Assignment 3- (FA15 BL CSCI P536 36187)

Rohit Nair – Ronair, Abhijit Karanjkar – Aykaranj

1. How exactly is synchronization achieved using semaphore in our assignment?

Answer: We use 2 semaphores to implement the synchronized producer consumer problem in our assignment. In the calling class prodcons.c, we initialize 2 semaphores viz, `produced = semcreate(0)` and `consumed = semcreate(1)`. And passes it to producer.c and consumer.c respectively (along with the count if passed as argument by the user else the default count). The processor first starts executing Producer process. Here, the producer issues a `wait(consumed)` to enter its critical section so that the consumer cannot execute any instruction as long as explicitly suggested to. In this section, the producer executes any critical statements (here, it is simply displaying and incrementing the counter value). Then the producer issues the command - `signal(produced)` which suggests to the consumer that the producer is now out of its critical section. Consumer runs in a similar fashion: first, asking the producer to wait, next executing critical statements without interruptions from the producer (here, it just displays the value of count) and finally signaling the producer that the consumer is out of its critical section.

- Thus, thread synchronization is achieved

2. Can the above synchronization be achieved with just one semaphore? Why or why not?

Answer: No

Problems: When we execute the producer-consumer problem using a single semaphore, it leads to 2 issues:

i) Busy waiting:

ii) Consumer consumes only last produced value

Explanation:

>> ready_queue = [producer, consumer]

a. <u>If we initialize the semaphore with 0:</u>

Here, when the producer executes wait(sem), sem becomes -1. Since sem is negative, producer process gets blocked. Now resched() goes to the ready queue and transfers control to consumer(). Consumer executes wait(sem), sem becomes -2. Since sem is negative, consumer process gets blocked

>> Both processes are never resume

b. <u>If we initialize the semaphore with 1:</u>

Here, when the producer executes wait(sem), it makes sem=0. Since sem is non-negative, wait() returns. Critical section of producer() is executed. signal(sem) increments sem making it 1. Since there's no process in its queue, it goes into the next iteration. wait(sem) again makes sem=0 and this goes on till all values have been produced. Once count values are produced, loop breaks. Note that sem is 1 at the end. Now, resched() calls consumer() where wait(sem) makes sem=0. Since sem is non-negative, wait() returns. Critical section of consumer() is executed, consuming the latest produced value. signal(sem) increments sem making it 1. Since there's no process in its queue, it goes into the next iteration. Now count is reached and loop breaks

- Thus producer() doesn't leave control till all values are produced and consumer() consumes only the latest produced value

For default count (= 10):

```
xsh $ prodcons
Taking the default no of args as 10
Produced: 1
xsh $ Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5
Produced: 6
Consumed: 6
Produced: 7
Consumed: 7
Produced: 8
Consumed: 8
Produced: 9
Consumed: 9
Produced: 10
Consumed: 10
```

For count = 4:

```
xsh $ prodcons 4
Produced: 1
xsh $ Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
```

For count = 13:

```
xsh $ prodcons 13
Produced: 1
xsh $ Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5
Produced: 6
Consumed: 6
Produced: 7
Consumed: 7
Produced: 8
Consumed: 8
Produced: 9
Consumed: 9
Produced: 10
Consumed: 10
Produced: 11
Consumed: 11
Produced: 12
Consumed: 12
Produced: 13
Consumed: 13
```

3. Program Functions :

a. include/**prodcons.h**

```
#include<xinu.h>
#include <stddef.h>
#include <stdio.h>

/*Global variable for producer consumer*/
extern int n; /*this is just declaration*/

/* Declare the required semaphores */
extern sid32 consumed, produced;

/*function Prototype*/
//void consumer(int count);
//void producer(int count);
void producer(sid32, sid32, int);
void consumer(sid32, sid32, int);
```

b. shell/**xsh_prodcons.c**

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

int n = 1;                  //Definition for global variable
'n'

//Defination for semaphores
sid32 produced, consumed;

//Checks whether the argument passed is a number
int isNumber(const char *val)
{
     while(*val != '\0')
     {
          if(*val < '0' || *val > '9')
               return 0;
          val++;
     }
     return 1;
}

shellcmd xsh_prodcons(int nargs, char *args[])
{
     n=1;
     int count=10;  //local varible to hold count
```

```
     //Argument verifications and validations
     if(nargs<2)
     {
          printf("Taking the default no of args as 10\n");
          produced = semcreate(0);
          consumed = semcreate(1);
          //create the process producer and consumer and put
them in ready queue.
          resume( create(consumer, 1024, 20, "consumer", 3,
produced, consumed, count) );
          resume( create(producer, 1024, 20, "producer", 3,
produced, consumed, count) );
/*        if (n>=count)
          if(n==7)
          {
               semdelete(produced);
               semdelete(consumed);
          }
*/   }
     else
          if(nargs>2)
               printf("Too many arguments!\n\n");
          else //nargs==2
               if(strncmp(args[1], "--help", 7) == 0)
                    printf("\nThis command executes producer
& consumer with Semapho!\n");
               else
                    if(isNumber(args[1]) == 1)
                    {
                         count = atoi(args[1]);
                         /*Initialise semaphores*/
                         produced = semcreate(0);
                         consumed = semcreate(1);
                         //create the process producer and
consumer and put them in ready queue.
                         resume( create(consumer, 1024, 20,
"consumer", 3, produced, consumed, count) );
                         resume( create(producer, 1024, 20,
"producer", 3, produced, consumed, count) );
/*                       //if (n>=count)
                         if(n==7)
                         {
                              semdelete(produced);
                              semdelete(consumed);
                         }
*/                  }
                    else
```

```
                                printf("Please provide an integer
argument\n");
}
```

c. apps/**produce.c**

```
#include <prodcons.h>

void producer(sid32 produced, sid32 consumed, int count)
{
      //Code to produce values less than equal to count,
      //produced value should get assigned to global
variable 'n'.

      //Use system call wait() and signal() with predefined
semaphores produced and consumed to synchronize critical
section
      //Code to produce values less than equal to count,
      //produced value should get assigned to global
variable 'n'.
      //print produced value e.g. produced : 8

     while(n<=count)
     {
          wait(consumed);
          printf("Produced: %d\n",n);
          signal(produced);
          if (n==count)
               break;
          else
               n++;
     }
}
```

d. apps/**consume.c**

```
#include <prodcons.h>

void consumer(sid32 produced, sid32 consumed, int count)
{
     //Code to consume values of global variable 'n' until
the value of n is less than or equal to count

     while(1)
```

```
        {
                wait(produced);
                printf("Consumed: %d\n", n);
                signal(consumed);
/*              if (n==count)
                {
                        semdelete(produced);
                        semdelete(consumed);
                        break;
                }
                else
                        signal(consumed);*/
        }
        semdelete(produced);
        semdelete(consumed);
}
```

4. Teamwork:

Sr. No	Abhijit Karanjkar(aykaranj)	Rohit Nair(ronair)
1.	Modified xsh_prodcons	Studied the working of semaphores
2.	Modified Producer.c	Modified Consumer.c
3.	Tested output for different count values	Implemented Field validation
4.	Discussed with Rohit and prepared answer for Q2	Discussed with Abhijit and prepared answer for Q1