Student name: Ha Minh Duc

Student ID: 20161067

Class: ICT 02 – K61

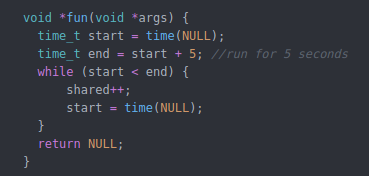
**Practical exercises of chapter 6**

Question 1: the results of the program are random numbers smaller than 3000. The reason iAf that when a thread called *exploit()*, it accessed the shared resource *rsc*, which is also being processed by other threads. Therefore, the value it get from *getRsc()* is not the updated value.

Question 2: the results are always 3000, because *synchronized* is applied for the shared resource, which means only 1 thread can access and process the resource at a time, other threads have to wait.

Question 3: the results are always 3000, because only one thread can hold the lock at a time and only the thread with the lock can use the shared resource, other threads have to wait until the lock is released.

Question 4:

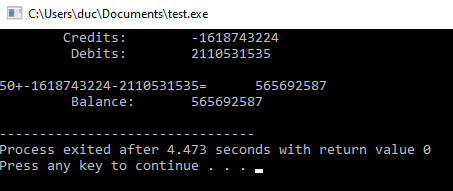
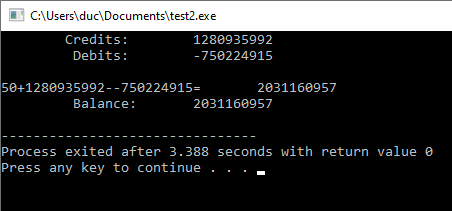
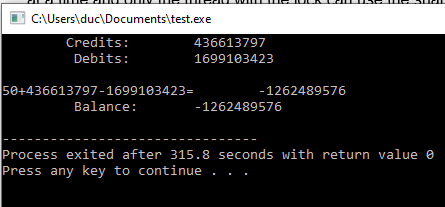


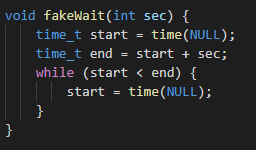
Question 5: Because *balance, credits, debits* are shared resources, and without locking mechanism, all threads can freely access them. Therefor, when a thread perform *transactions(),* the resources it use for calculation have already been changed by other threads → incorrect result.

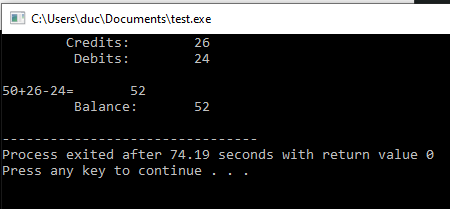
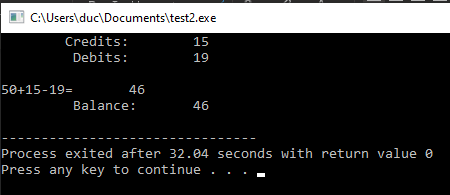
Question 6: The problem is that *lock* is also a shared resource, which means when the lock is released, multiple threads can hold the lock at the same time.

Question 7: The *shared* value is equal the expected value. The improvement is that the lock is guaranteed to be atomic in operation and the act of acquiring a lock cannot be interrupted, which means only 1 thread can be in critical session at a time.

Question 8:

* Testing with *NUM\_TRANS* = 100000, *n\_threads* = 5, it can be seen that fine lock is faster than coarse lock
  + Coarse lock: 
  + Fine lock: 
* Testing with *NUM\_TRANS* = 10000000, *n\_threads* = 5, it can be seen that fine lock is faster than coarse lock:
  + Coarse lock: 
  + Fine lock:
* In real banking system, each transaction takes a few seconds to process. To simulate this, a small delay is added before releasing each lock using a function:



* Testing results with *NUM\_TRANS* = 5, *n\_threads* = 5, balance calculation delay = 1s, credit and deposit calculation delay = 2s. Fine lock is much faster than coarse lock in this case:
  + Coarse lock: 
  + Fine lock: 

Question 9: There is no output from this program, it stuck somewhere around iteration 6000 → 7000 due to deadlock. Deadlock is a state in which each member of a group is waiting for another member, including itself, to take action. In this case, suppose that *fun\_1* is locking a and wait for *fun\_2* to release lock b, but at the same time, *fun\_2* is locking b and wait for *fun\_1* to release lock a. Both lock a and b are locked and can only be unlocked if one of them is unlocked.