a esquerda queue. Por exemplo o elemento da posição 2 passa para a posição 1;
- A função "void printQueue(queue_t *queue)" dá print de todos os elementos da queue;
- A função "int top(queue_t *queue)" dá return do valor da última posição do queue.

- Funções pertencentes há manipulação da struct de processos:

- A função "process_t *create_process(int sz);" cria a estrutura e aloca espaço para o processo;
- A função "process_t *insert_process(int beg, int end, int queues, int arr[]);" insere um processo;
- A função "int find_PID(int PID, process_t *process_arr[], int n_process)" procura o PID no array e devolve a sua posição;
- A função "void update_run(int n_process, process_t *process_arr[])" decrementa a primeira posição do run do processo;
- A função "void update_blocked(int n_process, process_t *process_arr[])" decrementa a primeira posição do blocked do processo;

- A função "void update_index_run(int n_process, process_t *process_arr[], int size)" move os elementos do run do processo para a esquerda e verifique se existe algum número maior que 1000, caso exista irá mudar esse valor para 0;
- A função "void update_index_blocked(int n_process, process_t *process_arr[], int size)" move os elementos do blocked do processo para a esquerda e verifique se existe algum número maior que 1000, caso exista irá mudar esse valor para 0"

4 Execução

Quando o utilizador executar o **Simulador de Escalonamento** irá ser questionado acerca do nome do ficheiro de input e qual algoritmo pretende utilizar.

- Caso o utilizador escolha o **FCFS**, o **QUANTUM** irá ter um valor de **10000**, assim nunca irá obrigar o programa a passar para o estado ready.
- Caso o utilizador escolha o **Round Robin**, será perguntado ao utilizador qual o **Quantum** desejado.

5 Análise de Resultados

- Escalonamento com o Algoritmo **FCFS**:

- Com o input1.txt:

```
Instant 0 - Ready: 101 | Run: 100 | Blocked: Empty!
Instant 1 - Ready: 200 300 | Run: 101 | Blocked: 100
Instant 2 - Ready: 200 300 | Run: 101 | Blocked: 100
Instant 3 - Ready: 200 300 | Run: 101 | Blocked: 100
Instant 4 - Ready: 200 300 100 | Run: 101 | Blocked: Empty!
Instant 5 - Ready: 300 100 | Run: 200 | Blocked: 101
Instant 6 - Ready: 300 100 | Run: 200 | Blocked: 101
Instant 7 - Ready: 100 | Run: 300 | Blocked: 101 200
Instant 8 - Ready: 100 | Run: 300 | Blocked: 101 200
Instant 9 - Ready: 100 101 | Run: 300 | Blocked: 200
Instant 10 - Ready: 100 101 | Run: 300 | Blocked: 200
Instant 11 - Ready: 100 101 | Run: 300 | Blocked: 200
Instant 12 - Ready: 100 101 200 | Run: 300 | Blocked: Empty!
Instant 13 - Ready: 100 101 200 | Run: 300 | Blocked: Empty!
Instant 14 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 15 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 16 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 17 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 18 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 19 - Ready: 101 200 | Run: 100 | Blocked: 300
Instant 20 - Ready: 101 200 300 | Run: 100 | Blocked: Empty!
Instant 21 - Ready: 101 200 300 | Run: 100 | Blocked: Empty!
Instant 22 - Ready: 101 200 300 | Run: 100 | Blocked: Empty!
Instant 23 - Ready: 101 200 300 | Run: 100 | Blocked: Empty!
Instant 24 - Ready: 200 300 | Run: 101 | Blocked: 100
Instant 25 - Ready: 200 300 | Run: 101 | Blocked: 100
Instant 26 - Ready: 300 | Run: 200 | Blocked: 100
```

Instant 27 - Ready: 100 | Run: 300 | Blocked: 200
 Instant 28 - Ready: Empty! | Run: 100 | Blocked: 200
 Instant 29 - Ready: 200 | Run: 100 | Blocked: Empty!
 Instant 30 - Ready: 200 | Run: 100 | Blocked: Empty!
 Instant 31 - Ready: 200 | Run: 100 | Blocked: Empty!
 Instant 32 - Ready: 200 | Run: 100 | Blocked: Empty!
 Instant 33 - Ready: 200 | Run: 100 | Blocked: Empty!
 Instant 34 - Ready: Empty! | Run: 200 | Blocked: Empty!
 Instant 35 - Ready: Empty! | Run: 200 | Blocked: Empty!
 Instant 36 - Ready: Empty! | Run: 200 | Blocked: Empty!

- Com o input2.txt:

- Com o input3.txt:

- Escalonamento com o Algoritmo **RR (Round Robin)**:

- Com o input1.txt:

Instant 0 - Ready: 101 | Run: 100 | Blocked: Empty!
 Instant 1 - Ready: 200 300 | Run: 101 | Blocked: 100
 Instant 2 - Ready: 200 300 | Run: 101 | Blocked: 100
 Instant 3 - Ready: 200 300 | Run: 101 | Blocked: 100
 Instant 4 - Ready: 300 100 101 | Run: 200 | Blocked: Empty!
 Instant 5 - Ready: 300 100 101 | Run: 200 | Blocked: Empty!
 Instant 6 - Ready: 100 101 | Run: 300 | Blocked: 200
 Instant 7 - Ready: 100 101 | Run: 300 | Blocked: 200
 Instant 8 - Ready: 100 101 | Run: 300 | Blocked: 200
 Instant 9 - Ready: 101 300 | Run: 100 | Blocked: 200
 Instant 10 - Ready: 101 300 | Run: 100 | Blocked: 200
 Instant 11 - Ready: 101 300 200 | Run: 100 | Blocked: Empty!
 Instant 12 - Ready: 300 200 100 | Run: 101 | Blocked: Empty!
 Instant 13 - Ready: 200 100 | Run: 300 | Blocked: 101
 Instant 14 - Ready: 200 100 | Run: 300 | Blocked: 101
 Instant 15 - Ready: 200 100 | Run: 300 | Blocked: 101
 Instant 16 - Ready: 100 300 | Run: 200 | Blocked: 101
 Instant 17 - Ready: 300 101 | Run: 100 | Blocked: 200
 Instant 18 - Ready: 300 101 | Run: 100 | Blocked: 200
 Instant 19 - Ready: 300 101 200 | Run: 100 | Blocked: Empty!
 Instant 20 - Ready: 101 200 100 | Run: 300 | Blocked: Empty!
 Instant 21 - Ready: 200 100 | Run: 101 | Blocked: 300
 Instant 22 - Ready: 200 100 | Run: 101 | Blocked: 300
 Instant 23 - Ready: 100 | Run: 200 | Blocked: 300
 Instant 24 - Ready: 100 | Run: 200 | Blocked: 300
 Instant 25 - Ready: 100 | Run: 200 | Blocked: 300
 Instant 26 - Ready: Empty! | Run: 100 | Blocked: 300
 Instant 27 - Ready: 300 | Run: 100 | Blocked: Empty!

```
Instant 28 - Ready: 300 | Run: 100 | Blocked: Empty!
Instant 29 - Ready: 100 | Run: 300 | Blocked: Empty!
Instant 30 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 31 - Ready: Empty! | Run: Empty! | Blocked: 100
Instant 32 - Ready: Empty! | Run: Empty! | Blocked: 100
Instant 33 - Ready: Empty! | Run: Empty! | Blocked: 100
Instant 34 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 35 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 36 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 37 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 38 - Ready: Empty! | Run: 100 | Blocked: Empty!
Instant 39 - Ready: Empty! | Run: 100 | Blocked: Empty!
```

– Com o `input2.txt`:

– Com o `input3.txt`:

6 Problemas

Neste trabalho só existiu um problema e consiste no facto de que em algumas execuções o ultimo processo existente no ficheiro não é lido, mas este problema só acontece no **Linux**. Por outro lado no **Windows** este problema não ocorre.

7 Conclusão

Em suma, este trabalho fez-me entender melhor como funciona o escalonador e as condições que cada algoritmo usa para proceder há mudança dos processos entre os estados e as diferenças de tempo.