Universidade Federal do Rio de Janeiro 2020.1(REMOTO)
Computação concorrente - Silvana Rossetto
Alunos: Luana Gonçalves Ribeiro (117227266) e Lucas Tatsuya Tanaka (118058149)

Questão 1

a) A principal característica de um programa concorrente seria a mudança de paradigma de programação, na qual ocorre a troca do ideário da utilização apenas de uma thread para a troca do uso de múltiplas threads no programa visando o aproveitamento da arquitetura moderna de processadores. Com isso, é possível executar trechos de código em paralelo.

b) A aceleração máxima será de 2,5 caso sejam utilizados 4 Threads para o processamento da tarefa, caso seja utilizado mais Threads o tempo proposto de aceleração será maior de acordo com a lei de Amdahl.

c) Seção crítica é o trecho do código concorrente na qual pode ocorrer o surgimento de resultados indesejáveis devido ao acesso de múltiplas threads à mesma parte do código.

d) A sincronização por exclusão mútua é um tipo de mecanismo de programação concorrente na qual tem por objetivo controlar a ordem de execução sob a seção crítica do código. Nela, caso uma thread esteja executando em sua seção crítica não é permitido que outra thread execute também. Sendo assim, se fará necessário o término da execução da seção crítica da primeira thread para que outra possa assumir.

Questão 2

- **-3:** Seria necessário que as 3 operações de x-- fossem realizadas consecutivamente, porém isso não ocorre, o que faz com que o menor resultado possível seja -2.
- **-1:** é possível através das operações descritas:

1) T2(1) o valor de x se torna -1
2) T2(2) o valor se torna -2
3) T2(3) O if aceita o valor de x
4) T1(1) o valor de x se torna -1
5) T2(4) o valor printado será -1

- **1:** possível através das operações descritas:
    1) T3(1) o valor se torna 1
    2) T3(2) o valor se torna 2
    3) T3(3) o if aceita o valor de x
    4) T2(1) o valor x se torna 1
    5) T3(4) o valor printado será 1
- **3:** possível através das operações descritas
    1) T3(1) o valor se torna 1
    2) T3(2) o valor se torna 2
    3) T3(3) o if aceita o valor de x
    4) T1(1) o valor x se torna 3
    5) T3(4) o valor printado será 3

Questão 3

a) A seguinte implementação não garante exclusão mútua pois pode ocorrer de ser executado da seguinte maneira:

1) T1(1) a aplicação é rejeitada de entrar
2) T0(1) se adentra no secao critica
3) T1(2)
4) T1(3) se troca o valor de TURN para 0
5) T0(2) se executa a seção crítica do programa mesmo com as variáveis TURN não sendo consistente com o loop While.

Logo com o término do processamento se identifica que a garantia de exclusão mútua não existe. Isso ocorre porque a seção crítica não se dá de forma atômica, fazendo assim com que a seção crítica de uma outra thread possa referenciar a mesma variável.

b) Essa solução não atende aos requisitos de implementação uma vez que não proíbe que uma seção crítica seja executada com a exclusividade na qual o conceito de sincronização de espera ocupada se propõe.

Questão 4

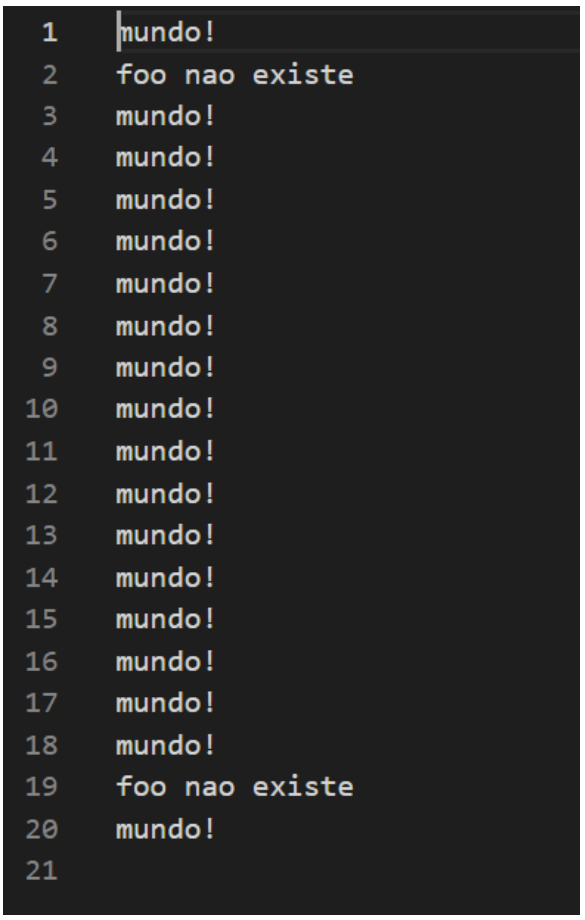
```c
int x = 0, y = 0;       //variaveis globais
pthread_mutex_t mutex; //variavel de lock para exclusao mutua

//-----------------------------Thread 1
void *T1(void *id)
{
    int a = 0;
    while (a < 2)
    {
        pthread_mutex_lock(&mutex);
        if (x == 0)
        {
            printf("x=%d\n", x);
        }
        pthread_mutex_unlock(&mutex);
        a++;
        printf("a=%d\n", a);
    }
}
//-----------------------------Thread 2
void *T2(void *id)
{
    int a = 2;
    while (a > 0)
    {
        pthread_mutex_lock(&mutex);
        if (x == 0)
        {
            printf("x=%d\n", x);
        }
        pthread_mutex_unlock(&mutex);
        a--;
        fprintf(file, "a=%d\n", a);
    }
}
//-----------------------------Thread 3
void *T3(void *id)
{
    pthread_mutex_lock(&mutex);
        y++;
    pthread_mutex_unlock(&mutex);
}
```

Foram retirados os x++ e x-- em sua execução por conta da redundância já que as operações sobre o x resultam sempre em x=0. Os locks foram posicionados entre a seção crítica com o objetivo de gerar a exclusão mútua.

.

Questão 5

a) Abaixo consta o conteúdo do arquivo bar.



b) Podemos identificar a ocorrência de condições de corrida uma vez que é impresso um conteúdo inesperado no arquivo bar. O que significa dizer que há uma certa dependência em relação à ordem de execução de certas instruções.
No específico caso, a instrução “rename(‘foo’, ‘bar’)” nem sempre é executada após o término da thread. O que acaba por permitir a impressão da frase “foo não existe”, pois a execução da instrução da rename é passível de ocorrer antes da instrução de condição “if (!fd_foo)”.