****

**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
   1. Empty semaphore acquired by producer
   2. mutex semaphore acquired by producer
   3. next character from string is written into buffer
   4. Empty semaphore released by producer
   5. mutex semaphore released by producer

Consumer function:

1. Initialise i=0
2. If I greater than string length then exit from consumer
3. Else
   1. Empty semaphore acquired by consumer
   2. mutex semaphore acquired by consumer
   3. next character from buffer is read and pointer to buffer is increased
   4. Empty semaphore released by consumer
   5. mutex semaphore released by consumer

**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>

#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));

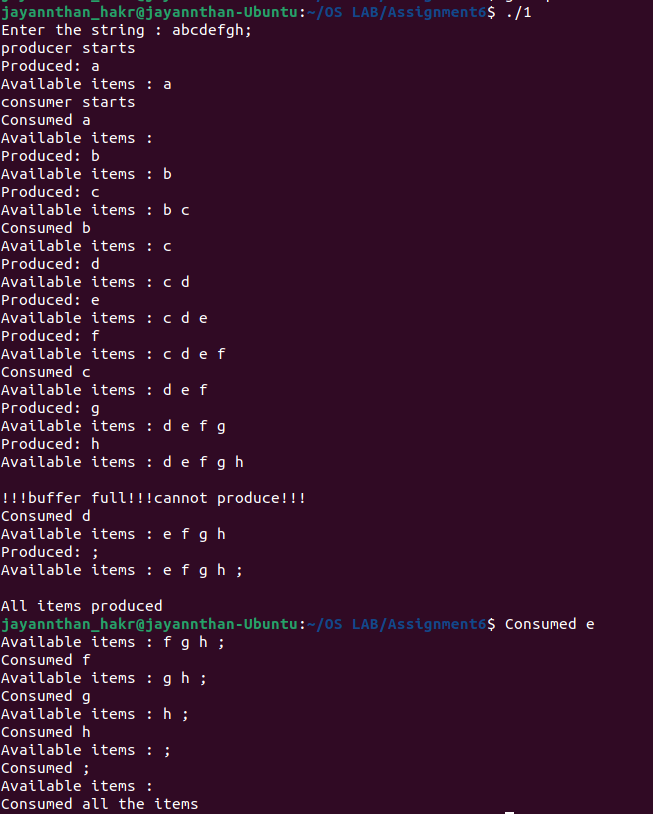
    sem\_destroy(&(shmptr->full));

    sem\_destroy(&(shmptr->mutex));

    return 0;

}

**Output:**



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
   1. Empty semaphore acquired by Server
   2. mutex semaphore acquired by Server
   3. a new random number written into buffer and N is decremented
   4. Empty semaphore released by Server
   5. 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
8. Empty semaphore acquired by Client
9. mutex semaphore acquired by Client
10. a new random read from buffer and N is decremented
11. Empty semaphore released by Client
12. mutex semaphore released by Client

**Code:**

*/\*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

**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);

}

*/\*Client 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

**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));

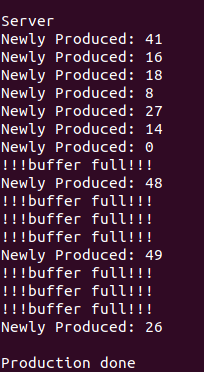
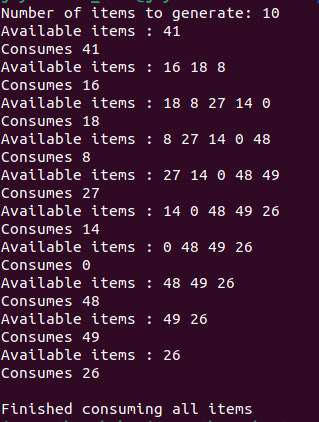
    sem\_destroy(&(shmptr->full));

    sem\_destroy(&(shmptr->mutex));

    exit(1);

}

**Output:**

**Learning Outcome:**

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