UNIVERSIDADE DE ÉVORA

Sistemas Operativos
Trabalho 1

Simulador de SO com modelo de 5 estados

Trabalho Realizado por:
Chen Cheng Nº38147
Luís Maurício Nº37722

Introdução

Neste trabalho irá ser implementado um simulador de Sistema Operativo com um modelo de 5 estados(NEW, READY, RUNNING, BLOCKED, EXIT) com recurso a uma implementação de uma fila de espera do tipo FIFO(First In First Out).

Serão implementados 2 algoritmos de escalonamento, o primeiro Round-Robin com quantum de valor 3 e o segundo Virtual Round-Robin com quantum também de valor 3.

Os processos serão representados por um conjunto de pseudo-instruções-máquina na forma de números. O primeiro número indica o instante inicial de cada processo, ou seja, quando este passará ao estado NEW.

Os números seguintes indicam alternadamente o tempo que o processo está no estado RUNNING, seguido do tempo que o processo está no estado BLOCKED.

Estruturas

Fila de Espera FIFO

Esta fila de espera tem 3 atributos inteiros(head, tail, size), 1 atributo inteiro sem sinal(capacity) e um ponteiro para um array de inteiros(array) que guarda os valores que representam os ids de processo(pid).

O primeiro método desta implementação corresponde ao construtor de queues. Recebe como argumento a capacidade total da queue a inicializar. Retorna uma queue vazia com apenas o atributo capacidade preenchido.

Existem também os métodos comuns a filas:

- bool isFull(Queue *queue) - retorna true se a queue estiver cheia, false em caso contrário;
- bool isEmpty(Queue *queue) - retorna true se a queue estiver vazia, false em caso contrário;
- bool enqueue(Queue *queue, int num_proc) - tenta adicionar o pid que recebe no num_proc à tail da queue, retorna false sem adicionar se não for possível e true caso o pid seja adicionado;
- int dequeue(Queue *queue) - retira e retorna o pid que se encontra na head da fila no caso da fila não estar vazia. Caso esteja vazia retorna o valor 0;
- int size(Queue *queue) - retorna o número de pids presentes na queue(atributo size da queue).

Processo

A estrutura *process* representa um processo e é composta pelos seguintes componentes:

- int instances[MAX] - cada posição neste array é uma pseudo-instrução-máquina, estas são lidas do ficheiro de input. MAX é uma constante definida no início do programa.
- int pid - indica o número do processo. Este número é atribuído pela ordem de leitura do ficheiro.
- int count - indica a posição do array \textit{instances} que está a ser lida neste momento pelo SO.
- int entry - primeira coluna do ficheiro de input, indica o momento de entrada do processo no SO.
- int size - número total de pseudo-instruções-máquina preenchidas no array \textit{instances}.
- int state - indica o estado em que o processo se encontra atualmente(0 - NEW, 1 - READY, 2 - RUN, 3 - BLOCKED, 4 - BLOCKED HEAD, 5 - AUX, 6 - EXIT).

Funcionamento

Ao executar o programa o utilizador deve passar 2 argumentos, o algoritmo que deseja simular (rr ou vrr) e o formato de input (hard ou file), a partir desta escolha o programa importa a lista de programas e chama a função respectiva.

Mas primeiro o array de input é convertido num array processos para ser possível separar o momento de entrada das instruções, contar o número de instruções e popular o array instances do processo com as instruções.

Escalonamento Round-Robin

Para a implementação deste algoritmo utilizamos 2 filas, a ready e a block.
Temos ainda 2 flags, uma para indicar se o processador está livre (r) e outra para indicar se as operações I/O estão disponíveis (b), para estas duas flags existem 2 respectivos sinais.

O contador de tempo é incrementado a cada ciclo para emular os ciclos do processador e este ciclo corre até que não haja mais processos por terminar.

Escalonamento Virtual Round-Robin

A implementação deste algoritmo é bastante semelhante à do Round-Robin mas adiciona-se outra queue para servir de auxiliar.

Ao adicionar esta queue adicionou-se também uma flag para indicar se existe um processo que esteja na aux de forma a que esta tenha prioridade em relação a processos vindos da ready. Aliado a esta flag é utilizado também um novo sinal para alterar o valor da flag apenas no final de cada ciclo.

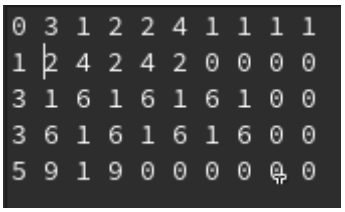
Exemplo Input/Output

Input

Hardcoded:

```
int programas[5][10] = {
        {0,3,1,2,2,4,1,1,1,1},
        {1,2,4,2,4,2,0,0,0,0},
        {3,1,6,1,6,1,6,1,0,0},
        {3,6,1,6,1,6,1,6,0,0},
        {5,9,1,9,0,0,0,0,0,0}
};
```

Ficheiro:



Output

Round-Robin Scheduler

TEMPO	P1	P2	P3	P4	P5
0	NEW				
1	RUN	NEW			
2	RUN	READY			
3	RUN	READY	NEW	NEW	
4	BLOCK*	RUN	READY	READY	
5	READY	RUN	READY	READY	NEW
6	READY	BLOCK*	RUN	READY	READY
7	READY	BLOCK*	BLOCK	RUN	READY
8	READY	BLOCK*	BLOCK	RUN	READY
9	READY	BLOCK*	BLOCK	RUN	READY
10	RUN	READY	BLOCK*	READY	READY
11	RUN	READY	BLOCK*	READY	READY
12	BLOCK	READY	BLOCK*	READY	RUN
13	BLOCK	READY	BLOCK*	READY	RUN
14	BLOCK	READY	BLOCK*	READY	RUN
15	BLOCK	RUN	BLOCK*	READY	READY
16	BLOCK*	RUN	READY	READY	READY
17	BLOCK*	BLOCK	READY	RUN	READY
18	READY	BLOCK*	READY	RUN	READY
19	READY	BLOCK*	READY	RUN	READY
20	READY	BLOCK*	READY	BLOCK	RUN
21	READY	BLOCK*	READY	BLOCK	RUN
22	READY	READY	READY	BLOCK*	RUN
23	READY	READY	RUN	READY	READY
24	RUN	READY	BLOCK*	READY	READY
25	RUN	READY	BLOCK*	READY	READY
26	RUN	READY	BLOCK*	READY	READY
27	READY	RUN	BLOCK*	READY	READY
28	READY	RUN	BLOCK*	READY	READY
29	READY	EXIT	BLOCK*	RUN	READY
30	READY		READY	RUN	READY
31	READY		READY	RUN	READY
32	READY		READY	READY	RUN
33	READY		READY	READY	RUN
34	READY		READY	READY	RUN
35	RUN		READY	READY	BLOCK*
36	BLOCK*		RUN	READY	READY
37	READY		BLOCK*	RUN	READY
38	READY		BLOCK*	RUN	READY
39	READY		BLOCK*	RUN	READY
40	READY		BLOCK*	BLOCK	RUN
41	READY		BLOCK*	BLOCK	RUN
42	READY		BLOCK*	BLOCK	RUN
43	RUN		READY	BLOCK*	READY
44	BLOCK*		RUN	READY	READY
45	READY		EXIT	READY	RUN
46	READY			READY	RUN
47	READY			READY	RUN
48	READY			RUN	READY
49	READY			RUN	READY
50	READY			RUN	READY
51	RUN			READY	READY
52	EXIT			READY	RUN
53				READY	RUN
54				READY	RUN
55				RUN	EXIT
56				RUN	
57				RUN	
58				BLOCK*	
59				RUN	
60				RUN	
61				RUN	
62				RUN	
63				RUN	
64				RUN	
65				EXIT	

Virtual Round-Robin Scheduler

TEMPO	P1	P2	P3	P4	P5
0	NEW				
1	RUN	NEW			
2	RUN	READY			
3	RUN	READY	NEW	NEW	
4	BLOCK*	RUN	READY	READY	
5	AUX	RUN	READY	READY	NEW
6	RUN	BLOCK*	READY	READY	READY
7	RUN	BLOCK*	READY	READY	READY
8	BLOCK	BLOCK*	RUN	READY	READY
9	BLOCK	BLOCK*	BLOCK	RUN	READY
10	BLOCK*	AUX	BLOCK	RUN	READY
11	BLOCK*	AUX	BLOCK	RUN	READY
12	AUX	RUN	BLOCK*	READY	READY
13	AUX	RUN	BLOCK*	READY	READY
14	RUN	BLOCK	BLOCK*	READY	READY
15	RUN	BLOCK	BLOCK*	READY	READY
16	RUN	BLOCK	BLOCK*	READY	READY
17	READY	BLOCK	BLOCK*	READY	RUN
18	READY	BLOCK*	AUX	READY	RUN
19	READY	BLOCK*	AUX	READY	RUN
20	READY	BLOCK*	RUN	READY	READY
21	READY	BLOCK*	BLOCK	RUN	READY
22	READY	AUX	BLOCK*	RUN	READY
23	READY	AUX	BLOCK*	RUN	READY
24	READY	RUN	BLOCK*	BLOCK	READY
25	READY	RUN	BLOCK*	BLOCK	READY
26	RUN	EXIT	BLOCK*	BLOCK	READY
27	BLOCK		BLOCK*	BLOCK	RUN
28	BLOCK		AUX	BLOCK*	RUN
29	BLOCK*		AUX	AUX	RUN
30	AUX		RUN	AUX	READY
31	AUX		BLOCK*	RUN	READY
32	AUX		BLOCK*	RUN	READY
33	AUX		BLOCK*	RUN	READY
34	RUN		BLOCK*	READY	READY
35	BLOCK		BLOCK*	READY	RUN
36	BLOCK		BLOCK*	READY	RUN
37	BLOCK*		AUX	READY	RUN
38	AUX		RUN	READY	BLOCK*
39	RUN		EXIT	READY	AUX
40	EXIT			READY	RUN
41				READY	RUN
42				READY	RUN
43				RUN	READY
44				RUN	READY
45				RUN	READY
46				BLOCK*	RUN
47				AUX	RUN
48				AUX	RUN
49				RUN	READY
50				RUN	READY
51				RUN	READY
52				READY	RUN
53				READY	RUN
54				READY	RUN
55				RUN	EXIT
56				RUN	
57				RUN	
58				BLOCK*	
59				RUN	
60				RUN	
61				RUN	
62				RUN	
63				RUN	
64				RUN	
65				EXIT	