**Trent University: Operating Systems (COIS3320)**

**Dr. Bin Guo**

**Lab 6 – Pthreads Mutex Locks**

**Banking System Problem**

Consider a simple banking system that maintains bank accounts for its users. Every bank

account has a balance (represented by an integer variable **amount**). The bank allows

deposits and withdrawals from these bank accounts, represented by the two functions:

**deposit** and **withdraw**. These functions are passed an integer value that is to be

deposited or withdrawn from the bank account. Assume that a husband and wife share a

bank account. The husband only withdraws from the account and the wife only deposits into

the account using the **withdraw** and **deposit** functions respectively. Race condition is

possible when the shared data (**amount**) is accessed by these 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:

1) Take two command line arguments. First argument is the amount to be deposited (an

integer value) and the second argument is the amount to be withdrawn (an integer value).

2) Create a total of 6 threads that run concurrently using the Pthreads API.

a. 3 of the 6 threads call the deposit() function, and

b. 3 of the 6 threads call the withdraw()function.

3) Create the threads calling the deposit() function using the pthread\_Create()

function. While creating these threads you need to pass the thread identifier, the attributes for the thread, deposit() function, and the first integer command line argument argv[1] (which is the amount to be deposited).

4) Similarly, create the threads calling the withdraw() function.

5) To achieve mutual exclusion use mutex locks provided by the Pthreads API.

6) You are to provide print statements that output an error message if an error occurs while

creating threads, mutex locks etc.

7) Your program should print the value of the shared variable amount, whenever it is

modified.

8) Finally, the parent thread should output the final amount value after all threads finish

their execution.

**Make sure you use pthread\_join() for all the threads created. This will ensure that the**

**parent thread waits for all the threads to finish and the final amount reported by main**

**is correct for every execution of the program.**

**Notes:**

1. Look at the Pthreads example mentioned in Lab 4 to see how command line arguments

are passed and used in the function called by a thread.

2. See lecture slides on Chapters 6&7 for using mutex locks provided by the pthreads

API.

3. 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

for this practice lab.

**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