Enunciado do trabalho2 de Sistemas Operativos 1

Ano letivo 2020/21 - Universidade de Évora

Pretende-se implementar um simulador de Sistema Operativo com um modelo de 5 estados que simula programas constituídos por um conjunto instruções seguindo as especificações apresentadas abaixo. O trabalho pode ser realizado individualmente ou em grupos de dois ou três alunos, devendo implementar sistemas de gestão de memória do seguinte modo:

- trabalho individual: BEST FIT

- grupos de 2 alunos: um deverá ser responsável por NEXT FIT; o outro será responsável pela gestão de memória com FIRST FIT.

- grupos de 3 alunos: cada um dos alunos será responsável pelos algoritmos FIRST FIT NEXT FIT e BEST FIT.

Para o teste deverão existir versões dos programas com os seguintes nomes:

- particionamento dinâmico BEST FIT – so_best

- particionamento dinâmico NEXT FIT – so_next

- particionamento dinâmico FIRST FIT – so_first

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 20.04LTS, ao qual apenas foi adicionada o compilador gcc pelo que deverá instalar essa versão ou 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 incluída, 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 exatamente 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 para ser despachada; este tempo só conta para o processo que está na cabeça da fila blocked.
- o escalonamento de processos é feito de acordo com o algoritmo preemptivo Round-Robin. O quantum default deve ser igual a 3 instantes de tempo, mas deve ser configurável.
- quando no mesmo instante, um processo vindo de NEW, 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 de NEW;
- No estado NEW, cada processo consome 1 instante de tempo.
- No estado EXIT, cada processo consome 1 instante de tempo, após o que é apagado do sistema.
- a execução da instrução HLT (halt) termina o processo, passando assim para estado EXIT.
- quando se tenta criar um processo novo (a partir do input ou devido a um fork) será necessário alocar o espaço necessário na memória principal. Caso não seja possível, a criação do processo aborta e há uma mensagem de erro para o standard ouput:
- "impossível criar processo: espaço em memória insuficiente", ou
- "impossível fazer fork: espaço em memória insuficiente"

Input: Programas e instruções

Os programas são definidos, através duma matriz hardcoded, ou dum ficheiro de entrada *input.txt* com o seguinte formato:

```
INI t1          (início do programa 1; t1 é o instante inicial)
instrução1
instrução2
instrução3
instrução4
instrução5
instrução6
etc …
```

INI t2 (início do programa 2; t2 é o instante inicial)
instrução1
instrução2
instrução3
instrução4
instrução5
instrução6
instrução7
etc …
INI t3 (início do programa 3; t3 é o instante inicial)
etc...

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

Instrução	Var.	Descrição	Código
ZER	X	Var_0 = X	0
CPY	var_X	Copia var_0 para var_X (var_x = var0)	1
DEC	X	Decrementa var_X	2
FRK	X	Fork (devolve o output para var_X) var_X será zero ser for o filho, ou será 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	Pedido de I/O (e passa em Blocked)	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 primeiro inicia no instante 0 e o segundo inicia no instante 1:

```
INI     0       (instante inicial é 0)
ZER   500       (var_0  = 500 )
CPY    13       (var_13 = var_0 que é igual a 500  )
CPY     7       (var_7   = var_0 que é igual a 500  )
PRT    13       (imprime o valor de  var_13 para output)
HLT  -999       (termina)
INI     1       (instante inicial é 1)
ZER   100       (var_0  = 100 )
CPY     5       (var_5 = var_0 que é igual a 100)
DEC     5       (var_5 = var_5-1)
JFW     3       (salta para a 3ª instrução em frente)
HLT    99       (termina)
PRT     5       (imprime o valor de  var_5 para output)
JBK     2       (salta para a 2ª instrução, para trás)
```

Memória – codificação dos programas e dados

O programa é guardado em memória, representada por um array **int mem[]** com a dimensão de **200**.

Dados

Cada processo tem sempre espaço para variáveis inteiras. No mínimo é obrigado a ter espaço para uma variável, o v_0 (“v_zero”) mas pode ter mais. É necessário verificar qual o endereço de dados (Var_x) mais elevado e reservar espaço para todas essas variáveis. Por exemplo no primeiro programa o var_x mais elevado é 13, logo é preciso reservar espaço para 14 variáveis (de var_0 a var_13).

Nota: naturalmente este é um processo muito simplificado de atribuição de espaço de memória para as variáveis de programa, e que é pouco eficiente, (das 14 variáveis apenas três estão efetivamente usadas – var_0, var_7 e var_13) um compilador faria algo semelhante mas de modo muito mais eficiente, reservando apenas o espaço de memória necessário para as três variáveis. No entanto, é este o método que deve ser seguido neste trabalho.

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:

```
INI      1      (instante inicial é 1)
ZER    100      (var_0  = 100 )
CPY      5      (var_5 = var_0 que é igual a 100)
DEC      5      (var_5 = var_5-1)
JFW      3      (salta para a 3ª instrução em frente)
HLT   -999      (termina)
PRT      5      (imprime o valor de  var_5 para output)
JBK      2      (salta para a 2ª instrução para trás)
```

Representação das instruções na memória (a cada instrução segue-se um parâmetro).

0	ZER
100	100
1	CPY
5	5
2	DEC
5	5
4	JFW
3	3
99	HLT
-999	Qualquer valor
8	PRT
5	5
5	JBK
2	2

Representação dos dados na memória:

100	**Var 00**
1	Var 01
-2	Var 02
3	Var 03
-4	Var 04
99	**Var 05**

Deste modo seria necessário reservar espaço para dois segmentos: um de 14 espaços para o executável, e outro de 6 espaços para os dados.

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). 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:

- Segmentação (apenas dois segmentos) com particionamento dinâmico: BEST FIT ou NEXT FIT, e FIRST FIT.

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[]).**

Escalonamento Round Robin, Quantum 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.

Output

O output para o stdout deve ter o seguinte formato mostrando os processos em cada fila:

```
. . .
 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**".
- ao fim de 100 instantes o simulador deve terminar mesmo que ainda existam processos no sistema.

Pontos extra/opcionais:

- input através de ficheiro
- partilha dos segmentos executáveis entre processos pai e filho
- sistema de desfragmentação (que se ativa de 50 em 50 instantes)