CHAPTER 6: SYNCHRONIZATION

Question 1: Launch this program several times. What do you notice? Explain it!

After running the program for several times, the final value of rsc variable of the instance resource varies and less than 3000 most of the times.

When several threads are running and accessing the resource at a time, there will be the case that 2 threads will increase the value so instead of increasing by 2, the variable increases only 1, so the final result is less than 3000.

Question 2: Change the code of the general executable program by replacing ThreadedWorkerWithoutSync with ThreadedWorkerWithSync to initiate three instances worker1-3. What is the difference between the output of this program and that of question 1? Explain it!

The final result of the variable is always 3000 when running the program for several times because synchronized block of the same object can have only one thread executing them at the same time.

Question 3: Change the code of the general executable program by replacing ThreadedWorkerWithSync with ThreadedWorkerWithLock to initiate three instances worker1-3. What is the difference between the output of this program and the output in question 1? Explain it!

The final result of the variable is always 3000. Each thread locks the resource for up to 10 seconds, and variable's value is added during this time when there is only one thread accessing it. Therefore, the program runs synchronously in this case.

Question 4: Complete this file above (in the part YOUR-CODE-HERE) with a loop to increase the variable shared by 1 for 5 seconds. (hint: time(NULL) will return the present system time in second).

```
void *fun(void *args)
{
    time_t start = time(NULL);
    time_t end = start + 5; //run for 5 seconds
    while(time(NULL) < end){
        shared++;
    }
    return NULL;
}
```

Question 5: Try to increase the value of threads and the value of the constant NUM_TRANS after each execution time until you obtain the different results between Balance and INIT_BALANCE+credits-debits. Explain why do you get this difference.

If NUM_TRANS increases to a certain value, the for loop took noticeable time to execute, so when one thread access to the balance and credit/debit value, the values it received may be different from what the later thread received since the execution hasn't finished.

If the number of threads is large enough, you can see the difference in results because the value of credits or debits is still held by other thread where it supposes to be 0 if it runs correctly.

Question 6: Try to build and run this program. Launch it repeatedly until you see the difference between Shared and Expect values. Analyze the source code to understand the problem that leads to this difference.

The difference is caused due to the duplicate of threads access. If a number of threads can be accessed at the same time so the value only increased by 1 instead of increasing multiple times. This behavior leads to the Shared value to be less than the Expected value.

Question 7: Now, you have to modify the code of the file without-lock.c in implementing the mutex lock above (you can name it differently like mutex-lockbanking.c). Try to launch it repeatedly and evaluate the obtained output. What is the improvement after using mutex lock?

The improvement is significant. With mutex lock, now the result is still correct regardless of the number of threads increasing to a big number.

Question 8: compare the run times of the two strategies to prove that Fine Locking is faster and much faster on larger load sets.

Actually Fine Locking performance is poorer due to the fact that the critical section where the lock took effect is just a simple task.

Question 9: Run this program and what do you get as output? Explain what the deadlock is.

- Can’t get the output since the program has fallen into a deadlock. Might fall into the case where the thread request the resource b while it’s still being locked by the previous thread so it’s gonna wait forever.