

Número:

Nome:

**LEIC/LERC – 2011/12 - 1º Teste de Sistemas Operativos**  
**19 de Novembro de 2011**

**Responda no enunciado, apenas no espaço fornecido.**  
**Identifique todas as folhas.**  
**Justifique todas as respostas.**  
**Duração: 1h30m**

**Grupo I [7 Val]**

1. Considere um sistema operativo de tempo partilhado, com preempção, e que o seguinte pedaço de código se executa em nível utilizador num dado processo P1:

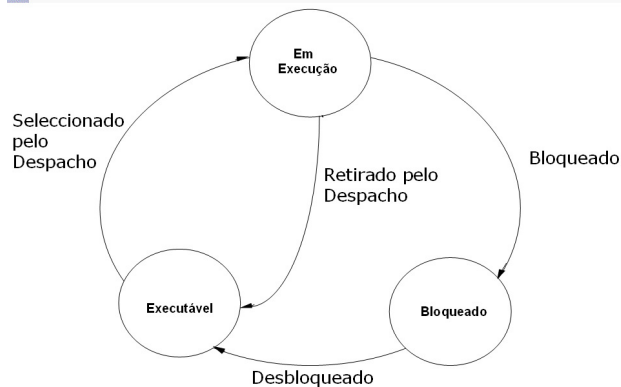
```
while (1);
```

Obviamente, deve ter em conta que existem outros processos no sistema em causa.

- a. [0.9 Val] Quando P1 executa o código acima indicado por uma tarefa real dentro de P1, diga em que condições e quando é que pode perder o processador.


- b. [0.8 Val] Como compara a comutação de tarefas reais e de pseudo-tarefas no que diz respeito ao desempenho?


2. Considere o seguinte diagrama de estados dos processo de um sistema operativo de tempo partilhado, com preempção:



- a. [0.8 Val] Dê dois exemplos de funções que, se chamadas por um processo no estado “Em Execução”, podem fazer com que este transite para o estado “Bloqueado”. Indique qual ou quais os objectos do sistema operativo que são implicados.


- b. [0.8 Val] Considere que um processo P1 se encontra no estado “Em Execução”. Descreva de forma detalhada as circunstâncias em que P1 passa para o estado “Executável” sem ter chegado ao fim do seu *time-slice*, por ter sido alvo de preempção. (Sugestão: refira explicitamente quais as funções sistema relevantes.)


3. Considere o sistema operativo Unix.

- a. [0.7 Val] Para que serve a pilha núcleo de cada processo?


- b. [0.7 Val] Quando é que um dado processo tem uma pilha núcleo não vazia?


- c. [0.8 Val] Considere a frase seguinte: “As prioridades dos processos variam de modo a que a utilização de recursos seja equitativa”. Em que circunstância é que a frase anterior não é verdadeira?


d. Considere as estruturas user e proc.

- i. [0.8 Val] Qual o tipo de informação contém cada estrutura? Dê exemplos de 3 campos de cada estrutura.


- ii. [0.7 Val] Qual a razão da separação da informação de contexto dos processos em duas estruturas?


### Grupo II [8v]

Considere o seguinte jogo, chamado paintShoot, com as seguintes regras:

1. O paintShoot é jogado por diferentes jogadores, divididos em  $E$  equipas ( $E > 1$ ).
2. Cada equipa é distinguida por uma cor, cor essa que é identificada por um inteiro maior que 0 (vermelho=1, verde=2, azul=3, amarelo=3, etc). Cada jogador tem uma arma de *paintball* com balas de tinta da cor da sua equipa.
3. O jogo desenrola-se num recinto composto por  $N > 0$  alvos. Cada alvo encontra-se inicialmente limpo (mais precisamente, com cor branco=0).
4. O objectivo de cada jogador é encontrar alvos ainda em branco e pintá-los com a cor da sua equipa. Repetidamente, cada jogador escolhe um alvo à sorte e verifica se está ainda em branco e se não há outro jogador prestes a atirar ao mesmo alvo. Se ambas as condições se verificam, o jogador atira ao alvo, pintando-o da cor da sua equipa e inutilizando-o para outras equipas.
5. Quer tenha atirado ao alvo, quer não tenha, o jogador volta a escolher aleatoriamente outra sala e repete o mesmo procedimento.
6. Com o decorrer do jogo, chegar-se-á a um momento em que todos os alvos ficam pintados por alguma equipa. O jogador que pintou o último alvo branco que restava é responsável por, após esse momento, limpar todos os alvos, assim como actualizar e apresentar a nova pontuação do jogo. Assim que cada alvo seja limpo, pode de imediato ser usado pelos outros jogadores, mesmo que restem ainda outros alvos por limpar.

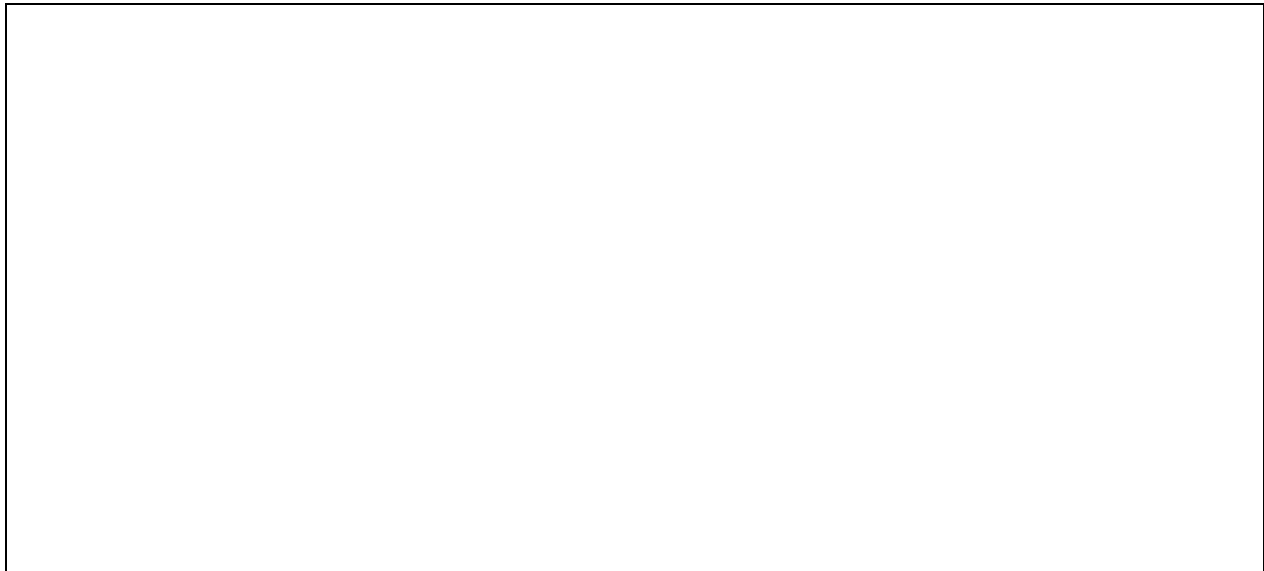
Considere a seguinte implementação da função *jogador* que a tarefa de cada jogador executará.

```
int pontuação[E] = {0, 0, ..., 0};
int alvo[N] = {0, 0, ..., 0};
int alvosLimpos = N;

void jogador(int cor) {
1  while (true) {
2      int alvoEscolhido = random(0..N-1);
3      if (alvo[alvoEscolhido] == 0) {
4          alvo[alvoEscolhido] = cor;
5          if ((--alvosLimpos)==0)
6              limpaTodos();
7      }
8  }
9  }

void limpaTodos() {
8  for (int i=0; i<N; i++) {
9      pontuação[alvo[i]]++;
10     alvo[i] = 0;
11     alvosLimpos++;
12 }
13 }
```

1. [2v] Adicione a sincronização necessária ao programa acima para que o jogo decorra correctamente numa execução com múltiplas tarefas. Escreva a sua nova solução abaixo, tentando maximizar o paralelismo do seu programa.  
Pode usar qualquer número de trincos lógicos (mutexes) e semáforos. Qualquer nova variável (trinco lógico, semáforo, ou outra) deverá ser declarada e inicializada no início do programa.  
Nota: não tem de copiar as linhas do programa original que se mantenham intactas na sua solução. Para as incluir na sua resposta, basta escrever o número das linhas originais (1 a 10).



2. [2v] Assuma agora uma nova variante do jogo em que existe uma tarefa adicional de limpeza, que executa a nova função *void limpeza()*. Nesta nova variante, o jogador que pinta o último alvo deixa de ser responsável por limpar os alvos.

A tarefa de limpeza deverá estar bloqueada enquanto exista, pelo menos, um alvo limpo. Apenas quando o último dos alvos limpos é pintado, deve a tarefa de limpeza desbloquear-se e começar a limpar, executando a função *limpaTodos*.

Apresente abaixo uma nova solução que implemente esta nova variante. Mais uma vez, pode adicionar quaisquer trincos lógicos ou semáforos, desde que devidamente declarados e inicializados no início do programa.



3. [2v] Assuma agora outra variante do jogo, em que cada jogador leva 2 armas de paintball, uma em cada mão. Nesta variante, cada jogador escolhe aleatoriamente 2 alvos (e não apenas 1), e só atira a ambos se ambos estiverem limpos e nenhum outro jogador estiver prestes a atirar sobre eles. Apresente abaixo as alterações que terá de efectuar à sua resposta à alínea 1 para suportar esta nova variante. Idealmente, a sua solução deverá assegurar que **nunca ocorre interbloqueio**. Mais uma vez, pode adicionar quaisquer trincos lógicos ou semáforos, desde que devidamente declarados e inicializados no início do programa.

```
int pontuação[E] = {0, 0, ..., 0};
int alvo[N] = {0, 0, ..., 0};
int alvosLimpos = N;

void jogador(int cor) {

    int alvoEscolhido1 = random(0..N-1);
    int alvoEscolhido2 = random(0..N-1);

    if (alvo[alvoEscolhido1] == 0 &&
        alvo[alvoEscolhido2] == 0) {

        alvo[alvoEscolhido1] = cor;
        alvo[alvoEscolhido2] = cor;

    }

}
```

4. [2v] Implemente a variante pedida na alínea 2, mas agora recorrendo a apenas um monitor global. Em particular, não pode usar nem trincos lógicos nem semáforos. Assuma que tem ao seu dispor a API: `enter(monitor)`, `exit(monitor)`, `wait(monitor)`, `notify(monitor)` e `notifyAll(monitor)`.

**Grupo III [5 Val]**

1. [1 Val] No Unix, como é que o sistema operativo determina a localização dos blocos de um ficheiro dado o nome absoluto do ficheiro (*pathname*)? Ilustre a sua resposta através de um exemplo.


2. Considere uma variante do Ext2 em que cada inode inclui apenas: 10 entradas que referenciam directamente os primeiros 10 blocos do ficheiro, seguidos de 1 entrada de 1 nível de indirectão e 1 entrada de 2 níveis de indirectão. Assuma que cada referência para um bloco tem a dimensão de 4 bytes e que cada bloco de disco tem a dimensão de 4 Kbytes.

- a. [0.8 Val] Qual a dimensão máxima de um ficheiro que ocupe apenas os blocos que são referenciados directamente pelo inode? Justifique.


- b. [0.8 Val] Qual a dimensão máxima de um ficheiro? Justifique.


- c. [0.8 Val] Quantos blocos são utilizados para suportar um ficheiro tenha 50 Kbytes? Justifique.


3. [0,8v] Considere a figura ao lado, que ilustra um sistema de ficheiros do tipo FAT. Diga quais os números dos blocos que constituem o ficheiro F1, cujo primeiro bloco é o 6.


0	32
1	3
2	EOF
3	4
4	5
5	10
6	7
7	8
8	9
9	12
10	14
11	15
12	13
13	2
14	34

4. [0,8v] A tabela ilustrada está incompleta. Para um disco com 200 Gbytes, blocos de 8 Kbytes e referências de 4 bytes, qual a dimensão total da Tabela de Alocação? Indique número de entradas e dimensão total (em bytes).

--