CS 3305A: Operating Systems
Department of Computer Science
Western University
Assignment 5
Fall 2022
Due Date: Dec 2nd 2022

Purpose

The goals of this assignment are the following:

- Get hands-on experience in developing mutual exclusion/semaphore/critical section techniques and algorithms.
- Gain experience with the C programming language from an OS's process synchronization perspective.

Assignment Description

Using C programming language, you will develop a mutual exclusion algorithm for a process synchronization problem. You must ensure that your mutual exclusion algorithm guarantees only one process access to the critical section portion of your code at a given time. You are allowed to use any mutual exclusion/semaphore related C function calls.

Description of the problem:
Assume that there are a set of n bank accounts ($n \geq 1$) shared by a set of x clients ($x \geq 1$). Clients can perform two different types of transactions with each bank account: deposit and withdraw funds. If a particular transaction results in a negative account balance, the transaction should be ignored (i.e., an account balance should never be less than 0).

Example and structure of the input file:

In the following example, there are two bank accounts (account1 and account2) shared by a total of ten clients (c1 to c10). The clients are allowed to deposit money into both accounts and withdraw money from both the accounts. The initial balances of the accounts are specified in the input file. An input file is provided below for illustrative purposes.

```
account1 balance 7000
account2 balance 4500
c1 deposit account1 100 withdraw account2 500
c2 withdraw account1 2500
...
...
c9 withdraw account1 1000 withdraw account2 500
c10 deposit account1 50 deposit account2 200
```

Illustration:

(i) account1 balance 7000

The above line specifies the initial balance of account #1 as $7000

(ii) c1 deposit account1 100 withdraw account2 500

The above line specifies the operations performed by client #1. client #1 deposits $100 into account #1, then withdraws $500 from Account #2.

The input file will be provided as "assignment_5_input.txt", and it must be hard-coded inside your program as "assignment_5_input.txt". A different input file will be used to evaluate your program for marking purposes where the structure of this input file name will remain the same, and only the data will be different.

For every client, you must use a separate thread. The transactions for a unique client should be handled by a distinct thread. For appropriate synchronization among the threads (i.e., critical sections are protected against random access by the threads) you must use mutual exclusion-related system functions (such as pthread_mutex_lock(), pthread_mutex_unlock() etc. which will be discussed in lecture #14). You must output the balances of each bank account after all the transactions have been performed. For each bank account, your output should display the account followed by the account balance. For example:

```
account1 balance 500
account2 balance 300
```

Your C program should output results to the screen.

Sample Input and Output File

You must not use any other format, such as user input from the terminal, command-line argument, etc., for providing input to the program. Also, **you must not modify the format of the provided input file**. The output of your program must follow the format of the sample output given above. For testing purposes, we have designed the sample input in a way where the final account balances will always remain the same after completing the transactions. However, when testing your program with another input file, note that due to the non-deterministic nature of threads, the final account balances may vary when running your program multiple times.

Mark Distribution

This section describes a tentative allocation of points assigned for the desired features.

A. Supplying ***assignment_5_input.txt*** to the program: 10 points
B. Parsing the input file correctly: 10 points
C. Printing the correct number of accounts and clients: 10 points
D. Using semaphore or mutual exclusion technique: 20 points
E. Using a separate thread for each client: 25 points
F. Correct balance of the accounts after all transactions: 25 points

Computing Platform for Assignments

You are responsible for ensuring that your program compiles and runs without error on the computing platform mentioned below. **Marks will be deducted** if your program fails to compile or runs into errors on the specified computing platform (see below).

- Students have virtual access to the MC 244 lab, which contains 30 Fedora 28 systems. Linux machines available to you are **linux01.gaul.csd.uwo.ca** through **linux30.gaul.csd.uwo.ca**.
- It is your responsibility to ensure that your code compiles and runs on the above systems. You can SSH into MC 244 machines (please see the Assignment 1 file transfer tutorial).
- If you are off-campus, you have to SSH to **compute.gaul.csd.uwo.ca** first (this server is also known as sylvia.gaul.csd.uwo.ca, in honor of Dr. Sylvia Osborn), and then to one of the MC 244 systems **(linux01.gaul.csd.uwo.ca** through **linux30.gaul.csd.uwo.ca**) (please see the Assignment 1 file transfer tutorial).
- https://wiki.sci.uwo.ca/sts/computer-science/gaul

Assignment Submission

You need to submit only one C file. The name of your submitted C file must be **"assignment5.c".** Marks will be deducted if your submitted C file name is different. You must submit your assignment through OWL. Be sure to test your code on one of MC 244 systems (see "Computing Platform for Assignments" section above). **Marks will be deducted** if your program fails to compile or runs into errors on the computing platform mentioned above.

Assignment 5 FAQ will be made available on OWL as needed. Also, consult TAs and the Instructor for any questions you may have regarding this assignment.