Respostas do Roteiro

André Toyama

PARTE 1 – SINCRONIZAÇÃO EM C++ THREADS

Exercício 1:
arquivo thread1.cpp no diretório

MUTEX

Exercício 2:
O objeto do tipo mutex exige que o programa chame as funções .lock() e .unlonck() em alguns momentos específicos, já o lock_guard e o unique_lock realizam essas funções quando necessário, sem que seja necessário chamar a função no código.

Exercício 3:
A vantagem de ter usado essa técnica, é que não foi necessário informar para o programa quando ele deveria executar o lock e o unlock do objeto.

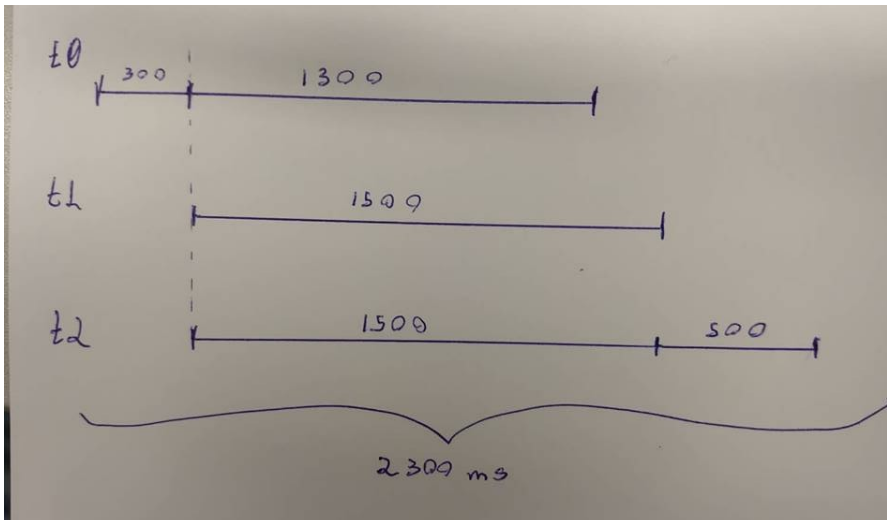
CONDITIONAL VARIABLE

Exercício 4:
O código nunca atinge o fim do trabalho B. Acredito que a causa do problema seja que no trabalho A, a notificação na variável cA está ocorrendo antes do trabalho B pedir para a mesma esperar a notificação.

Exercício 5:
Arquivo rdv.cpp no diretório

Exercício 6:
5.1 segundos

Exercício 7:



Exercício 8:
Arquivo cond_var.cpp no diretório

SEMAPHORES

Exercício 9:
Para um semáforo são necessários para a implementação o uso de mutex e lock, onde usaremos o lock para bloquear a thread de acordo com o contador do semáforo, até que aconteça um release.

Exercício 10:

```
void acquire(){

        count lock

        n = n-1

        se n > 0{
                queue lock
        }
        count unlock
}
```

Exercício 11:

```
void release(){

        count lock

        n = n+1;

        se n <= 0{
                queue unlock
        }
}
```

Exercício 12:
Arquivo rdv_sm.cpp no diretório.

PARTE 2 – PRODUTOR CONSUMIDOR

Arquivo pc.cpp no diretório