Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS DE COMPUTADORES – 2023/2024
Exame Época Normal

Versão A

Apenas autorizada a consulta da folha oficial
Duração: 2h

Nome:_____ Nº _____

Folha de Respostas
(14/20 valores)

NOTAS:

1. **Em todas as questões deverá assinalar <u>apenas uma resposta.</u>**
2. **Se a resposta assinalada for incorreta sofrerá uma penalização de 1/3 da cotação da pergunta.**
3. **Apenas as respostas às questões de escolha múltipla assinaladas na Folha de Respostas serão consideradas.**
4. **A parte prática deve ser respondida numa folha separada devidamente identificada.**
5. **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 – Em Linux, as chamadas ao sistema (*system calls*):

a) São invocadas e executadas em *user-space*, permitindo a um processo aceder aos serviços disponibilizados pelo sistema operativo.
b) São invocadas em *user-space* e executadas em *kernel-space*, permitindo a um processo aceder aos serviços disponibilizados pelo sistema operativo.
c) Não são usadas porque os processos em *user-space* podem aceder diretamente aos serviços disponibilizados pelo sistema operativo.
d) Nenhuma das anteriores.

2 – Um sistema operativo é normalmente complexo, sendo por isso construído através de um conjunto de componentes. A estruturação com base numa abordagem monolítica pura:

a) Agrega todos os componentes num único processo que corre num único espaço de endereçamento.
b) Executa alguns componentes críticos em *user space* por uma questão de performance.
c) Exige um mecanismo de troca de mensagens entre *user space* e *kernel space*.
d) Permite carregar módulos dinamicamente, evitando a recompilação do núcleo quando se adicionam novas funcionalidades.

3 – As mudanças de estado de um processo são determinadas quer pelo seu fluxo de execução, quer pelo escalonador do sistema operativo. Qual das seguintes transições entre estados é possível?

a) "Bloqueado" para "em execução".
b) "Em execução" para "pronto a executar".
c) "Pronto a executar" para "bloqueado".
d) "Bloqueado" para "Terminado".

4 – Durante a execução de um processo qual das seguintes situações pode ocorrer?

a) Um processo bloqueia usando um mecanismo de espera ativa pelo acesso a um recurso, liberta o CPU e é colocado numa fila de espera associada ao recurso.
b) O tempo que o escalonador tinha atribuído ao processo (*time quantum*) termina, o processo liberta o CPU e mantém-se no estado "em execução".
c) O tempo que o escalonador tinha atribuído ao processo (*time quantum*) termina, o processo liberta o CPU e passa para o estado "pronto a executar".
d) Um processo cria outro processo, ficando de seguida à espera de que o seu filho termine invocando a função *wait,* não libertando o CPU e mantendo-se em execução.

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

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) Impede 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 – A gestão do acesso e reserva de memória pelos processos em execução:

a) Não é necessária num sistema operativo que permita a execução concorrente de processos.
b) É substituída pelo conceito de memória virtual em Linux.
c) É crítica em qualquer sistema operativo que permita a execução concorrente de processos.
d) Obriga os processos a indicarem endereços físicos quando pretendem aceder à memória.

7 – A comunicação entre processos permite:

a) A troca de dados e a sincronização de ações, apenas quando os processos partilham o mesmo espaço de endereçamento.
b) A troca de dados e a sincronização das ações, mesmo sem partilha do mesmo espaço de endereçamento.
c) Apenas a troca de dados, sem sincronização das suas ações, independentemente da partilha do espaço de endereçamento.
d) Apenas a sincronização das ações, sem troca de dados, independentemente da partilha do espaço de endereçamento.

8 – A utilidade do uso de sinais como método de comunicação entre processos é limitada porque:
a) Só podem ser usados entre as várias *threads* do mesmo processo.
b) Funcionam de forma assíncrona.
c) O programador não consegue associar mais dados ao sinal para além do seu número.
d) Um processo tem sempre a opção de ignorar a recepção de todos os sinais do sistema.

9 – Uma solução eficiente para o problema da secção crítica deve garantir:
a) Diferentes prioridades para os processos, envelhecimento (*aging*), ausência de interbloqueio (*deadlock*).
b) Acesso exclusivo, preempção, progressão.
c) Acesso exclusivo, progressão, espera limitada.
d) Ausência de preempção, privação de recursos (*resource starvation*), inversão de prioridades.

10 – Os mecanismos de sincronização de processos são implementados:
a) Apenas em *software*.
b) Apenas em *hardware*.
c) Tanto em *software* como em *hardware*.
d) Apenas para garantir exclusão mútua.

11 – Em sistemas concorrentes, a privação de recursos (*resource starvation*) é uma situação em que:
a) Dois processos (P1 e P2) se bloqueiam mutuamente devido a P1 ter bloqueado o semáforo S1, P2 ter bloqueado o semáforo S2, P1 necessitar de aceder a uma zona crítica protegida por S2 (sem libertar S1) e P2 necessitar de aceder a uma zona crítica protegida por S1 (sem libertar S2).
b) Dois processos (P1 e P2) se bloqueiam mutuamente devido a P1 ter bloqueado o semáforo S1, P2 ter bloqueado o semáforo S2, P1 necessitar de aceder a uma zona crítica protegida por S2 (depois de libertar S1) e P2 necessitar de aceder a uma zona crítica protegida por S1 (depois de libertar S2).
c) Em consequência da política de escalonamento do CPU, um recurso passa alternadamente dum processo P1 para um outro processo P2, deixando um terceiro processo P3 indefinidamente bloqueado sem acesso ao recurso.
d) Nenhuma das anteriores.

12 – Decidir o número de semáforos necessários para uma correta sincronização de um conjunto de processos pode ser um exercício difícil. A abordagem de granularidade abrangente (*coarse grained*) tem como consequência:
a) Diminuir o grau de concorrência das aplicações e o custo (*overhead*) do protocolo de sincronização.
b) Aumentar o grau de concorrência das aplicações e o custo (*overhead*) do protocolo de sincronização.
c) Diminuir o grau de concorrência das aplicações, mas aumentar o custo (*overhead*) do protocolo de sincronização.
d) Um maior número de dependências entre os semáforos, por vezes subtis, que podem originar situações de interbloqueio (*deadlocks*).

13 – A estratégia de escalonamento que permite que qualquer processo em execução monopolize o CPU até ao fim do seu código ou até o libertar voluntariamente é denominada por:
a) Escalonamento não preemptivo.
b) Escalonamento preemptivo.
c) Escalonamento por prioridades fixas.
d) Escalonamento por prioridades dinâmicas.

14 – Aplicar ao algoritmo de escalonamento *Round Robin*:
a) Um *time quantum* muito pequeno converte-o na prática no algoritmo *First In First Out*.
b) Um *time quantum* muito pequeno aumenta consideravelmente a performance do sistema.
c) Um *time quantum* muito pequeno aumenta consideravelmente o custo (*overhead*) do escalonamento dos processos.
d) Um *time quantum* muito pequeno converte-o na prática no algoritmo *Shortest Job First*.

15 – Considere os seguintes processos P1 e P2. Assuma que existem dois semáforos (S1, S2) partilhados entre eles, ambos inicializados a 1 (um). As funções `up(s)` e `down(s)` permitem incrementar e decrementar um semáforo, respetivamente, de forma atómica.

P1	**P2**

`down(S1);`
`down(S2);`
/* secção crítica */
`up(S2);`
`up(S1);`
	`down(S2);`
	`down(S1);`
	/* secção crítica */
	`up(S1);`
	`up(S2);`

Qual das seguintes afirmações é verdadeira?

a) O código poderá causar um interbloqueio (*deadlock*) porque S1 é libertado antes de S2 por P1.
b) O código poderá causar um interbloqueio (*deadlock*) porque P1 e P2 estão a tentar decrementar os semáforos pela mesma ordem.
c) O código poderá causar um interbloqueio (*deadlock*) porque P1 e P2 estão a tentar decrementar os semáforos por ordens diferentes.
d) O código nunca poderá causar um interbloqueio (*deadlock*) porque S1 e S2 são sempre libertados depois da secção crítica.

16 – Considere um processo com diversas *threads*. Qual das seguintes secções do espaço de endereçamento de um processo é privada para cada uma das *threads* desse processo:

a) *Heap*.
b) *Stack*.
c) Variáveis globais.
d) Código do programa.

17 – Considere os seguintes processos P1 e P2 que executam num único processador. Assuma que existem dois semáforos (S1, S2), em que S1 é inicializado a zero (0) e S2 é inicializado a um (1), e que as funções `up(s)` e `down(s)` permitem incrementar e decrementar um semáforo, respetivamente, de forma atómica.

P1	**P2**

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

Indique qual das seguintes afirmações é verdadeira:

a) P1 executa sempre o bloco C antes de P2 executar o bloco D.
b) P1 nunca executa o bloco C antes de P2 executar o bloco D.
c) Nada pode ser garantido em relação à ordem de execução dos blocos A e B.
d) P2 executa sempre o bloco B antes de P1 executar o bloco A.

18 – Considere um sistema com um único processador e um algoritmo de escalonamento *Round Robin* com um ***time quantum* igual a 2**. 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	2222I2	1
P3	333I33	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. Indique a sequência de execução destes processos, sabendo que na solução o símbolo "–" significa que o processador não está a executar qualquer processo.

a) 11223312231233
b) 33221132213321
c) 3322113-221-33-21
d) 32111-1222-233-33

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=2; P3=3(mais baixa)). Indique a sequência de escalonamento para estes processos (note-se que o símbolo "–" significa que o processador não está a executar qualquer processo).

a) 32111-1222-233-33
b) 321112122323-33
c) 11223312231233
d) 33221132213321

20 – O código seguinte apresenta uma solução para o problema dos "Produtores/Consumidores":

Produtor

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

Consumidor

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

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

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

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS DE COMPUTADORES – 2023/2024
Exame Época Normal

Apenas autorizada a consulta da folha oficial

Nome:_____ Nº _____

Parte prática
(6/20 valores)

NOTAS:

1. **Responder numa folha separada, devidamente identificada.**
2. **Não é necessário indicar o nome das bibliotecas usadas na resolução.**

Implemente um programa em C que simule uma fila de impressão com várias impressoras e vários clientes usando *threads*.

Requisitos:

- Crie uma estrutura de dados para representar um trabalho de impressão.
- Crie uma fila de impressão utilizando uma estrutura adequada com um tamanho máximo pré-determinado (*bounded buffer*) de 4 trabalhos.
- Implemente a função que adiciona trabalhos de impressão à fila (executada pelas *threads* cliente).
- Implemente a função que remove trabalhos de impressão da fila (executada pelas *threads* impressora).
- Garanta que o acesso à fila é protegido por um *mutex*.
- Crie 5 *threads* cliente. Assuma que cada cliente adiciona 1 trabalho a cada 2 segundos e que nunca termina.
- Crie 3 *threads* impressora. Assuma que demora 5 segundos a processar um trabalho e que uma impressora nunca termina.
- Sincronize o acesso à fila de impressão usando variáveis de condição, garantindo que uma *thread* cliente bloqueia se a fila estiver cheia e uma *thread* impressora bloqueia se a fila estiver vazia.