PROGRAM – 10

AIM: Program to design consumer producer problem using semaphores.

INTRODUCTION: Semaphore is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multi processing environment.
The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1. only.
In computing, the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.
The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time

C PROGRAM :
Producer Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/sem.h>

#define MUTEX 0
#define FULL 1
#define EMPTY 2

union semun{
        int val;
        struct semid_ds *buf;
        ushort *array;
        struct seminfo *__buf;
};

//create a semaphore
int createSem(int semNo){
        int id;
        key_t key = ftok("os10",1);

        id = semget(key,semNo,IPC_CREAT | 0660);
```

```
	if (id<0){
		perror("semget: ");
		exit(1);
	}

	return id;
}

//initialize the semaphore
void initializeSem(int semId, int semNo, int value){
	union semun un;
	un.val = value ;

	if (semctl(semId,semNo,SETVAL, un)<0){
		perror("semctl: ");
		exit(1);
	}

}

//wait
void wait(int semId,int semNo){
	struct sembuf buff = {semNo, -1, 0};

	if (semop(semId,&buff,1) < 0){
		perror("semop: ");
		exit(1);
	}
}

//signal
void signal(int semId, int semNo){
	struct sembuf buff = {semNo, 1, 0};

	if (semop(semId,&buff,1) < 0){
		perror("semop: ");
		exit(1);
	}
}

//producer
```

```
int main(){
	char ch ;
	int semid = createSem(3);
	initializeSem(semid, MUTEX, 1);
	initializeSem(semid, FULL, 0);
	initializeSem(semid, EMPTY, 10);

	int i = 0;

	do{
		//entry section
		wait(semid, EMPTY);
		wait(semid, MUTEX);

		//***critical section***
		FILE *buffer = fopen("buffer.txt","a+");

		//produce
		i = i + 1;

		fprintf(buffer, "Product %d\n",i );

		fclose(buffer);

		printf("Product %d is produced.\n",i);

		//exit section
		signal(semid, MUTEX);
		signal(semid, FULL);

		printf("Want to Produce more ?(Y or N): ");
		scanf("%c",&ch);
		getchar();
	}while(ch=='Y'||ch=='y');


	return 0;
}
```

Consumer Code :

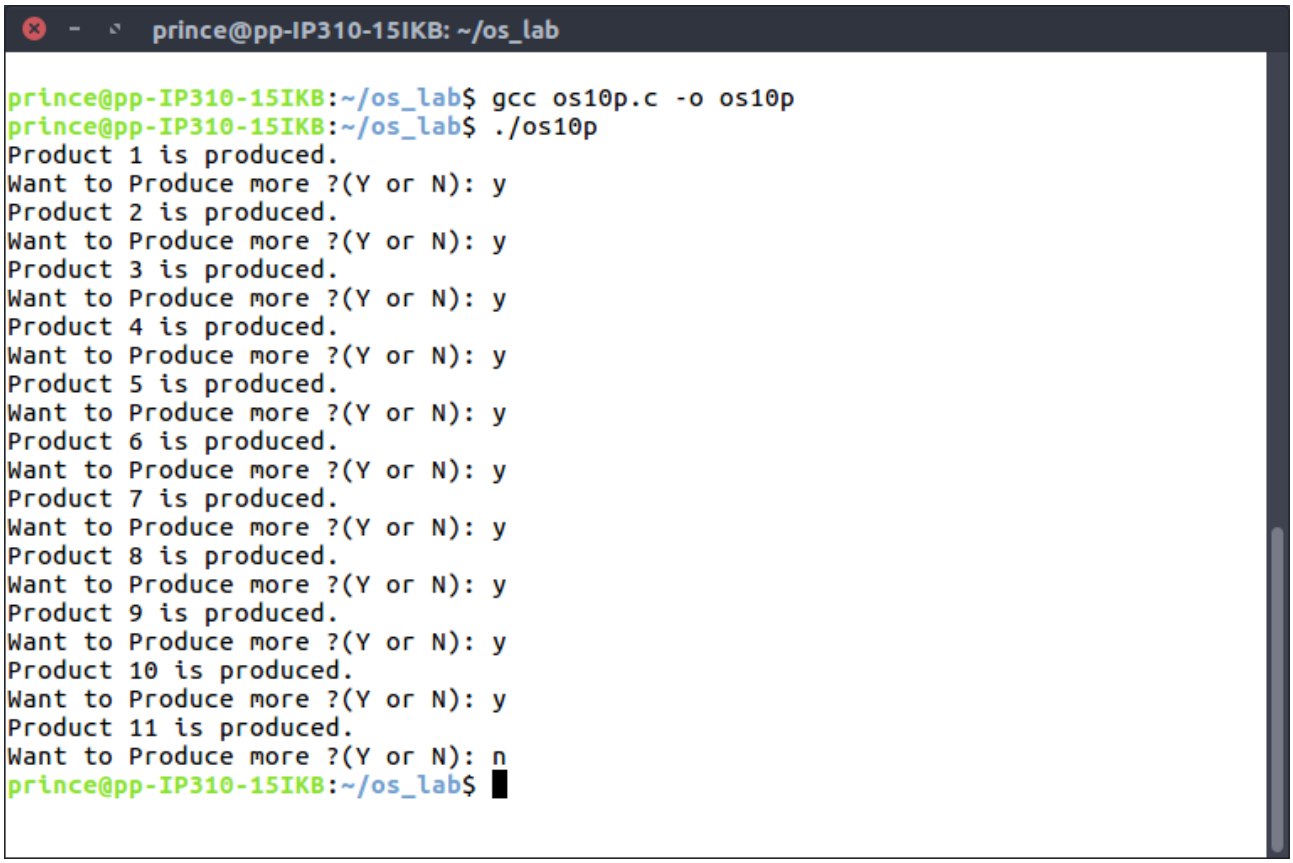
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

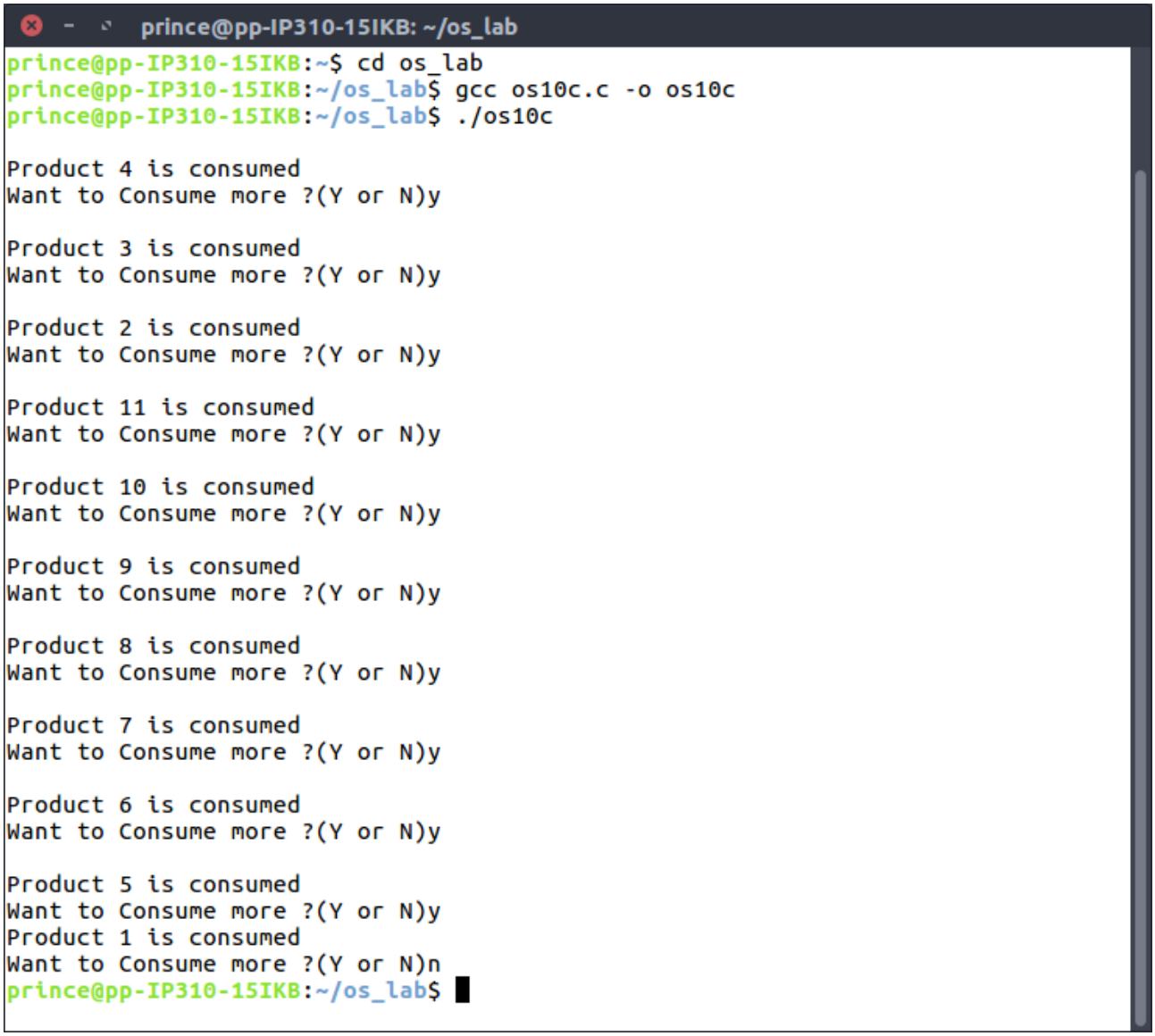
```
#include <sys/sem.h>

#define MUTEX 0
#define FULL 1
#define EMPTY 2

//create a semaphore
int createSem(int semNo){
	int id;
	key_t key = ftok("os10",1);
	int semFlag = IPC_CREAT | 0660 ;

	id = semget(key,semNo,semFlag);

	if (id<0){
		perror("semget: ");
		exit(1);
	}

	return id;
}

//wait
void wait(int semId,int semNo){
	struct sembuf buff = {semNo, -1, 0};

	if (semop(semId,&buff,1) < 0){
		perror("semop: ");
		exit(2);
	}
}

//signal
void signal(int semId, int semNo){
	struct sembuf buff = {semNo, 1, 0};

	if (semop(semId,&buff,1) < 0){
		perror("semop: ");
		exit(2);
	}
}
```

```
//Consume product
void consumeProduct(char *fileName,char str[]){
	FILE *buffer = fopen(fileName,"a+");
	FILE *temp = fopen("temp.txt","w");

	char ch;
	int len;
	int line = 0,t;

	while((ch = fgetc(buffer)) != EOF){
		if (ch == '\n'){
			line ++;
		}
	}
	line = line - 1;
	rewind(buffer);

	while((ch = fgetc(buffer)) != EOF){
		if (ch == '\n'){
			t++;
		}
		if (t!=line){
			fputc(ch,temp);
		}else{
			len = strlen(str);
			str[len] = ch;
			str[len+1] = '\0';

		}
	}

	fclose(buffer);
	fclose(temp);
	remove(fileName);
	rename("temp.txt",fileName);
}

//consumer
int main(){
	char ch;
	char str[25] = {'\0'};
	int semid = createSem(3);
```

```
        do{
                //entry section
                wait(semid, FULL);
                wait(semid, MUTEX);

                //***critical section***
                str[0] = '\0';
                consumeProduct("buffer.txt",str);

                printf("%s is consumed\n",str);
                //exit section
                signal(semid, MUTEX);
                signal(semid, EMPTY);

                printf("Want to Consume more ?(Y or N)");
                scanf("%c",&ch);
                getchar();
        }while(ch=='Y'||ch=='y');
        return 0;
}
```

OUTPUT:





LEARNING OUTCOMES: We learnt the the usage and implementation of semaphore and conditions at which consumer and producer come into action.