Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS DE COMPUTADORES – 2014/2015 2ºSemestre
Época especial – Conclusão antecipada

Sem consulta
Duração: 1 hora

Nome:_____ Nº _____

Folha de Respostas

NOTAS:

1. **Em todas as questões deverá assinalar <u>apenas uma resposta.</u>**
2. **Se a resposta assinalada for incorrecta sofrerá uma penalização de 1/3 da cotação da pergunta.**
3. **Apenas as respostas assinaladas na Folha de Respostas serão consideradas.**
4. **Devem ser entregues <u>todas as folhas</u> do exame.**

1 -	a) □	b) □	c) □	d) □		11 -	a) □	b) □	c) □	d) □
2 -	a) □	b) □	c) □	d) □		12 -	a) □	b) □	c) □	d) □
3 -	a) □	b) □	c) □	d) □		13 -	a) □	b) □	c) □	d) □
4 -	a) □	b) □	c) □	d) □		14 -	a) □	b) □	c) □	d) □
5 -	a) □	b) □	c) □	d) □		15 -	a) □	b) □	c) □	d) □
6 -	a) □	b) □	c) □	d) □		16 -	a) □	b) □	c) □	d) □
7 -	a) □	b) □	c) □	d) □		17 -	a) □	b) □	c) □	d) □
8 -	a) □	b) □	c) □	d) □		18 -	a) □	b) □	c) □	d) □
9 -	a) □	b) □	c) □	d) □		19 -	a) □	b) □	c) □	d) □
10 -	a) □	b) □	c) □	d) □		20 -	a) □	b) □	c) □	d) □

1 – Quando um processo cria outro invocando a *system call fork()* qual das seguintes características não é herdada pelo processo filho:

a) Espaço de endereçamento.
b) Indentificador do utilizador (UID).
c) Identificador do processo (PID).
d) Descritores de ficheiros abertos.

2 – Se um processo pai modificar o valor de uma variável global num programa em C, qual das seguintes afirmações descreve o seu efeito?

a) O processo filho vê imediatamente a alteração ao valor da variável.
b) O processo filho irá continuar a ver o valor antigo na variável global com o mesmo nome no seu espaço de endereçamento.
c) O processo filho vê a alteração ao valor da variável apenas se esta tiver sido alocada na *heap* antes do processo filho ter sido criado.
d) O processo filho vê imediatamente a alteração ao valor da variável apenas se pai e filho estiveram a usar o mesmo semáforo de exclusão mútua no acesso à variável.

3 – Com a execução concorrente de processos numa sistema com um único CPU podemos ter:

a) Um processo em execução enquanto outro está bloqueado numa operação de I/O usando um mecanismo de espera activa.
b) Um processo em execução enquanto outro está bloqueado numa operação de I/O usando um mecanismo de espera passiva.
c) Mais do que um processo em execução simultânea, quer existam ou não processos bloqueados em operações de I/O.
d) Apenas um processo bloqueado em operações de I/O, quer existam ou não processos em execução.

4 – No diagrama de transições de estado de um processo discutido nas aulas, a transição directa de "bloqueado" para "em execução" indica que:

a) Um processo foi preemptado por outro processo.
b) Um processo deixou de estar bloqueado num semáforo.
c) A espera do processo pela operação de I/O ou evento terminou.
d) Essa transição nunca poderá acontecer.

5 – A técnica de memória virtual agrega recursos de *hardware* e *software* com três funções básicas: realocação, protecção e paginação. A função de paginação:

a) Delega nos processos o mapeamento de endereços virtuais em endereços físicos.
b) Permite que um processo use mais memória do que a RAM fisicamente existente.
c) Usa registos do CPU para impedir que um processo utilize um endereço que não lhe pertence.
d) Assegura que cada processo tem o seu próprio espaço de endereçamento contínuo que começa no endereço 0.

6 – O uso de *buffers* entre processos comunicantes:

a) Dá a ilusão de que existe mais memória do que a realmente existente de forma física.
b) Reduz o número de operações de escrita em memória.
c) Permite que os processos acedam à memória sem qualquer mecanismo de sincronização.
d) Permite que os processos operem de forma assíncrona.

7 – A principal vantagem do uso de memória partilhada em relação ao uso de *pipes* como mecanismo de comunicação entre processos é:

a) A sua melhor performance.
b) A sincronização implícita na troca de dados.
c) A necessidade de acesso sequencial aos dados partilhados, o que facilita as leituras e escritas.
d) Impedir a leitura simultânea do mesmo bloco de dados por vários processos consumidores.

8 – Em programação concorrente, uma secção crítica é uma:

a) Parte do programa em que são acedidos dados potencialmente partilhados por vários processos.
b) Parte do programa em que o processo requisita ao sistema operativo a reserva de mais memória de forma dinâmica.
c) Parte do programa em que um *bug* causa garantidamente o término do processo.
d) Parte do programa que é executada em *kernel space*.

9 – Para que ocorra um interbloqueio (*deadlock*) entre processos é necessário que se manifestem simultaneamente as condições:

a) Ausência de exclusão mútua, posse e espera, preempção dos recursos, espera circular.
b) Exclusão mútua, ausência de posse e espera, preempção dos recursos, espera circular.
c) Exclusão mútua, posse e espera, ausência de preempção dos recursos, espera circular.
d) Exclusão mútua, posse e espera, preempção dos recursos, ausência de espera circular.

10 – Assuma que vários processos acedem e manipulam os mesmos dados de forma concorrente e o resultado final depende da ordem particular em que os acessos têm lugar. A este cenário é dado o nome de:

a) Comunicação entre processos através de memória partilhada.
b) Sincronização de processos.
c) Condição de concorrência (*race condition*).
d) Interbloqueio (*deadlock*).

11 – No escalonamento de processos baseado em prioridades, o protocolo de herança de prioridades refere-se a:

a) Uma redução da prioridade de um processo por este bloquear constantemente em operações de I/O.
b) Um aumento da prioridade de um processo de modo a que este liberte mais rapidamente um recurso requerido por um processo de maior prioridade.
c) Uma situação em que um processo de maior prioridade é obrigado a esperar por um processo de menor prioridade.
d) À atribuição de prioridades aos processos pela ordem inversa dos seus tempos de execução.

12 – Decidir o número de semáforos necessários para uma correcta sincronização de um conjunto de processos pode ser um exercício difícil. A abordagem de granularidade fina tem como consequência:

a) Diminuir o grau de concorrência das aplicações e o *overhead* do protocolo de sincronização.
b) Aumentar o grau de concorrência das aplicações e o *overhead* do protocolo de sincronização.
c) Diminuir o grau de concorrência das aplicações, mas aumentar o *overhead* do protocolo de sincronização.
d) Um maior número de dependências entre os semáforos, por vezes subtis, que podem originar *deadlocks*.

13 – Considere um processo com diversas *threads*. Qual dos seguintes recursos é partilhado por todas as *threads* desse processo:

a) *Heap*.
b) *Stack*.
c) Variáveis locais.
d) Conjunto de registos do CPU.

14 – Admita que um processo inicializa uma variável global X a zero e, de seguida, cria três *threads*. Cada uma delas lê o valor actual de X, incrementa-o e actualiza o seu valor sem qualquer mecanismo de sincronização. Qual o valor final de X após as três *threads* terminarem?

a) X = 3
b) X = 1 ou X = 3.
c) X =1 ou X = 2 ou X = 3.
d) X = 0 ou X = 1 ou X = 2 ou X = 3.

15 – Considere o pseudo-código seguinte, referente aos processos P1 e P2 que executam num único processador. Assuma que existem dois semáforos (S1, S2), em que ambos são inicializados a zero (0), e que as funções *up(s)* e *down(s)* permitem incrementar e decrementar um semáforo, respetivamente.

P1	**P2**

…
up(S1);
/* Executa bloco A */
up(S2);
down(S1);
/* Executa bloco C */
	…
	down(S2);
	/* Executa bloco B */
	up(S2);
	/* Executa bloco D */
	down(S1);

Indique qual das seguintes afirmações é verdadeira:

a) P1 executa sempre o bloco C antes de P2 executar o bloco D.
b) P2 executa sempre o bloco D antes de P1 executar o bloco A.
c) P1 executa sempre o bloco A depois de P2 executar o bloco B.
d) P2 executa sempre o bloco B depois de P1 executar o bloco A.

16 – O escalonamento não preemptivo é uma estratégia que:

a) Suspende temporariamente a execução de um processo sem requisitar a sua cooperação, antes do seu *time quantum* expirar ou o processo terminar.
b) Garante que um processo vítima de *resource starvation* execute.
c) Troca de processo apenas quando este bloqueia numa operação de I/O ou termina.
d) Garante a ausência de *deadlocks*.

17 – Não assumindo qualquer conhecimento do comportamento dos processos em execução, com qual dos seguintes algoritmos de escalonamento é possível garantir a ausência de *resource starvation* no acesso ao CPU?

a) Escalonamento por prioridades fixas.
b) *Shortest Job First*.
c) *Round Robin*.
d) Nenhum dos anteriores.

18 – Considere um sistema com um único processador e um algoritmo de escalonamento *Shortest Remaining Time Next*. Considerando os seguintes tempos de chegada ao sistema e perfis de execução para os processos P1, P2 e P3

Processo	Perfil do processo	Tempo de chegada
P1	111I1	2
P2	222I2	1
P3	333I333	0

em que um 1, 2 ou 3 representa, respectivamente, o processo P1, P2 ou P3 em execução durante uma unidade de tempo e I representa o bloqueio do processo em I/O, usando espera passiva. Indique uma possível sequência de execução destes processos, sabendo que na solução o símbolo "–" significa que o processador não está a executar qualquer processo.

a) 111-1222-2333-333
b) 32221211313-333
c) 33322211133321
d) 32111212323-333

19 – Assuma o mesmo conjunto de processos, respetivos perfis de execução e tempos de chegada, mas um algoritmo de **escalonamento preemptivo de prioridades fixas**, em que os processos têm prioridades (P1=1 (mais alta); P2=3; P3=2) e usam espera passiva para aceder aos recursos. Indique a sequência de escalonamento para estes processos (note-se que o símbolo "–" significa que o processador não está a executar qualquer processo.

a) 3111-1222-233-333
b) 32111212323-333
c) 3311131233322-2
d) 33322211133321

20 – Considere o seguinte código:

Escritor

```
1. void write(int elem){
2.   pthread_mutex_lock(&buffer_mux);
3.   while(n_elems == MAX)
4.     pthread_cond_wait(&not_full, &buffer_mux);
5.   buffer[write_pos++] = elem;
6.   …
7.   pthread_mutex_unlock(&buffer_mux);
8. }
```

Leitor

```
1. void read(int &elem){
2.   pthread_mutex_lock(&buffer_mux);
3.   …
4.   n_elem--;
5.   if(n_elem == MAX – 1)
6.     pthread_cond_broadcast(&not_full);
7.   pthread_mutex_unlock(&buffer_mux);
8. }
```

Admita que existem duas *threads* escritoras bloqueadas na linha 4 e uma *thread* leitora a executar a linha 6.

a) A correcção das escritas não é garantida porque ambas as *threads* escritoras acordam ao mesmo tempo e executam simultaneamente a sua linha 5.
b) A correcção das escritas é garantida porque, apesar de ambas acordarem ao mesmo tempo, apenas uma *thread* escritora avança de cada vez para a sua linha 5.
c) A correcção das escritas é garantida porque a *thread* leitora apenas consegue acordar uma das *threads* escritoras.
d) Não é possível ter duas *threads* escritoras simultaneamente na linha 4 porque a segunda *thread* só passa da linha 2 quando a anterior executar a sua linha 7.