| ESTGF POLITÉCNICO DO PORTO | Tipo de Prova: **Exame modelo**<br>Curso:<br>U. C.: **Sistemas Operativos** | Data: Janeiro 2017<br>Hora: 10:00<br>Duração: 2h30m |
|---|---|---|

Observações: **Com consulta de documentação própria.**
**O tempo previsto para responder a cada questão é apresentado entre parêntesis reto.**
**A cotação atribuída a cada pergunta é apresentada entre parêntesis curvo.**

# Grupo I

1. (1,0 valores)
Para cada uma das seguintes afirmações deverá indicar se as considera verdadeiras ou falsas. Caso considere alguma afirmação como **falsa** deverá **rescreve-la, transformando-a numa afirmação verdadeira**. À simples negação não será atribuída nenhuma cotação.

   a) [2,5 min] Em escalonamento *preemptivo*, após a atribuição do CPU ao processo, este é executado até ao fim.

   b) [2,5 min] Uma situação de bloqueio acontece sempre que surge uma espera circular com resolução.

   c) [2,5 min] Só se mantém um processo na fila de recursos prontos em sistemas multi-núcleo.

   d) [2,5 min] A técnica de evitar *deadlocks* necessita do suporte para o *rollback* sobre recursos .

2. [10 min] (1,0 valores)
"Um processo, quando solicita uma operação de I/O, sai de imediato da fila de processos *waiting*".

   **Comente** a afirmação, indicando também se **concorda ou não** com a mesma. Fundamente a sua resposta com um **exemplo concreto**.

3. [10 min] (1,0 valores)
"Um processo filho criado por execução, quebra a sua relação com o processo pai".

   **Comente** a afirmação, indicando também se **concorda ou não** com a mesma. Fundamente a sua resposta com um **exemplo concreto**.

4. [10 min] (1,5 valores)
Assumindo um sistema com 64K de memória RAM que implementa memória virtual, por *pagging*, com páginas de 16K. Indique, recorrendo à técnica MMU e à seguinte tabela, a que endereços físicos correspondem os endereços virtuais: **13820**; **301**.

| | |
|---|---|
| 11 | 1 |
| 00 | 0 |
| 10 | 1 |
| 00 | 0 |
| 00 | 1 |
| 01 | 1 |
| 00 | 0 |
| 00 | 0 |

5. [10 min] (1,5 valores)
Considere um computador com **1024KB** de memória que utiliza um sistema operativo que faz a gestão de memória pelo algoritmo *buddy*. Apresente uma representação de como a memória ficaria dividida considerando o estado atual e após a lista de acontecimentos apresentados de seguida:

**Estado atual:**
L1024:
L512: (H|512)
L256: (H|256)
L128: (P1|0)
L64: (P2|128)-(H|192)
...

**Lista de eventos**
1. Novo processo (P3) com 229K tamanho
2. Novo processo (P4) com 63K tamanho
3. Novo processo (P5) com 137K tamanho
5. Novo processo (P6) com 26K tamanho

6. [15 min] (2,0 valores)
Considere o seguinte conjunto de processos. Assuma que os processos chegam no instante de tempo indicado na tabela seguinte:

| Processo | Instante de chegada | Duração |
|---|---|---|
| P1 | 0,0 | 1,0 |
| P2 | 0,0 | 1,5 |
| P3 | 0,5 | 1,0 |
| P4 | 0,5 | 0,6 |
| P5 | 1,0 | 0,4 |

Calcule o tempo médio de turnaround, considerando que o algoritmo de escalonamento é o **SRTF**. Fundamente a sua resposta com todos os cálculos que sentir necessidade de efectuar.

7. [15 min] (2,0 valores)
Assuma um sistema com os tipos de recursos (A, B, . . . ), processos (P1, P2, . . . ) e caracterização como apresentada nas tabelas seguintes.

**Alocação**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| P1 | 1 | 2 | 0 | 1 | 1 |
| P2 | 1 | 0 | 1 | 2 | 0 |
| P3 | 0 | 0 | 1 | 0 | 0 |
| P4 | 1 | 0 | 4 | 1 | 2 |
| P5 | 1 | 1 | 1 | 1 | 1 |

**Necessidades máximas**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| P1 | 5 | 4 | 1 | 7 | 3 |
| P2 | 2 | 3 | 2 | 4 | 1 |
| P3 | 5 | 4 | 1 | 0 | 6 |
| P4 | 5 | 1 | 4 | 2 | 3 |
| P5 | 2 | 2 | 3 | 5 | 4 |

**Disponibilidades**

| A | B | C | D | E |
|---|---|---|---|---|
| 3 | 1 | 2 | 6 | 3 |

Valide, recorrendo ao algoritmo do banqueiro, se existe um sequência de execução que mantenha o sistema num estado seguro. Justifique a sua resposta

## Grupo II

8. [20 min] (3,0 valores)
Considere a classe *SemaphoreTest* apresentada na listagem seguinte:

```
1  import java.util.concurrent.*;
2  public class SemaphoreTest{
3      static Semaphore s = new Semaphore(0);
4
5      public void fun(final char c, final int r) throws Exception {
6          new Thread(new Runnable(){
7              public void run(){
8                  try{
9                      System.out.println("acquire␣"+r);
10                     s.acquire(r);
```

```
11                      System.out.println(c+"_"+r);
12                  } catch(Exception e){ e.printStackTrace(); }
13              }
14          }).start();
15          Thread.sleep(500);
16      }
17
18      public static void main(String[]args) throws Exception{
19          SemaphoreTest f = new SemaphoreTest();
20
21          f.fun('B',2);
22          f.fun('F',6);
23          f.fun('A',1);
24          f.fun('C',3);
25          f.fun('D',4);
26          f.fun('E',5);
27
28          while(s.hasQueuedThreads()){
29              Thread.sleep(1000);
30              System.out.println("release␣"+1+",␣available␣"+(s.
                    availablePermits()+1));
31              s.release(1);
32          }
33      }
34  }
```

Qual será o resultado esperado se se substituir as chamadas às funções *fun()* (linhas 21 a 26) pelas seguintes:

```
1  f.fun('C',7);
2  f.fun('B',2);
3  f.fun('A',5);
```

9. [50 min] (7,0 valores)

Elabore duas classes: **Produtor** e **Consumidor**. Estas classes deverão trocar informação entre elas recorrendo a um *buffer* com 5 carateres de capacidade máxima. À classe **Produtor** compete a geração de carateres e a sua escrita no *buffer*. À classe **Consumidor** compete a leitura de caracteres do *buffer*.

A classe **Produtor** deve poder escrever até que encha o *buffer*, altura em que deve ser suspensa até que o *buffer* volte a ficar vazio. A classe **Consumidor** só devem conseguir ler do *buffer* quando este estiver completamente cheio, adormecendo quando este estiver vazio. O acesso ao *buffer* deve ser executando sempre em exclusividade de acesso, independentemente de ser um acesso de escrita ou leitura.