

Lista 1 de Computação Concorrente

Gabriel da Fonseca Ottoboni Pinho - DRE 119043838

15/04/2021

Questão 1

A

Um programa concorrente possui mais de um fluxo de execução. Isso significa que múltiplas tarefas podem ser executadas ao mesmo tempo. Por outro lado, um programa sequencial só é capaz de executar um fluxo ao mesmo tempo.

B

Se uma das 5 tarefas leva tempo t para ser executada, as 4 tarefas que serão executadas concorrentemente levarão o mesmo tempo t para serem executadas. A 5ª tarefa, que será executada depois das 4 primeiras, também levará tempo t , então $t_{\text{concorrente}} = t + t = 2t$. Se as 5 tarefas fossem executadas sequencialmente, cada uma levaria tempo t , então $t_{\text{sequencial}} = 5t$.

$$\text{Aceleração} = \frac{t_{\text{sequencial}}}{t_{\text{concorrente}}} = \frac{5t}{2t} = \frac{5}{2} = 2.5$$

C

Seção crítica é uma parte do código onde há alguma operação que não deve ser executada por mais de uma thread simultaneamente. Por exemplo, a alteração de uma variável global por duas threads simultaneamente causa uma condição de corrida, então essa seria uma seção crítica.

D

A sincronização por exclusão mútua tem como objetivo proteger uma seção crítica de código, garantindo que apenas uma thread execute uma parte do código ao mesmo tempo. Para acessar o trecho crítico, a thread precisa adquirir uma *lock*, que permite o acesso. Essa *lock* só pode ser adquirida por uma thread ao mesmo tempo. Se uma thread já tiver a *lock* e outra thread requisitá-la, essa ficará bloqueada até que aquela libere a *lock*.

Questão 2

- O valor -3 *não* é possível.
- O valor -1 é possível. T2 executa as três primeiras linhas, atribuindo -2 a x e passando pelo *if*. Depois, T1 executa sua primeira linha e muda x para -1 . Por fim, T2 imprime -1 .
- O valor 1 é possível. T1 executa as três primeiras linhas, atribuindo 0 a x e passando pelo *if*. Depois, T3 executa sua primeira linha e muda x para 1 . Por fim, T1 imprime 1 .
- O valor 3 é possível. T3 executa as três primeiras linhas, atribuindo 2 a x e passando pelo *if*. Depois, T1 executa sua primeira linha e muda x para 3 . Por fim, T3 imprime 3 .

Questão 3

A

Sim, a exclusão mútua é garantida. O *loop* garante que uma thread precisa esperar que TURN tenha o valor adequado antes de entrar na seção crítica. Desse modo, T1 e T0 executarão a seção crítica alternadamente, garantindo a exclusão mútua.

B

A *independência do restante do código* não é atendida. Se T1 acabar sua execução, T0 só poderá entrar na seção crítica uma única vez, ficando presa no *loop* na próxima vez (ou viceversa).

Questão 4

```
int x = 0, y = 0;
pthread_mutex_t x_mutex = PTHREAD_MUTEX_INITIALIZER;

void *T1(void *id) {
    int a = 0;
    while (a < 2) {
        pthread_mutex_lock(&x_mutex);
        x++;
        x--;
        if (x == 0)
            printf("x=%d\n", x);
        pthread_mutex_unlock(&x_mutex);
        a++;
        printf("a=%d\n", a);
    }
}

void *T2(void *id) {
    int a = 2;
    while (a > 0) {
        pthread_mutex_lock(&x_mutex);
        x++;
        x--;
        if (x == 0)
            printf("x=%d\n", x);
        pthread_mutex_unlock(&x_mutex);
        a--;
        fprintf(file, "a=%d\n", a);
    }
}

void *T3(void *id) {
    pthread_mutex_lock(&x_mutex);
    x--;
    x++;
    pthread_mutex_unlock(&x_mutex);
    y++;
}
```

Foi criada a *mutex* global `x_mutex`, que é usada para controlar o acesso à variável global `x`. O mesmo não foi necessário para `y`, pois essa variável só é acessada dentro de uma thread, ou seja, não há condição de corrida para `y`. Analogamente, não foi necessário controlar o acesso a `file`, pois apenas uma thread escreve no arquivo.

Questão 5

A

Há três possibilidades:

- "Ola mundo!\nfoo nao existe\n":
"Ola" é escrito em foo;
foo é renomeado para bar;
"mundo!\n" é escrito em bar;
"foo nao existe\n" é escrito em bar
- "Ola foo nao existe\n"
"Ola" é escrito em foo;
"mundo!\n" é escrito em bar;
foo é renomeado para bar (antes de fopen("foo", "r"));
"foo nao existe\n" é escrito em bar
- "Ola "
"Ola" é escrito em foo;
"mundo!\n" é escrito em bar;
foo é renomeado para bar (depois de fopen("foo", "r"))