|  |  |
| --- | --- |
| **Name:** | Bodhisatya Ghosh |
| **Class:** | CSE DS |
| **Batch:** | B |
| **UID:** | 2021700026 |
| **Experiment:** | 10 |

**Aim**: The program r.c initializes n number of semaphores. It first assign count equal -1, which is then used by process p and q. This count is protected by semaphore. It also allocates shared memory of size 40 ints. It waits for process p and q to enter all n1 and n2 elements through different terminals. This program r.c sorts shared data in ascending order. It waits to finish p and q. At end, The program r.c detaches and deletes n semaphores and print the sorted list.  
  
**Code:**

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <semaphore.h>

#define TOTAL\_ELEMENTS 40

int main()

{

    printf("Main:\n");

    key\_t shmkey = ftok("my\_unique\_key\_gg", 0);

    sem\_t \*sem = sem\_open("my\_semaphore\_for\_count", O\_CREAT, S\_IRUSR | S\_IWUSR, 0);

    sem\_t \*over\_sem = sem\_open("my\_semaphore\_for\_r", O\_CREAT, S\_IRUSR | S\_IWUSR, 0);

    if (sem == SEM\_FAILED || over\_sem == SEM\_FAILED)

    {

        perror("sem\_open");

        exit(1);

    }

    // Intial value of semaphore set to 0

    sem\_init(sem, 1, 0);

    sem\_init(over\_sem, 1, 0);

    int shmid = shmget(shmkey, (TOTAL\_ELEMENTS + 1) \* sizeof(int), IPC\_CREAT | 0666);

    // Last one for maintaining count

    if (shmid == -1)

    {

        perror("shmget");

        exit(1);

    }

    int \*array = (int \*)shmat(shmid, NULL, 0);

    if (array == (int \*)-1)

    {

        perror("shmat");

        exit(1);

    }

    array[TOTAL\_ELEMENTS] = -1;

    printf("Waiting for r to complete\n");

    sem\_wait(over\_sem);

    printf("\nAll operation done\n");

    shmdt(array);

    shmctl(shmid, IPC\_STAT, NULL);

    sem\_destroy(sem);

    sem\_close(sem);

    sem\_destroy(over\_sem);

    sem\_close(over\_sem);

    sem\_unlink("my\_semaphore\_for\_count");

    sem\_unlink("my\_semaphore\_for\_r");

    return 0;

}

**-----------------------------------------**

**p.c**

#include <stdio.h>

#include "semaphore.h"

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <time.h>

#include <fcntl.h>

#include <sys/stat.h>

#define TOTAL\_ELEMENTS 40

int main()

{

    printf("Process p:\n");

    key\_t key;

    key = ftok("my\_unique\_key\_gg", 0);

    // sem\_t \*sem;

    // sem = sem\_open("my\_semaphore\_for\_count", 0);

    sem\_t \*sem = sem\_open("my\_semaphore\_for\_count", O\_CREAT, S\_IRUSR | S\_IWUSR, 1);

    if (sem == SEM\_FAILED)

    {

        perror("sem\_open");

        exit(1);

    }

    int shmid = shmget(key, (TOTAL\_ELEMENTS + 1) \* sizeof(int), IPC\_CREAT | 0666);

    int \*arr = (int \*)shmat(shmid, NULL, 0);

    srand(time(NULL));

    printf("Waiting for instruction from r\n\n");

    while (1)

    {

        sem\_wait(sem);

        int count = arr[TOTAL\_ELEMENTS];

        if (count >= TOTAL\_ELEMENTS - 1)

        {

            sem\_post(sem);

            break;

        }

        int val = rand() % 500;

        printf("Index: %d, Value: %d\n", count + 1, val);

        arr[count + 1] = val;

        arr[TOTAL\_ELEMENTS] = count + 1;

        sem\_post(sem);

        sleep(1);

    }

    shmdt(arr);

    return 0;

}

**-----------------------------------------**

**q.c**

#include <stdio.h>

#include "semaphore.h"

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <time.h>

#include <fcntl.h>

#include <sys/stat.h>

#define TOTAL\_ELEMENTS 40

int main()

{

    printf("Process q:\n");

    key\_t key;

    key = ftok("my\_unique\_key\_gg", 0);

    // sem\_t \*sem;

    // sem = sem\_open("my\_semaphore\_for\_count", 0);

    sem\_t \*sem = sem\_open("my\_semaphore\_for\_count", O\_CREAT, S\_IRUSR | S\_IWUSR, 1);

    if (sem == SEM\_FAILED)

    {

        perror("sem\_open");

        exit(1);

    }

    int shmid = shmget(key, (TOTAL\_ELEMENTS + 1) \* sizeof(int), IPC\_CREAT);

    int \*arr = (int \*)shmat(shmid, NULL, 0);

    srand(time(NULL));

    printf("Waiting for instruction from r\n\n");

    while (1)

    {

        sem\_wait(sem);

        int count = arr[TOTAL\_ELEMENTS];

        if (count >= TOTAL\_ELEMENTS - 1)

        {

            sem\_post(sem);

            break;

        }

        int val = rand() % 500;

        printf("Index: %d, Value: %d\n", count + 1, val);

        arr[count + 1] = val;

        arr[TOTAL\_ELEMENTS] = count + 1;

        sem\_post(sem);

        sleep(1);

    }

    shmdt(arr);

    return 0;

}

**-----------------------------------------**

**r.c**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <semaphore.h>

#define TOTAL\_ELEMENTS 40

int main()

{

    printf("Process r:\n");

    key\_t key;

    key = ftok("my\_unique\_key\_gg", 0);

    // sem\_t \*sem;

    // sem = sem\_open("my\_semaphore\_for\_count", 0);

    sem\_t \*sem = sem\_open("my\_semaphore\_for\_count", O\_CREAT, S\_IRUSR | S\_IWUSR, 1);

    sem\_t \*over\_sem = sem\_open("my\_semaphore\_for\_r", O\_CREAT, S\_IRUSR | S\_IWUSR, 0);

    if (sem == SEM\_FAILED)

    {

        perror("sem\_open");

        exit(1);

    }

    int shmid = shmget(key, (TOTAL\_ELEMENTS+1) \* sizeof(int), IPC\_CREAT | 0666);

    int \*arr = (int \*)shmat(shmid, NULL, 0);

    printf("Waiting for arr to be filled.....\n");

    sem\_post(sem);

    sleep(1);

    while(1){

        sem\_wait(sem);

        if(arr[TOTAL\_ELEMENTS] >= TOTAL\_ELEMENTS-1){

            sem\_post(sem);

            break;

        }

        sem\_post(sem);

        sleep(1);

    }

    // bubble sort the arr

    for (int i = 0; i < TOTAL\_ELEMENTS; i++)

    {

        for (int j = i; j < TOTAL\_ELEMENTS; j++)

        {

            if (arr[i] > arr[j])

            {

                int t = arr[i];

                arr[i] = arr[j];

                arr[j] = t;

            }

        }

    }

    int min = arr[0];

    printf("Sorted array: ");

    for (int i = 0; i < TOTAL\_ELEMENTS; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

    sem\_post(over\_sem);

    shmdt(arr);

    sem\_close(sem);

    sem\_close(over\_sem);

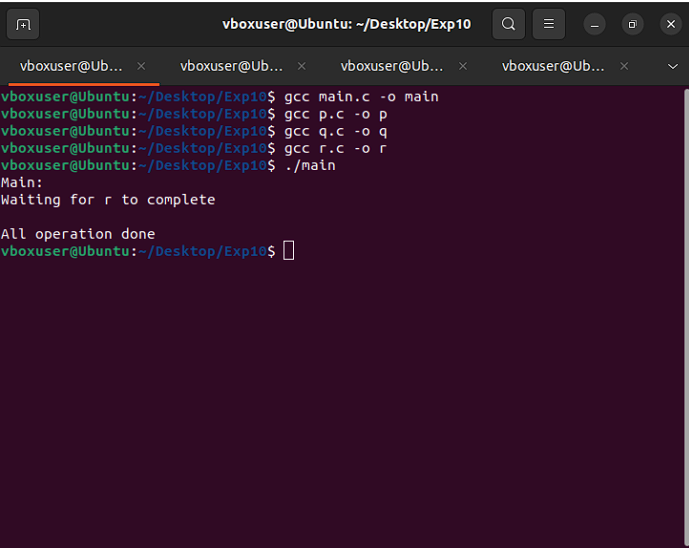
    // sem\_unlink("my\_semaphore\_for\_count");

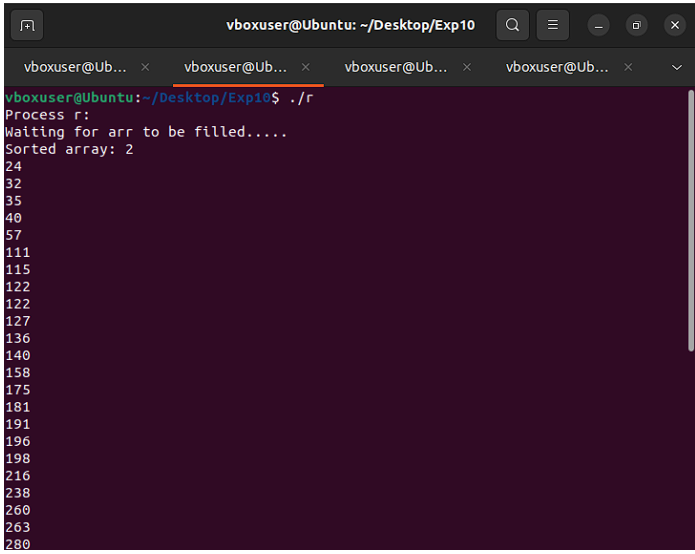
    // sem\_unlink("my\_semaphore\_for\_r");

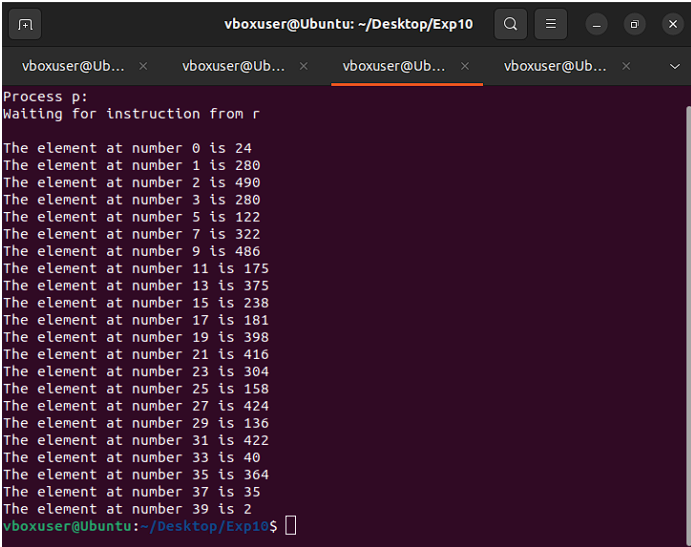
    return 0;

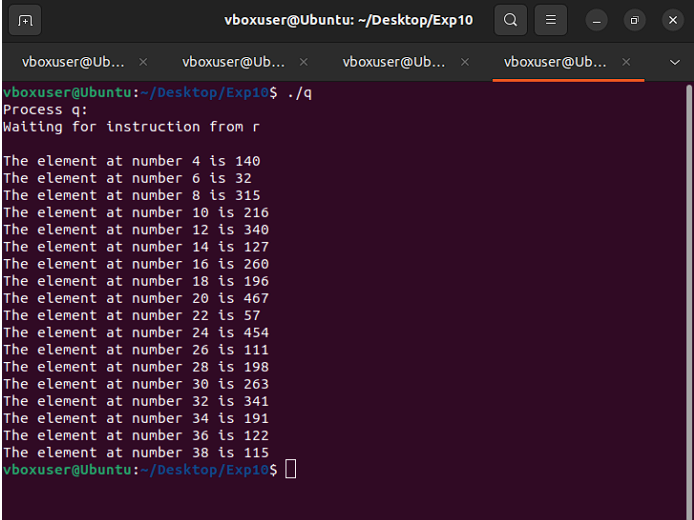
}

**Output:**

****

****

****

**  
  
Conclusion:** In this experiment we have learnt how to implement semaphores.