Practical exercise for Chapter 6

Student name: Doan Cao Thanh Long

Student ID: 20162513

Class: ICT-02.k61

Question 1:

Sometimes, *rsc* variable return result < 3000. This could be explained as because there is no synchronization in the program, 2 or more thread may read the same value of *rsc* variable and update it at the same time, making the program considers as 1 operation.

Question 2:

Because all threads are in synchronization, only one thread can access resource at a time, thus the result is 3000.

Question 3:

With lock, a worker can only access shared resources when it is free. After finish executing the task, the shared resource will be unlocked for other thread workers to access.

Question 4:

The completed code

```
while (time(NULL) <= end){
    shared++;
    sleep(1);
}
```

Question 5:

Since every threads can access resources freely, the balance and credit variables can be accessed at a same time by several threads. Without synchronization, the variable values one thread used may not be updated by another thread. For example, when thread A is calculating the new value of balance, thread B access the balance value before thread A is done calculating, thus the balance value that thread B use is not updated.

Question 6:

The following code responsible for the difference

```
while (lock > 0);
lock = 1;
shared++;
```

```
lock = 0;
```

When lock `= 0`, the while stops. Other threads could have the lock value `= 1` and increase share at the same time.

Question 7:

When using mutex lock, the number of errors reduced drastically. The error rate is nearly zero.

Question 8:

With `10000` threads, Coarse Locking runs in ~`3`s while Fine Locking takes ~`1.5`s, and both locking strategies achieve same performance.

Question 9:

The program is stuck in deadlock state so there is no output.

Deadlock is the phenomenon where threads waiting for each other resources to be unlocked for it to unlock its current used resource.

In this case, thread `1` is holding resource A waiting for thread `2` to release resource B for it to unlock resource A. On the other hand, thread `2` is holding resource B waiting for thread `1` to release resource A for it to unlock resource b. Two threads are waiting for each other forever because the requirement for each thread to unlock its resources is not met.