

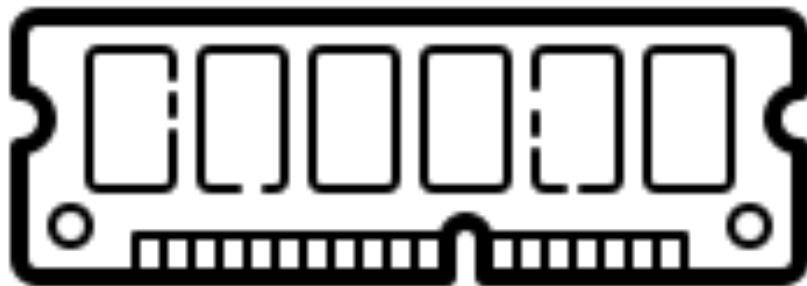


UNIVERSIDADE DE ÉVORA

Simulador de Sistema Operativo

Modelo de 5 Estados com Threads e Gestão
de Memória por Segmentação

Relatório do Trabalho Prático



Curso: Engenharia Informática

Disciplina: Sistemas Operativos

Docentes: Luís Rato

Entregue junho 2022

Alunos:

Joana Carrasqueira nº48566

João Condeço nº48976

João Oliveira nº48979

Descrição da estrutura do programa

De modo, a simular um sistema operativo foram criadas várias structs, PCB, TCB e SO. A **struct PCB** é utilizada para gerir a alocação da memória relativa a cada processo, tendo assim, nesta armazenadas as posições do início dos blocos de memória que contêm as instruções e variáveis de um processo. Para gerir a informação relativamente aos estados e a execução, informação utilizada no algoritmo Round Robin, recorreremos à **struct TCB** armazenando nesta o Program Counter, o ID da thread, o BurstTime, se esta tiver a espera de um thread por causa da instrução Join, e outras informações específicas de cada processo que auxiliam o funcionamento do programa. Por fim, a **struct SO** é responsável por conter o array representativo da memória (mem), informar qual o processo/thread que se encontra no estado RUNNING, indicar o quantum para a execução do Round Robin, o último ID atribuído a um processo (lastID) e a variável instsAlocated que verifica se já foi alocado o espaço para as instruções da thread. Na struct SO, também residem as queues ready e blocked que gerem, respetivamente, os estados READY e BLOCKED e a lista joinBlocked que contém os processos a correr a instrução JOIN.

O programa começa por inicializar todas as variáveis das structs com os valores iniciais pretendidos. Tendo isto feito, irá ser recolhido do ficheiro de input (formato txt) as informações relativas as instruções que devem ser executadas por cada processo e thread. Ao longo desta recolha de informação, o programa irá também registar a variável mais alta de cada processo e thread (de forma a posteriormente alocar a memória para as estas), o número de threads e o número de instruções dos processos e threads. De seguida, são inicializadas as variáveis da struct SO com base nas informações recolhidas e nalguns casos, como o quantum, com base nos valores indicados pelo enunciado.

Instrução	Var.	Descrição	Código
ZERO	X	Var_0 = X	0
COPY	X	Copia var_0 para var_X (var_x = var0)	1
DECR	X	Decrementa var_X	2
NWTH	X	Cria uma Thread número X. O parâmetro de entrada da thread está na VAR_X do processo principal, e será copiado para VAR0 da thread (var. Local da thread), o número da thread X será copiado para VAR1 (var. Local da thread)	3
JFRW	X	Jump forward X instruções	4
JBCK	X	Jump back X instruções	5
DISK	X	Pedido de I/O (processo passa a Blocked)	6
JIFZ	X	Jump if X = zero (se X= zero então salta duas instruções, senão segue para a próxima instrução)	7
PRNT	X	Print/ imprime o valor de Var_X	8
JOIN	X	Espera pelo fim da thread X	9
ADDX	X	Soma Var_0 com Var_x e o resultado vai para Var_0	10
MULX	X	Multiplica Var_0 com Var_x e o resultado vai para Var_0	11
RETN	X	Termina a thread e retorna ao processo principal	12
HALT	X	Halt / termina o processo	20 ou qualquer outro valor

Fig.1 – Informações relativas à recolha de informação


Por fim, é iniciada a execução dos processos dentro de um ciclo while onde estes são geridos através do algoritmo Round Robin, com quantum = 3 e a memória necessária para a execução destes é alocada segundo o algoritmo **NEXT FIT**. Dentro deste ciclo, primeiro irá ser verificado para todos os processos e respetivas threads se algum destes pode passar do estado BLOCKED para READY (no caso dos processos é também verificada se podem sair de modo JOIN para READY). De seguida, é feita a mesma verificação para as transições RUN para EXIT/BLOCKED/RUN/READY, NEW para READY, READY para RUN, se existe algum processo novo de que inicia no instante em questão (ou seja, entra no estado NEW), se existir algum processo que já terminou e seja

necessário limpar o espaço que este ocorre na memória, lida com o processo no estado RUN por esta mesma ordem. Finalmente, é verificado se já todos os processos saíram do programa e, caso se confirme, o ciclo termina juntamente com o programa.

Dificuldades encontradas

A dificuldade mais relevante no desenvolvimento do trabalho foi o facto do programa, em situações com mais de um processo, não apresentar o estado EXIT de um processo quando este sai de BLOCKED e executa a instrução HALT no output mas passar por este estado na prática.

Outputs



T	NEW	READY	RUN	BLOCKED	EXIT
00	P1				
01			P1		
02			P1		
03			P1		
04			P1		
05		TH1	P1		
06		TH1 TH2	P1		
07		P1 TH2	TH1		
08		P1 TH2	TH1		
09		P1 TH2	TH1		
10		P1 TH1	TH2		
11		P1 TH1	TH2		
12		P1 TH1	TH2		
13		TH2	TH1	P1	
14		TH2	TH1	P1	
15			TH2	P1	
16		P1	TH2		
17		P1	TH2		
18			P1		
19			P1		
20			P1		
21			P1		
22			P1		
> print 25					
23			P1		
24					P1

Fig.2 – Output do exemplo 1 do enunciado