UCS1411 Operating Systems Lab
AY: 2021-22

Name: Krithika Swaminathan
Roll No.: 205001057

Assignment 6 – Implementation of Producer-Consumer Problem using Semaphores

Date: 18/04/2022

Aim:

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

ii) To modify the program as separate client/server process programs to generate 'N' random numbers in producer, write them into the shared memory and execute a consumer process to read them from the shared memory and display them on the terminal.

Algorithm:

i) Producer-Consumer problem using semaphores:

1. Start
2. Create a shared memory for the buffer and semaphores - *empty*, *full* and *mutex*. Get the shared memory identifier. Initialize the semaphores with values 'buffer-size', 0 and 1 respectively.
3. Create a parent and a child process with the parent acting as the producer and the child as the consumer.
4. Get a string as input from the user and fork the process.
5. In the producer process, produce an item by taking the first unvisited character from the string entered and placing it in the buffer array. Increment the semaphores *full* and *mutex*, and decrement the semaphore *empty* using the *wait* and *signal* operations suitably.
6. In the consumer process, consume an item by removing the first item in the buffer and displaying it on the terminal. Increment the semaphores *empty* and *mutex*, and decrement the semaphore *full* using the *wait* and *signal* operations appropriately.
7. Detach the pointer from the shared memory.
8. Destroy the shared memory space.
9. Stop

ii) Producer-Consumer problem with client/server process programs and random number generation:

Server side:

1. Start

2. Create a shared memory for the buffer and semaphores - *empty*, *full* and *mutex*. Get the shared memory identifier. Initialize the semaphores with values ‘buffer-size’, 0 and 1 respectively.
3. Get the number of numbers required as input from the user.
4. Randomly generate as many numbers as required and add them to the buffer one by one.
5. After every addition to the buffer, increment *full* and decrement *empty* using the wait and signal operations appropriately.
6. Detach the pointer from the shared memory.
7. Destroy the shared memory space.
8. Stop

Client side:

1. Start
2. Access the shared memory that contains the buffer.
3. Get the numbers stored in the buffer one by one and display them on the terminal.
4. Detach the pointer from the shared memory.
5. Stop

Programs:

1) Producer-Consumer problem using semaphores:

Code:

```
//Implementation of producer-consumer problem using semaphores

#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 10

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

```
    sem_t mutex;
        };
struct memory *shmptr;

char string[BUFSIZE];
int in_index=0;
int c=0;

void producer() {
        do {
                sem_wait(&(shmptr->empty));
                sem_wait(&(shmptr->mutex));

                shmptr->buffer[shmptr->count++]=string[in_index++];
                shmptr->buffer[shmptr->count]='\0';
                printf("Produced: %c\n",shmptr->buffer[shmptr->count-1]);
                printf("Buffer: %s\n",shmptr->buffer);
                sem_post(&(shmptr->mutex));
                sem_post(&(shmptr->full));

                sleep(1);
                } while(in_index<strlen(string));

        wait(NULL);
        printf("Producer operation completed!\n");
        exit(1);
        }

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("Buffer: %s\n",shmptr->buffer);
               sem_post(&(shmptr->mutex));
               sem_post(&(shmptr->empty));

               sleep(2);
                  } while(c<strlen(string));

            printf("Consumer operation completed!\n");
            exit(1);
        }
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
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 string: ");
        scanf("%s",string);

        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));
        printf("Complete. Exiting...\n");
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q61 prod_cons.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q61
Enter string: random
Produced: r
Buffer: r
Consumed: r
Buffer:
Produced: a
Buffer: a
Consumed: a
Buffer:
Produced: n
Buffer: n
Produced: d
Buffer: nd
Consumed: n
Buffer: d
Produced: o
Buffer: do
Produced: m
Buffer: dom
Consumed: d
Buffer: om
Consumed: o
Buffer: m
Consumed: m
Buffer:
Consumer operation completed!
Producer operation completed!
kri@kri-ubuntu:~/workspace$
```

Name: Krithika Swaminathan
Roll No.: 205001057

2) Producer-Consumer problem with client/server process programs and random number generation:

Code: Server side

//Server side implementation of producer-consumer problem with random number generation

```
#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 10

struct memory{
   int buffer[BUFSIZE];
   int count;
   sem_t full;
   sem_t empty;
   sem_t mutex;
   int n;
        };
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);
            shmptr->count=0;

            printf("No. of numbers: ");
            scanf("%d",&(shmptr->n));

            int i=shmptr->n, rnum;
            do {
                        sem_wait(&(shmptr->empty));
                        sem_wait(&(shmptr->mutex));

                        rnum = rand()%100;
```

```
                shmptr->buffer[shmptr->count++]=rnum;
                printf("Produced: %d\n",shmptr->buffer[shmptr->count-1]);
                i--;

                sem_post(&(shmptr->mutex));
                sem_post(&(shmptr->full));
                sleep(1);
                } while(i>0);

        printf("Producer operation completed!\n");

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

        printf("__Process completed__\n");
        exit(1);
        }
```

Code: Client side

```
//Client side implementation of producer-consumer problem with random number generation

#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 10

struct memory{
        int buffer[BUFSIZE];
        int count;
        sem_t full;
        sem_t empty;
        sem_t mutex;
        int n;
        };
struct memory *shmptr;
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
int main() {
        srand(time(0));
        int shmid=shmget(111,sizeof(struct memory),IPC_CREAT|0666);
        shmptr=(struct memory *)shmat(shmid,NULL,0);

        int c=0;
        do {
                sem_wait(&(shmptr->full));
                sem_wait(&(shmptr->mutex));

                printf("Consumed: %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(1);
                } while(c<shmptr->n);

        printf("\nConsumer operation completed!\n");

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

        printf("__Process complete. Exiting..._\n");
        exit(1);
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q62s sema_server.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q62s
No. of numbers: 5
Produced: 89
Produced: 63
Produced: 94
Produced: 20
Produced: 5
Producer operation completed!
__Process completed__
kri@kri-ubuntu:~/workspace$
```

```
kri@kri-ubuntu:~/workspace$ gcc -o q62c sema_client.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q62c
Consumed: 89
Consumed: 63
Consumed: 94
Consumed: 20
Consumed: 5

Consumer operation completed!
__Process complete. Exiting..._
kri@kri-ubuntu:~/workspace$
```

Name: Krithika Swaminathan
Roll No.: 205001057

Learning outcomes:

- It was understood that semaphores can be used to solve various synchronization problems and can be implemented efficiently.
- The producer/consumer bounded buffer problem was understood by implementing it using semaphores.
- The method of handling semaphores between a server and a client was understood.
