EX:06 **PRODUCER CONSUMER PROBLEM USING SEMAPHORES**

-SRINITHYEE S K
-185001166

SAMPLE PROGRAM:

```
#include
<stdio.h>
                #include <stdlib.h>
                #include <string.h>
                #include <unistd.h>
                #include <semaphore.h>
                #include <pthread.h>
                #include <sys/ipc.h>
                #include <sys/shm.h>
                #include <sys/sem.h>#include <stdio.h>
                #include <stdlib.h>
                #include <string.h>
                #include <unistd.h>
                #include <semaphore.h>
                #include <pthread.h>
                #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 = 1 byte */
                #define INPUTSIZE 20
                #define SHMPERM 0666 /* shared memory permissions */
                int segid; /* ID for shared memory buffer */
                int empty_id;
                int full_id;
```

```
int mutex_id;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int p = 0, c = 0;
char *buff, *input_string;
//
// 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 acquire Semaphore Empty
\n",getpid());
        sem_wait(empty);
        printf("\nProducer %d successfully acquired Semaphore
Empty \n",getpid());
        printf("\nProducer %d trying to acquire Semaphore Mutex
\n",getpid());
        sem_wait(mutex);
        printf("\nProducer %d successfully acquired Semaphore
Mutex \n",getpid());
        buff[p]=input_string[i];
        printf("\nProducer %d Produced Item [ %c ]
\n",getpid(),input_string[i]);
        i++;
        p++;
        printf("\nItems produced: %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(random()%5);
    }
}
//
// 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 acquire Semaphore Full
\n",getpid());
        sem_wait(full);
        printf("\nConsumer %d successfully acquired Semaphore Full
\n",getpid());
        printf("\nConsumer %d trying to acquire Semaphore Mutex
\n",getpid());
        sem_wait(mutex);
        printf("\nConsumer %d successfully acquired Semaphore
Mutex\n",getpid());
        printf("\nConsumer %d Consumed Item [ %c ]
\n",getpid(),buff[c]);
        buff[c]=' ';
        c++;
        printf("\nItems consumed: %d \n",i+1);
        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);
```

```
    }
}
//---------------------------------------------------------------------
//Main function
//---------------------------------------------------------------------
int main()
{
    int i=0;
    buff = (char *)malloc(100);
    input_string = (char*)malloc(100);
    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);
}
#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 = 1 byte */
#define INPUTSIZE 20
#define SHMPERM 0666 /* shared memory permissions */
int segid; /* ID for shared memory buffer */
int empty_id;
int full_id;
int mutex_id;
sem_t *empty;
sem_t *full;
sem_t *mutex;
int p = 0, c = 0;
char *buff, *input_string;
//
// 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 acquire Semaphore Empty
\n",getpid());
       sem_wait(empty);
       printf("\nProducer %d successfully acquired Semaphore Empty
\n",getpid());
       printf("\nProducer %d trying to acquire Semaphore Mutex
\n",getpid());
       sem_wait(mutex);
       printf("\nProducer %d successfully acquired Semaphore Mutex
\n",getpid());
       buff[p]=input_string[i];
       printf("\nProducer %d Produced Item [ %c ]
\n",getpid(),input_string[i]);
       i++;
       p++;
       printf("\nItems produced: %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(random()%5);
```

```
    }
}
//
// 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 acquire Semaphore Full
\n",getpid());
        sem_wait(full);
        printf("\nConsumer %d successfully acquired Semaphore Full
\n",getpid());
        printf("\nConsumer %d trying to acquire Semaphore Mutex
\n",getpid());
        sem_wait(mutex);
        printf("\nConsumer %d successfully acquired Semaphore
Mutex\n",getpid());
        printf("\nConsumer %d Consumed Item [ %c ]
\n",getpid(),buff[c]);
        buff[c]=' ';
        c++;
        printf("\nItems consumed: %d \n",i+1);
        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);
    }
}
//---------------------------------------------------------------------
//Main function
//---------------------------------------------------------------------
int main()
{
    int i=0;
    buff = (char *)malloc(100);
    input_string = (char*)malloc(100);
    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);
}
```

PRODUCER:

```
#include
<stdio.h>
                    #include <stdlib.h>
                    #include <string.h>
                    // For semaphore operations sem_init,sem_wait,sem_post
                    #include <semaphore.h>
                    #include <pthread.h>
                    #include <unistd.h>
                    #include <sys/ipc.h>
                    #include <sys/shm.h>
                    #include <sys/sem.h>
                    #include <sys/wait.h>
                    #include <sys/errno.h>
                    #include <sys/types.h>
                    #include<unistd.h>
                    extern int errno;
                    #define SIZE 10 /* size of the shared buffer */
                    #define VARSIZE 1 /* size of shared variable = 1 byte */
                    #define INPUTSIZE 20
                    #define SHMPERM 0666 /* shared memory permissions */
                    int segid; /* ID for shared memory buffer */
                    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;
                    int main()
```

```
{
    int i = 0;
    pid_t temp_pid;
    segid = shmget(100, SIZE, IPC_CREAT | IPC_EXCL | SHMPERM );
    empty_id=shmget(101, sizeof(sem_t), IPC_CREAT | IPC_EXCL |
SHMPERM);
    full_id=shmget(102, sizeof(sem_t), IPC_CREAT | IPC_EXCL |
SHMPERM);
    mutex_id=shmget(103,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, 10);
    sem_init(full, 1, 0);
    sem_init(mutex, 1, 1);
    printf("\nProducer Process Started\n");
    while (i < 10)
    {
        int val = random()%10;
        printf("\nProducer %d trying to acquire Semaphore Empty\n",
getpid());
        sem_wait(empty);
        printf("\nProducer %d successfully acquired Semaphore
Empty\n", getpid());
        printf("\nProducer %d trying to acquire Semaphore Mutex\n",
getpid());
        sem_wait(mutex);
        printf("\nProducer %d successfully acquired Semaphore
Mutex\n", getpid());
        buff[p] = (char)(val + 48);
```

```
        printf("\nProducer %d Produced Item [%d]\n", getpid(),
val);
        i++;
        p++;
        printf("\nItems produced: %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);
    }
    shmdt(buff);
    shmdt(empty);
    shmdt(full);
    shmdt(mutex);
    printf("\nProducer Process Ended\n");
    return(0);
}
```

CONSUMER:

```
#include
<stdio.h>
                #include <stdlib.h>
                #include <string.h>
                //For semaphore operations - sem_init, sem_wait, sem_post
                #include <semaphore.h>
                #include <pthread.h>
                #include <unistd.h>
                #include <sys/ipc.h>
                #include <sys/shm.h>
                #include <sys/sem.h>
```

```
#include <sys/wait.h>
#include <sys/errno.h>
#include <sys/types.h>
#include<unistd.h>
extern int errno;
#define SIZE 10 /* size of the shared buffer */
#define VARSIZE 1 /* size of shared variable = 1 byte */
#define INPUTSIZE 20
#define SHMPERM 0666 /* shared memory permissions */
int segid; /* ID for shared memory buffer */
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;
int main()
{
   int i = 0;
   pid_t temp_pid;
   segid = shmget (100, SIZE, IPC_EXCL | SHMPERM );
   empty_id = shmget(101, sizeof(sem_t), IPC_EXCL | SHMPERM);
   full_id = shmget(102, sizeof(sem_t), IPC_EXCL | SHMPERM);
   mutex_id=shmget(103, sizeof(sem_t), 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);
    printf("\nConsumer Process Started\n");
    while (i < 10)
    {
        printf("\nConsumer %d trying to acquire Semaphore Full\n",
getpid());
        sem_wait(full);
        printf("\nConsumer %d successfully acquired Semaphore
Full\n", getpid());
        printf("\nConsumer %d trying to acquire Semaphore Mutex\n",
getpid());
        sem_wait(mutex);
        printf("\nConsumer %d successfully acquired Semaphore
Mutex\n", getpid());
        printf("\nConsumer %d Consumed Item [%c]\n", getpid(),
buff[c]);
        buff[c]=' ';
        c++;
        printf("\nItems consumed: %d\n", i+1);
        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);
    }
    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("\nConsumer Process Ended\n");
    return(0);
}
```