CSE 325

Name : Saniya Chandanan

Section : K18TS

Roll No : 46

Regd No : 11805683

1. Write a program to sync 5 readers and 3 writers such that multiple readers should be allowed to read at same time but reader and writer cannot simultaneously enter into the critical section. Program should be implemented with the help of thread and use semaphore for synchronization.

Solution:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

sem_t wrt,mutex;
int num = 1;
int numreader = 0;

void *writer(void *wno)
```

```
{
   sem_wait(&wrt);
   num=num+1;
   printf("Writer %d edited data to %d\n",(*((int *)wno)),num);
   sem_post(&wrt);

}
void *reader(void *rno)
{
   sem_wait(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt);
   }
   sem_post(&mutex);
   printf("Reader %d reading data as %d\n",*((int *)rno),num);
   sem_wait(&mutex);
   numreader--;
```

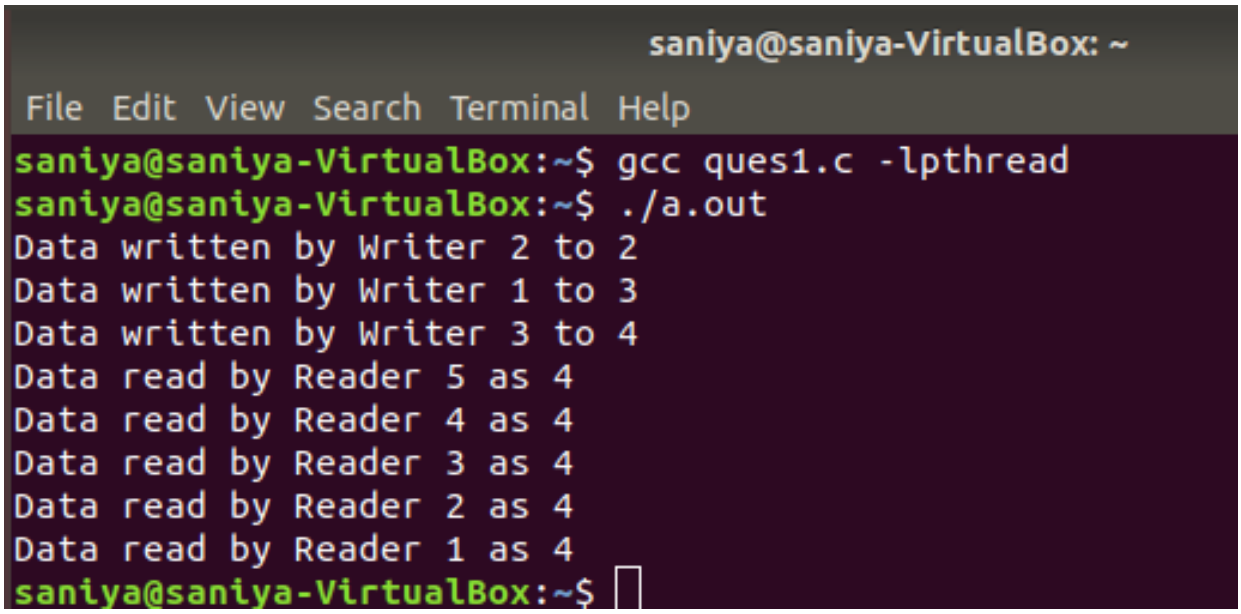
```
    if(numreader == 0) {
        sem_post(&wrt);
    }
    sem_post(&mutex);
}

int main()
{

    pthread_t read[5],write[3];
    sem_init(&mutex, 0,1);
    sem_init(&wrt,0,1);
    int per[5] = {1,2,3,4,5};
    for(int i = 0; i < 5; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&per[i]);
    }
        for(int i = 0; i < 3; i++) {
        pthread_create(&write[i], NULL, (void *)writer, (void *)&per[i]);
```

```
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(read[i], NULL);
    }
    for(int i = 0; i < 3; i++) {
        pthread_join(write[i], NULL);
    }
    sem_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;

}
```



2. Write a program to create two threads and both the threads are sharing a common variable whose value is initialized by 2. First thread should increment the shared value by 1 and the other thread should decrement by 1. In order to avoid context switching and to attain consistency use Mutex Lock. Create a scenario such that after every execution, the value of the shared variable should be consistent.

Solution:

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>

void *fun1();
void *fun2();
int shared=2;
pthread_mutex_t l;
int main()
{
	pthread_mutex_init(&l, NULL);
	pthread_t thread1, thread2;
	pthread_create(&thread1, NULL, fun1, NULL);
	pthread_create(&thread2, NULL, fun2, NULL);
	pthread_join(thread1, NULL);
	pthread_join(thread2,NULL);
	printf("\nFinal value of shared variable : %d\n",shared);
}

void *fun1()
```

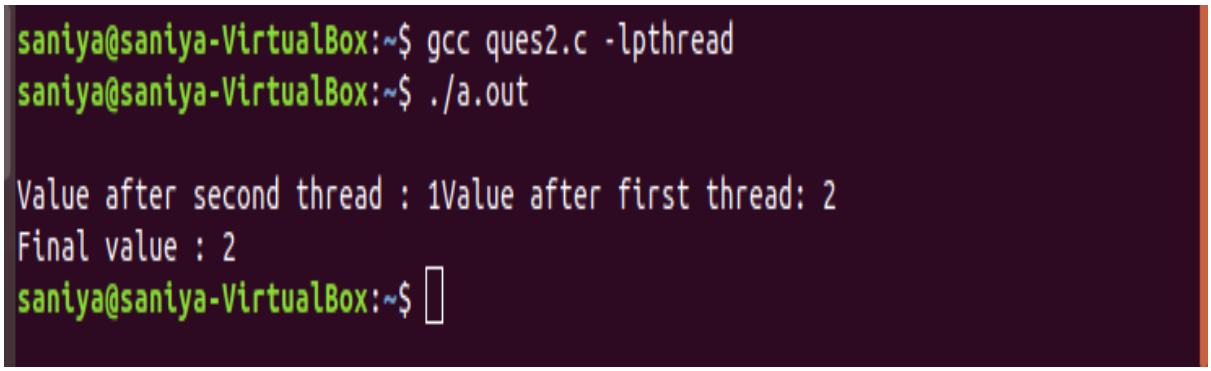
```
{
    int x;
    pthread_mutex_lock(&l);
    x=shared;
    x++;
    sleep(1);
    shared=x;
    printf("Value of shared variable in thread 1 after incrementation : %d",shared);
    pthread_mutex_unlock(&l);
}

void *fun2()
{
    int y;
    pthread_mutex_lock(&l);
    y=shared;
    y--;
    sleep(1);
    shared=y;
```

```
    printf("\nValue of shared variable in thread 2 after decrementation : %d",shared);

    pthread_mutex_unlock(&l);

}
```



3. Write a program to create three threads and three binary semaphore S1, S2 and S3 that are initialized with value 1. Threads should acquire semaphore in such a manner that deadlock is achieved. (Take reference from the deadlock in semaphore that we studied in CSE316.)

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include<semaphore.h>

void *fun1();
void *fun2();
void *fun3();
```

```
sem_t s1,s2,s3;

int main()
{
	sem_init(&s1,0,1);
	sem_init(&s2,0,1);
	sem_init(&s3,0,1);

	pthread_t thread1,thread2,thread3;
	pthread_create(&thread1,NULL,fun1,NULL);
	pthread_create(&thread2,NULL,fun2,NULL);
	pthread_create(&thread3,NULL,fun2,NULL);
	pthread_join(thread1,NULL);
	pthread_join(thread2,NULL);
	pthread_join(thread3,NULL);

}

void *fun1()
{
```

```
    sem_wait(&s1);
    printf("thread one acquired semaphore 1\n");
    sleep(1);
    sem_wait(&s2);
    printf("\nthread one acquired semaphore 2\n");
    sleep(1);
    sem_wait(&s3);
    printf("\nthread one acquired semaphore 3\n");

    sem_post(&s3);
    printf("\nthread one release semaphore 3\n");

    sem_post(&s2);
    printf("\nthread one release semaphore 2\n");

    sem_post(&s1);
    printf("\nthread one release semaphore 1\n");
}

void *fun2()
```

```
{
        sem_wait(&s3);
        printf("thread two acquired semaphore 3\n");
        sleep(1);
        sem_wait(&s2);
        printf("\nthread two acquired semaphore 2\n");
        sleep(1);
        sem_wait(&s1);
        printf("\nthread two acquired semaphore 1\n");

        sem_post(&s1);
        printf("\nthread two release semaphore 1\n");

        sem_post(&s2);
        printf("\nthread two release semaphore 2\n");

        sem_post(&s3);
        printf("\nthread two release semaphore 3");

}
```

```
void *fun3()
{
	sem_wait(&s2);
	printf("thread three acquired semaphore 2\n");
	sleep(1);
	sem_wait(&s1);
	printf("\nthread three acquired semaphore 1\n");
	sleep(1);
	sem_wait(&s3);
	printf("\nthread three acquired semaphore 3\n");

	sem_post(&s3);
	printf("\nthread three release semaphore 3\n");

	sem_post(&s1);
	printf("\nthread three release semaphore 1\n");

	sem_post(&s2);
```

```
        printf("\nthread three release semaphore 2\n");
}
```

```
saniya@saniya-VirtualBox:~$ gcc ques3.c -lpthread
saniya@saniya-VirtualBox:~$ ./a.out
Thread : 2 using Semaphore : 3

Thread : 2 using Semaphore : 2

Thread : 2 using Semaphore : 1
```

4.Write a program to implement producer consumer problem such that Maximum number of items produced by a producer are five. Producer should not produce any item if the buffer is full and should say “ BUFFER IS ALREADY FULL” and the consumer should not consume if the buffer is empty. Problem should be implemented with the help of thread, semaphore and mutex lock.

Solution:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
```

```
#include <stdio.h>

#define MaxItems 5
#define BufferSize 5

sem_t empty,full;
int in = 0, out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

int a=0;


void *producer()
{
   int item=1;
       sem_wait(&empty);
       pthread_mutex_lock(&mutex);
       buffer[in] = item;
```

```
        printf("Producer produces %d item at %d\n", buffer[in],in+1);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        a++;
}
void *consumer()
{
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer removes %d item from %d\n",item, out+1);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        a--;
}
```

```
int main()
{
	int n;
   pthread_t pro,con;
   pthread_mutex_init(&mutex, NULL);
   sem_init(&empty,0,BufferSize);
   sem_init(&full,0,0);
	printf("\n1.Producer\n2.Consumer\n3.Exit");


	while(1)
	{
		printf("\nEnter your choice:");
		scanf("%d",&n);
		switch(n)
		{
			case 1:if(a!=5)
				{
					pthread_create(&pro,
NULL,producer,NULL);
```

```
                        pthread_join(pro,NULL);
                }
                else
                        printf("Buffer is already full!");
                break;

case 2: if(a!=0)
                {
                        pthread_create(&con,
NULL,consumer,NULL);
                        pthread_join(con,NULL);
                }
else

                        printf("Buffer is empty!!");
                break;
            case 3:
                    exit(0);
                    break;
        }
```

```
    }
  pthread_mutex_destroy(&mutex);
  sem_destroy(&empty);
  sem_destroy(&full);

  return 0;

}
```

```
saniya@saniya-VirtualBox:~$ gcc ques4.c -lpthread
saniya@saniya-VirtualBox:~$ ./a.out
Choose

1.Consume
2.Produce
3.Quit
Enter :1
Producer produces 1 item at 1

Enter :1
Producer produces 1 item at 2

Enter :1
Producer produces 1 item at 3

Enter :1
Producer produces 1 item at 4

Enter :1
Producer produces 1 item at 5

Enter :1
Buffer is full
Enter :1
Buffer is full
Enter :1
Buffer is full
```

```
Producer produces 1 item at 5

Enter :1
Buffer is full
Enter :1
Buffer is full
Enter :1
Buffer is full
Enter :2
Consumer removes 1 item from 1

Enter :2
Consumer removes 1 item from 2

Enter :2
Consumer removes 1 item from 3

Enter :2
Consumer removes 1 item from 4

Enter :2
Consumer removes 1 item from 5

Enter :2
Empty Buffer
Enter :2
Empty Buffer
Enter :
```