Simulador de Escalonamento - Trabalho Prático

Sistemas Operativos I

Tomás Dias nº42784

Introdução

Foi proposto na cadeira de Sistemas Operativos I a realização de um trabalho prático com o objetivo de implementar um simulador de escalonamento ***FCFS*** (*first come, first served*) e ***Round Robin***. O procedimento feito tendo como objetivo a implementação do simulador encontra-se descrito de seguida.

Estruturas de dados usadas e função main()

A principal estrutura de dados utilizada foi uma estrutura de filas. Estas tinham como finalidade gerir filas de processos. Na implementação das filas, para além da definição da struct que representa uma fila, foram utilizadas as seguintes funções:

- **struct Queue* createQueue()**: Esta função cria uma fila recebendo como argumento a capacidade da mesma.
- **int isFull()**: Esta função indica se a fila está lotada, ou seja, se o tamanho da fila é igual à sua capacidade. Recebe como argumento a fila.
- **int isEmpty()**: Esta função indica se a fila está vazia, ou seja, se o tamanho da fila é igual a 0. Recebe como argumento a fila.
- **void enqueue()**: Esta função adiciona um processo à parte mais atrás da fila, recebendo como argumentos a fila à qual vai ser adicionado o processo e o próprio processo.

- **int dequeue()**: Esta função remove o processo que se encontra na frente da fila, recebendo como argumento a fila à qual o processo será removido e retorna o processo removido.
- **int front()**: Esta função indica qual o processo que está na frente da fila, recebendo como argumento a fila.
- **int rear()**: Esta função indica qual o processo que está na parte de trás da fila, recebendo como argumento a fila.

Foi também definida uma struct de forma a representar um processo.

Na função **main()**, para além de serem chamadas as funções que implementam os algoritmos de escalonamento e de serem criadas as filas que correspondem aos estados run, blocked e ready, é também lido o ficheiro que serve como input. A estrutura desta função é maioritariamente baseada em ciclos, em que a partir de o input introduzido pelo utilizador, cada linha do ficheiro é guardada num array de duas dimensões e para cada linha do ficheiro é passada a informação dessa linha para cada membro da estrutura do processo correspondente. Este ciclo repete-se tendo em conta o número de processos existentes.

Funções que implementam os algoritmos de escalonamento FCFS e Round Robin

A arquitetura da função que implementa o FCFS e da função que implementa o Round Robin é praticamente idêntica. Ambas as funções assentam em ciclos e em estruturas de decisão/condição. Com a utilização de ciclos procurou-se estudar cada processo individualmente, de forma a determinar o seu comportamento num determinado instante e repetir o procedimento para o resto dos instantes. O comportamento que o processo iria ter foi fundamentalmente determinado por condições de maneira a garantir que um processo só agia de uma determinada maneira quando cumpridas essas mesmas condições.

Para a implementação das funções correspondentes a FCFS e Round Robin, foram implementadas as seguintes funções auxiliares:

- **void coloca_posicao()**: Esta função coloca o processo numa determinada posição da fila. Recebe como argumentos o processo, a posição da fila na qual se pretende colocar o processo e a fila.

- **int elimina()**: Esta função elimina o processo do array de processos, returnando o tamanho atualizado do array. Recebe como argumentos o array de processos, o tamanho do array e o processo.

- **int verifica()**: Esta função verifica se o processo se encontra numa determinada fila returnando 1 se se verificar e 0 se não se verificar. Recebe como argumento o processo e a fila.

- **void ordena_menor()**: Esta função ordena os processos de forma crescente de tempos de chegada (t_inicio). Recebe como argumentos o array de processos e o tamanho do array.

- **void ordena_estado()**: Esta função ordena os processos pelo estado em que se encontram (RUN -> BLOCKED -> READY). Recebe como argumentos o array de processos, o tamanho do array e as três filas correspondentes aos estados (run, blocked e ready).

- **void output()**: Esta função é responsável pela estrutura do output (printa a fila). Recebe como argumento a fila.

Com isto, implementou-se os algoritmos FCFS e Round Robin que correspondem às funções:

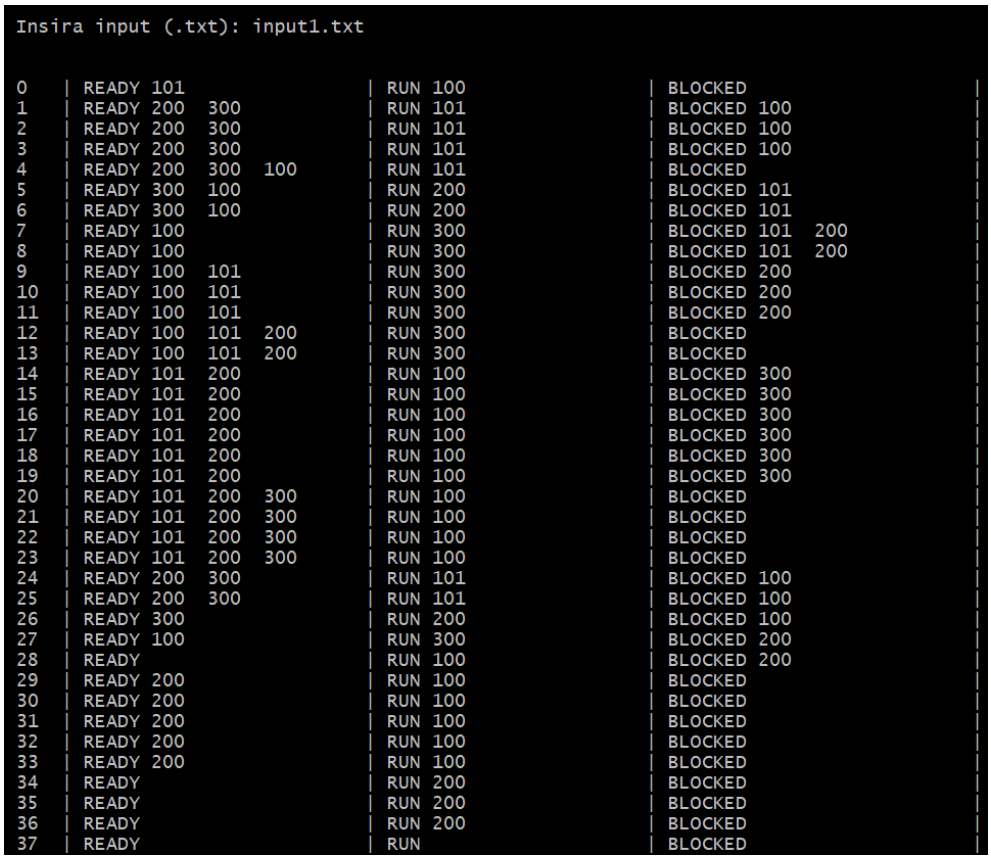
- **void fcfs()**: Esta função implementa o algoritmo de escalonamento FCFS. Como dito anteriormenete, esta função é constituída essencialmente por ciclos e estruturas de condição. O ciclo 'principal' inicia quando o instante é 0 e os processos encontram-se ordenados pelo menor tempo de chegada e pelo estado em que se encontram, e finaliza após ser atingido o número de instantes dado para o algoritmo ser executado. Definiu-se como 60 instantes como limite máximo de vezes que o ciclo é executado, sendo que os processos podem demorar mais ou menos tempo a finalizar dependendo do input dado. Recebe como argumentos o array de processos, o número de processos e as filas correspondentes aos estados (run, blocked, ready).

- **void rr()**: Esta função implementa o algoritmo de escalonamento Round Robin. Com a arquitetura semelhante à função fcfs, esta por

sua vez diferencia-se pela utilização de um quantum que modifica o comportamento dos processos em cada instante bem como as prioridades de acesso às filas correspondentes ao estados. À semelhança da função fcfs, o limite máximo de instantes que o ciclo 'principal' é executado é também 60. Recebe como argumentos o array de processos, o número de processos, o valor de quantum e as filas.

Resultados do output dos algoritmos de escalonamento FCFS e Round Robin

O output do FCFS para o input1.txt foi o seguinte:



```
Insira input (.txt): input1.txt


0   | READY 101               | RUN 100               | BLOCKED                 |
1   | READY 200  300          | RUN 101               | BLOCKED 100             |
2   | READY 200  300          | RUN 101               | BLOCKED 100             |
3   | READY 200  300          | RUN 101               | BLOCKED 100             |
4   | READY 200  300  100     | RUN 101               | BLOCKED                 |
5   | READY 300  100          | RUN 200               | BLOCKED 101             |
6   | READY 300  100          | RUN 200               | BLOCKED 101             |
7   | READY 100               | RUN 300               | BLOCKED 101  200        |
8   | READY 100               | RUN 300               | BLOCKED 101  200        |
9   | READY 100  101          | RUN 300               | BLOCKED 200             |
10  | READY 100  101          | RUN 300               | BLOCKED 200             |
11  | READY 100  101          | RUN 300               | BLOCKED 200             |
12  | READY 100  101  200     | RUN 300               | BLOCKED                 |
13  | READY 100  101  200     | RUN 300               | BLOCKED                 |
14  | READY 101  200          | RUN 100               | BLOCKED 300             |
15  | READY 101  200          | RUN 100               | BLOCKED 300             |
16  | READY 101  200          | RUN 100               | BLOCKED 300             |
17  | READY 101  200          | RUN 100               | BLOCKED 300             |
18  | READY 101  200          | RUN 100               | BLOCKED 300             |
19  | READY 101  200          | RUN 100               | BLOCKED 300             |
20  | READY 101  200  300     | RUN 100               | BLOCKED                 |
21  | READY 101  200  300     | RUN 100               | BLOCKED                 |
22  | READY 101  200  300     | RUN 100               | BLOCKED                 |
23  | READY 101  200  300     | RUN 100               | BLOCKED                 |
24  | READY 200  300          | RUN 101               | BLOCKED 100             |
25  | READY 200  300          | RUN 101               | BLOCKED 100             |
26  | READY 300               | RUN 200               | BLOCKED 100             |
27  | READY 100               | RUN 300               | BLOCKED 200             |
28  | READY                   | RUN 100               | BLOCKED 200             |
29  | READY 200               | RUN 100               | BLOCKED                 |
30  | READY 200               | RUN 100               | BLOCKED                 |
31  | READY 200               | RUN 100               | BLOCKED                 |
32  | READY 200               | RUN 100               | BLOCKED                 |
33  | READY 200               | RUN 100               | BLOCKED                 |
34  | READY                   | RUN 200               | BLOCKED                 |
35  | READY                   | RUN 200               | BLOCKED                 |
36  | READY                   | RUN 200               | BLOCKED                 |
37  | READY                   | RUN                   | BLOCKED                 |
```

O output do Round Robin para o input1.txt foi o seguinte:

```
Insira input (.txt): input1.txt


0   | READY 101                | RUN 100                | BLOCKED                |
1   | READY 200  300           | RUN 101                | BLOCKED 100            |
2   | READY 200  300           | RUN 101                | BLOCKED 100            |
3   | READY 200  300           | RUN 101                | BLOCKED 100            |
4   | READY 300  100  101      | RUN 200                | BLOCKED                |
5   | READY 300  100  101      | RUN 200                | BLOCKED                |
6   | READY 100  101           | RUN 300                | BLOCKED 200            |
7   | READY 100  101           | RUN 300                | BLOCKED 200            |
8   | READY 100  101           | RUN 300                | BLOCKED 200            |
9   | READY 101  300           | RUN 100                | BLOCKED 200            |
10  | READY 101  300           | RUN 100                | BLOCKED 200            |
11  | READY 101  300  200      | RUN 100                | BLOCKED                |
12  | READY 300  200  100      | RUN 101                | BLOCKED                |
13  | READY 200  100           | RUN 300                | BLOCKED 101            |
14  | READY 200  100           | RUN 300                | BLOCKED 101            |
15  | READY 200  100           | RUN 300                | BLOCKED 101            |
16  | READY 100  300           | RUN 200                | BLOCKED 101            |
17  | READY 300  101           | RUN 100                | BLOCKED 200            |
18  | READY 300  101           | RUN 100                | BLOCKED 200            |
19  | READY 300  101  200      | RUN 100                | BLOCKED                |
20  | READY 101  200  100      | RUN 300                | BLOCKED                |
21  | READY 200  100           | RUN 101                | BLOCKED 300            |
22  | READY 200  100           | RUN 101                | BLOCKED 300            |
23  | READY 100                | RUN 200                | BLOCKED 300            |
24  | READY 100                | RUN 200                | BLOCKED 300            |
25  | READY 100                | RUN 200                | BLOCKED 300            |
26  | READY                    | RUN 100                | BLOCKED 300            |
27  | READY 300                | RUN 100                | BLOCKED                |
28  | READY 300                | RUN 100                | BLOCKED                |
29  | READY 100                | RUN 300                | BLOCKED                |
30  | READY                    | RUN 100                | BLOCKED                |
31  | READY                    | RUN                    | BLOCKED 100            |
32  | READY                    | RUN                    | BLOCKED 100            |
33  | READY                    | RUN                    | BLOCKED 100            |
34  | READY                    | RUN 100                | BLOCKED                |
35  | READY                    | RUN 100                | BLOCKED                |
36  | READY                    | RUN 100                | BLOCKED                |
37  | READY                    | RUN 100                | BLOCKED                |
38  | READY                    | RUN 100                | BLOCKED                |
39  | READY                    | RUN 100                | BLOCKED                |
40  | READY                    | RUN                    | BLOCKED                |
```