Q1. How exactly is synchronization achieved using semaphore in our assignment?

Answer

```
sid32 produced,consumed;
```

Consumer Code

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

Producer Code

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

In our assignment we have used 2 semaphores in order to achieve synchronization.

a. produced – initial value 0
b. consumed – initial value 1

Now let's assume both the processes (producer and consumer) are activated simultaneously.

Consumer process execution

Consumer process will issue a wait call on *produced semaphore*. Wait call will decrement the value by 1. Since the initial value is 0, the current value would be -1. As the value is negative, the consumer process will be blocked. The process will stay in blocked state until some process signals the *produced semaphore* thereby incrementing the value to some non-negative value.

Producer process execution

Producer process will issue a wait call on *consumed semaphore*. Wait call will decrement the value by 1. Since the initial value is 1, the current value would be 0. As the value is non-negative, the producer process would be allowed entry in the critical section. The process executes the critical section and then executes the signal call on *produced semaphore*, incrementing the value to 0 (-1 made by consumer process).

Now if the operating system passes the control back to producer process, it will again issue a wait call on *consumed semaphore*. But this time the value would be decremented to -1 (since the current value is 0) so the producer would be in blocked state.

Consumer process execution

As the value of *produced semaphore* is incremented to non-negative by producer process, consumer process is granted an access in the critical section. The consumer consumes the value produced by producer and then calls signal on *consumed semaphore* incrementing the value to 0 and granting the access of critical section to producer.

Conclusion

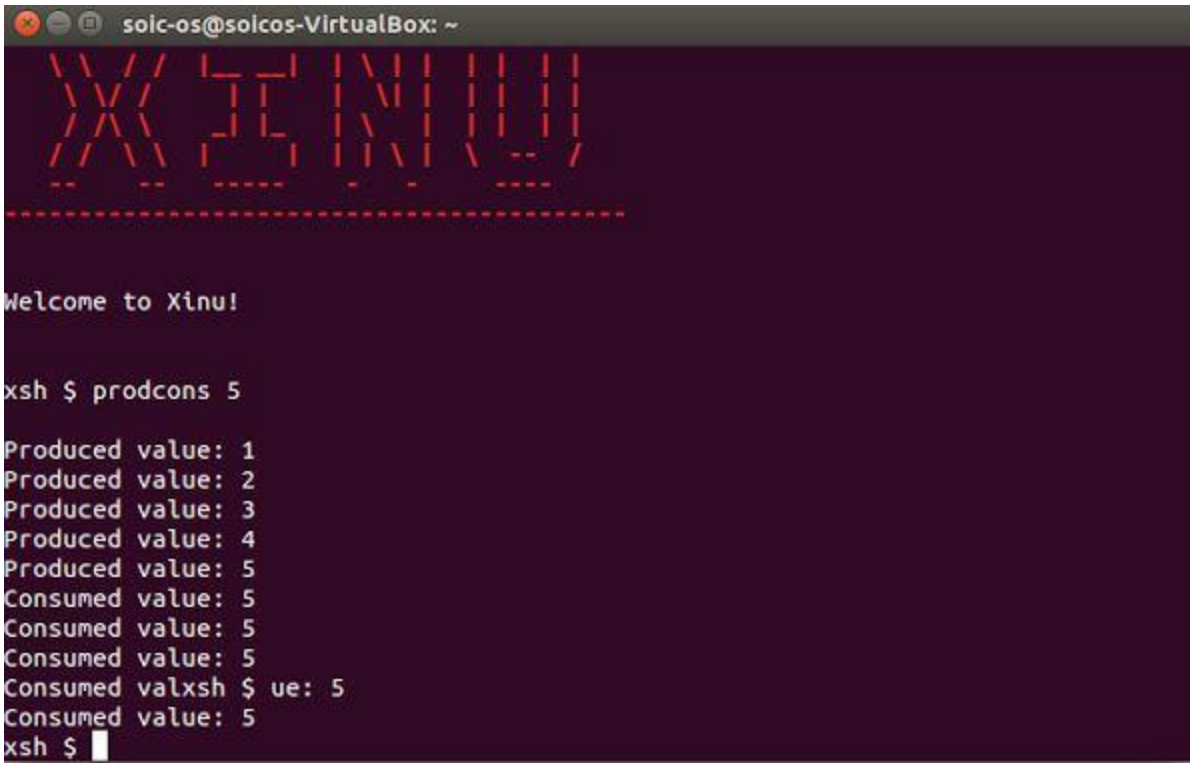
In this way producer and consumer are granted access alternatively to the critical section i.e. producer will produce one value and it won't be able to produce the next value until the previous value is consumed by consumer. Similarly, consumer won't be able to consume any value until producer produces them.

Q2. Can the above synchronization be achieved with just one semaphore? Why or why not?

Answer

The above synchronization cannot be achieved using just one semaphore. Let's assume that we just have one semaphore with initial value as 1.
Suppose that operating system firsts gives control to producer. Producer would call wait on the semaphore, decrementing the value of the semaphore. Since the value is non-negative, producer is granted access to the critical section. Now no other process can enter the critical section. Producer will produce a value and send signal call on the semaphore, incrementing the value of semaphore. Now other waiting processes (i.e. producer or consumer) may access the critical section. The operating system would decide which process can access the critical section next. It may so happen that the operating system may give the control back to the producer. The producer would produce new value even before the consumer has consumed the old value, over writing the old value.
Also we need to make sure that the code for producer is invoked before the code for consumer is invoked.

Screenshot of output with one semaphore



Contribution:
Producer code: Anand Nahar.
Consumer code: Pratik Patel.