How exactly synchronization is achieved using semaphore in our assignment?

Synchronization is achieved between Producer and Consumer with Producer producing before the values are consumed in Consumer Process. Two binary semaphores have been used in the implementation .The semaphore consumed is initialized to the value of 1, so wait will not block when first time it is called in Producer, so Producer will produce the value of n, which is a shared variable between the two processes and hence forms a part of the critical section. In our case it is getting incremented by 1.However the produced is initialized to value 0 and hence it gets blocked in Consumer for the first time .In effect the Consumer waits for the semaphore produced before printing (or consuming) the values of the global variable n . When the execution proceeds the producer and consumer coordinate and consumer prints all values of n from 1 to count or 2000 in case the value of count is not given.

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

The above synchronization can not be achieved using one semaphore as the use of single binary semaphore ensures mutual exclusion but it would not ensure synchronization between the two processes as any process can enter the critical section at any point . There is not a precedence relationship between the two processes. The implementation for the single semaphore is given below.

Pseudo Code

PRODUCER

```
void producer(int count)
{
int i;
        for(i=1 ; i<=count ; i++ )
        {
                wait(m);
                if(flag==0)
                {
```

```
                n++;
                printf("\nProduced :%d ",n);
                flag=1;
                printf("\n %d",flag);
        }
        signal(m);
    }
}
```

CONSUMER

```
void consumer(int count)
{
int i;
    for(i=1; i<=count; i++)
    {
        wait(m);
        if(flag==1)
        {
            printf("\nConsumed: %d ", n);
            flag=0;
            printf("\n %d",flag);
        }

        else
        {
```

```
                printf("\nxyz");
        }
        signal(m);

    }
}
```

CODE WITH 2 SEMAPHORES

PRODUCER

```
void producer(int count)
{
    int i;
    for( i=1 ; i<=count ; i++ )
    {
        wait(consumed);
        n++;
        printf("Produced :%d \n",n);
        signal(produced);
    }
}
```

CONSUMER

```
void consumer(int count)
{
    int i;
    for( i=1 ; i<=count ; i++ )
    {
        wait(produced);
```

```
                printf("Consumed: %d \n", n);
                signal(consumed);
        }
        semdelete(consumed);
        semdelete(produced);
}
```

xsh_prodcons.c

```
#include<prodcons.h>
int n;
int flag;
sid32 produced, consumed;
//sid32 m;
shellcmd xsh_prodcons(int nargs, char *args[])
{
n=0;
//flag=0;
int count;
        if(nargs==2 && strncmp(args[1], "--help", 7) == 0) {
                printf("Usage: %s\n\n", args[0]);
                printf("Description:\n");
                printf("\tProducer Consumer Problem\n");
                printf("Give only one argument\n");
                printf("Don't give more than one argument\n");
                return 0;
```

```
	}


	if(nargs>2){
		printf("Too many arguments.\n");
		return 0;
	}


	if(nargs==2){
		count=atoi(args[1]);
	}
	else
		count=2000;
	consumed = semcreate(1);
	produced = semcreate(0);
	//m=semcreate(1);
	resume(create(producer, 1024, 20, "producer", 1, count));
	resume(create(consumer, 1024, 20, "consumer", 1, count));
	return 0;
}
```

prodcons.h

```
#include <xinu.h>


extern sid32 produced, consumed;
```

```
extern sid32 m;

extern int n;

extern int flag;


void producer(int count);

void consumer(int count);
```

TASK DIVISION

Producer Code : Rudrani

Consumer Code : Kushal

Report : Kushal, Rudrani