***Formal Element***

***Semaphores***



**Name:** Tyrone Verburgt

**Student ID:** C11365256

**Module:** Operating Systems

**Lecturer:** Martin O’Hanlon

Contents

[Introduction 3](#_Toc373742168)

[Objective 4](#_Toc373742169)

[1. One Consumer & One Producer 4](#_Toc373742170)

[2. One Consumer & Two Producers 4](#_Toc373742171)

[3. Two Consumers & Two Producers 4](#_Toc373742172)

[Code 5](#_Toc373742173)

[Code for Producer 5](#_Toc373742174)

[Code for Consumer 7](#_Toc373742175)

[Results 10](#_Toc373742176)

[1. One Consumer & One Producer 10](#_Toc373742177)

[2. One Consumer & Two Producers 11](#_Toc373742178)

[11](#_Toc373742179)

[3. Two Consumers & Two Producers 12](#_Toc373742180)

[Conclusion 13](#_Toc373742181)

[Discussions 13](#_Toc373742182)

[Resources 13](#_Toc373742183)

# Introduction

A semaphore, in its most basic form, is a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment. In this report we are using semaphores to allow one process to access shared memory one at a time.

In simple terms semaphores allow one process to do a job and then once it is finished its job it signals for another process to take over.

Hold the value (2-9) of where the consumer should take the next item. (Initially start with 2)

Holds the value(2-9) of where the producer should put the next item

(Initially start with 2)

Producer

Consumer

1

0

9

8

7

6

5

4

3

2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

8 Spaces for integer values

2 Spaces for producer & consumer

A total size of 10 bytes shared memory

We will use the first space of memory (location 0) to hold a value to indicate where to write. And we will use to second space (location 1) to indicate where to read.

# Objective

This assignment involves an implementation of the Producer Consumer problem using shared memory and semaphores. You will make use of the system calls to create and use semaphores that we covered in the lab. You will have two programs (and hence two process) one called the Producer and the other Consumer. These programs will be run in separate terminal windows. The Producer will ask the user to enter number and will put them in the shared memory area. No more than 8 numbers should be in memory are at a time (buffer is length 8). If there are 8 items in the buffer the Producer should block. The Consumer should take the numbers out of the shared memory area and print them to the screen. If there are no items in the buffer the consumer should block.

In this report we are going to write and examine 3 multi-processing environments.

## 1. One Consumer & One Producer

## 2. One Consumer & Two Producers

## 3. Two Consumers & Two Producers

# Code

The producer code has been made so that it can be copied and pasted depending on how many producers one would like. That also follows the same for the consumer.

## Code for Producer

**#include<sys/types.h>**

**#include<sys/ipc.h>**

**#include<sys/sem.h>**

**#include<string>**

**#include<sys/shm.h>**

**#include <iostream>**

**#define SEMKEY 10000**

**#define SHM\_KEY 5000**

**using namespace std;**

**int main()**

**{**

**struct sembuf vsembuf, psembuf, v1sembuf, p1sembuf, v2sembuf, p2sembuf;**

**int semid, shmid;**

**int \*ptr;**

**//1 represent no of semaphores we want.**

**semid = semget(SEMKEY,1,0777|IPC\_CREAT);**

**//Initialise semaphore 0 represents which semaphore in set**

**semctl(semid,0, SETVAL, 0); //data**

**semctl(semid,1, SETVAL, 8); //space**

**semctl(semid,2, SETVAL, 1); //free**

**//Create an 9 byte block of shared memory.**

**shmid = shmget(SHM\_KEY, 9,0777|IPC\_CREAT);**

**//ptr points to first location in mem.**

**ptr = (int\*)shmat(shmid,0,0);**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SEMAPHORES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//This is the wait(data). //Semaphore 1.**

**psembuf.sem\_num= 0;**

**psembuf.sem\_op = -1;**

**psembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(data).**

**vsembuf.sem\_num = 0;**

**vsembuf.sem\_op = 1;**

**vsembuf.sem\_flg = SEM\_UNDO;**

**//This is the wait(space). //Semaphore 2.**

**p1sembuf.sem\_num= 1;**

**p1sembuf.sem\_op = -1;**

**p1sembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(space).**

**v1sembuf.sem\_num = 1;**

**v1sembuf.sem\_op = 1;**

**v1sembuf.sem\_flg = SEM\_UNDO;**

**//This is the wait(free). //Semaphore 3.**

**p2sembuf.sem\_num= 2;**

**p2sembuf.sem\_op = -1;**

**p2sembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(free).**

**v2sembuf.sem\_num = 2;**

**v2sembuf.sem\_op = 1;**

**v2sembuf.sem\_flg = SEM\_UNDO;**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SEMAPHORES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**ptr[0] = 2; //We initialise value within vector to 2 as it indicates where**

**//the writing vector will write.**

**while(true)**

**{**

**semop(semid, &p1sembuf, 1); //wait(space)**

**semop(semid, &p2sembuf, 1); //wait(free)**

**cout <<"Enter a number (Position: " <<ptr[0]<<") -> ";**

**cin >> ptr[ptr[0]];**

**ptr[0] = ptr[0] + 1; //The value of the pointer is incremented.**

**if(ptr[0] > 9)**

**{**

**ptr[0] = 2;**

**}**

**semop(semid, &v2sembuf, 1); //signal(free)**

**semop(semid, &vsembuf, 1); //signal(data)**

**}**

**}**

## Code for Consumer

**#include<sys/types.h>**

**#include<sys/ipc.h>**

**#include<sys/sem.h>**

**#include<sys/shm.h>**

**#include <iostream>**

**#define SEMKEY 10000**

**#define SHM\_KEY 5000**

**using namespace std;**

**main()**

**{**

**struct sembuf vsembuf, psembuf, v1sembuf, p1sembuf, v2sembuf, p2sembuf;**

**int semid, shmid;**

**int \*ptr;**

**//1 represent no of semaphores we want.**

**semid = semget(SEMKEY,1,0777|IPC\_CREAT);**

**//Create an 9 byte block of shared memory.**

**shmid = shmget(SHM\_KEY, 9,0777|IPC\_CREAT);**

**//ptr points to first location in mem.**

**ptr = (int\*)shmat(shmid,0,0);**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SEMAPHORES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//This is the wait(data). //Semaphore 1.**

**psembuf.sem\_num= 0;**

**psembuf.sem\_op = -1;**

**psembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(data).**

**vsembuf.sem\_num = 0;**

**vsembuf.sem\_op = 1;**

**vsembuf.sem\_flg = SEM\_UNDO;**

**//This is the wait(space). //Semaphore 2.**

**p1sembuf.sem\_num= 1;**

**p1sembuf.sem\_op = -1;**

**p1sembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(space).**

**v1sembuf.sem\_num = 1;**

**v1sembuf.sem\_op = 1;**

**v1sembuf.sem\_flg = SEM\_UNDO;**

**//This is the wait(free). //Semaphore 3.**

**p2sembuf.sem\_num= 2;**

**p2sembuf.sem\_op = -1;**

**p2sembuf.sem\_flg = SEM\_UNDO;**

**//This is the signal(free).**

**v2sembuf.sem\_num = 2;**

**v2sembuf.sem\_op = 1;**

**v2sembuf.sem\_flg = SEM\_UNDO;**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SEMAPHORES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**ptr[1] = 2;**

**while(1)**

**{**

**semop(semid, &psembuf, 1); //wait(data)**

**semop(semid, &p2sembuf, 1); //wait(free)**

**cout <<"Number (Position: " <<ptr[1]<<") -> "<< ptr[ptr[1]] << endl;**

**ptr[1] = ptr[1] + 1; //The value of the pointer is incremented.**

**if(ptr[1] > 9)**

**{**

**ptr[1] = 2;**

**}**

**semop(semid, &v2sembuf, 1); //signal(free)**

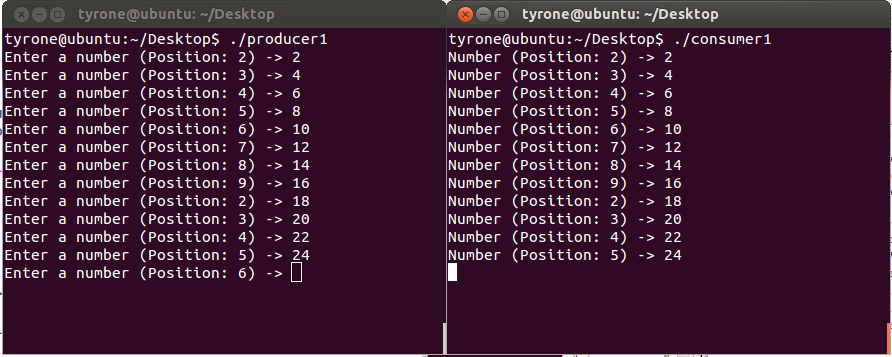
**semop(semid, &v1sembuf, 1); //signal(space)**

**}**

**}**

# Results

## 1. One Consumer & One Producer

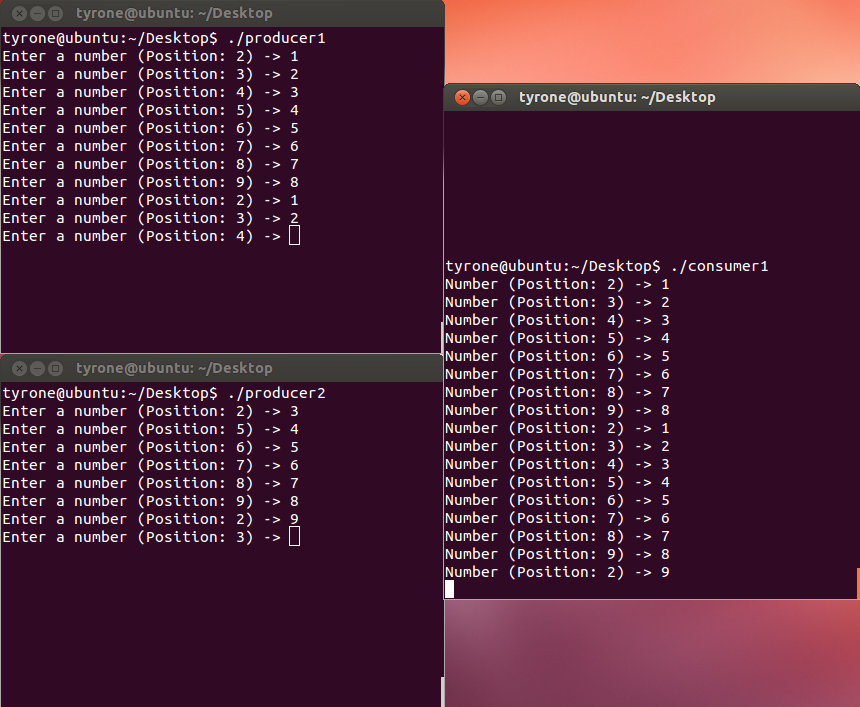


**Figure 1**

In this program, when the user inputs into the producer the consumer automatically updates itself with what was inputted. As shown figure 1 above it can be seen that whenever an integer was typed, it naturally updated on the right.

Note: the positions indicate where in the buffer (2-9) the integer was saved or taken from.

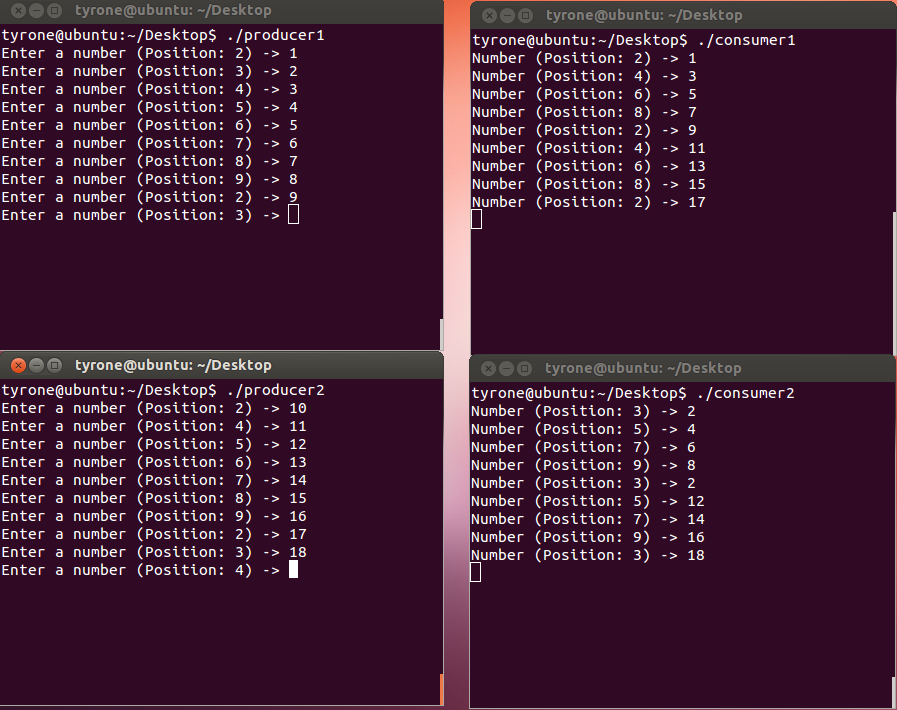
## 2. One Consumer & Two Producers



**Figure 2**

Shown in figure 2 above we have 3 processes running (2 Producers and 1 Consumer). It can be seen that whenever something was entered in one of the producers, it was updated on the consumer.

## 3. Two Consumers & Two Producers



**Figure 3**

It is very much the same thing for 2 producers and 2 consumers as shown above in figure 3. But there is one exception. The consumers take turns in printing out what the user enters in. Both consumers are in continuous need of the processor, so they have to form a queue and take turns. Figure 3 above demonstrates this queue operation.

# Conclusion

The objective is complete and I was able to complete what was required. Due to this lab and report I feel more knowledgeable on the topic of semaphores. I had difficulties grasping the concept because I didn’t understand the idea. Eventually after a little time and playing around with the code I was able to get an idea of what was required.

I feel I have learned a lot from the formal element.

# Discussions

One of the problems I encounter most was the fragmentation fault. I learned that this happens when a vector goes over the boundaries of allocated shared memory.

A recommendation I can give is always set your SEMKEY & SHM\_KEY to values greater than 5000. Initially they were set to around 1000. In this case the program gives segmentation faults and in some rare instances not.

# Resources

http://en.wikibooks.org/wiki/Operating\_System\_Design/Processes/Semaphores