AOS Assignment 3

In producer-consumer problem, both the producer and the consumer simultaneously work on a single piece of data at the same time. In our code, the producer produces values in a counter and assigns it to global variable ‘n’ and consumer at the same time prints the produced values. In this case, there should be proper synchronization between the producer and consumer processes so that every value produced is consumed.

In this assignment, we need to achieve this synchronization of critical sections in producer and consumer by using pre implemented semaphores in Xinu.

For this synchronization, we need two semaphores one for the producer process and other for the consumer process. The two semaphores: produced and consumed are first created using the semcreate() command in xsh_prodcons.c file. The two producer and consumer processes are created and put in the ready queue. The produced semaphore is for the producer process and the consumed semaphore is for the consumer process. The producer code is executed first as per our code.

In producer code, we first execute the wait command on consumed i.e. wait(consumed).

wait(consumed):

The wait command internally reduces the count of consumed semaphore and checks if the count is less than zero. If the count is less than zero, then it puts the consumed semaphore to wait state and puts the current process i.e. **producer** in wait queue of **consumed** and then it calls **resched()** system call which allows processor to execute next process waiting in ready queue. The critical section is not executed if the count is less than zero.

If the count is greater than zero, the critical section is executed and then we issue signal on produced.

signal(produced):

The signal command checks if the count is less than zero. If the count is less than zero, then it dequeues the process from its queue and increments the count. Once the process is dequeued from wait queue, it goes into ready state and the process is executed.

Similarly, the same process in followed inside the consumer code where the producer is first asked to wait, the critical section of consumer is executed and then the producer is signaled to start This way the synchronization is achieved between the two processes.

In our case, initially produced = 0 and consumed = 1.

Producer executes. wait(consumed) makes consumed = 0 and since it is > -1, the critical section is executed and then signal(produced) checks for any process in ready queue which is empty so it simply increments produced to 1.

Now, produced = 1 and consumed = 0.

Producer executes again. wait(consumed) makes consumed = -1 and since it is < 0, the critical section is skipped and producer is added to wait queue of consumed semaphore. resched() now transfer control to consumer process which is executed.

Consumer executes. wait(produced) makes produced = 0 and critical section is executed. signal(consumed) now checks if there is any process in its queue which is producer process in our case which is de-queued and executed and consumed is incremented to 0.

This process continues till the last values is produced and consumed.

Producer Code (producer.c): Amruta

```
#include <prodcons.h>

void producer(int count){

        while(n < count){

                sleep(1);

                wait(consumed);
                n++;
                printf("Produced value : %d \n", n);

                signal(produced);

        }

        printf("Producer stopped.\n");
}
```

Consumer Code (consumer.c): Ajinkya

```
#include <prodcons.h>

 void consumer(int count){

        while(1){
```

```
                sleep(1);

                wait(produced);
                printf("consumed value : %d \n", n);

                if(n == count){
                        break;
                }
                signal(consumed);
        }

        printf("Consumer stopped.\n");
        printf("Deleting semaphores.\n");

        semdelete(produced);
        semdelete(consumed);

        printf("Semaphores deleted.\n");
}
```

xsh_prodcons.c: Ajinkya and Amruta

```
#include <prodcons.h>
#include <stdlib.h>

int n;              //Definition for global variable 'n'

int isNumeric(const char *str)
{
   while(*str != '\0')
   {
      if(*str < '0' || *str > '9')
         return 0;
      str++;
   }
   return 1;
}

/*Now global variable n will be on Heap so it is accessible all the processes i.e. consume and produce*/

shellcmd xsh_prodcons(int nargs, char *args[])
{
        //local varible to hold count
        int count = 2000;

        //Argument verifications and validations
        if(nargs > 2){
                printf("Too many arguments.\nType prodcons --help for details.\n");
                return 0;
```

```
	}

	if (nargs == 2) {
		if(strncmp(args[1], "--help", 7) == 0){
			printf("Use: %s [file...]\n", args[0]);
			printf("Description:\n");
			printf("\tProducer-Consumer problem demo\n");
			printf("\tdisplays garbage data\n");
			printf("\t--help\t display this help and exit\n");
			printf("Arguments:\n");
			printf("\tmax value of shared variable (integer){default value is 2000}\n");
			return 0;
		}

		//int arg= (atoi)(args[1]);
		//check, if command line arg is integer or not
		//if(arg == 0){
		if(!isNumeric(args[1])){
			printf("Invalid argument.\nType prodcons --help for details.\n");
			return 0;
		}
		count = (atoi)(args[1]);
	}



	//reset shared variable
	n=0;

	//create and schedule producer thread
	resume( create(producer, 1024, 20, "producer", 1, count) );

	//create and schedule consumer thread
	resume( create(consumer, 1024, 20, "consumer", 1, count) );

	return 0;
}
```

Producer-Consumer with one semaphore:

If this synchronization is tried using one semaphore, we will get either busy waiting or the consumer will consume only the last produced value.

Initially, the ready queue has producer and consumer processes. Single semaphore is created using semcreate() and it can be initialized with either count 1 or 0.

Semaphore initialized with count 0:

If the semaphore is initialized with 0, in producer.c when the wait (semaphore) is issued the count reduces to -1. Thus, the count becomes -1 and the critical section is skipped and the producer is blocked. resched () in wait() now starts the consumer process which in turn issues wait (semaphore) which further reduces the value of semaphore to -2. Thus both the processes are now blocked in wait queue and the critical section is never executed. This is called busy waiting.

Semaphore initialized with count 1:

If the semaphore is initialized with 1, in producer.c when the wait (semaphore) is issued the count reduces to 0. The critical section of producer is executed and signal (semaphore) is issued which checks if semaphore count is less than 0 which in our case is not. Thus it just increments the semaphore value and control remains with the producer which is executed again. Thus the producer executes till all the values are produced. Once all the values are produced, the control comes out of the loop. The semaphore value is 1 at the end. Now, resched() calls consumer where wait (semaphore) reduces the count to zero. Critical section of consumer is executed. Here the last produced value is consumed. signal (semaphore) increments count to 1. But as the counter is already at its max value, the loop breaks.

Thus, in this case all the values are first produced and only the last value is consumed by the consumer.