FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

SISTEMAS OPERATIVOS – 2010/2011 - 2º semestre

Exame da Época de Recurso 7/Julho/2011

PARTE A – com consulta
Duração: 15 minutos

NOME DO ESTUDANTE: _____ **Nº:** $\mathcal{EI}$_____

1. [4]

Indique quais das seguintes afirmações são verdadeiras e quais são falsas, assinalando respectivamente com V ou F:

	Multiprogramação e multiprocessamento são sinónimos.
	O escalonamento do tipo *Round-Robin* com uma fatia de tempo muito grande acaba por ser equivalente a um escalonamento do tipo *First-Come-First-Served.*
	Uma *thread* que pretende devolver/transferir um valor para a *thread* que a invocou pode guardar esse valor numa variável local e retornar o endereço dessa variável.
	Uma "secção crítica" é um bloco de código que tem obrigatoriamente de ser executado em "modo supervisor".
	Não faz sentido inicializar o contador de um semáforo com um valor negativo.
	Os *deadlocks* só podem ocorrer em processos que usem semáforos.
	Quando a frequência de falta de páginas de um processo é muito grande devem ser-lhe atribuídos mais quadros (*frames*) e, se isso não for possível, o processo deve ser "suspenso" (*swapped out*).
	A *PID* de um processo que executou uma chamada execlp() é modificada após a chamada.
	A instalação de um *handler* para o sinal SIGPIPE não tem qualquer efeito útil.
	Em Unix/Linux, a chamada dup2(STDOUT_FILENO,fd) permite redireccionar a saída standard do processo que a executar para um ficheiro cujo descritor é fd.

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

SISTEMAS OPERATIVOS – 2010/2011 - 2º semestre

Exame da Época de Recurso 7/Julho/2011

PARTE B – com consulta
Duração: 2 horas

Responda às questões 2, 3 e 4 na mesma folha

2. [2]

a) Em sistemas operativos do tipo Unix/Linux qual o significado de dizer que um processo está no estado "zombie" ?

b) Qual a justificação para que um processo seja mantido, pelo sistema operativo, nesse estado?

c) Por que é, particularmente em algumas aplicações como, por exemplo, um interpretador de comandos (*shell*), devem ser tomados cuidados para evitar que os processos executados fiquem nesse estado?

d) Indique duas abordagens diferentes para evitar a existência de processos "zombie".

3. [2.5]

Considere as seguintes declarações de dados, guardados numa região de memória, partilhada por vários processos, e funções, em código *C-like*, que podem ser invocadas por esses processos:

```
// variáveis comuns e inicialização de semáforos
int message;          // "mensagem" a ler por vários processos
int N = 3;            // nº de processos que vão ler a mensagem
int num = 0;          // variável auxiliar
semaphore m = 1;      // cria e inicializa semáforo c/valor 1
semaphore s1 = 0;     // cria e inicializa semáforo c/valor 0
semaphore s2 = 0;
```

```
void send(int msg) {
    message = msg;
    for (int i = 1; i <= N; i++) {
       sem_signal(s1);
    }
    printf("All done\n");
    sem_wait(s2);
}
```

```
void receive( ) {
   sem_wait(s1);
   printf("message = %d\n", message);
   sem_wait(m);
   num++;
   if (num == N) {
      for (int i = 1; i <= N+1 ; i++) {
         sem_signal(s2);
      }
   }
   sem_signal(m);
   sem_wait(s2);
}
```

Estas funções serão usadas por processos que pretendem trocar mensagens entre si. Um processo que queira enviar uma mensagem a outros processos deve invocar a função send(); o número de processos receptores é indicado pela variável partilhada N. Por sua vez, cada processo que pretenda ler a mensagem deve invocar a função receive(). Considere o seguinte conjunto de processos:

P1	P2	P3	P4
//código P1-parte A	//código P2- parte A	//código P3- parte A	//código P4- parte A
receive();	receive();	receive();	send(31);
//código P1-parte B	//código P2- parte B	//código P3- parte B	//código P4- parte B

a) Considere que os processos P1 a P4 eram lançados em execução, por esta ordem. Qual o estado dos processos e a contagem dos semáforos durante a execução da instrução printf("All done\n") pelo processo P4, considerando que não ocorreu nenhuma comutação de contexto desde que este processo iniciou a execução da função send() e que cada um dos restantes processos iniciou a execução da função receive()?

b) Justifique a necessidade do semáforo m?

c) Para que servem os semáforos s1 e s2?

d) Descreva sucintamente o que acontece quando estes processos forem executados concorrentemente. Diga qual a condição que tem de se verificar para que as partes B de cada processo, incluindo a parte B do processo emissor (P4-parte B, neste caso), sejam executadas. Justifique a resposta.

4. [1.5]

Considere a técnica de gestão de memória virtual baseada em paginação a pedido (*demand paging*).

a) Descreva sucintamente os conceitos de "memória virtual" e de "paginação a pedido".

b) Qual o suporte de *hardware* necessário para a implementação desta técnica de gestão de memória?

Responda às questões 5, 6 e 7 em três folhas separadas

5. [3]

Um processo inicial **P** pretende lançar um conjunto de **np** processos descendentes em que cada um destes deverá escrever no terminal a mensagem "**processo nr. <pid> filho de <pid>**". Cada um dos **np+1** processos só deve terminar após o(s) seu(s) filho(s) o terem feito (se um processo não tiver filhos deve terminar logo após ter escrito a mensagem). O valor de **np** é o parâmetro passado na linha de comando do programa.

a) Escreva o código do programa **P** que cria todos os **np** descendentes como seus filhos directos. Utilize um ciclo **for** para a criação dos **np** processos.

b) Escreva uma segunda versão do programa **P**, onde cada processo, incluindo o inicial, só cria um filho directo, com a excepção do **np**-ésimo que não tem filhos. Utilize igualmente um ciclo **for** para a criação dos **np** processos.

6. [3]

Um processo necessita de comunicar a outro uma *string* (que no máximo poderá conter 80 caracteres significativos) e uma sequência de 100 valores inteiros. Pretende fazê-lo utilizando um bloco de memória partilhada, de nome **shm.serv**, que cria para o efeito.

a) Escreva o código de criação do bloco de memória partilhada e seu preenchimento. Assuma que dispõe já das funções:

- **char *get_name()**, que retorna a *string* a colocar na memória partilhada e
- **void fill_values(int *vals)**, que preenche um *array* de 100 inteiros, cujo endereço lhe é passado como parâmetro.

b) Descreva um possível mecanismo simples para o segundo processo indicar ao primeiro que já não necessita da zona de memória partilhada, de modo a que este a possa destruir.

7. [4]

Considere o seguinte segmento de código C, utilizado para implementar um jogo do tipo roleta. Analise-o e responda às questões colocadas a seguir ao código.

```
01 …
02 typedef struct{
03      unsigned int numero;    // número do jogador
04      unsigned int njogadas;  // número de jogadas que o jogador pretende fazer
05      unsigned int nvitorias; // número de vitórias alcançadas
06 }jogador;
07 unsigned int numeroSorteado = 0;
08 void* roleta(void* p){
09      int fonteAleatoria = open("/dev/random", O_RDONLY);
10      unsigned int cnt = (int) p;
11      while(cnt-- > 0){
12         read(fonteAleatoria, (void*) &numeroSorteado, sizeof numeroSorteado);
13         numeroSorteado %= 20; // para simplificar
14         // printf("Número Sorteado=%u\n",numeroSorteado);
15      }
16      close(fonteAleatoria);
17      return NULL;
18 }
19 void* fazApostas(void* p){
20      jogador* j = (jogador*) p;
21      unsigned int aposta = 0;
22      unsigned int cnt = j->njogadas;
23      //j->nvitorias = 0; // o número de vitórias deve vir inicializado na estrutura
24      while(cnt-- > 0) {
25         printf("Jogador %u, escolha um número:",j->numero);
26         fflush(stdout);
27         scanf("%u",&aposta);
28         if(aposta == numeroSorteado) {
29            j->nvitorias++;
30            printf("Parabéns ao jogador %u! Ganhou!!!\n",j->numero);
31         } else
32            printf("Jogador %u, tente novamente para a próxima (%u != %u)\n",
33                   j->numero,aposta,numeroSorteado);
34      }
35      return NULL;
36 }
37 int main()
38 {
39 …
40 }
```

a) Qual é a função/necessidade de utilizar a função **`fflush`** na linha 26?

b) Ignorando, para já, qualquer questão relativa à necessidade de sincronização, escreva o código da função **`main`** para inicializar as variáveis necessárias, criar duas *threads*, uma para executar o código **`roleta`** e outra para o código **`fazApostas`**, esperar que elas terminem e, no final, indicar o número de vitórias alcançadas pelo jogador. Ambas as *threads* devem executar os seus ciclos internos 10 vezes, isto é, serão feitas 10 jogadas.

c) Repita a alínea anterior mas, agora, considerando que o código **`fazApostas`** é executado, em paralelo, por múltiplas *threads*. Isto é, há mais de um jogador no jogo. O número de jogadores e o número de jogadas são recebidos como parâmetros pela função **`main`**. No final deve ser apresentado, no ecrã, o número de vitórias de cada jogador.

d) Sem alterar a estrutura do código, para a situação de apenas um jogador e uma roleta (situação da alínea b)), recorrendo apenas a semáforos, adicione o código em falta para garantir a seguinte sequência de operações por jogada (***NOTA****-nesta alínea e na seguinte basta indicar o código adicional e a sua localização*):

1. a roleta sorteia o número
2. o jogador faz a aposta
3. a aposta é comparada com o número sorteado

e) Repita a alínea anterior, continuando a usar apenas semáforos, mas agora para uma situação mais realista (nesta situação o **`printf`**, comentado na linha 14, pode ser descomentado sem perigo de revelar o número sorteado antes de ocorrer a aposta):

1. o jogador faz a aposta
2. a roleta sorteia o número
3. a aposta é comparada com o número sorteado

FIM