Lab 5b6a – Semaphores in C on Linux

Operating Systems CS SH3 Term 2, Winter 2018

Dr. Neerja Mhaskar

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

Lab Format: The practice labs will be posted a day before or on the day of the lab on the course website. You can choose to solve it beforehand and come in with your solutions and check the correctness of your solution with your TA.

The TAs will also be available to answer any questions you might have on your assignments.

Solutions to practice labs will not be posted online.

Outline

In this lab you are to modify your C program created for practice lab 4b and 5a to ensure the following conditions are met:

1. Withdrawals don't take place if `amount <=0`
2. Deposits don't take place if `amount >=400`
3. The amount of money deposited or withdrawn at a given time is 100.
4. Your program should create a total of **10** threads that run concurrently. **7 of 10** threads call the `deposit()` function and **3 of 10** threads call the `withdraw()` function.

In particular your solution needs to do the following:

1) Take one command line argument. Since the amount of money withdrawn/ deposited at a given time is 100. The value of the command line argument is 100.
2) You are to use mutex locks provided by the Pthreads API to achieve mutual exclusion as explained in the practice lab 4a/5a.
3) You are to use **two** semaphores to ensure **condition 1 and condition 2** are met.
4) Additionally, you need to set the initial value of these semaphores correctly for your solution to work.
5) You are to provide print statements that output an error message if an error occurs while creating threads, mutex locks semaphores etc.
6) You are to provide print statements in the deposit and withdraw functions that output the value of the shared variable '`amount`' after each modification.

7) You are to provide print statements at the beginning of the `deposit()` and `withdraw()` function. This way you can see (through the print statements) when threads beginning their execution in their functions.
8) Finally, the parent thread should output the final amount value after all threads finish their execution. Since deposit function gets executed 7 times and withdraw () function gets executed 3 times the correct final amount = 7*100 – 3*100 = 700-300 = 400.
   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 this thread is correct for every execution of the program.

Notes:

1. Look at practice labs 4a/5a to see how command line arguments are passed and used in the function called by a thread.
2. See lecture slides on Chapter 5 for using mutex locks provided by the `pthreads` API and using semaphores provided by the `Pos sem extension`.
3. **If you see negative amount values, your solution is incorrect.**

Sample Output: ./lab6b7a 100

Executing deposit function

Amount after deposit = 100

Executing Withdraw function

Amount after Withdrawal = 0

Executing Withdraw function

Executing Withdraw function

Executing deposit function

Amount after deposit = 100

Amount after Withdrawal = 0

Executing deposit function

Amount after deposit = 100

Amount after Withdrawal = 0

Executing deposit function

Amount after deposit = 100

Executing deposit function

Amount after deposit = 200

Executing deposit function

Amount after deposit = 300

Executing deposit function

Amount after deposit = 400

Final amount = 400