**CS446 Lab 6 – Pthread Mutex locks**

**Labs that are not scheduled for a lab test are not mandatory. These are practice labs, designed to help you on your assignments.**

**Solutions to practice labs will not be posted online.**

**Outline**

**Banking System Problem**

Consider a banking system that maintains an account balance **(integer variable amount)** with two functions: **deposit** and **withdraw**. These two functions pass the amount to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account.

Concurrently, the husband calls the **withdraw**function, and the wife calls**deposit**. A race condition is possible when theshared data **(amount)** is accessed by the two functions concurrently.

In this lab, you are to write a C program that provides a critical section solution to the Banking System Problem using mutex locks provided by the POSIX Pthreads API. **In particular, your solution needs to do the following:**

**Steps to follow:**

1. Initialize amount = 0
2. Take two command-line arguments. The first argument is the amount to be deposited (an integer value), and the second argument is the amount to be withdrawn (an integer value).
3. Create a total of 6 threads that run concurrently. Three threads call the deposit function, and the other three call the withdraw function.
4. When you create the thread calling the deposit function, you need to pass the thread identifier, the attributes for the thread, name of the deposit function, and the first integer parameter that was provided on the command line, avg[1] (which is the amount to be deposited).

1. Similarly, when creating the thread calling the withdraw function, you need to pass the thread identifier, the attributes for the thread, name of the withdraw function, and the second integer parameter that was provided on the command line, argv[2] (which is the amount to be withdrawn).
2. You are to use mutex locks provided by the Pthreads API to achieve mutual exclusion.
3. You are to provide print statements that output an error message if an error occurs while creating threads, mutex locks, etc.
4. You are to also provide print statements in the deposit and withdraw functions that output the value of the shared variable ‘amount’ after any modification.
5. Finally, the parent thread should output the final amount value after all threads finish their execution.

**Hint:**

* **Look at the Pthread in lecture notes 4.24 – 4.29 for Chapter 4 (Threads) to see how command line arguments are passed and used in the function called by a thread.**
* **Look at the Pthread mutex in slides 6.26 – 6.28 for Chapters 6 and 7 (Synchronization Tools and Examples) to see how to declare, create mutex, and implement the acquire and release functions.**

**Note:**

* **You might see that the amount is negative. This could happen if the threads calling the withdraw function are scheduled to run on the CPU before the deposit function. This is OK.**

**Sample Output: /lab6 100 50**



Withdrawal amount = -50

Withdrawal amount = -100

Withdrawal amount = -150

Deposit amount = -50

Deposit amount = 50

Deposit amount = 150

Final amount = 150