Cse660 **Lab 3. Assignment** February 11, 2011.

Due date: Friday, February 18, 2011.

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

Assignment Part A: There are three processes (plus one described later) and each process move $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 the total amount of money contained in the three accounts at that moment. Also, there is the fourth process that just reads all accounts. Note that the sum has to be 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 only to add some statements without changing the existing code (except for syntax error if any).

Code of Proc1

```
int main()
  {int i, internal_reg;
/* here create and initialize all semaphores */
/* here created: shared memory array Account of size 3 */
   Account[0]=10,000;
   Account[1]=10,000;
   Account[2]=10,000;
/* synchronize with Proc2, Proc3 and Proc4 (4 process 4 way
synch.)*/
     for (i = 0, i < 1000; 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;
        }
    /* here add a code that prints contents of each account and
    their sum after 100th, 200th, 300th, .... and 1000th
    iteration*/
  }
/*in the code above include some wait and signal operations on
semaphores. Do not over-synchronize. */
```

Code of Proc2

```
int main()
  {int i, internal_reg;
/*synchronize with Proc1, Proc3 and Proc4 (4 process 4 way
sync.)*/
     for (i = 0, i < 1000; 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;
         }
  /* here add a code that prints contents of each account
    and their sum after 100th, 200th, 300th, .... and 1000th
    iteration*/
}
/*in the code above include some wait and signal operations on
semaphores. Do not over-synchronize. */
```

The above file can be found at: /usr/class/cis660/lab3bfile.

Proc1 sleeps for 1 sec after 300th and 600th iterations, Proc2 sleeps 1 sec after 400th and 700th iterations and Proc3 sleeps 1 sec after 500th and 800th iterations.

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. Proc4 sleeps 1 sec after 4000 iterations. Before terminating, Proc4 removes all semaphores and shared *memory created* for this problem.

First copy codes of processes Proc 1, Proc2 and Proc3 in separate file, and after updating, compile each file. Then write code for Proc4 and compile it. First run Proc1 in one terminal window, then Proc2, Proc3 and Proc4 each in a different terminal window in any order..

Submit your source code files using command:

submit c660ab lab3a Proc1.c Proc2.c Proc3.c Proc4.c

Also, provide a hard copy of your source codes and a hard copy of the output of your programs from one of executions, plus compilation commands.

Assignment Part B: This problem is 'One Consumer and both Producers' problem with the following 3 processes: ProcX, ProcY and ProcZ. 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.

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

1xx YYY1 2xx YYY2 3xx YYY3 4xx YYY4 5xx YYY5 6xx YYY6 ... 498xx YYY498 499xx YYY499 500xx YYY500

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

After putting every 100 items in its buffer ProcX sleeps for 1 sec, ProcY sleeps for 2 seconds after putting every 75 items in its buffer, while ProcZ sleeps for 1 sec after taking every 100*2 items.

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

Use semaphores as the only synchronization mechanism and do not over-synchronize.

Submit your source code files using command:

submit c660ab lab3b ProcX.c ProcY.c, ProcZ.c

Also, provide a hard copy of your source codes and a hard copy of the output of your program from one of executions, plus compilation commands.