Name: Tan Han Nguyen

Net ID: TXN200004

Chapter 11 Part 6 Reflection and Summary

Chapter 11 part 6 discussing thread synchronization when there are multiple threads of control sharing the same memory. First, synchronization is not needed when there is no shared memory (variables, structures, etc.) between the threads, or if the data is read-only without any modification from any thread. But when there are modification of shared memory between multiple threads, synchronization must be in place to avoid consistency in our data. The chapter gives an example of 2 threads trying to increment the same global variable, since the increment is not an atomic operation (writing to and reading from register steps), if both threads try to increment the variable without synchronization, the result will be incorrect.In order to synchronize our control flows, we must make it a sequentially consistent environment, which means no possible ordering of the two threads can result in any other different result of the variable. Using mutexes is a way to ensure our data is only accessed by one thread at a time. A mutex is a lock that we set before a thread modify the data, after the modification, we can unlock it to allow other thread to access the data. Mutex ensures only one thread process at a time, and we must ensure all of our threads follow the same synchronization rule, otherwise, if only one thread is not following, the data may turn out to be inconsistent. A mutex variable is represented by pthread\_mutex\_t data type, we must initialize before using and destroy it after done. The chapter suggests that we must be careful to when using mutex to avoid deadlock. A good practice is that we must ensure all threads lock all mutexes at the same order, but it can get complicated in some application’s structures, in this case, we can use pthread\_mutex\_trylock to simplify the structure. The chapter also discusses spin locks. Spin lock is like a mutex but instead of sleeping the process is blocked by busy-waiting until the lock can be acquired, this can help with reduce the cost of the process being descheduled. Spin locks are useful for low-level premitives to implement other types of locks, but it could lead to wasting CPU resources, so it should only be held for a short time period. Spin locks are useful for nonpreemptive kernal, but are not aseful at user level. The interfaces of spin locks are similar to other type of locks using pthread\_spin\_t, pthread\_spin\_lock, pthread\_spin\_unlock, etc. But many mutex implementation are so efficient, making spin locks only useful in limited situations. In conclusion, this chapter is a good introduction to multi-thread programming and control flow, especially how to synchronization works between multiple threads.