P536: ADVANCED OPERATING SYSTEMS

ASSIGNMENT 3: REPORT

TEAM: ROHIT PATIL, SAMEEKSHA VAITY

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

Answer: Semaphores are usually used when two processes engage in mutual exclusion and have to access a shared resource at a same given time.

In our assignment the producer file functions to produce values which have to be consumed by the consumer function in a serialized/ synchronized manner. For having the desired output we use the concept of semaphores.

In our program we create two semaphores '*produced'* and '*consumed'* in the xsh_prodcons.c file. Initially the semaphore are initialized such that *produced = semcreate(0)* and consumed *= semcreate(1)*. The processor starts executing the Producer process thread first. The Producer process first calls a wait on the semaphore '*consumed'* before it enters the critical section. Critical section is the lines of code that are to be shared by two or more processes and must be isolated and executed by a single process at a given time. Once the wait on semaphore '*consumed'* is called, it would decrement the semaphore '*consumed'* value by 1 and check if it is less than 0, if true it will add the current process to the waiting queue of the '*consumed*' semaphore. After which it will call the processes in the ready queue by the *resched()* function which in our case would call the consumer process. If the semaphore '*consumed*' value is not less than 0 it will continue the normal execution of the critical section of the program i.e it will increment the counter value in our program by 1 and then call the signal function on semaphore '*produced*'. The signal function will then check If the semaphore value is less than 0, if yes then it will dequeue the waiting processes of the semaphore '*produced*' and put them in the ready queue of the processes. Otherwise it would just increment the semaphore '*produced'* value by 1.

Similarly the consumer process will call a wait on semaphore '*produced*' before entering the critical section and then will perform signal operation on the semaphore '*consumed*' after completing the execution of the critical section. So this way we enforce an order in the execution of concurrent programs so they execute in a synchronized manner producing the desired result.

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

Answer: No. The desired synchronized output cannot be achieved with a single semaphore.
For implementing single semaphore concept we can initialize the semaphore count value to either 1 or 0.' Let's consider both the cases:

1. When semaphore count is =0 : [*singleSem = semcreate(0)*]

In this case when the Producer process call wait on the semaphore *singleSem*, it will decrement the semaphore *singleSem* value by 1 and add the producer process to its waiting queue of processes and will *resched()* that is call the consumer process for execution.
Then the consumer process has wait on the same semaphore i.e. *singleSem* which will further be decremented by 1 causing it to add the Consumer process to its waiting queue and calls the *resched()* function for calling the ready processes but since now we have both the Producer and Consumer processes waiting on the same semaphore they will never be ready and the system will enter the busy waiting state.
Thus we will never get the desired output this way.

2. When semaphore count is = 1: [*singleSem = semcreate(1)*]

In this case when the Producer process calls wait on the semaphore *singleSem*, it will decrement the semaphore *singleSem* value by 1 and since it is not less than 0 it would continue the execution of the critical section of the program. Once it is out of the critical section it would signal on the same semaphore '*singleSem*' incrementing its value by 1 and again since the value is not less than 0 it will continue to its next iteration. This goes on until the

producer has produced all the values i.e. completed its execution after which the system will call the next ready process i.e. Consumer process.

Then when the consumer process starts with its execution it will call the wait on semaphore '*singleSem*', decrementing it by 1 and since it is not less than 0 it will continue normal execution of the critical section of the program i.e. printing the last produced value. After which it calls the signal function on the same semaphore '*singleSem*' incrementing its value by 1 and since it is not less than 0 it will continue executing until Consumer process has completed all its iterations.

In this approach the producer then has finished producing all the values before the consumer could even start consuming and by the time the Consumer process executes since the producer process has already finished its execution it just gets the last produced value and keeps printing it.

FUNCTIONS USED IN THE PROJECT:

1. Producer Function:

```
void producer(int count)
{
int i;
//Code to produce values less than equal to count
//produced value should get assigned to global variable 'n'.
//print produced value
for( i = 1;i<=count;i++)
{
wait(consumed);
n=i;
printf("Produced value : %d\n", n);
signal(produced);
}
}
```

2. Consumer Function:

```
void consumer(int count)
 {
   int j ;
    //Code to consume values of global variable 'n' until the value of n is less than or equal to count
    //print consumed value
   for(j = 1;j<=count;j++)
```

```
{
wait(produced);
printf("Consumed Value :%d\n",n);
signal(consumed);
}
printf("Ending the program, semaphores deleted..");
semdelete(produced);
semdelete(consumed);
}
```

3. Check if user input is a valid number

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

TASKS OF THE GROUP MEMBERS:

Task	Implemented By
Modifications in Producer Function.	Sameeksha Vaity
Modifications in Consumer Function.	Rohit Patil
Modifications in xsh_prodcons.c	Sameeksha Vaity
Modifications for Exception Handling	Rohit Patil
Question 1: How is synchronization achieved in our assignment?	Sameeksha Vaity/Rohit Patil
Tried to implement synchronization using single semaphore.	Sameeksha Vaity/Rohit Patil
Question 2: Can the above synchronization be achieved with just one semaphore?	Sameeksha Vaity/Rohit Patil