SSN COLLEGE OF ENGINEERING, KALAVAKKAM
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

Batch: 2018-22 **Academic Year: 2019-20**
Class: CSE B

Lab Exercise 6: Implementation of Producer/Consumer Problem using Semaphores

Study the following system calls

Semaphores – sem_init, sem_wait, sem_post, sem_destroy – POSIX, pthread

semget , semctl, semop (BSD – for your understanding)

Shared memory - shmget, shmat, shmdt, shmctl

Assignment 1:

Aim:

To write a C program to create parent/child processes to implement the producer/consumer problem using semaphores in pthread library.

Procedure:

1. Create a Shared memory for buffer and semaphores - empty, full, mutex

2. Create a parent and a child process one acting as a producer and the other consumer.

3. In the producer process, produce an item, place it in the buffer. Increment full and decrement empty using wait and signal operations appropriately.

4. In the consumer process, consume an item from the buffer and display it on the terminal. Increment empty and decrement full using wait and signal operations appropriately.

5. Compile the sample program with pthread library

cc prg.c - lpthread

Assignment 2:

Modify the program as separate client / server process programs to generate ‘N’ random numbers in producer and write them into shared memory. Consumer process should read them from shared memory and display them in terminal

Sample Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <semaphore.h>
#include <pthread.h> // for semaphore operations sem_init,sem_wait,sem_post
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <sys/errno.h>
#include <sys/types.h>
extern int errno;
#define SIZE 10 /* size of the shared buffer*/
#define VARSIZE 1 /* size of shared variable=1byte*/
#define INPUTSIZE 20
#define SHMPERM 0666 /* shared memory permissions */
int segid; /* id for shared memory bufer */
int empty_id;
int full_id;
int mutex_id;
char * buff;
char * input_string;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int p=0,c=0;
//
// Producer function
//
void produce()
{
        int i=0;
        while (1)
        {
                if(i>=strlen(input_string))
                {
                        printf("\n Producer %d exited \n",getpid());
                        wait(NULL);
                        exit(1);
                }
                printf("\nProducer %d trying to aquire Semaphore Empty \n",getpid());
                sem_wait(empty);
                printf("\nProducer %d successfully aquired Semaphore Empty \n",getpid());
                printf("\nProducer %d trying to aquire Semaphore Mutex \n",getpid());
                sem_wait(mutex);
                printf("\nProducer %d successfully aquired Semaphore Mutex \n",getpid());
```

```
                buff[p]=input_string[i];
                printf("\nProducer %d Produced Item [ %c ] \n",getpid(),input_string[i]);
                i++;
                p++;
                printf("\nItems in Buffer %d \n",p);
                sem_post(mutex);
                printf("\nProducer %d released Semaphore Mutex \n",getpid());
                sem_post(full);
                printf("\nProducer %d released Semaphore Full \n",getpid());
                sleep(2/random());
        } //while
} //producer fn
//
// Consumer function
//
void consume()
{
        int i=0;
        while (1)
                {
                if(i>=strlen(input_string))
                        {
                        printf("\n Consumer %d exited \n",getpid());
                        exit(1);
                        }
                printf("\nConsumer %d trying to aquire Semaphore Full \n",getpid());
                sem_wait(full);
                printf("\nConsumer %d successfully aquired Semaphore Full \n",getpid());
                printf("\nConsumer %d trying to aquire Semaphore Mutex \n",getpid());
                sem_wait(mutex);
                printf("\nConsumer %d successfully aquired Semaphore Mutex\n",getpid());
                printf("\nConsumer %d Consumed Item [ %c ] \n",getpid(),buff[c]);
                buff[c]=' ';
                c++;
                printf("\nItems in Buffer %d \n",strlen(input_string)c);
                i++;
                sem_post(mutex);
                printf("\nConsumer %d released Semaphore Mutex \n",getpid());
                sem_post(empty);
                printf("\nConsumer %d released Semaphore Empty \n",getpid());
                sleep(1);
        } //while
} //consumer fn
//---------------------------------------------------------
Main function
//---------------------------------------------------------
int main()
{
        int i=0;
        pid_t temp_pid;
```

```
        segid = shmget (IPC_PRIVATE, SIZE, IPC_CREAT | IPC_EXCL | SHMPERM );
        empty_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        full_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        mutex_id=shmget(IPC_PRIVATE,sizeof(sem_t),IPC_CREAT|IPC_EXCL|
        SHMPERM);
        buff = shmat( segid, (char *)0, 0 );
        empty = shmat(empty_id,(char *)0,0);
        full = shmat(full_id,(char *)0,0);
        mutex = shmat(mutex_id,(char *)0,0);
        // Initializing Semaphores Empty , Full & Mutex
        sem_init(empty,1,SIZE);
        sem_init(full,1,0);
        sem_init(mutex,1,1);
        printf("\n Main Process Started \n");
        printf("\n Enter the input string (20 characters MAX) : ");
        input_string=(char *)malloc(20);
        scanf("%s",input_string);
        printf("Entered string : %s",input_string);
        temp_pid=fork();
        if(temp_pid>0) //parent
                {
                produce();
                }
        else //child
                { consume();
                }
        shmdt(buff);
        shmdt(empty);
        shmdt(full);
        shmdt(mutex);
        shmctl(segid, IPC_RMID, NULL);
        semctl( empty_id, 0, IPC_RMID, NULL);
        semctl( full_id, 0, IPC_RMID, NULL);
        semctl( mutex_id, 0, IPC_RMID, NULL);
        sem_destroy(empty);
        sem_destroy(full);
        sem_destroy(mutex);
        printf("\n Main process exited \n\n");
        return(0);
} //main
```