**Question 1:** Does your program output any garbage value? If yes, why?

**Answer:**

**No**, the program does not output any garbage value. This is so, because the values that the *consumer program* prints are the values of the global variable *n,* whichthe *producer program* manages in its code. There is just a loss of synchronicity between the *producer* and the *consumer* and as such, garbage values are not printed.

However, when running it for values of count more than 100, it produced some weird output as below:



This is not *garbage* but not what is expected.

**Question 2:** Are all the produced values getting consumed? Check your program for a small count like 20.

**Answer:**

**No,** all the produced values are not getting consumed. The *producer* *program* produces all the values from *0* to *count*, however, the *consumer program* just prints the last value (*19*, in this case as the iterator variable *i* starts from *0*). This is because, the *consumer program* is concerned with printing the value of n, which involves formatting the output (due to the printf() function) and getting the control of the console for printing. All these instructions are time consuming; and as such the processor switches back to the producer program which in turn starts executing. Thus, although the values from 0 to 19 are generated, they are not consumed by the consumer program and they are effectively lost.

**Question 3:** Write down all the functions in the project.

**Answer:**

1. Create():

Function Prototype: int create(caddr, ssize, prio, name, nargs[,argument]\*)

This function creates a new process which will begin execution at the memory address *caddr*, The stack allocated is of *ssize* bytes and the initial priority of the process will be *prio.* The process would be identified by thename *name*. Also, *nargs* here is the number of arguments that can be passed to the create() function. Also, this is followed by the optional actual arguments, as indicated in the function prototype above. And the return type is *int*.

1. Resume():

Function Prototype: int resume(pid)

*Resume* takes a process *pid* out of sleep and resumes it by making it ready and puts them in the ready queue. Also, the return type of this function is *int*.

1. Produce():

Function Prototype: void produce(int count)

This function takes the value of *count* as an argument, and generates *count* values. These produced values are then assigned to the global variable *n*.

1. Consume():

Function Prototype: void consume(int count)

This function takes the value of *count* as an argument, and consumes the values that the producer has produced. It in turn, prints out (I.e., consumes) the values of the global variable *n.*

**Question 4:** Note the tasks of the group members. Who developed the various components?

**Answer:**

Sagar and I worked together as a group to create the shell command, coded *xsh\_prodcons.c* along with its error handling functionality as well as created the *prodcons.h* header file which was used to declare the global variable *n* and the *producer* and *consumer* methods. Then, Sagar coded the logic for the *produce.c* file, while I did it for the *consume.c* file. Answers above were written together post discussion.