Enunciado do trabalho-2 de Sistemas Operativos

Ano letivo 2024/25 - Universidade de Évora

Este trabalho vem na sequência do trabalho 1. O trabalho deve ser realizado exclusivamente em grupos de dois ou três alunos. 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, os ficheiros de output gerados e um relatório em PDF. O código fonte deverá ser escrito na linguagem C e incluir um ficheiro README com instruções, ou em alternativa um makefile, ou um shell script que permitam a compilação. O código compilado deverá gerar dois executáveis, um para executar cada parte do trabalho 2. Os outputs deverão ser gerados para os inputs fornecidos no Moodle e nomeados conforme indicado na descrição do trabalho. O relatório deve conter uma descrição sucinta da estrutura do programa proposto (máximo 6 pág) composto por explicação das estruturas de dados utilizadas e funções do(s) simulador(es).
O trabalho está dividido em 2 partes: parte 1) Implementar um simulador de memória que pode utilizar duas estratégias de alocação diferente para paginação; e parte 2) integrar o simulador de memória no trabalho 1, adicionando novos estados.

Parte 1: Simulador Gestor de Memória

Pretende-se implementar um simulador de gestão de memória com paginação e memória virtual de acordo com:

1) A memória principal tem tamanho 21kB (kilobyte) e cada Página/Frame tem tamanho 3kB.

2) O espaço de endereçamento de cada processo/programa é dado por:

```
int inputP1Mem00[] = {5000, 11000, 2000, 6000, 4000};
```

O **índice +1** indica o identificador do processo e o valor indica a memória que o processo ocupa em bytes, por exemplo, o processo 1 ocupa 5kB e o processo 2 ocupa 11kB. Um processo ocupa no máximo 11kB.

3) A ordem de execução e endereço de memória a aceder, usada apenas nesta parte do trabalho, é dada por

```
int inputP1Exec00[] = {1, 4000, 2, 6000, 3, 1000, 3, 2000, 4, 3000, 5, 2444, 5, 2, 4, 1, 2, 4000, 2, 10000, 1, 4999, 2, 0};
```

Em que cada par de valores indica o identificador do processo e o endereço de memória a aceder, segundo a ordem pela qual deverão ser executados. Por exemplo, o processo 1 executa primeiro e carrega a página onde se encontra o endereço 4000 em memória, e o segundo processo a ser executado é o processo 2 e carrega a página onde se encontra o

endereço `6000` em memória. Quando não houver frames livres em memória, será necessário executar um algoritmo de substituição (ver ponto 7).
Nesta Parte 1 do trabalho 2, para simplificar, assume-se que **cada processo executa por um instante de tempo e acede sempre a um endereço de memória**. Esta condição vai ser alterada na Parte 2.

4) À semelhança dos sistemas reais que utilizam memória virtual, neste trabalho, um processo para executar precisa ter a página onde se encontra o endereço que quer aceder carregada na memória principal. Quando não está carregada em memória, assume-se que a página encontra-se guardada em disco.

5) Quando existe a tentativa de aceder a um endereço de memória que excede o espaço de endereçamento do processo, acontece um Segmentation Fault. Nesse caso, o simulador deverá imprimir *SIGSEGV* e terminar o processo. Independentemente do input dado, um processo que terminou, não volta a executar durante a simulação.

6) Quando um processo já não tem mais execuções planeadas, considera-se que não termina e as suas páginas continuam em memória da mesma forma que um processo ativo, até serem substituídas pelo algoritmo de substituição. Caso o processo gere um erro (SIGEGV), a memória onde as suas páginas estão guardadas passam a estar livres.

7) Para selecionar as páginas a substituir, deverá aplicar um algoritmo de substituição por página do processo, por ordem crescente de página no contexto do processo. Para isso, terá de implementar dois algoritmos:

a) First-in-first-out (FIFO)
b) Least Recently Used (LRU)

Devem ser gerados dois tipos diferentes de output por tipo de algoritmo e nomeados, respetivamente, fifoXX.out e lruXX.out em que XX é o identificador dos inputs por exemplo 00 é o identificador de `inputP1Mem00 e inputP1Exec00`.

8) Como output, o simulador deverá imprimir no terminal o estado da memória principal em cada instante, para cada processo devem ser indicadas as frames alocadas por processo e ordenadas pela página do processo. O formato deve seguir o seguinte exemplo utilizando o algoritmo FIFO para a configuração de memória e execução dadas pelos `inputMem00 e inputExec00`:

time inst	proc1	proc2	proc3	proc4	proc5
0	F0				
1	F0	F1			
2	F0	F1	F2		
3	F0	F1	SIGSEGV		
4	F0	F1		F2	
5	F0	F1		F2	F3
6	F0	F1		F2	F3
7	F0	F1		F4,F2	F3
8	F0	F5,F1		F4,F2	F3

```
9          F0                  F5,F1,F6                        F4,F2               F3
10         F0                  F5,F1,F6                        F4,F2               F3
11                             F0,F5,F1,F6                     F4,F2               F3
```

Para o mesmo input, mas executando o algoritmo LRU, o output será:

```
time inst proc1              proc2               proc3              proc4              proc5
0          F0
1          F0                F1
2          F0                F1                  F2
3          F0                F1                  SIGSEGV
4          F0                F1                                     F2
5          F0                F1                                     F2                 F3
6          F0                F1                                     F2                 F3
7          F0                F1                                     F4,F2              F3
8          F0                F5,F1                                  F4,F2              F3
9          F0                F5,F1,F6                               F4,F2              F3
10         F0                F5,F1,F6                               F4,F2              F3
11         F0                F1,F5,F6                               F4,F2              F3
```

Repare que as Frames estão separadas por vírgula e cada bloco de Frames está alinhado com os processos na primeira linha.
Sugestão: Utilize a função sprintf ou snprintf para criar uma string (um buffer de chars) com as frames de determinado processo. Depois pode utilizar a função printf com uma string no argumento e definir um tamanho fixo de caracteres (e.g., `%-23s`) para obter um output estruturado em tabela no terminal.

9) Caso esteja mais que uma frame livre na memória principal, seleciona a que tem o índice menor na memória principal.

Parte 2: Simulador de escalonamento e memória

A parte 2 do trabalho consiste em integrar o simulador de escalonamento realizado no 1º trabalho com o gestor de memória desenvolvido na Parte 1 deste trabalho.
Para isso deverá ter em conta as seguintes alterações:

1) A ordem dos processos a executar em RUN é dada pela solução desenvolvida no trabalho 1.

2) Assim como na Parte 1, a memória principal tem tamanho 21kB e cada Página/Frame tem tamanho 3kB.

3) Nesta parte do trabalho, para selecionar as páginas a substituir utilize o algoritmo LRU. Em caso de empate entre páginas deverá ser selecionada para substituição a página cujo ID da Frame é menor.

4) São adicionadas novas instruções em relação ao primeiro trabalho que implicam o acesso a endereços de memória:

- LOAD/STORE - instruções de 1 000 a 15 999, indicam que um endereço de memória é acedido. O endereço é obtido ao subtrair 1000 à instrução, por exemplo 11 003 indica o endereço 10 003

- SWAP/MEMCPY (bónus +2 valores) - instruções de 1 000 000 000 a 2 109 999 999. O primeiro endereço é dado pelos dígitos 1 a 5 por ordem de mais significativo a menos, ao resultado destes dígitos subtraímos 10 000. O segundo endereço é dado pelos dígitos 6 a 10. Por exemplo, a instrução 2 103 310 238 acede ao endereço de memória 11 033 e ao endereço de memória 10 238.
  A primeira página a ser carregada é dada pelos dígitos mais significativos. O tempo de acesso é igual para ambos os endereços. Caso uma página onde se encontra um dos endereços seja selecionada pelo algoritmo de substituição e seja necessário alocar duas páginas, a página em questão mantêm-se em memória.

5) Assim como na parte 1, se existir uma tentativa de aceder a um endereço de memória que não está reservado ao processo, a instrução é cancelada, o simulador deverá mostrar o erro SIGSEGV e terminar o processo. No caso da instrução SWAP/MEMCPY, nenhuma página é carregada em memória.

6) A instrução JUMP é alterada passando a existir as instruções JUMPB e JUMPF.

- A instrução JUMPB, identificada pelas instruções de 101 a 199, têm o mesmo comportamento do trabalho 1. A lembrar, fazem com que o Program Counter do processo ande para **trás** *n* instruções, em que *n* é corresponde aos dois dígitos menos significativos. Por exemplo, a instrução 103 indica que o Program Counter vai recomeçar na 3ª instrução anterior, 101 indica a instrução imediatamente anterior à instrução JUMPB.
- A instrução JUMPF, identificada pelas instruções de 1 a 100, têm um comportamento idêntico ao da instrução JUMPB, no entanto, o Program Counter anda para a frente em vez de andar para trás. Ou seja, fazem com que o Program Counter do processo ande para **a frente** *n* instruções, em que *n* é corresponde aos dois dígitos menos significativos. Por exemplo, a instrução 3 indica que o Program Counter vai recomeçar na 3ª instrução seguinte, 101 indica a instrução imediatamente seguinte à instrução JUMPF.

7) As instruções JUMP (JUMPF e JUMPB) podem tentar aceder a instruções fora do programa, caso em que o simulador deverá mostrar o erro SIGILL e terminar o processo.

8) Quando um programa não contêm a instrução Halt (0) e tentar executar uma instrução depois da última deverá mostrar o erro SIGEOF e terminar o processo.

9) Quando for emitido um erro, o processo deverá terminar passando para Exit.

10) Os processos quando mudam para o estado Exit, ficam nesse estado 3 instantes, independente se devido à instrução Halt ou não.

11) Assim como no trabalho 1, considere que são fornecidos os pseudo-programas e que apenas um processo é lançado correndo o primeiro programa. No entanto, considere ainda que a primeira linha (antes destinada a instruções) agora indica o espaço de endereçamento do processo que executa determinado programa. As instruções iniciam na segunda linha.
exemplo:
Considere os seguintes 3 pseudo-programas. Neste sistema é sempre iniciado um processo no instante 1 executando o 1º programa (1ª coluna). Os restantes poderão ser executados a partir do primeiro usando as instruções EXEC.
Os valores na primeira linha indicam o espaço de endereçamento do processo em bytes que executa determinado programa, por exemplo, o programa 1 ocupa 10Kb e o programa 2 ocupa 11Kb. Um programa ocupa no máximo 11Kb.

```
int input00[8][20] = {
    { 10000, 11000,  4440, 1000 },
    {   203,   204,  1000, 1500 },
    {  9000,  2000,   202, 2000 },
    {    -4,  4000,  5000,    0 },
    {  4000,  7000,     4,    0 },
    { 10999, 10000,   400,    0 },
    {  1000,  4000,   400,    0 },
    {  9000,   105,     0,    0 }};
```

12) O output deverá ser uma combinação da Parte 1 do trabalho 2 e do trabalho 1. Em que depois de se imprimir cada estado, entre parênteses retos ([ ]) são impressas as Frames por ordem das páginas dos processos. Por exemplo, o output para os inputs dados por `input2T00` é:

```
time inst  proc1                proc2                proc3                proc4
...
1          NEW
2          NEW
3          RUN []               NEW
4          RUN [F0]             NEW
5          RUN [F0]             READY []
6          BLOCKED [F0]         RUN [F1]
7          BLOCKED [F0]         RUN [F1]             NEW
8          READY [F0]           RUN [F1,F2]          NEW
```

…continua

13) Quando o processo termina (após o estado EXIT) as frames do processo são libertas da memória, ou seja, ficam livres para utilização e não são impressas no output.

14) Se todos os processos terminarem antes dos 100 instantes, o simulador deverá terminar assim que deixar de haver estados para imprimir.

15) O output para cada teste `input2TXX` deverá ser guardado num ficheiro de texto em separado, nomeado output**2T**XX.out