CECS 326 Extra Credit Assignment (5 points)

Due: 12/7/2020 for Monday Section

12/9/2020 for Wednesday Section

Submit by lab class time on BeachBoard

Lab 11 and this Extra Credit Assignment are still 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. Recall you had used the POSIX implementation of shared memory and named semaphores in Assignment 7 to provide the shared access and control.

With named semaphores, however, you would need to generate a unique name for every semaphore and associate each name with a bank account. This approach will become difficult to manage when the number of bank accounts is large. In Lab 10 you have learned the POSIX unnamed semaphores and all the APIs for using them. In this assignment, you will control access to the individual accounts by using POSIX unnamed semaphores, which can be declared in the same shared memory segment where the accounts are stored. (Hint: Declare an array of semaphores in the ACCOUNTS struct, one semaphore for every account.)

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. Note that the two programs would have to include a header file similar to the one in Assignment 7 but added with an array of unnamed semaphores.

Suppose the program execution is launched as follows:

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

*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 all existing accounts in the ACCOUNTS struct in shared memory
* Initialize 2 semaphores in shared memory with value 1 [ the number 2 is based on data from commandline]
* Create a child process to execute *transfer*, with parameters (*my\_shm\_name*, *1*, *2*, *50*) indicating to transfer $*50* from account *1* to account *2* that are maintained in shared memory named *my\_shm\_name*.
* Create a second child process to execute *transfer*, with parameters (*my\_shm\_name*, *2*, *1*, *25*) indicating to transfer $*25* from account *2* to account *1* that are maintained in shared memory named *my\_shm\_name*.
* Wait for child processes to terminate
* Upon receiving termination signals from all child processes, output current content of all existing accounts in the ACCOUNTS struct
* Destroy semaphores
* Close access to shared memory segment, unlink and exit

*transfer* should do the following:

* Acquire access to shared memory segment
* Acquire the semaphore lock associated with the first account listed in formal arguments
* Sleep for a time ranged from 0 to 5 seconds
* Acquire the semaphore lock associated with the second account listed in formal arguments
* Make necessary transfer as indicated in parameters
* Output actions taken
* Release semaphore locks
* 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 in your experiments on the Cover Page. Make sure that enough clear output is produced to enable understanding of logics in the programs.

Do the following for this assignment:

1. Develop two C programs that work as described above. Discuss what the programs are designed to show and what you have observed through the multiple runs 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 in the program comments 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 successful runs, and a cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear description of what the programs are about as well as your observations. 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.