CSc 332 - Operating Systems

Task 5 - Process Synchronization

Dad-Son Problem

Max Points: 25 **Due**: April 30, 2020

The given bank.c program has 3 processes namely, the dad process and two son processes. The critical (CS) in the given problem is a son withdrawing money from the bank and the dad depositing money in the bank, at randomly selected time intervals. The program in the given form has synchronization errors, i.e., these 3 processes get into a race condition when accessing the shared bank_balance variable. The program itself compiles correctly (i.e., there are no syntax errors).

In **Step 1**, you need to run the program and analyze the execution traces, whereupon you have to identify the synchronization errors. After identifying the synchronization errors, you need to insert the "P(sem)" and "V(sem)" operations at the right places in the code that fix the synchronization errors.

In **Step 2**, you work on a measurement component. A mutex algorithm is associated with a "bounded wait" property, i.e., how long a process P is forced to wait for entry into a CS after P has expressed interest in the CS. Let's call the wait time as T(P), where T(P) is the number of times processes other than P enter the CS after P expresses interest in the CS but before P actually enters the CS. You need to compute T(P) for all three processes in the problem and display it at the end of "N" different attempts. Measure T(P) for at least 5 different N's and include the values in your report.

Semaphores:

Semaphores are system variables used for process synchronization. You may think of a semaphore, sem, as a variable maintained by the system. A semaphore can be obtained by a **semget()** system call. Its initial value can be set by the **semctl()** system call. There are two common operations that a process can perform on a semaphore, sem, namely:

P(sem) or wait(sem): If the value of sem is greater than 0, then this operation decrements the value of sem and the calling process continues. Otherwise, if sem is 0, then the calling process is blocked on s.

V(sem) or signal(sem): If any process is blocked on sem, then this unblocks (wakes up) the earliest among the processes blocked on sem. Otherwise, the value of the semaphore is incremented.

In UNIX/Linux, both P(s) and V(s) can be done with the **semop**() system call with appropriate parameters.

Instructions

- You need to have **sem.h** header file in your present working directory to run this program and invoke semaphore operations.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
```

- The bank.c file is documented. Once you are done with your solution, insert comments at the places where you made changes to fix the synchronization errors and add the wait time measurement.
- This synchronization problem should be solved with as few semaphore variables as possible.

Submission Instructions

Submit the bank.c file with semaphore operations inserted at appropriate places and measurement components. Write a brief report (at most 1 page) on how your solution prevents race condition between the dad and son processes and measure the wait time of each process. Zip both the files into a single folder as: task5_firstname_lastname.zip. Email your zip file with the subject line, "Task 5 - CSc 332 - firstname_lastname".