LAB – 8

Name : Shubham Shingala

Roll no. : CE143

ID : 19CEUOS159

1. **Explain the use of semaphore library functions with syntax and examples.**

- **Use of Semaphore:** Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

a. **sem_init :**

➢ **Description:**

initialize an unnamed semaphore . sem_init() function first argument is we pass address of type sem_t variable where sem_t is structure. second argument is pshared means process shared . if we pass pshared is equal to 0 then semaphore is shared between the threads of a process And if we pass pshared is equal to non zero then the semaphore is shared between processes. third argument value is initial value of semaphore. sem_init( ) return 0 on success otherwise it will return -1.

➢ **Syntax:**

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

➢ **Example:**

```
sem_t sem1
sem_init(&sem1,0,5);
```

in above example it will initialize an unnamed semaphore. first arg we pass address of sem1.second arg we pass as 0 so semaphore is shared between the threads of a process and initial value of semaphore is 5.

b. sem_wait:

- **Description:**
  lock a semaphore . sem_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement, or a signal handler interrupts the call. sem_post() returns 0 on success; on error returns -1

- **Syntax:**

```
#include <semaphore.h>
int sem_wait(sem_t *sem);
```

- **Example:**

```
sem_t sem1

sem_wait(&sem1);
```

  in above example lock a semaphore pointed by sem1

c. sem_post:

- **Description:**
  unlock a semaphore. sem_post() increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait(3) call will be woken up and proceed to lock the semaphore. sem_post() returns 0 on success; on error returns -1

- **Syntax:**

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- **Example:**

```
sem_t sem1

sem_post(&sem1);
```

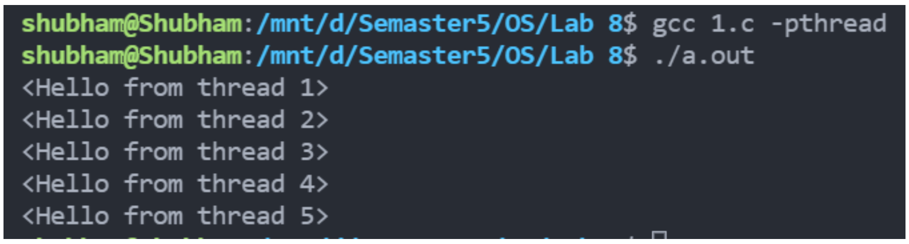
  in above example unlock a semaphore pointed by sem1

2. **Write a program in which 5 threads are sharing and incrementing the value of a global variable using semaphores.**

Program:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
int count = 0;
sem_t semptr;

void *fun()
{
    sem_wait(&semptr);
    count++;
    printf("<Hello from thread %d>\n", count);
    sem_post(&semptr);
}

void main()
{
    sem_init(&semptr, 0, 1);
    pthread_t p[5];
    for (int i = 0; i < 5; i++)
        pthread_create(&p[i], NULL, fun, NULL);
    for (int i = 0; i < 5; i++)
        pthread_join(p[i], NULL);
    sem_destroy(&semptr);
}
```

Output:



```
shubham@Shubham:/mnt/d/Semaster5/OS/Lab 8$ gcc 1.c -pthread
shubham@Shubham:/mnt/d/Semaster5/OS/Lab 8$ ./a.out
<Hello from thread 1>
<Hello from thread 2>
<Hello from thread 3>
<Hello from thread 4>
<Hello from thread 5>
```

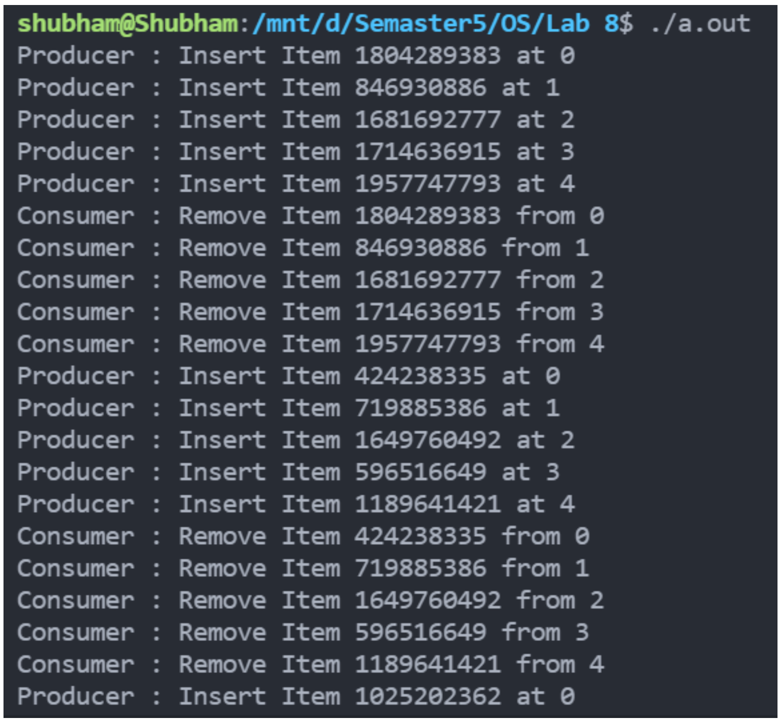
3. **Write a program to implement the solution to bounded buffer Producer-Consumer Problem using semaphores.**

Program:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define BufferSize 5 // Size of the buffer

sem_t s, n, e; //s=process,n=number of item in buffer,e=number of empty spac
e in buffer
int in = 0, out = 0, buffer[BufferSize];

void *producer(void *pno)
{
    int item;
    while (1)
    {
        item = rand(); // Produce an random item
        sem_wait(&e);
        sem_wait(&s);
        buffer[in] = item;
        printf("Producer : Insert Item %d at %d\n", buffer[in], in);
        in = (in + 1) % BufferSize;
        sleep(1); //for observe output
        sem_post(&s);
        sem_post(&n);
    }
}

void *consumer(void *cno)
{
    while (1)
    {
        sem_wait(&n);
        sem_wait(&s);
        int item = buffer[out];
        printf("Consumer : Remove Item %d from %d\n", item, out);
        out = (out + 1) % BufferSize;
        sleep(1);
        sem_post(&s);
        sem_post(&e);
    }
}
```

```c
int main()
{
    sem_init(&s, 0, 1);          //1 means 1 process can access critical sec
tion at a time
    sem_init(&n, 0, 0);          //0 means initial buffer hasno item
    sem_init(&e, 0, BufferSize); //5 means initial buffer has 5 empty space
    pthread_t produce, consume;
    pthread_create(&produce, NULL, producer, NULL);
    pthread_create(&consume, NULL, consumer, NULL);
    pthread_join(produce, NULL);
    pthread_join(consume, NULL);
    return 0;
}
```

Output:



```
shubham@Shubham:/mnt/d/Semaster5/OS/Lab 8$ ./a.out
Producer : Insert Item 1804289383 at 0
Producer : Insert Item 846930886 at 1
Producer : Insert Item 1681692777 at 2
Producer : Insert Item 1714636915 at 3
Producer : Insert Item 1957747793 at 4
Consumer : Remove Item 1804289383 from 0
Consumer : Remove Item 846930886 from 1
Consumer : Remove Item 1681692777 from 2
Consumer : Remove Item 1714636915 from 3
Consumer : Remove Item 1957747793 from 4
Producer : Insert Item 424238335 at 0
Producer : Insert Item 719885386 at 1
Producer : Insert Item 1649760492 at 2
Producer : Insert Item 596516649 at 3
Producer : Insert Item 1189641421 at 4
Consumer : Remove Item 424238335 from 0
Consumer : Remove Item 719885386 from 1
Consumer : Remove Item 1649760492 from 2
Consumer : Remove Item 596516649 from 3
Consumer : Remove Item 1189641421 from 4
Producer : Insert Item 1025202362 at 0
```

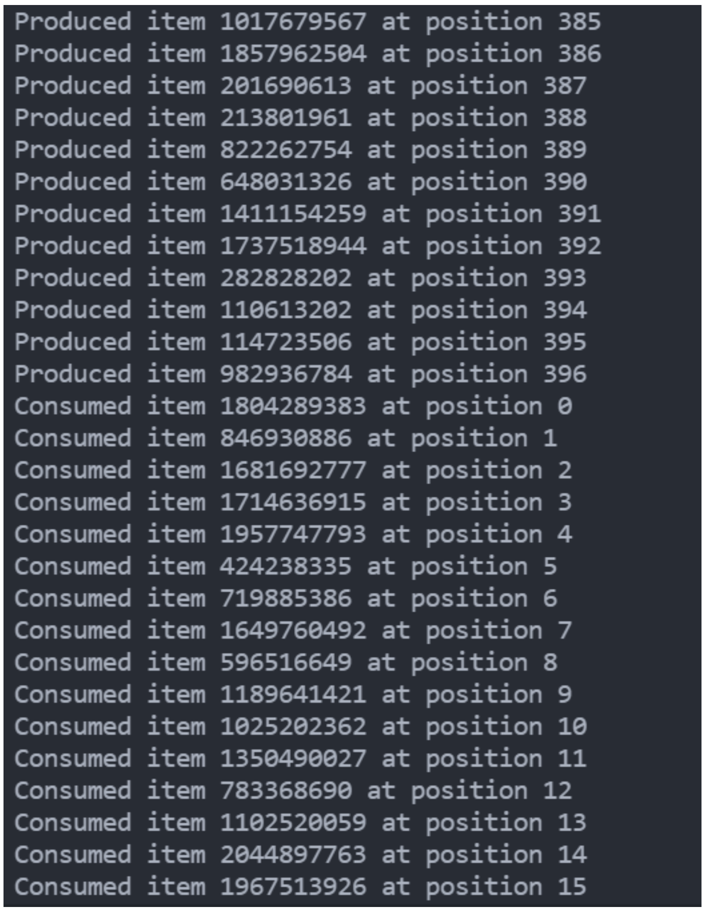
❖ **Assignment:**

➢ **Write a program to implement the solution to infinite buffer Producer-Consumer Problem using semaphores.**

Program:

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <limits.h>
sem_t process, n; //s=process,n=number of item in buffer
int in = 0, out = 0, buffer[INT_MAX];
void *producer()
{
    int item;
    while (1)
    {
        item = rand();
        sem_wait(&process);
        buffer[in] = item;
        printf("Produced item %d at position %d\n", item, in);
        in = in + 1;
        sleep(1); //for observe output
        sem_post(&process);
        sem_post(&n);
    }
}
void *consumer()
{
    int item;
    while (1)
    {
        sem_wait(&n);
        sem_wait(&process);
        item = buffer[out];
        printf("Consumed item %d at position %d\n", item, out);
        out = out + 1;
        sleep(1);
        sem_post(&process);
    }
}
void main()
{
    sem_init(&process, 0, 1); //1 means 1 process can access critical sectio
n at a time
    sem_init(&n, 0, 0);        //0 means initial buffer hasno item
    pthread_t produce, consume;
    pthread_create(&produce, NULL, producer, NULL);
    pthread_create(&consume, NULL, consumer, NULL);
    pthread_join(produce, NULL);
    pthread_join(consume, NULL);
}
```

Output:



```
Produced item 1017679567 at position 385
Produced item 1857962504 at position 386
Produced item 201690613 at position 387
Produced item 213801961 at position 388
Produced item 822262754 at position 389
Produced item 648031326 at position 390
Produced item 1411154259 at position 391
Produced item 1737518944 at position 392
Produced item 282828202 at position 393
Produced item 110613202 at position 394
Produced item 114723506 at position 395
Produced item 982936784 at position 396
Consumed item 1804289383 at position 0
Consumed item 846930886 at position 1
Consumed item 1681692777 at position 2
Consumed item 1714636915 at position 3
Consumed item 1957747793 at position 4
Consumed item 424238335 at position 5
Consumed item 719885386 at position 6
Consumed item 1649760492 at position 7
Consumed item 596516649 at position 8
Consumed item 1189641421 at position 9
Consumed item 1025202362 at position 10
Consumed item 1350490027 at position 11
Consumed item 783368690 at position 12
Consumed item 1102520059 at position 13
Consumed item 2044897763 at position 14
Consumed item 1967513926 at position 15
```