AOS Assignment 2

Producer-Consumer Problem:

The producer-consumer problem is an example of multi-process synchronization. The problem describes two processes, the producer and the consumer, who work on the same data where the producer edits the data and consumer utilizes the edited data.

In our code, the consumer process is displaying values i.e. values of global variable 'n' which is being simultaneously incremented by producer process.

The consumer process has to format and write the output, which requires huge machine instructions, the consumer has to wait for the output device to display all the characters. When consumer waits, the producer loops until the count is maximum. As a result, all produced values are not displayed by the consumer.

Printf() synchronization:

Both printf() statements of producer and consumer are being executed simultaneously on one output stream. Because of this we get garbage values in output. There is no guarantee of ordering with respect to displayed data. Both the producer and consumer actions are not synchronized and are trying to generate output at the same time.

Functions in the code:

1. **shellcmd xsh_prodcons(int nargs, char *args[])** - Ajinkya and Amruta

   In this function, we are assigning a value to count. The value is either hard-coded or accepted from the user. Producer and Consumer processes are created in this function and are put in ready queue.

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

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

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

```
    str++;
  }
  return 1;
}

/*Now global variable n will be on Heap so it is accessible all the processes i.e. consume and produce*/

shellcmd xsh_prodcons(int nargs, char *args[])
{
	//local varible to hold count
	int count = 2000;

	//Argument verifications and validations
	if(nargs > 2){
		printf("Too many arguments.\nType prodcons --help for details.\n");
		return 0;
	}

	if (nargs == 2) {
		if(strncmp(args[1], "--help", 7) == 0){
			printf("Use: %s [file...]\n", args[0]);
			printf("Description:\n");
			printf("\tProducer-Consumer problem demo\n");
			printf("\tdisplays garbage data\n");
			printf("\t--help\t display this help and exit\n");
			printf("Arguments:\n");
			printf("\tmax value of shared variable (integer){default value is 2000}\n");
			return 0;
		}

		//int arg= (atoi)(args[1]);
		//check, if command line arg is integer or not
		//if(arg == 0){
		if(!isNumeric(args[1])){
			printf("Invalid argument.\nType prodcons --help for details.\n");
			return 0;
		}
		count = (atoi)(args[1]);
	}
```

```
	//reset shared variable
	n=0;

	//create and schedule producer thread
	resume( create(producer, 1024, 20, "producer", 1, count) );

	//create and schedule consumer thread
	resume( create(consumer, 1024, 20, "consumer", 1, count) );

	return 0;
}
```

2. **void producer(int count) –** Ajinkya

The producer function increments the value of global variable ‘n’ count number of times.

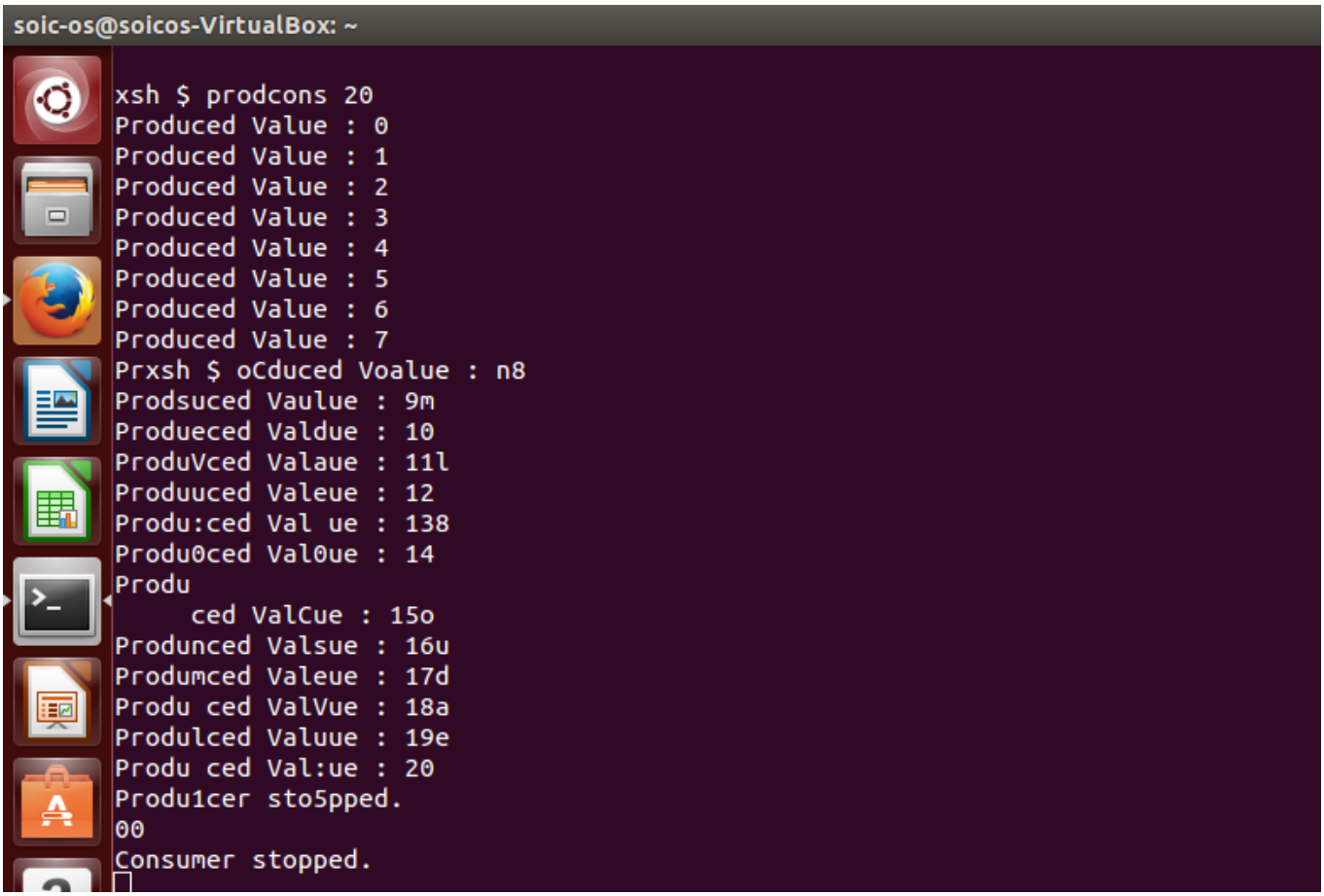
```
#include <prodcons.h>
#include <string.h>
#include <stdlib.h>

/*
* Assign new value to shared variable
*/
 void producer(int count)
 {
	while(n<=count){

		//display shared varible
		printf("Produced Value : %d\n", n);

		//increment value of shared variable if less than count
		if(n==count){
			break;
		}else{
			n++;
		}
	}

	printf("Producer stopped.\n");
	return;
 }
```

3. **void consumer(int count) -** Amruta

The consumer just displays the value of global variable 'n'.

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

/*
* Consume value generated by producer
* Display value of shared variable 'n'
* Does not perform any operations on 'n'
   n : shared variable
*/
 void consumer(int count)
 {

        while(n<=count){

                //display shared variable
                //n*100 is calculated to differentiate consumed value from produced value
                printf("Consumed Value : %d\n", n*100);

                //if n reaches limit break the loop and exit
                if(n == count) {
                        break;
                }

        }

        printf("Consumer stopped.\n");
        return;

 }
```

Output:-



Observations:

From the output we observe that, since the producer and consumer are not synchronized, both are trying to print to the output stream at the same time. As a result, we get garbage values in the output.

Also, it is possible that consumer process starts even before the producer. The resume function puts the process in ready state. It is uncertain which process will get the CPU first. Thus, either the producer or the consumer will be executed first.