104062203 陳涵宇

1. Problem Set

1. 7.4 In Section 7.4.4, we describe a situation in which we prevent deadlock by ensuring that all locks are acquired in a certain order. However, we also point out that deadlock is possible in this situation if two threads simultaneously invoke the transaction() function. Fix the transaction() function to prevent deadlocks.

```
void transaction(Account from, Account to, double amount) {
        mutex lock1, lock2;
        lock1 = get_lock(from);
        lock2 = get_lock(to);
        acquire(lock1);
                acquire(lock2);
                        withdraw(from, amount);
                        deposit(to, amount);
                release(lock2);
        release(lock1);
}
```

Add the third lock to ensure that the two threads, withdraw and deposit, can't be executed at the same time.

The fixed code is:

```
void transaction(Account from, Account to, double amount) {
        Semaphore lock1, lock2, lock3;
        wait(lock3);
        lock1 = get_lock(from);
        lock2 = get_lock(to);
        wait(lock1);
                wait(lock2);
                        withdraw(from, amount);
                        deposit(to, amount);
                        signal(lock3);
                signal(lock2);
        signal(lock1);
}
```

2. 7.7 Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free.

Assume the system is deadlocked. To satisfy the condition of deadlock, each process has at least one resource and is waiting for another. Since there are four resources while process number is only three, one of them is able to get one more resource, and it is no longer need resources. After that process finishes executing, it will release the resource, and thus no deadlock will happen.