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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1411 - OPERATING SYSTEMS LAB

LAB EXERCISE 6

Implementation of Producer/Consumer Problem using Semaphores

Submission Date:21-04-2022

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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

Algorithm:

1) For the segment, shared memory is allotted using shmget and returned id is stored in segid
2) For the empty, shared memory is allotted using shmget and returned id is stored in empty_id
3) For the full, shared memory is allotted using shmget and returned id is stored in full_id
4) For the mutex, shared memory is allotted using shmget and returned id is stored in mutex_id
5) Attach buffer to segid, empty to empty_id, full to full_id, mutex to mutex_id
6) Initialise semaphore to empty, full and mutex
7) Get the string from user and store it in str
8) Fork the process using call fork() and store it in m_pid
9) If m_pid greater than 0 then call producer function
10) Else call consumer function
11) Detach buffer, empty, full, mutex from memory
12) Destroy all shared memory
13) Destroy semaphores

Producer function:

1) Initialise i=0
2) If I greater than string length then exit from producer
3) Else
    a) Empty semaphore acquired by producer
    b) mutex semaphore acquired by producer
    c) next character from string is written into buffer
    d) Empty semaphore released by producer
    e) mutex semaphore released by producer

Consumer function:

1) Initialise i=0
2) If I greater than string length then exit from consumer
3) Else
    a) Empty semaphore acquired by consumer
    b) mutex semaphore acquired by consumer
    c) next character from buffer is read and pointer to buffer is increased
    d) Empty semaphore released by consumer
    e) mutex semaphore released by consumer

Code:

```c
#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#define BUFSIZE 5

struct memory
{
    char buffer[BUFSIZE];
    int count;
    sem_t full;
    sem_t empty;
    sem_t mutex;
};

struct memory *shmptr;

char original[100], input_string[BUFSIZE];
int input_index = 0;
int c = 0;

void producer()
{
    do
    {
        if (shmptr->count >= BUFSIZE)
        {
            wait(NULL);
            continue;
        }

        sem_wait(&(shmptr->empty));
        sem_wait(&(shmptr->mutex));

        shmptr->buffer[shmptr->count++] = input_string[input_index++];
        shmptr->buffer[shmptr->count] = '\0';
        printf("Produced: %c\n", shmptr->buffer[shmptr->count - 1]);
```

```
        printf("Available items : ");
        for (int i = 0; i < strlen(shmptr->buffer); i++)
        {
            printf("%c ", shmptr->buffer[i]);
        }
        printf("\n");
        sem_post(&(shmptr->mutex));
        sem_post(&(shmptr->full));

        sleep(1);
    } while (input_index < strlen(original));
    printf("\nAll items produced\n");
}

void consumer()
{

    do
    {

        sem_wait(&(shmptr->full));
        sem_wait(&(shmptr->mutex));

        printf("Consumed %c\n", shmptr->buffer[0]);
        memmove(shmptr->buffer, shmptr->buffer + 1, strlen(shmptr->buffer));
        shmptr->count--;
        c++;

        printf("Available items : ");
        for (int i = 0; i < strlen(shmptr->buffer); i++)
            printf("%c ", shmptr->buffer[i]);
        printf("\n");
        sem_post(&(shmptr->mutex));
        sem_post(&(shmptr->empty));

        sleep(3);
    } while (c < strlen(input_string));

    printf("Consumed all the items\n");
    // exit(1);
}

int main()
{
    int shmid = shmget(IPC_PRIVATE, sizeof(struct memory), IPC_CREAT | 0666);
    shmptr = (struct memory *)shmat(shmid, NULL, 0);
    sem_init(&(shmptr->full), 1, 0);
    sem_init(&(shmptr->empty), 1, BUFSIZE);
    sem_init(&(shmptr->mutex), 1, 1);
    shmptr->count = 0;

    printf("Enter the string : ");
    scanf("%s", original);
```

```
    int pid = fork();

    if (pid == -1)
    {
        printf("Fork error\n");
    }
    else if (pid == 0)
    {
        consumer();
    }
    else
    {
        producer();
    }

    shmdt(shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    sem_destroy(&(shmptr->empty));
    sem_destroy(&(shmptr->full));
    sem_destroy(&(shmptr->mutex));
    return 0;
}
```

Output:

```
jayannthan_hakr@jayannthan-Ubuntu:~/OS LAB/Assignment6$ ./1
Enter the string : abcdefgh;
producer starts
Produced: a
Available items : a
consumer starts
Consumed a
Available items :
Produced: b
Available items : b
Produced: c
Available items : b c
Consumed b
Available items : c
Produced: d
Available items : c d
Produced: e
Available items : c d e
Produced: f
Available items : c d e f
Consumed c
Available items : d e f
Produced: g
Available items : d e f g
Produced: h
Available items : d e f g h

!!!buffer full!!!cannot produce!!!
Consumed d
Available items : e f g h
Produced: ;
Available items : e f g h ;

All items produced
jayannthan_hakr@jayannthan-Ubuntu:~/OS LAB/Assignment6$ Consumed e
Available items : f g h ;
Consumed f
Available items : g h ;
Consumed g
Available items : h ;
Consumed h
Available items : ;
Consumed ;
Available items :
Consumed all the items
```

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.

Algorithm for server:

1) For the segment, shared memory is allotted using shmget and returned id is stored in segid

2) For the empty, shared memory is allotted using shmget and returned id is stored in empty_id
3) For the full, shared memory is allotted using shmget and returned id is stored in full_id
4) For the mutex, shared memory is allotted using shmget and returned id is stored in mutex_id
5) Attach buffer to segid, empty to empty_id, full to full_id, mutex to mutex_id
6) For the key, shared memory is allotted using shmget and returned id is stored in shmid
7) Loop until N becomes zero
    f) Empty semaphore acquired by Server
    g) mutex semaphore acquired by Server
    h) a new random number written into buffer and N is decremented
    i) Empty semaphore released by Server
    j) mutex semaphore released by Server

Algorithm for client:

1) For the segment, shared memory is allotted using shmget and returned id is stored in segid
2) For the empty, shared memory is allotted using shmget and returned id is stored in empty_id
3) For the full, shared memory is allotted using shmget and returned id is stored in full_id
4) For the mutex, shared memory is allotted using shmget and returned id is stored in mutex_id
5) Attach buffer to segid, empty to empty_id, full to full_id, mutex to mutex_id
6) For the key, shared memory is allotted using shmget and returned id is stored in shmid
7) Loop until N becomes zero
    a) Empty semaphore acquired by Client
    b) mutex semaphore acquired by Client
    c) a new random read from buffer and N is decremented
    d) Empty semaphore released by Client
    e) mutex semaphore released by Client

Code:

```c
/*Server Code*/
#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <time.h>
#define BUFSIZE 5
```

```c
struct memory
{
    int buffer[BUFSIZE];
    int count;
    sem_t full;
    sem_t empty;
    sem_t mutex;
    int n;
    int nstatus;

};

struct memory * shmptr;

int main()
{
    srand(time(0));
    int shmid = shmget(111, sizeof(struct memory), IPC_CREAT | 0666);
    shmptr = (struct memory *) shmat(shmid, NULL, 0);
    shmptr->nstatus = 0;
    shmptr->count = 0;
    printf("\nServer\n");
    while (1)
    {
        if (shmptr->nstatus != 0)
        {
            int i = shmptr->n;
            do {    sem_wait(&(shmptr->empty));
                sem_wait(&(shmptr->mutex));
                shmptr->buffer[shmptr->count++] = rand() % 100;
                printf("Newly Produced: %d\n", shmptr->buffer[shmptr->count - 1]);
                i--;
                sem_post(&(shmptr->mutex));
                sem_post(&(shmptr->full));
                sleep(0);
            } while (i > 0);
            printf("\nProduction done\n");
            if (i == 0) break;
        }
    }
    shmdt(shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    sem_destroy(&(shmptr->empty));
    sem_destroy(&(shmptr->full));
    sem_destroy(&(shmptr->mutex));
    exit(1);
}
```

```c
/*Client Code*/
```

```c
#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <time.h>
#define BUFSIZE 5

struct memory
{
    int buffer[BUFSIZE];
    int count;
    sem_t full;
    sem_t empty;
    sem_t mutex;
    int n;
    int nstatus;

};

struct memory * shmptr;

int main()
{
    srand(time(0));
    int shmid = shmget(111, sizeof(struct memory), IPC_CREAT | 0666);
    shmptr = (struct memory *) shmat(shmid, NULL, 0);
    sem_init(&(shmptr->full), 1, 0);
    sem_init(&(shmptr->empty), 1, BUFSIZE);
    sem_init(&(shmptr->mutex), 1, 1);
    if (shmptr->nstatus == 0)
    {
        printf("Number of items to generate: ");
        scanf("%d", &(shmptr->n));
        shmptr->nstatus = 1;
    }
    int c = 0;
    do {
        sem_wait(&(shmptr->full));
        sem_wait(&(shmptr->mutex));
        printf("Available items : ");
        for (int i = 0; i < shmptr->count; i++)
        {
            printf("%d ", shmptr->buffer[i]);
        }
        printf("\n");
        printf("Consumes %d\n", shmptr->buffer[0]);
        memmove(shmptr->buffer, shmptr->buffer + 1, sizeof(shmptr->buffer));
```

```
        shmptr->count--;
        c++;
        sem_post(&(shmptr->mutex));
        sem_post(&(shmptr->empty));
        sleep(4);
    } while (c < shmptr->n);

    printf("\nFinished consuming all items\n");
    shmdt(shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    sem_destroy(&(shmptr->empty));
    sem_destroy(&(shmptr->full));
    sem_destroy(&(shmptr->mutex));

    exit(1);

}
```

Output:

```
Server
Newly Produced: 41
Newly Produced: 16
Newly Produced: 18
Newly Produced: 8
Newly Produced: 27
Newly Produced: 14
Newly Produced: 0
!!!buffer full!!!
Newly Produced: 48
!!!buffer full!!!
!!!buffer full!!!
!!!buffer full!!!
Newly Produced: 49
!!!buffer full!!!
!!!buffer full!!!
!!!buffer full!!!
Newly Produced: 26

Production done
```

```
Number of items to generate: 10
Available items : 41
Consumes 41
Available items : 16 18 8
Consumes 16
Available items : 18 8 27 14 0
Consumes 18
Available items : 8 27 14 0 48
Consumes 8
Available items : 27 14 0 48 49
Consumes 27
Available items : 14 0 48 49 26
Consumes 14
Available items : 0 48 49 26
Consumes 0
Available items : 48 49 26
Consumes 48
Available items : 49 26
Consumes 49
Available items : 26
Consumes 26

Finished consuming all items
```

Learning Outcome:

- Executed semaphore functions and system calls
- Executed server-side and client-side program using shared memory and semaphores