Due time & date: Please check Carmen

Purpose: Learn how Linux/Unix processes can synchronize using semaphores and shared memory.

Assignment Part A: There are three processes (plus Proc4 described later) and each process moves $200 between two accounts in each of its iterations. Proc1 moves from Account0 to Account1, Proc2 moves from Account1 to Account2, while Proc3 moves from Account2 to Account0. Initially, each account contains $10,000, so the total amount of money in the three accounts is $30,000. Note that it is allowed that an account gets in red. After going through 100 iterations every process prints contents of each account and the total amount of money contained in the three accounts at that moment. Note that if your programs are correct, the sum is always $30,000.

You are given parts of codes for Proc1 and Proc2, while code for Proc3 is similar to code of Proc2. You are supposed to add some statements without changing the existing code.

Code of Proc1

```c
int main()
{
  int i, internal_reg;
  /* here create and initialize all semaphores */
  /* here created: shared memory array Account of size 3 */
  Account[0]=10000;
  Account[1]=10000;
  Account[2]=10000;
  /* synchronize with Proc2, Proc3 and Proc4 (4 process 4 way synch.) */
  for (i = 0, i < 2000; i++)
  {
    internal_reg = Account[0];
    internal_reg = internal_reg - 200;
    Account[0] = internal_reg;
    /* same thing, except we're adding $200 to account1 now... */
    internal_reg = Account[1];
    internal_reg = internal_reg + 200;
    Account[1] = internal_reg;
  }
  /* Add a code that prints contents of each account and their sum after 100th, 200th, 300th, .... and 1900th iteration */
}
/* in the code above include some wait and signal operations on semaphores. Do not over-synchronize. */
```
**Code of Proc2**

```c
int main()
{
    int i, internal_reg;
    //synchronize with Proc1, Proc3 & Proc4 (4 process 4 way sync.)
    for (i = 0, i < 2000; i++)
    {
        internal_reg = Account[1];
        /*Proc3 takes from Account[2]*/
        internal_reg = internal_reg - 200;
        Account[1] = internal_reg;
        /* same thing, except we're adding $200 to Account[2] now... */
        /*Proc3 adds into Account[0]*/
        internal_reg = Account[2];
        internal_reg = internal_reg + 200;
        Account[2] = internal_reg;
    }
    /* Add a code that prints contents of each account
     and their sum after 100th, 200th, 300th, .... and 1900th
     iteration*/
}
/*in the code above include some wait and signal operations on
semaphores. Do not over-synchronize. */
```

Proc1 sleeps for 0.2 sec after 600th, 1200th and 1800th iterations (total 0.6 sec), Proc2 sleeps 0.2 sec after 400th, 800th, 1200th and 1600th iterations (total 0.8 sec) and Proc3 sleeps 0.3 sec after 500th and 1500th iterations (total 0.6 sec).

Proc4 has the 5000 iteration loop and it should synchronize with other 3 processes in 4 process 4 way synchronization before entering its loop. In each iteration, contents of all 3 accounts are read and at the end, Process 4 prints a number of times it checked the sum and a number of times the sum was not equal 30000 (if your programs are correct, the sum is always $30,000). Proc4 sleeps 0.05 sec after each of its 1000 iterations (total 0.25 sec). Before terminating, Proc4 removes all semaphores and shared memory created for this problem.

Based on instructions and given codes above, write C-codes for processes Proc1, Proc2 and Proc3 in files Proc1.c, Proc2.c, and Proc3.c, respectively and compile each file. Then write code for Proc4 in Proc4.c and compile it. First run Proc1 in one terminal window, then Proc2, Proc3 and Proc4 each in a different terminal window in any order.
On/before due day/time:

1. electronically submit your source codes Proc1.c Proc2.c
   Proc3.c and Proc4.c on Carmen Cse2431 at Assignment Lab3a
   and
2. turn in hard copies of your source codes (indicate your
   compilation commands) and a hard copy of the output of your
   programs from the last executions.

Assignment Part B: This is ‘One Consumer and all three
Producers’ problem with the following 4 processes: ProcX, ProcY,
ProcZ and ProcW. Each of processes has 500 iterations.

ProcX (one of producers) places items into empty slots of its
buffer (BufferA with 20 slots and each slot with an integer and
2 characters), but only one item in each iteration. Each item
should include the item number (1, 2, 3,..., 499, 500) for the
integer and characters ‘xx’.

ProcY (another of producers) places items into empty slots of
its buffer (BufferB, with 30 slots and each slot with 3 characters
and an integer), but one item in each iteration. Each item
should include characters ‘YYY’ followed by the item number
(1, 2, 3, ..., 499, 500) for the integer.

ProcZ (third producer) places items into empty slots of its
buffer (BufferC, with 25 slots and each slot with 2 characters
and an integer), but one item in each iteration. Each item
should include characters ‘ZZ’ followed by the item number (1,
2, 3, ..., 499, 500) for the integer.

In each iteration, ProcW (as a consumer) takes one item from
each buffer (total of three) and prints them. If the processes
are properly synchronized, your output will be:

1xx YYY1 ZZ1 2xx YYY2 ZZ2 3xx YYY3 ZZ3 4xx YYY4 ZZ4 5xx YYY5 ZZ5
6xx YYY6 ZZ6 ... 498xx YYY498 ZZ498 499xx YYY499 ZZ499 500xx
YYY500 ZZ500.

ProcX or ProcY or ProcZ waits if there is no empty slot for the
current item. ProcW waits if all slots in any buffer are empty.

After putting every 100 items in its buffer ProcX sleeps for 0.3
sec, ProcY sleeps for 0.3 seconds after putting every 75 items
in its buffer, ProcZ sleeps for 0.3 seconds after putting every
60 items in its buffer, while ProcW sleeps for 0.3 sec after
taking every 100*3 items.

ProcW creates and initializes all necessary semaphores and
shared memories, and you should run ProcW first (in one window).
Thus, you can run those three programs (from different terminal
windows) in any order. ProcX removes all semaphores and memories created for this problem. ProcX, ProcY and ProcZ are synchronized according to three way three process synchronization at the beginning of their codes.


---------------------------
On/before due day/time:

1. electronically submit your source codes ProcX.c ProcY.c ProcZ.c and ProcW.c on Carmen Cse2431 at Assignment Lab3b and
2. turn in hard copies of your source codes (indicate your compilation commands) and a hard copy of the output of your programs from the last executions.