Escola de Ciências e Tecnologia

Departamento de Informática

Licenciatura em Engenharia Informática

Unidade curricular Sistemas Operativos

Ano letivo 2020/2021

Relatório

Trabalho prático de Sistemas Operativos

Docentes

Professor Luís Rato

Discentes

Duarte Anastácio nº40090

Ludgero Teixeira nº41348

Ricardo Oliveira nº42647

Évora, abril de 2021

Descrição do trabalho

É pretendida a implementação um simulador de Sistema Operativo considerando um modelo de 5 estados, este modelo deve incluir os estados READY, RUNNING, BLOCKED, e ainda os estados NEW e EXIT.
Foi pedida a implementação do simulador dos processos neste sistema tendo em consideração que:

1) Admite-se que a mudança entre estados é infinitamente rápida; depois os processos ficam nos diversos estados 1 instante de tempo, seguindo-se outra mudança de estados, e assim sucessivamente.

2) Os processos quando são criados passam para o estado NEW e permanecem nesse estado 1 instante de tempo.

3) Os processos quando saem de NEW passam para o estado READY, respeitando a ordem pela qual entraram em NEW. No entanto, se a fila de READY estiver vazia, e o CPU (estado RUNNING) não estiver ocupado um processo pode passar de NEW para READY e deste para RUNNING/CPU "instantaneamente"

4) Os instantes na fila BLOCKED, para cada processo só conta para o processo que está na cabeça da fila.

5) Os processos quando terminam, passam do CPU (estado RUNNING) para o EXIT, onde permanecem 1 instante de tempo.

6) Os processos depois de estarem 1 instante de tempo em EXIT desaparecem do sistema.

7) O escalonamento a implementar deve compreender 2 algoritmos: o Round-Robin Quantum =3; e o VRR Virtual Round Robin com Quantum = 3.

8) Os processos que saem de BLOCKED, passam para READY. No caso de RR passam para a fila READY; no caso do VRR passam para a fila READY-AUX (no VRR a fila READY-AUX tem prioridade face à READY). Note que o modelo de 5 estados define um estado READY, no entanto, na prática ao usarmos o VRR estamos a funcionar com um estado extra (um estado READY e um estado READY-AUX).

9) O número máximo de programas a dar entrada no sistema é de 10, e cada programa tem a dimensão máxima de

10 (1 instante inicial, mais 9 instruções alternadas de CPU e BLOCK). 10) Se no mesmo instante puderem entrar processos na fila de READY vindos de RUNNING, BLOCKED, e NEW, estes entrarão na fila de READY pela seguinte ordem: 1º os de BLOCKED; 2ª o de RUNNING/CPU; e por último os de NEW.

11) O programa deve ter como output, o estado de cada processo em cada instante.

Descrição das funções

Queue* createQueue ( ): Esta função cria e retorna uma queue.

void enqueue (Queue* queue, Processo* processo): Esta função adiciona um processo à Queue, inserindo o mesmo no final da fila.

void dequeue (Queue* queue): Esta função remove o primeiro processo da Queue.

Processo* peek(Queue* queue): Esta função permite nos aceder ao primeiro elemento da fila da Queue sem o termos que remover.

Processo* criarProcesso(Programa programa): Esta função "cria" e retorna um processo.

OS* createCPU( ): Esta função serve de constructor para a nossa simulação de sistema operativo. Inicializa a mesma com as queues necessárias e com os processos que vão ser executados.

void InputProgramas(OS* os): Esta função recebe e processa o input colocando os valores dados dentro do array de ciclos do programa. Criando um processo a partir de cada linha do input.

void printEstados( ): Esta função imprime os estados em que os processos se encontram a cada instante.

void round_robin(OS* os): Esta função começa por iterar todos os processos, e caso algum não tenha sido inicializado **(STATE == NONE),** o processo avança para **NEW** e incrementa o pc.

Caso seja encontrado um processo que se encontre no estado **EXIT**, incrementa-se os ciclos que já decorreram desse processo.

Se a **BlockedQueue** não estiver vazia retornamos o processo em primeiro lugar da fila dessa queue, se o mesmo já percorreu todos os seus instantes no estado **BLOCKED,** então passa para **READY**, caso esta condição não se verifique apenas se incrementa um ciclo desse mesmo processo no estado **BLOCKED**.

Um processo que se encontra em estado **RUN** caso ainda não tenham passado todos os seus instantes necessários nesse estado, incrementa-se mais 1 ciclo que o programa passa nesse estado, caso esse mesmo programa já tenha percorrido todos os seus ciclos, o mesmo vai para **EXIT.**
Os processos quando passam do estado **RUN** para **EXIT** permanecem lá durante 1 instante de tempo.
Caso estas condições não se verifiquem o processo vai para o estado **BLOCKED**, senão, após correrem o **Quantum** voltam para o estado **READY,** caso nenhuma destas condições se verifiquem apenas se incrementa o número de ciclo que o processo passou no estado.

Todos os processos que se encontram no estado **NEW**, apenas permanecem lá 1 instante, seguindo de seguida para **READY**.
Sendo que se a fila **READY** não estiver vazia, olha-se para o primeiro elemento dessa queue e passamo-lo para **RUN.**

Caso ainda existam processos por ser executados, o programa continua, senão nega se a variável **correrProcesso** para se terminar o programa.

Void virtual_round_robin(OS* os): Esta função corre o escalonamento virtual round robin, que em parte é idêntico ao escalonamento Round Robin.

Sendo que neste escalonamento ao contrário do Round Robin caso um processo se encontro estado **BLOCKED** ao invés de ir para o estado **READY**, o mesmo vai para uma queue auxiliar previamente criada **AUX**.

Caso existam processos na queue **AUX**, retornamos o processo no 1 lugar dessa queue e o mesmo vai para **RUN** e incrementamos os ciclos nesse estado.

Caso a queue **AUX** esteja vazia, e a queue **READY** tiver processos a espera, os mesmos passam para **RUN**.

int main( ): A função main começa por pedir para que seja escolhido que tipo de escalonamento é pretendido para executar o input.
Após isso, a função createCPU() é chamada juntamente com a função InputProgramas() e de seguida consoante a escolha do escalonamento a função do mesmo é chamada.

Estruturas

Implementámos as seguintes structs representando os processos, as queues e o Sistema Operativo.

```
typedef struct
{
int ciclos [MAX_PROGRAM_SIZE];
int pc;

}Programa;


typedef struct
{
Programa programa;
enum STATE state;
int ciclosDecorridos;
}Processo;


typedef struct
{
Processo* data[MAX_processos];
int front;
int size;
int tail;

}Queue;


typedef struct
{
Processo* processos[MAX_processos];
int ciclos;
int numeroDeProcessos;
Processo* runningProcess;
Queue* readyQueue;
Queue* blockedQueue;
Queue* auxQueue;

}OS;
```

I/O exemplo

Foi nos dado pelo docente o seguinte input para testarmos o nosso programa.

```
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 } };
```

Os outputs obtidos com base neste input foram.

➢ Round Robin

*nota o 4 processo corre até
ao instante 66, não tendo sido possível apanhar o mesmo
na captura de ecrã.

instante	proc1	proc2	proc3	proc4	proc5
01	NEW				
02	RUN	NEW			
03	RUN	READY			
04	RUN	READY	NEW	NEW	
05	BLCK	RUN	READY	READY	
06	READY	RUN	READY	READY	NEW
07	READY	BLCK	RUN	READY	READY
08	READY	BLCK	BLCK	RUN	READY
09	READY	BLCK	BLCK	RUN	READY
10	READY	BLCK	BLCK	RUN	READY
11	RUN	READY	BLCK	READY	READY
12	RUN	READY	BLCK	READY	READY
13	BLCK	READY	BLCK	READY	RUN
14	BLCK	READY	BLCK	READY	RUN
15	BLCK	READY	BLCK	READY	RUN
16	BLCK	RUN	BLCK	READY	READY
17	BLCK	RUN	READY	READY	READY
18	BLCK	BLCK	READY	RUN	READY
19	READY	BLCK	READY	RUN	READY
20	READY	BLCK	READY	RUN	READY
21	READY	BLCK	READY	BLCK	RUN
22	READY	BLCK	READY	BLCK	RUN
23	READY	READY	READY	BLCK	RUN
24	READY	READY	RUN	READY	READY
25	RUN	READY	BLCK	READY	READY
26	RUN	READY	BLCK	READY	READY
27	RUN	READY	BLCK	READY	READY
28	READY	RUN	BLCK	READY	READY
29	READY	RUN	BLCK	READY	READY
30	READY	EXIT	BLCK	RUN	READY
31	READY		READY	RUN	READY
32	READY		READY	RUN	READY
33	READY		READY	READY	RUN
34	READY		READY	READY	RUN
35	READY		READY	READY	RUN
36	RUN		READY	READY	BLCK
37	BLCK		RUN	READY	READY
38	READY		BLCK	RUN	READY
39	READY		BLCK	RUN	READY
40	READY		BLCK	RUN	READY
41	READY		BLCK	BLCK	RUN
42	READY		BLCK	BLCK	RUN
43	READY		BLCK	BLCK	RUN
44	RUN		READY	BLCK	READY
45	BLCK		RUN	READY	READY
46	READY		EXIT	READY	RUN
47	READY			READY	RUN
48	READY			READY	RUN
49	READY			RUN	READY
50	READY			RUN	READY
51	READY			RUN	READY
52	RUN			READY	READY
53	EXIT			READY	RUN
54				READY	RUN
55				READY	RUN
56				RUN	EXIT
57				RUN	
58				RUN	
59				BLCK	
60				RUN	
61				RUN	
62				RUN	

➢Virtual Round Robin

*nota o 4 processo corre até
ao instante 66, não tendo sido possível apanhar o mesmo
na captura de ecrã.

instante	proc1	proc2	proc3	proc4	proc5
01	NEW				
02	RUN	NEW			
03	RUN	READY			
04	RUN	READY	NEW	NEW	
05	BLCK	RUN	READY	READY	
06	AUX	RUN	READY	READY	NEW
07	RUN	BLCK	READY	READY	READY
08	RUN	BLCK	READY	READY	READY
09	BLCK	BLCK	RUN	READY	READY
10	BLCK	BLCK	BLCK	RUN	READY
11	BLCK	AUX	BLCK	RUN	READY
12	BLCK	AUX	BLCK	RUN	READY
13	AUX	RUN	BLCK	READY	READY
14	AUX	RUN	BLCK	READY	READY
15	RUN	BLCK	BLCK	READY	READY
16	RUN	BLCK	BLCK	READY	READY
17	RUN	BLCK	BLCK	READY	READY
18	READY	BLCK	BLCK	READY	RUN
19	READY	BLCK	AUX	READY	RUN
20	READY	BLCK	AUX	READY	RUN
21	READY	BLCK	RUN	READY	READY
22	READY	BLCK	BLCK	RUN	READY
23	READY	AUX	BLCK	RUN	READY
24	READY	AUX	BLCK	RUN	READY
25	READY	RUN	BLCK	BLCK	READY
26	READY	RUN	BLCK	BLCK	READY
27	RUN	EXIT	BLCK	BLCK	READY
28	BLCK		BLCK	BLCK	RUN
29	BLCK		AUX	BLCK	RUN
30	BLCK		AUX	AUX	RUN
31	AUX		RUN	AUX	READY
32	AUX		BLCK	RUN	READY
33	AUX		BLCK	RUN	READY
34	AUX		BLCK	RUN	READY
35	RUN		BLCK	READY	READY
36	BLCK		BLCK	READY	RUN
37	BLCK		BLCK	READY	RUN
38	BLCK		AUX	READY	RUN
39	AUX		RUN	READY	BLCK
40	RUN		EXIT	READY	AUX
41	EXIT			READY	RUN
42				READY	RUN
43				READY	RUN
44				RUN	READY
45				RUN	READY
46				RUN	READY
47				BLCK	RUN
48				AUX	RUN
49				AUX	RUN
50				RUN	READY
51				RUN	READY
52				RUN	READY
53				READY	RUN
54				READY	RUN
55				READY	RUN
56				RUN	EXIT
57				RUN	
58				RUN	
59				BLCK	
60				RUN	
61				RUN	
62				RUN	