Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS DE COMPUTADORES – 2023/2024
Exame Época Recurso

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, quando um processo cria outro invocando a chamada ao sistema (*system call*) *fork()*, qual das seguintes características do processo pai não é copiada para o processo filho:

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

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

a) O processo pai vê imediatamente a alteração ao valor da variável, se essa variável for global.
b) O processo pai irá continuar a ver o valor antigo na variável, independentemente do segmento do espaço de endereçamento onde a variável foi reservada.
c) O pai vê a alteração ao valor da variável apenas se esta tiver sido reservada na *heap* antes do processo filho ter sido criado.
d) Nenhuma das anteriores.

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

a) Um processo em execução, enquanto outro está bloqueado numa operação de sincronização usando um mecanismo de espera ativa.
b) Um processo em execução, enquanto outros estão bloqueados numa operação de sincronização usando um mecanismo de espera passiva.
c) Mais do que um processo em execução simultânea, quer existam ou não processos bloqueados em operações de sincronização usando mecanismos de espera passiva.
d) Apenas um processo bloqueado em operações de sincronização usando mecanismos de espera passiva, quer existam ou não processos em execução.

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

a) Um processo em execução foi preemptado por outro processo escolhido pelo escalonador.
b) Um processo deixou de estar bloqueado num semáforo.
c) A espera passiva do processo pela operação de I/O ou evento terminou.
d) Um novo processo acabou de ser criado.

5 – A técnica de memória virtual agrega recursos de *hardware* e *software* com três funções básicas: realocação, proteção 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) 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 – O uso de um *buffer* circular numa zona de memória partilhada 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 *pipes* em relação ao uso de memória partilhada como mecanismo de comunicação entre processos é:

a) A sua melhor performance.
b) A sincronização implícita na troca de dados.
c) A sua maior flexibilidade no acesso a apenas parte dos dados enviados.
d) Permitir 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 que devem ser alterados em exclusão mútua.
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 – Não assumindo qualquer conhecimento do comportamento dos processos em execução, com qual dos seguintes algoritmos de escalonamento é possível garantir a ausência de privação de recursos (*resource starvation*) no acesso ao CPU?

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

10 – Os mecanismos de sincronização de processos disponibilizados pelo sistema operativo são:

a) Dependentes da ordem e velocidade de execução dos processos.
b) Independentes da ordem e velocidade de execução dos processos.
c) Usados apenas para garantir exclusão mútua no acesso aos recursos partilhados.
d) Nenhuma das anteriores.

11 – No escalonamento de processos baseado em prioridades, a inversão 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 para 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) Uma atribuição de prioridades aos processos pela ordem inversa dos seus tempos de execução.

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 fina (*fine 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) Diminuir número de dependências entre os semáforos, reduzindo a possibilidade de situações de interbloqueio (*deadlocks*).

13 – A estratégia de escalonamento que impede que um 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 grande converte-o na prática no algoritmo *First In First Out*.
b) Um *time quantum* muito grande aumenta consideravelmente a performance do sistema.
c) Um *time quantum* muito grande aumenta consideravelmente o custo (*overhead*) do escalonamento dos processos.
d) Um *time quantum* muito grande converte-o na prática no algoritmo *Shortest Job First*.

15 – Considere o seguinte excerto de código:

```
void *f1(void *arg){
  printf("S1\n");
  pthread_exit(NULL);
}
```

```
for(i = 0; i < 3; i++){
  pid = fork();
  if (pid > 0){
    execlp("ls", "ls", NULL);
    pid = fork();
  }
  else{
    pthread_create(&t[i], NULL, f1, NULL);
    printf("S1\n");
  }
  printf("S2\n");
}
```

Quantas vezes irá ser impresso "S1"? Assuma que a invocação da função `execlp()` nunca falha.

a) 18.
b) 6.
c) 36.
d) 3.

16 – Qual das seguintes afirmações melhor descreve o fluxo de controlo excecional num sistema operativo?

a) O fluxo de controlo excecional ocorre quando um processo a executar em *user-space* altera diretamente o fluxo de execução para *kernel-space* sem qualquer forma de mediação.
b) O fluxo de controlo excecional diz respeito a eventos como interrupções, *traps* e exceções que fazem com que o CPU mude de *user-space* para *kernel-space* para lidar com condições ou eventos inesperados.
c) O fluxo de controlo excecional é um mecanismo que garante que todos os processos recebam igual tempo de CPU, interrompendo preemptivamente os processos em execução em intervalos fixos.
d) O fluxo de controlo excecional só ocorre durante o encerramento do sistema operativo para garantir que todos os processos são corretamente terminados.

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 um (1) e S2 é inicializado a zero (0), e que as funções `up(s)` e `down(s)` permitem incrementar e decrementar um semáforo, respetivamente, de forma atómica.

P1	**P2**

…
`down(S1);`
`up(S2);`
/* Executa bloco A */
`up(S1);`
/* Executa bloco C */
	…
	`down(S2);`
	/* Executa bloco B */
	`down(S1);`
	`up(S2);`
	/* 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 C e D.
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 *Shortest Remaining Time First*. 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, respetivamente, 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) 111-1222-2333-333
b) 3331331121222-2
c) 33322211133321
d) 321112122323-33

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

a) 33111-133-332222-2
b) 32111212323-333
c) 3311131233222-2
d) 33322211133321

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.  printf("SCOMP\n");
8.  pthread_mutex_unlock(&mux);
9. }
```

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

a) A linha 7 da *thread* consumidora será sempre executada antes de qualquer uma das *threads* produtoras conseguir executar a sua linha 5.
b) A linha 5 das *threads* produtoras será sempre executada em simultâneo pelas duas *threads* antes da *thread* consumidora executar a sua linha 7.
c) Cada uma das *threads* produtoras executa sempre de forma exclusiva a sua linha 5 antes da *thread* consumidora executar a sua linha 7.
d) Não é possível garantir qualquer ordem de execução entre a linha 7 da *thread* consumidora e da linha 5 das *threads* produtoras.

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, através do uso de *threads*, simule um sistema de pontuações numa prova de triatlo com 300 atletas. O objetivo é determinar a pontuação final de cada atleta de acordo com a seguinte fórmula:

```
final_score = 0.30 * (sc_swim + sc_bike + sc_run)/3 + 0.7 * total_time
```

onde `sc_swim`, `sc_bike` e `sc_run` são as pontuações das provas de natação, ciclismo e corrida, respetivamente, e `total_time` é o tempo total do atleta.

Considere a seguinte estrutura de dados para armazenar os dados de avaliação de cada atleta:

```
typedef struct {
  int number;              // número do atleta
  int event_scores[3];     // pontuações de natação, ciclismo e corrida
  int total_time;          // tempo total
  float final_score;       // pontuação final
}athlete_score;
```

Assuma que existe um *array* `athlete_score scores[300]` para armazenar os dados de avaliação de todos os atletas. O programa deve criar apenas 2 *threads* (T1 e T2) que se sincronizam através do uso de *mutexes* e variáveis de condição.

A *thread* T1 deverá gerar aleatoriamente uma pontuação entre 0 e 100 para as pontuação de cada evento e um valor entre 50 e 100 para o tempo total de um atleta. De seguida, sinaliza a *thread* T2 que deverá preencher o campo `final_score` desse atleta, de acordo com a fórmula indicada acima.

Para que não se percam os valores gerados para cada um dos atletas, **deve garantir que as *threads* operam de forma alternada**, isto é, T1 gera os valores parciais de um atleta e T2 calcula o seu resultado final, antes de serem gerados os valores para o próximo atleta.

No final, a *thread* principal imprime as pontuações finais de cada atleta e remove todos os *mutexes* e variáveis de condição usados.

Nota: admita que já existe uma função `int gera_num(int inicio, int fim)` que gera um valor aleatório no intervalo `[inicio, fim]`.