Exemplos de Questões para o 1º teste de FSO
(incluídas em testes anteriores da autoria de J. C. Cunha)

1) Pretende-se implementar as operações P(s) e V(s) sobre semáforos genéricos, segundo a definição de Dijkstra. Para esta implementação assuma que dispomos de semáforos binários, que só podem assumir os valores 0 ou 1 e que suportam as correspondentes operações Pb e Vb (em que o indice b indica tratar-se de um semáforo binário).

2) Apresente, em pseudo-código, uma implementação, em modo utilizador, das operações P(s) e V(s) sobre semáforos genéricos baseando-se em operações em semáforos binários. Nesta solução assuma que apenas dispõe de dois semáforos binários previamente criados e inicializados: mutex com o valor inicial 1 e block com o valor inicial 0. Assuma também que tem acesso ao valor inteiro s do semáforo que se pretende implementar.

3) Considere o seguinte programa concorrente em que os dois processos P1 e P2 executam acções sobre as seguintes variáveis partilhadas: um inteiro turn e um vector de inteiros flag[]:

```
int turn; /* o valor inicial de turn é irrelevante para o programa */

int flag[2]; /* os valores iniciais são flag[0] = flag[1] = 0 */
```

```
Processo P1                                          Processo P2

while (TRUE) {                                       while (TRUE) {

   1. flag[0] = 1;                                      1. flag[1] = 1;

   2. turn = 1;                                         2. turn = 0;

   3. while (flag[1] == 1 && turn == 1) ;               3. while (flag[0] == 1 && turn == 0) ;

   4. REGIÃO CRÍTICA                                    4. REGIÃO CRÍTICA

  5. flag[0] = 0;                                       5. flag[1] = 0;

}                                                    }
```

As acções numeradas de 1 a 3 (para a entrada) e 5 (para a saída) em cada processo pretendem implementar um protocolo de acesso que deve garantir a exclusão mútua da execução das duas regiões críticas indicadas. Diga se esta é uma solução correcta:

(i) Garante a exclusão mútua? Justifique.

(ii) Garante que não ocorrem situações de impasse em que os processos se bloqueiem mutuamente na tentativa de entrada nas suas regiões críticas? Justifique.

(iii) Evita situações de injustiça, em que um processo veja a sua tentativa de entrada eternamente adiada pelo facto de o outro processo aceder repetidamente à região crítica (em ciclo: entrando, acedendo e saindo)? Justifique.

4) Considere uma zona de memória partilhada por N processos concorrentes e com a capacidade máxima para conter, a cada momento, um único bloco de 1Kbytes. Admita que esta zona foi previamente reservada pelo SO e se mantém permanentemente em memória. Inicialmente a zona está vazia, ou seja tem um conteúdo indefinido.

O processo de identificador (pid) P1 produz um bloco (também de 1Kbytes) de cada vez e depois tenta escrevê-lo naquela zona de memória. Cada um dos restantes processos, de identificadores P2, ... PN, deve tentar ler o conteúdo de cada bloco escrito por P1. Só após todos os processos (P2, ... PN) terem lido o conteúdo de um dado bloco, será este considerado lido e a zona de memória será considerada livre (permitindo que P1 escreva outro bloco na mesma zona de memória).

- O processo P1 invoca a função pôrBloco(char *buf) que o bloqueia até que a zona de memória esteja livre e, então, copia o conteúdo apontado por buf para a zona de memória partilhada, invocando uma função auxiliar colocar_bloco(char *buf), que se assume já implementada (mas não garante condições de sincronização). O bloco buf tem também o tamanho de 1Kbytes.

- Cada um dos processos P2, ... PN invoca a função lerBloco(char *buf), a qual executa a seguinte sequência de passos:

1. bloqueia o processo até que exista um novo bloco para ler;

2. obtém uma cópia do bloco, para uma zona de memória do processo apontada pelo argumento buf. Para obter a cópia do bloco, dispõe-se de uma função auxiliar obter_bloco(char *buf), devolvendo o apontador para a memória para onde copia o bloco lido e que se assume já implementada, incluindo a reserva da memória (mas não garante condições de sincronização).

3. aguarda, bloqueando-se até que todos os outros processos (do conjunto P2PN) tenham também lido esse bloco e só então a função lerBloco retorna ao processo invocador. NOTA: não é necessário permitir que as leituras do bloco pelos processos P2-PN sejam simultâneas.

Apresente o código C das funções pôrBloco(buf) e lerBloco(buf), que devem garantir todas as condições de sincronização deste problema, baseando-se exclusivamente em memória partilhada e em semáforos segundo a definição de Dijkstra.

5) Explique as diferenças entre os conceitos de Processo (tal como criado por fork no Unix) e de Thread (tal como criado por pthread_create). Para cada conceito, Processo e Thread, descreva em que consiste o contexto que define o respectivo ambiente de execução.

6) Considere a função int launch (char *file_exec, char *argv[])

Esta função cria um processo para executar o programa contido no ficheiro executável "prog_exec", sendo "argv" o vector contendo os argumentos do programa a executar. A função retorna “de imediato”, sem esperar que o processo criado termine a sua execução, devolvendo, ao pai, o identificador ``process_id'' do processo criado, ou o valor -1, em caso de erro.

a) Programe, em C, a função "launch", usando chamadas ao sistema UNIX.

b) Admitindo a função da alínea a), considere a seguinte sequência de operações, invocadas por um processo inicial p0. Admita que p1 e p2 são dois ficheiros executáveis e argv1 e argv2 são os seus respectivos vectores de argumentos:

pid1 = launch(p1, argv1);

pid2 = launch(p2, argv2);

Imagine que os programas de p1 e de p2 têm as seguintes sequências de operações:

p1	p2
...	...
parte1	parte2
...	...
parte3	parte4
...	...

Ao executar os programas dos dois processos p1 e p2, num sistema como o Unix, indique quais são as possíveis ordens de execução das operações parte1/2/3/4. Justifique a resposta.

c) Se, na alínea b), pretendermos que o processo inicial p0 espere pela terminação dos dois processos p1 e p2, antes de prosseguir, indique, usando chamadas ao sistema Unix, como garantir isso.

d) Se, na alínea b) pretendermos que o processo inicial p0 aguarde apenas pela terminação do processo p2, antes de prosseguir, indique, usando chamadas ao sistema Unix, como garantir isso.