Enunciado do trabalho final de Sistemas Operativos 1

Ano lectivo 2019/20 - Universidade de Évora

Pretende-se implementar um simulador de Sistema Operativo com um modelo de 3 ou 5 estados que consome programas constituídos por um conjunto instruções seguindo as especificações apresentadas abaixo. O trabalho pode ser realizado individualmente ou em grupos. Cada aluno deverá ser responsável por um sistema de gestão de memória. Dependendo do número de alunos os sistemas de gestão de memória a implementar são os seguintes:

1 aluno – memória com paginação;

2 alunos – paginação e um dos outros dois (à escolha) particionamento fixo ou dinâmico.

3 alunos – as três abordagens (paginação, particionamento fixo e dinâmico)

No caso do particionamento dinâmico deve implementar com BEST FIT ou NEXT FIT (à escolha).

Para o teste deverão existir versões dos programas com os seguintes nomes (dependendo das escolha dos algoritmos):

- paginação – so_pag

- particionamento fixo – so_fixo

- particionamento dinâmico BEST FIT – so_best

- particionamento dinâmico NEXT FIT – so_next

No Moodle deverá submeter um .zip com os números de aluno no nome do ficheiro, ex "14444_22222.zip" e deverá conter o código fonte do trabalho assim como um relatório em PDF.

Os trabalhos serão testados num sistema Ubuntu 18.04LTS, ao qual apenas foi adicionada o compilador gcc pelo que deverá instalar uma máquina virtual com virtualbox ou Vmware de modo a garantir que esteja a usar o ambiente correto.

Se for necessário a compilação de vários ficheiros deverá ser incluído, ou uma *makefile* ou um *shell script* para a compilação de cada executável com as opções que utilizou.

Especificações

Transição entre estados

Neste sistema considere que:

- todas a instruções consomem extamente 1 instante de tempo de CPU;
- cada chamada de I/O (que implicará a passagem do processo para o estado blocked) demora exatamente 5 instantes de tempo de CPU para ser despachada;
- o escalonamento de processos é feito de acordo com o algoritmo preemptivo Round-Robin com quantum igual a 3 instantes de tempo;
- quando no mesmo instante, um processo novo, e um processo de BLOCKED, e /ou do RUN querem transitar para o estado READY, o processo vindo do BLOCKED tem prioridade, seguido do de RUN, e por fim o processo novo;
- apesar de não ser necessário, se pretender usar um estado NEW e um estado EXIT
- 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 execução da instrução HLT (halt) termina o processo.
- quando se tenta criar um processo novo será necessário alocar o espaço necessário na memória principal. Caso não seja possível, a criação do processo aborta e há uma mensagem de erro "impossível criar processo" para o standard ouput.

Input: Programas e instruções

Os programas são definidos num ficheiro de entrada *input.txt* com o seguinte formato:

```
INI t1                    (início do programa 1)
instrução1
instrução2
instrução3
instrução4
instrução5
instrução6
etc …
INI t2                    (início do programa 2)
```

```
instrução1
instrução2
instrução3
instrução4
instrução5
instrução6
instrução7
etc …
INI t3                    (início do programa 3)
etc...
```

As instruções que constituem os programas são definidas do seguinte modo:

Instrução	Var.	Descrição	Código
ZER	X	Var_x = 0	0
INC	X	Incrementa var_X	1
DEC	X	Decrementa var_X	2
FRK	X	Fork (devolve o output para var_X) var_X é zero ser for o filho, ou o PID do processo criado se for o pai	3
JFW	X	Jump forward X instruções	4
JBK	X	Jump back X instruções	5
DSK	qq	Acede ao disco (fica em Blocked 5 instantes de tempo)	6
JIZ	X	Jump if X = zero (se X= zero então salta duas instruções, senão segue para a próxima instrução.	7
PRT	X	Print/ imprime o valor de Var_X	8
HLT	qq	Halt / termina o processo	9 ou qualquer outro valor

Exemplo de ficheiro com dois programas de entrada, o 1º inicia no instante 0 e o segundo inicia no instante 1:

```
INI 00
ZER 05
INC 05
INC 05
PRT 05
HLT 00
INI 01
ZER 06
INC 06
```

```
INC 06
INC 06
JFW 03
HLT 00
PRT 06
JBK 02
```

Memória – codificação dos programas e dados

O programa é guardado em memória, representada por um array **mem[]** de inteiros (int, long int ou char) com a dimensão de **200**.

Dados

Cada processo tem sempre espaço para 10 (e apenas 10) variáveis inteiras.

Codificação dos programas

Cada instrução é codificada por dois números inteiros: o primeiro é o código da instrução e o segundo é um parâmetro da instrução (de acordo com a tabela acima que descreve as instruções).

No caso das instruções DSK e HLT o valor numérico que se segue é irrelevante para a execução, no entanto deverá existir sempre um valor de modo a manter a uniformidade do formato das instruções. Deste modo **cada instrução ocupa sempre dois valores na memória** (duas variáveis inteiras).

Exemplo de programa:

```
ZER 04
INC 04
PRT 04
HLT 00
```

Representação das instruções na memória (cada instrução mais parâmetro.

0	Código de ZER
4	Var 04
1	Código de INC
4	Var 04
8	Código de PRT
4	Var 04
8	Halt
0	Qualquer valor

Representação dos dados na memória:

-1	Var 00
0	Var 01
4	Var 02
1	Var 03
4	Var 04
8	Var 05
4	Var 06
8	Var 07
2	Var 08
3	Var 09

Segurança e isolamento entre processos

Caso um processo tente executar uma instrução fora do espaço alocado para o código, o processo é terminado e há uma mensagem de erro "erro de segmentação do processo X" para o standard ouput.

Caso um processo tente saltar para fora do seu espaço alocado para o código, o processo é terminado e há uma mensagem de erro "erro de segmentação do processo X" .

Instrução Fork

É necessário implementar a função FRK (fork) e que esta seja completamente funcional.

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

Caso não haja espaço suficiente em memória para criar o novo processo (no estado READY) prossegue apenas o processo original devolvendo uma mensagem de erro para o stdout "fork sem sucesso"

Gestão de Memória

Pretende-se implementar (dependendo dos elementos do grupo) gestores de memória de acordo com:

1) Particionamento fixo:

- 1 slot de 40
- 2 slots de 30
- 5 slots de 20

2) Particionamento dinâmico: ou BEST FIT ou NEXT FIT.

3) Paginação com páginas de 10.

Note que à medida que os processos terminam, o seu espaço em memória é libertado. É portanto uma condição essencial ter uma gestão dos **espaço livre** (a informação sobre o espaço livre não poderá estar guardada no próprio array **mem[]).**

Output

O output para o stdout deve ter o seguinte formato:

```
. . .
  T  | stdout | READY                    | RUN  |  BLOCKED
. . .
09   |        | P2   P3   P4             | P5   |   P6
10   |   3    | P2   P3                  | P4   |   P5   P6
11   |        | P2   P3                  | P4   |   P5   P6
> erro de segmentação
12   |        | P2                       | P3   |   P5   P6
13   |   4    | P2                       | P3   |   P5   P6
14   |        | P2                       | P3   |   P5   P6
> fork sem sucesso
15   |        | P2                       | P3   |   P5   P6
etc . . .
```

Notas e esclarecimentos adicionais:

- os saltos (forward e back) são o número de linhas que salta para trás ou para a frente. Se a instrução for um salto para a frente "**JFW 1**" significa que passa para a próxima instrução (que é o "normal") portanto se quiser saltar por cima da próxima instrução terá de fazer um "**JFW 2**".