FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

SISTEMAS OPERATIVOS – 2010/2011 - 2º semestre

Exame da Época Normal 14/Junho/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:

1	A multiprogramação só é possível em sistemas com múltiplos processadores.
2	Um processo que esteja no estado "a executar" passará sempre pelo estado "bloqueado" antes de voltar a entrar no estado "pronto a executar".
3	O sistema operativo guardará 3 *program counters* no *Process Control Block* de um processo que crie 2 *threads* auxiliares.
4	O processador nunca é retirado a um processo que esteja a executar código de uma secção crítica.
5	Em computadores que não suportem instruções do tipo Test-and-Set ou Swap é muito difícil implementar um mecanismo de sincronização como os semáforos.
6	A existência de "espera circular" é condição suficiente para a ocorrência de *deadlocks*.
7	O "princípio da localidade de referência" é o fundamento para a aplicação das técnicas de gestão de "memória virtual".
8	Quando dois processos têm o mesmo ficheiro aberto em "modo de escrita", em simultâneo, o sistema operativo garante que os dados que são escritos por cada processo nunca se sobrepõem uns aos outros.
9	Em Unix/Linux, a chamada wait() (*não confundir com a operação de espera sobre um semáforo, genericamente designada por wait*) permite que o processo-pai espere que um dos seus filhos termine e vice-versa.
10	Em Unix/Linux, um "*pipe* sem nome" usado para comunicação entre 2 processos filhos de um mesmo processo tem de ser criado pelo processo-pai.

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

SISTEMAS OPERATIVOS – 2010/2011 - 2º semestre

Exame da Época Normal 14/Junho/2011

PARTE B – com consulta
Duração: 2 horas

Responda às questões 2, 3 e 4 na mesma folha

2. [1.5]
Caracterize os seguintes mecanismos de comunicação entre processos quanto à possibilidade de trânsito bidireccional de dados entre um "processo-pai" e um "processo-filho":

a) variáveis globais do programa;

b) argumentos das chamadas exec();

c) *pipes*;

d) memória partilhada.

Alguma(s) destas formas de comunicação requer(em) a utilização de mecanismos auxiliares de sincronização? Justifique as respostas.

3. [3]

Considere o seguinte código, em linguagem *C-like*, através do qual se pretende simular a travessia de uma ponte que tem apenas uma via, pelo que a circulação de veículos tem de ser feita alternadamente, no sentido Norte-Sul ou Sul-Norte. As rotinas **NorthSouth_car()** e **SouthNorth_car()** pretendem simular um veículo que circula, respectivamente, no sentido Norte-Sul ou Sul-Norte. Em simultâneo, podem existir vários veículos que pretendem atravessar a ponte em cada um dos sentidos. As funções **sem_init(semaphore,int)**, **sem_wait(semaphore)** e **sem_signal(semaphore)** representam as operações básicas que podem ser executadas sobre um semáforo.

```
// variáveis comuns e inicialização de semáforos
int ncount = scount = 0;
semaphore m1, m2, bridge;
...
sem_init(m1,1);
sem_init(m2,1);
sem_init(bridge,1);
```

```
NorthSouth_car () {
   sem_wait(m1);
   if (ncount==0) sem_wait(bridge);
   ncount = ncount + 1;
   sem_signal(m1);
   cross_the_bridge(); //atravessa a ponte
   sem_wait(m1);
   ncount = ncount - 1;
   if (ncount == 0) sem_signal(bridge);
   sem_signal(m1);
}
```

```
SouthNorth_car () {
   sem_wait(m2);
   if (scount==0) sem_wait(bridge);
   scount = scount + 1;
   sem_signal(m2);
   cross_the_bridge(); //atravessa a ponte
   sem_wait(m2);
   scount = scount - 1;
   if (scount == 0) sem_signal(bridge);
   sem_signal(m2);
}
```

Em relação a este código, responda às seguintes questões, justificando as respostas:

a) Qual o valor dos semáforos na seguinte situação: após um longo período de ausência de tráfego, chegou um veículo no sentido Sul-Norte; quando este veículo estava a "atravessar a ponte" (isto é, a executar **cross_the_bridge**) chegou um veículo no sentido Norte-Sul. Indique também o estado deste segundo veículo.

b) É possível a circulação simultânea de vários veículos no mesmo sentido? Em caso afirmativo, há algum limite (imposto nesta simulação) para o número de veículos que podem circular simultaneamente?

c) Seria possível usar um único semáforo, **m**, em vez dos semáforos **m1** e **m2**?

d) Estão criadas condições para a ocorrência de *deadlocks*? E de inanição?

4. [1.5]

Considere um sistema de gestão de memória baseado em segmentação.

a) Como se processa a tradução de endereços lógicos em endereços físicos?

b) Como é que um sistema operativo consegue garantir que cada processo não acede a regiões de memória a que não deve ter acesso? Qual a mensagem de erro que surge tipicamente no ecrã quando, em sistemas operativos do tipo Unix/Linux, um processo tenta aceder a regiões de memória que não estão reservadas para o processo?

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

5. [3]

Um processo lança vários filhos em execução. Quando esses filhos terminam retornam um valor positivo no seu código de terminação. O processo pai deverá recolher todos esses códigos e calcular a sua média quando todos tiverem terminado, devendo também imprimi-la antes da sua própria terminação.

```
// variáveis globais
int np;
double mean;


…
while (np > 0)
  pause();
…
printf("Média: %f\n", mean);
…
```

a) O processo pai recolhe os códigos de terminação e calcula a média destes usando o sinal que os filhos geram quando terminam. Admitindo que o processo pai usa as instruções mostradas ao lado e **np** contém o número de processos filhos, escreva o *handler* do sinal e a sua instalação usando o serviço **sigaction** (sem a instalação de máscaras extra) de modo a que o **printf** mostrado escreva o valor correcto da média.

b) Quando o código do *handler* está em execução não pode ser interrompido por nova ocorrência do sinal que levou à sua execução. Existe o risco, neste esquema utilizado, de se perder a terminação de algum processo filho? Explique.

6. [3]

Vários processos clientes têm necessidade de transmitir a um processo servidor uma *string*, o seu PID e um carácter. Para isso o servidor já criou um FIFO chamado **fifo.serv** no directório **/tmp**. A *string* tem no máximo 9 caracteres. Um determinado cliente tem necessidade de uma única transmissão, tendo a *string* o valor "**mkhead**" e sendo o carácter '**B**'.

a) Um determinado programador implementou essa transmissão a partir desse cliente nos seguintes passos: definição de uma estrutura, declaração de uma variável da estrutura, preenchimento da variável, abertura do FIFO, escrita da informação no FIFO. Escreva o código correspondente aos passos enunciados.

b) Outro programador usou 3 variáveis distintas (*string*, PID e carácter) para transmissão da informação ao servidor. Escreva o código correspondente aos mesmos passos da alínea anterior, mas transmitindo cada uma das 3 variáveis separadamente.

c) Embora as duas versões possam funcionar correctamente em algumas situações, uma delas poderá apresentar problemas. Identifique o possível problema e justifique.

d) Em qualquer uma das versões das alíneas a) e b) será necessário fechar o FIFO? Justifique.

7. [4]

Pretende-se desenvolver um programa que simule um jogo de dados simples. A implementação deste programa deve ser baseada em 4 *threads*, uma para ler a aposta do jogador (*thread* **t_aposta**), outra para obter, aleatoriamente, um número de 1 a 6 (**t_roda**), a terceira para comparar o número sorteado com a aposta e actualizar os resultados (**t_compara**) e, finalmente, uma quarta (**t_mostra**) para, periodicamente, exibir os resultados.

A *thread* **t_aposta** deve, em ciclo infinito, pedir ao jogador que escolha um número, entre 1 e 6, guardá-lo numa variável de nome **aposta**, colocar a variável **aposta_feita** a 1 e aguardar que esta variável (**aposta_feita**) volte a 0 (zero) para repetir o ciclo.

A *thread* **t_roda** deve, em ciclo infinito, gerar um número aleatório, guardá-lo numa variável de nome **face**, colocar a variável **numero_sorteado** a 1 e aguardar que esta variável volte a 0 (zero) para repetir o ciclo.

A *thread* **t_compara** deve, em ciclo infinito, aguardar que as variáveis **numero_sorteado** e **aposta_feita** sejam, ambas, iguais a 1 para comparar os valores das variáveis **aposta** e **face** e, em função desse resultado, actualizar os valor das variáveis **numero_jogadas**, **apostas_ganhas** e **apostas_perdidas**, as quais devem conter, respectivamente, o número de jogadas feitas, o número de apostas ganhas e o número de apostas perdidas. Antes de repetir o ciclo, esta *thread* deve colocar a 0 (zero) as variáveis **numero_sorteado** e **aposta_feita**.

A *thread* **t_mostra** deve, em ciclo infinito, de segundo a segundo, imprimir no ecrã o número de jogadas, o número de apostas ganhas e o número de apostas perdidas.

Notas:

1) todas as variáveis acima referidas são variáveis globais;
2) o resultado a exibir deve ser coerente, isto é, deve-se verificar, sempre, a equação **numero_jogadas** = **apostas_ganhas** + **apostas_perdidas**.

a) Escreva os códigos seguintes:

a.1 - código da *thread* **t_aposta**

a.2 - código da *thread* **t_roda**

a.3 - código da *thread* **t_compara**

a.4 - código da *thread* **t_mostra**

a.5 - código do programa principal.

b) Se o jogador for muito rápido a jogar é possível que a *thread* **t_mostra** omita, por vezes, alguns resultados, impedindo deste modo a visualização da evolução do resultado, jogada a jogada. Por outro lado, se o jogador for lento, serão visualizados resultados repetidos. Mantendo esta estrutura de implementação e querendo garantir a correcta visualização, jogada a jogada, modifique as *threads* **t_compara** e **t_mostra** usando dois semáforos, para sincronizar as acções destas *threads*.

FIM