CECS 326 Assignment 8 (5 points)

Due: 11/04/2020 for Wednesday Section

11/09/2020 for Monday Section

Submit by lab class time on BeachBoard

Lab 8 and Assignment 7 are for an implementation of the bank account transfer code shown on slides 4-5 of the Deadlocks Lecture Note. You will need to create two programs to perform the following simple logics.

* create two bank accounts
* put some money in the two accounts
* run one transaction to transfer some money from account-1 to account-2
* run another transaction to transfer some money from account-2 to account-1

Bank accounts are shared data that may be accessed and manipulated by different processes. Therefore, they must be maintained in shared memory. Their access would require mutual exclusion through semaphores. Use the POSIX implementation of shared memory and semaphore for this assignment.

For this assignment, you will need to write two C programs named *master.c* and *transfer.c*, and compile them into executables *master* and *transfer*, respectively, using gcc and link with –lpthread. The two programs should include the following header file that defines the structure of the shared memory segment:

/\* accounts.h \*/

/\* Header file to be used with

\* master.c and transfer.c

\*/

struct ACCOUNTS {

int nAccounts; /\* number of active accounts \*/

int accounts[50]; /\* space to hold up to 50 accounts \*/

};

Suppose the program execution is launched as follows:

./master *400 200 my\_shm\_name sem1 sem2*

*master* should do the following:

* Create a POSIX shared memory segment named *my\_shm\_name* [ *my\_shm\_name* is from comandline ]
* Set up 2 accounts in shared memory with amounts *400* and *200* [ based on data from commandline]
* Output the current content of the ACCOUNTS struct in shared memory
* Create two named semaphores with names *sem1* and *sem2* [ *sem1* & *sem2* from commandline]
* Create a child process to execute *transfer*, with parameters (*my\_shm\_name*, *sem1*, *sem2*, *1*, *2*, *50*) indicating to transfer $*50* from account *1* to account *2* that are maintained in shared memory named *my\_shm\_name*. Semaphores *sem1* and *sem2* are for mutual exclusion to accounts 1 & 2, respectively.
* Create a seconf child process to execute *transfer*, with parameters (*my\_shm\_name*, *sem2*, *sem1*, *2*, *1*, *25*) indicating to transfer $2*5* from account *2* to account *1* that are maintained in shared memory named *my\_shm\_name*. Semaphores *sem1* and *sem2* are for mutual exclusion to accounts 1 & 2, rspectively.
* Wait for child process to terminate
* Upon receiving termination signal from child process, output updated content of the ACCOUNTS struct
* Close and unlink semaphores
* Close access to shared memory segment, unlink and exit

*transfer* should do the following:

* Acquire access to shared memory segment
* Acquire first lock listed in formal arguments
* Sleep for a time ranged from 0 to 5 seconds
* Acquire second lock listed in formal arguments
* Make necessary transfer as indicated in parameters
* Output actions taken
* Close semaphores
* Close access to shared memory segment and exit

The program must run successfully on Linux. Run the programs with zero (0) sleep time first, and then experiment with different amount of sleep times. The idea is to see if deadlock could occur. Discuss what happens on the Cover Page. Make sure that enough clear output is produced to enable understanding of logics in the programs.

Since named semaphores are persistent semaphores. They will remain in the system until they are unlinked or when the system shuts down. Therefore, to account for the possibility that a named semaphore created in the previous run did not get removed, you should do an error check at the sem\_open() call, and do a sem\_unlink there if error is found.

Do the following for this assignment:

1. Develop two C programs that work as described above. Discuss what the programs are designed to show on the Cover Page. Make sure your source programs are properly formatted as well as adequately and clearly commented. Detailed explanations on all system calls are required and must be in your own words. Simply copying those from the man pages are not acceptable.
2. Submit on BeachBoard the two C programs, a screenshot that shows successful compile of both programs as well as a successful run, and a cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear program description detailing what the programs are about. Format of the cover page should follow the cover page template on BeachBoard. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.