

[Pai...](#) / [Meus cu...](#) / [INE5410-03208A/B \(2...](#) / Unidade 2: Fundamentos de Programação Con... / [AF 2.6 - Mutexes e semáforos \(questi...](#)

Iniciado em	quinta-feira, 15 mai. 2025, 16:46
Estado	Finalizada
Concluída em	quinta-feira, 15 mai. 2025, 17:50
Tempo empregado	1 hora 4 minutos
Notas	7,00/7,00
Avaliar	10,00 de um máximo de 10,00(100%)

Questão 1

Correto

Atingiu 1,00 de 1,00

Em sistemas operacionais, uma Região Crítica é

Escolha uma opção:

- ☐ a. um trecho de programa que deve ser executado em paralelo com a seção crítica de outro programa.
- ☒ b. um trecho de programa no qual existe o compartilhamento de algum recurso que deve ser acessado com exclusão mútua. ✓
- ☐ c. um trecho de programa cujas instruções podem ser executadas em paralelo e em qualquer ordem.
- ☐ d. um trecho de programa no qual existe um recurso ao qual somente o sistema operacional pode ter acesso.

A resposta correta é: um trecho de programa no qual existe o compartilhamento de algum recurso que deve ser acessado com exclusão mútua.

Questão 2

Correto

Atingiu 1,00 de 1,00

O problema do buffer limitado de tamanho N é um problema clássico de sincronização de processos: um grupo de processos utiliza um buffer de tamanho N para armazenar temporariamente itens produzidos; processos produtores produzem os itens, um a um, e os armazenam no buffer; processos consumidores retiram os itens do buffer, um a um, para processamento. O problema do buffer limitado de tamanho N pode ser resolvido com a utilização de semáforos, que são mecanismos de software para controle de concorrência entre processos. Duas operações são definidas para um semáforo s : $\text{wait}(s)$ e $\text{post}(s)$. Considere o problema do buffer limitado de tamanho N cujos pseudocódigos dos processos produtor e consumidor estão mostrados na tabela abaixo. Pode-se resolver esse problema com a utilização dos semáforos `mutex`, `cheio` e `vazio`, inicializados, respectivamente, com 1, 0 e N .

processo produtor	processo consumidor
<code>produz item</code> <code>comando_a</code> <code>comando_b</code> <code>coloca no buffer</code> <code>comando_c</code> <code>comando_d</code>	<code>comando_e</code> <code>comando_f</code> <code>retira do buffer</code> <code>comando_g</code> <code>comando_h</code> <code>consome o item</code>

A partir dessas informações, para que o problema do buffer limitado de tamanho N cujos pseudocódigos foram apresentados possa ser resolvido a partir do uso dos semáforos `mutex`, `cheio` e `vazio`, é necessário que `comando_a`, `comando_b`, `comando_c`, `comando_d`, `comando_e`, `comando_f`, `comando_g` e `comando_h` correspondam, respectivamente, às operações:

Escolha uma opção:

- ☒ a. `wait(vazio)`, `wait(mutex)`, `post(mutex)`, `post(cheio)`, `wait(cheio)`, `wait(mutex)`, `post(mutex)` e `post(vazio)`. ✓
- ☐ b. `wait(mutex)`, `wait(vazio)`, `post(cheio)`, `post(mutex)`, `wait(mutex)`, `wait(cheio)`, `post(vazio)` e `post(mutex)`.
- ☐ c. `wait(mutex)`, `wait(vazio)`, `post(cheio)`, `post(mutex)`, `wait(mutex)`, `wait(vazio)`, `post(cheio)` e `post(mutex)`.
- ☐ d. `wait(vazio)`, `post(mutex)`, `post(cheio)`, `wait(mutex)`, `wait(cheio)`, `post(mutex)`, `post(vazio)` e `post(mutex)`.
- ☐ e. `wait(cheio)`, `wait(mutex)`, `post(mutex)`, `post(vazio)`, `wait(vazio)`, `post(mutex)`, `post(mutex)` e `wait(cheio)`.

A resposta correta é: `wait(vazio)`, `wait(mutex)`, `post(mutex)`, `post(cheio)`, `wait(cheio)`, `wait(mutex)`, `post(mutex)` e `post(vazio)`.

Questão 3

Correto

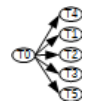
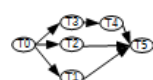
Atingiu 1,00 de 1,00

Seja o código abaixo:

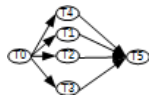
```
// Todas variáveis mutex inicializadas locked
void* T0() {
    ...
    printf("\n T0 finished\n");
    pthread_mutex_unlock(&lock1);
    pthread_mutex_unlock(&lock2);
    pthread_mutex_unlock(&lock3);
    return NULL;
}
void* T1() {
    pthread_mutex_lock(&lock1);
    ...
    printf("\n T1 finished\n");
    pthread_mutex_unlock(&lock4);
    return NULL;
}
void* T2() {
    pthread_mutex_lock(&lock2);
    ...
    printf("\n T2 finished\n");
    pthread_mutex_unlock(&lock5);
    return NULL;
}
void* T3() {
    pthread_mutex_lock(&lock3);
    ...
    printf("\n T3 finished\n");
    pthread_mutex_unlock(&lock6);
    return NULL;
}
void* T4() {
    pthread_mutex_lock(&lock4);
    ...
    printf("\n T4 finished\n");
    pthread_mutex_unlock(&lock7);
    return NULL;
}
void* T5() {
    pthread_mutex_lock(&lock5);
    pthread_mutex_lock(&lock7);
    pthread_mutex_lock(&lock6);
    ...
    printf("\n T5 finished\n");
    return NULL;
}
```

Indique o grafo que ilustra corretamente a ordem de execução das threads.

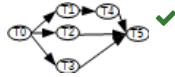
Escolha uma opção:

- ☐ a.
- 
- ☐ b.
- 

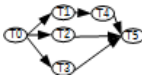
☐ c.



☒ d.



A resposta correta é:



Questão 4

Correto

Atingiu 1,00 de 1,00

A situação na qual vários processos manipulam um mesmo conjunto de dados concorrentemente e o resultado depende da ordem na qual os acessos são feitos é denominada:

Escolha uma opção:

- ☐ a. exclusão mútua
- ☐ b. espera ocupada (*busy wait*)
- ☒ c. condição de corrida (*race condition*) ✓
- ☐ d. *livelock*
- ☐ e. *deadlock*

A resposta correta é: condição de corrida (*race condition*)

Questão 5

Correto

Atingiu 1,00 de 1,00

Considere o problema dos Leitores e Escritores, no qual várias threads devem acessar a mesma base de dados (bd), algumas para ler e outras para escrever. Contudo, existem algumas restrições para a realização de leituras e escritas:

- I. Enquanto uma thread estiver escrevendo, nenhuma outra thread pode acessar a base de dados (bd).
- II. Enquanto uma thread estiver lendo, somente threads leitoras podem acessar a base de dados (bd).

Seja a solução abaixo:

```
pthread_mutex_t db;
pthread_mutex_t mutex;
int rc = 0; // Quantidade de leitores lendo ou querendo ler a base de dados (bd)

void reader() {
    while(1) { // Repete eternamente
        (1);
        rc=rc+1; // Um novo leitor
        if(rc==1)
            (2); // Caso este seja o primeiro leitor...
        (3);
        read_data_base(); // le a bd
        (4);
        rc=rc-1; // Um leitor a menos...
        if(rc==0)
            (5); // Caso este seja o ultimo leitor.
        (6);
        use_data_read(); // Utiliza as informações da bd
    } // Fim do while
}

void writer() {
    while(1) { // Repete eternamente
        think_up_data(); // pensa em informações para adicionar à bd
        (7);
        write_data_base(); // escreve novas informações na bd
        (8);
    }
}
```

Para a solução acima, que operações quais deveriam ser as operações 1, 2, 3, 4, 5, 6, 7 e 8?

Escolha uma opção:

- ☐ a. 1. pthread_mutex_unlock(&mutex)
2. pthread_mutex_unlock(&db)
3. pthread_mutex_lock(&mutex)
4. pthread_mutex_unlock(&mutex)
5. pthread_mutex_lock(&db)
6. pthread_mutex_lock(&mutex)
7. pthread_mutex_lock(&db)
8. pthread_mutex_unlock(&db)
- ☒ b. 1. pthread_mutex_lock(&mutex) ✓
2. pthread_mutex_lock(&db)
3. pthread_mutex_unlock(&mutex)
4. pthread_mutex_lock(&mutex)
5. pthread_mutex_unlock(&db)
6. pthread_mutex_unlock(&mutex)
7. pthread_mutex_lock(&db)
8. pthread_mutex_unlock(&db)
- ☐ c. 1. pthread_mutex_lock(&mutex)
2. pthread_mutex_lock(&db)
3. pthread_mutex_unlock(&mutex)
4. pthread_mutex_lock(&mutex)
5. pthread_mutex_unlock(&db)
6. pthread_mutex_unlock(&mutex)
7. pthread_mutex_lock(&mutex)
8. pthread_mutex_unlock(&mutex)
- ☐ d. 1. pthread_mutex_lock(&db)
2. pthread_mutex_lock(&mutex)
3. pthread_mutex_unlock(&mutex)

- 4. `pthread_mutex_lock(&mutex)`
 - 5. `pthread_mutex_unlock(&mutex)`
 - 6. `pthread_mutex_unlock(&db)`
 - 7. `pthread_mutex_lock(&db)`
 - 8. `pthread_mutex_unlock(&db)`
- ☐ e.
- 1. `pthread_mutex_lock(&db)`
 - 2. `pthread_mutex_lock(&db)`
 - 3. `pthread_mutex_unlock(&mutex)`
 - 4. `pthread_mutex_lock(&mutex)`
 - 5. `pthread_mutex_unlock(&mutex)`
 - 6. `pthread_mutex_unlock(&mutex)`
 - 7. `pthread_mutex_lock(&db)`
 - 8. `pthread_mutex_unlock(&db)`

A resposta correta é:

- 1. `pthread_mutex_lock(&mutex)`
- 2. `pthread_mutex_lock(&db)`
- 3. `pthread_mutex_unlock(&mutex)`
- 4. `pthread_mutex_lock(&mutex)`
- 5. `pthread_mutex_unlock(&db)`
- 6. `pthread_mutex_unlock(&mutex)`
- 7. `pthread_mutex_lock(&db)`
- 8. `pthread_mutex_unlock(&db)`

Questão 6

Correto

Atingiu 1,00 de 1,00

Sobre exclusão mútua é correto afirmar que:

Escolha uma opção:

- ☐ a. É uma técnica usada em programação concorrente para resolver *deadlock*.
- ☐ b. É uma técnica usada em programação concorrente para evitar que dois ou mais processos tenham acesso simultaneamente a múltiplos recursos privados.
- ☐ c. É uma técnica usada em programação concorrente para evitar *deadlock*.
- ☒ d. É uma técnica usada em programação concorrente para evitar que duas ou mais threads tenham acesso simultaneamente a um recurso compartilhado (seção crítica). ✓
- ☐ e. É uma técnica usada em programação não concorrente para evitar que duas ou mais threads tenham acesso simultaneamente a um recurso compartilhado (seção crítica).

A resposta correta é: É uma técnica usada em programação concorrente para evitar que duas ou mais threads tenham acesso simultaneamente a um recurso compartilhado (seção crítica).

Questão 7

Correto

Atingiu 1,00 de 1,00

Sejam as duas threads abaixo executadas concorrentemente e compartilhando as variáveis A e B.

Thread 0	Thread 1
<pre>A = 0; ... A = 1; ... if (B == 0) P1();</pre>	<pre>B = 0; ... B = 1; ... if (A == 0) P2();</pre>

Com relação à situação acima, julgue os itens seguintes.

- I. As funções P1() e P2() nunca serão executadas simultaneamente.
- II. As funções P1() e P2() poderão não ser executadas.
- III. Uma das funções (P1() ou P2()) necessariamente será executada.
- IV. Apenas P1() ou apenas P2() será executada.

Estão corretas as assertivas:

Escolha uma opção:

- ☐ a. I e III.
- ☒ b. I e II. ✓
- ☐ c. II e IV.
- ☐ d. III e IV.
- ☐ e. I, II, III e IV.

A resposta correta é: I e II.

[◀ AF 2.5 - Semáforos \(prática\)](#)

Seguir para...

[AF 2.7 - Deadlocks \(questionário\) ▶](#)