Enunciado do Trabalho prático de Sistema Operativos 1

Ano lectivo 2018/19 - Universidade de Évora

Suponha uma arquitetura sobre o modelo de 5 estados que consome programas constituídos por um conjunto instruções.

As intruções são codificadas por conjuntos de 3 números inteiros.

Tabela das instruções e da sua codificação:

Codificação	Instruções	Significado
0, X1, X2	SET_X	Var X1 = var X2
1, X, N	SET_N	Var X = N
2, X, qq	INC_X	Var X = Var X + 1
3, X, qq	DEC_X	Var X = Var X - 1
4, N, qq	BACK_N	Salta para trás, PC -= N, em que N é logo o valor do salto, não se vai consultar o valor da variável
5, N, qq	FORW_N	Salta para a frente, PC += N, em que N é logo o valor do salto, não se vai consultar o valor da variável
6, X, N	IF_X_N	IF X ==0, salta N linhas para a frente PC+=N; ELSE vai para próxima linha PC++
7, X, qq	FORK X	X = Fork(), X é zero ser for o filho, ou o PID do processo criado se for o pai
8, X, qq	DISK_SAVE_X	Guarda a variável X no disco
9, X, qq	DISK_LOAD_X	Carrega a variável do disco para X
10, X, qq	PRINT_X	Imprime variável X
11, qq, qq,	EXIT	Termina

Exemplo:

X1 = 3
IF X1 ==0 SALTA 4
PRINT X1
X1=X1-1
BACK 3
EXIT

```
 1, 1, 3,
 6, 1, 4,
10, 1, 0,
 3, 1, 0,
 4, 3, 0,
11, 0, 0
```

Deste modo o programa seria representado por:
 1 1 3 6 1 4 10 1 0 3 1 0 4 3 0 11 0 0

Escalonamento Round Robin, Quantum 4 mas configurável (#define).
Quando no mesmo instante, um processo vindo do NEW, do BLOCKED, e /ou do RUN querem entrar no READY, o vindo do BLOCKED tem prioridade, seguido do de RUN, e por fim o processo vindo de NEW.
Nas operações do programa, o primeiro dígito indica a instrução e o segundo e terceiro indicam a variável e/ou um valor constante onde se aplica essa operação. Assim existem 12 operações diferentes e até um máximo de 10 variáveis disponíveis por programa (X1, X2, ... até X10).

O programa é guardado em memória, representada por um array de inteiros, ficando o PCB (process control block) com a indicação dum PID (process ID); da localização do programa; com o PC (Program Counter); e o estado do respetivo programa.

- O desenvolvimento das instruções deve ser modular.
- Todas as instruções consomem um ciclo temporal no CPU (estado RUN)
- As instruções são executadas no estado RUN, no entanto as intruções 8, e 9 obrigam à passagem para o estado BLOCK, e a 11 para o estado EXIT. A instrução 7 faz um fork duplicando o processo que irá para o estado READY (o novo processo, uma vez que o processo original continua no RUN caso ainda não tenha esgotado o seu Quantum).
- As instruções 8, e 9 simulam uma escrita e leitura em disco enviam o processo para estado BLOCK, consumindo 3 ciclos temporais em espera no estado BLOCK.
- No estado NEW, cada processo consome 1 instante de tempo.
- No estado EXIT, cada processo consome 1 instante de tempo, antes de ser apagado do sistema.

A memória onde as instruções são colocadas é estática e é representada por um array MEM[] que deve obrigatoriamente ser do tipo **int,** e que tem um limite de **300 posições** (este limite deve estar "#define" no vosso programa para permitir futuros ajustes).

Por simplicidade, tenha em conta que qualquer programa em memória reserva espaço para as 10 variáveis, (mesmo que não as use todas).

O programa acima referido poderia ser representado no array da memória MEM[] da seguinte forma:

	10 Variáveis reservadas					Instruções do Programa							
...	X1	X2	X3	...	X10	01	01	03	06	01	04	Etc.	...

Deverá escolher a forma de arrumar as variáveis (no princípio, no fim, ou em regiões separadas, ...)

Instrução Fork

É necessário implementar a função FORK e que esta seja completamente funcional.

A instrução faz um fork duplicando o processo que irá para o estado READY (o novo processo), o processo original continua no RUN caso ainda não tenha esgotado o seu Quantum.

Caso não haja espaço suficiente para criar o processo duplicado (no estado READY) prossegue apenas o processo original devolvendo o valor -1 na função fork.

Gestão de Memória

À medida que os processos terminam, o seu espaço em memória é libertado, fazendo com que a memória fique fragmentada. Para lidar com isso, os novos processos a chegarem à memória tem de aproveitar esses espaços livres devido à memória ser limitada.

Para tal, pretende-se implementar dois gestores de memória:

Best Fit

Next Fit

Deverá existir uma variável “Global” que define o qual gestor a utilizar durante a execução.

Input

Os ficheiros de input é composto por várias linhas, sendo cada linha um processo, em que o primeiro elemento de cada linha representa o instante de entrada do programa, e.g. o programa “ 1 1 3 6 1 4 10 1 0 3 1 0 4 3 0 11 0 0” poderia entrar no instante 0, e depois no instante 3. O ficheiro de entrada seria:

0 1 1 3 6 1 4 10 1 0 3 1 0 4 3 0 11 0 0
3 1 1 3 6 1 4 10 1 0 3 1 0 4 3 0 11 0 0

No Moodle encontram-se ficheiros com inputs de testes.

Output
Deve haver 2 modos de output que são escritos em ficheiro devidamente formatado (*scheduler_simples.out e scheduler_complexo.out*).

- Ficheiro *scheduler_simples.out* com o seguinte formato:
  INSTANTE | ESTADO_P1 | ESTADO_P2 | . . . | ESTADO_Pn
  Exemplo:
  . . .
  9 | exit | exit | ready | block | run | block | block | new
  10 | exit | exit | run | ready | exit | block | block | ready
  11 | exit | exit | ready | run | exit | block | block | ready | new
  . . .

- Ficheiro *scheduler_complexo.out* adiciona ao output *scheduler_simples.out* o print da memória sempre que um novo processo é colocado ou removido da memória

No Moodle e devera submeter um .zip com os número de aluno no nome do ficheiro, ex "ll4444.zip" e deverá conter o código fonte do trabalho assim como um relatório em PDF.

O trabalho deve ser desenvolvido em fases, e terá avaliações intermédias.

1º Fase: compreender o enunciado!!
Deve colocar todas as dúvidas para poder planear o trabalho, e evitar decisões erradas no início que poderão ter um custo elevado no desenvolvimento futuro do trabalho.

2º Fase: definir o PCB e a transição entre estados.
Deve realizar esta fase…
sem implementar as instruções, e
sem mapear os processos em memória (no array MEM[]).
sem implementar escalonador Round Robin. Inicie considerando que o escalonador é FCFS (fila FIFO).

Considere apenas que tem 3 tipos de instruções:
1) EXIT (instrução 11) que, depois de gastar um instante de tempo de CPU, passa o processo para EXIT;
2) DISK (instruções 8 e 9) que, depois de gastar um instante de tempo de CPU, passa o processo para BLOCK;
3) todas as outras gastam 1 instante de tempo no CPU, mas não implementam função nenhuma.

Implemente filas (pode usar arrays, ou listas) para gerir os processos (PCBs) que existirão em cada estado. Note que no estado RUN só pode haver um processo, uma vez que se considera que existe apenas um CPU.

3º Fase, e seguintes …. Seguir passo a passo, as orientações nas aulas práticas de modo a desenvolver o programa faseadamente e esclarecer possíveis dúvidas atempadamente.