Lab 6: Synchronization

Problem 1:

This is a kind of synchronous problem, two or many processes dispute a shared resource. Specifically, the bank account is a shared resource which a couple (husband and wife) is using together. We can image that the husband is process A and the wife is process B and the shared bank account is a varible called *int money* . There are two functions : deposit (amount) and withdraw (amount). This problem is the same as the producer - comsumer problem.

Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit(). There will be two situation can happen:

- Process A and process B do their function alternately. The result will not be mistake because after process A complete its function, the system updates the value of *money.* And then process B do its function, the value of *money* is ready for using.

- Process A and B do thier function concurrently. This leads to the mistake of the result because 2 processes is using the shared resource at the same time, the system can not update 2 states at the same time and the result will be mistake.

To solve this problem, the bank should use the mutex lock model. When some one make a banking transaction, the account will be lock and notice to another user (if they want to make banking transaction with the same account) that "This account is processing, please try again later". It mean this account is processing a banking transaction and the user will have to wait until it've done.

Problem 2:

In this lab, we create a mutex lock and add it into pi_multi-thread.c file in the last lab. Comparing the performance, two program ensure that critical region is not disputed, ensure the accuracy of althorithm and high performance. But the program in lab 5, we use a array with n menbers which contains the value of each thread. This way will spend much more memory than using only one variable. So in the program of lab 6, we declare a variable called *count* and then all thread will use this shared variable. To avoid race condition when using shared variable, we using mutex lock to synchronize the value and make sure the accuracy of althorithm. The weakness of th lab 6's program is slower than the program in lab 5 a little bit. The reason is, every single thread have to wait for the shared variable unlock.