

CSE 660
Spring 2005

Lab 3

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

Assignment - Part A: There are three processes and each process move \$100 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 \$1000, so the total amount of money in the three accounts is \$3000. 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 (and it should be \$3000).

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]=1000;
    Account[1]=1000;
    Account[2]=1000;
    /* synchronize with Proc2 and Proc3 (3 process 3 way synch.)*/
    for (i = 0, i < 1000; i++)
    {
        internal_reg = Account [0];
        internal_reg = internal_reg - 100;
        Account[0] = internal_reg;
        /* same thing, except we're adding $100 to account1 now... */
        internal_reg = Account [1];
        internal_reg = internal_reg + 100;
        Account[1] = internal_reg;
        /* here add a code that prints contents of each account
        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 and Proc3 (3 process 3 way sync.)*/
    for (i = 0, i < 1000; i++)
    {
        internal_reg = Account [1];
        /*Proc3 takes from Account [2]*/
        internal_reg = internal_reg - 100;
        Account[1] = internal_reg;
        /* same thing, except we're adding $100 to Account [2] now... */
        /*Proc3 adds into Account [0]*/
        internal_reg = Account [2];
        internal_reg = internal_reg + 100;
        Account[2] = internal_reg;
        /* here add a code that prints contents of each account
        after 100th, 200th, 300th, .... and 1000th iteration*/
    }
}
/*in the code above include some wait and signal operations on
semaphores. Do not over-synchronize. */
```

Also, Proc3 is to remove all shared memories and semaphores created in this problem.

The above file can be found at: /usr/class/cis660/lab3bfile. First copy codes of each process in separate file.

After updating and compiling each file, run Proc1 first in one window, and then Proc2 and Proc3 in two different windows.

Submit your source code files using the command:
submit c660aa lab3a Proc1.c Proc2.c Proc3.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.

Assigned on 5/23 - Due on 6/1 in class

Assignment - Part B This problem is “One Consumer and both of two Producers” problem with the following 3 processes: ProcX, ProcY and ProcZ. Each of processes has 500 iterations.

ProcX 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 followed by the characters “xx”.

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

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

1xx 1YYY 2xx 2YYY 3xx 3YYY 4xx 4YYY 5xx 5YYY 6xx 6YYY ... 498xx 498YYY 499xx 499YYY
500xx 500YYY

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

After putting every 50 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 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 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 c660aa 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.

Assigned on 5/23 - Due on 6/1 in class

Assignment - Part C This problem is “One Consumer and any of two Producers” problem with the following 3 processes: ProcA, ProcB and ProcC.

ProcA places items into empty slots of its buffer (BufferX, with 20 slots and each slot with an integer and 1 character), but only one item in each iteration. ProcA has 255 iterations. Each item should include the item number (1, 2, 3,..., 254, 255) for the integer and the character “A”.

ProcB places items into empty slots of its buffer (BufferY, with 40 slots and each slot with an integer and 3 characters), but only one item in each iteration. ProcB has 345 iterations. Each item should include the item number (1, 2, 3,..., 344, 345) for the integer and the characters “BAB”.

In each iteration, ProcC takes one item from either buffer and prints it. ProcC has 600 iterations. ProcA or ProcB waits if there is no empty slot for the current item. ProcC waits if all slots in both buffers are empty.

After putting each 45 items in its buffer ProcA sleeps for 1 sec, ProcB sleeps for 2 seconds after putting each 65 items in its buffer, while ProcC sleeps for 3 seconds after taking each 190 items.

ProcC creates and initializes all necessary semaphores and shared memories, and you should run ProcC first in one window. ProcA and ProcB are synchronized according to two way two process synchronization at the beginning of their codes. Thus, you can run (from different windows) those two programs in any order.

Process B 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 c660aa lab3c 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.