LAB 8 ASSIGNMENT

Name: Birva Babaria

Roll no.: CE010

ID: 19CEUON064

Aim: Study of semaphore library functions.

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

❖ 'sem_init' system call

Library: #include<semaphore.h>

Syntax: int sem_init(sem_t *sem, int pshared, unsigned int value);

Description: sem_init system call initialize an unnamed semaphore.

First argument is address of type sem_t variable where sem_t is structure.

Second argument id pshared means process shared. If pshared is equal to 0 then semaphore is shared between the threads of process and if we pass pshared is equal to non-zero then semaphore is shared between the processes.

Third argument valuespecifies the initial value for the semaphore.

On success 0 is returned. On error, -1 is returned.

Example:
```
sem_t s;
sem_init(&s,0,5);
```

❖ 'sem_wait' system call

Library: #include<semaphore.h>

Syntax: int sem_wait(sem_t *sem);

Description: sem_wait system call decrements/locks the semaphore pointed to by sem.

If semaphore value is greater than equal to zero then the decrement proceeds, and the function returns.

If semaphore has the value zero at present then the call blocks until either it becomes possible to perform the decrement i.e. a signal handler interrupts the call

On success, the old and new file descriptor can be used interchangeably.

Example: `sem_t s;`

```
sem_wait(&s);
```

❖ ‘sem_post’ system call

Library: #include<semaphore.h>

Syntax: int sem_post(sem_t *sem);

Description: 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 call will be woken up and proceed to lock the semaphore.

On success 0 is returned. On error, -1 is returned.

Example: `sem_t s;`

```
sem_post(&s);
```

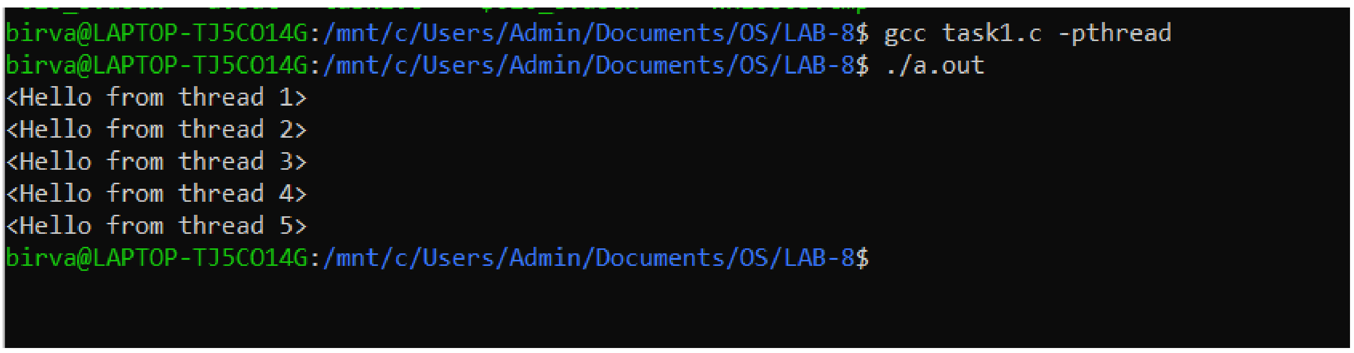
2). Write a program in which 5 threads are sharing and incrementing the value of a global variable using semaphores.

CODE:

```
//WAP for creating 5 threads, which share and increment a global variable count.

//Threads should print the count value.

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

int count = 0;

sem_t s;

void *fun()

{

      sem_wait(&s);

      count++;

      printf("<");

      printf("Hello from thread %d",count);
```

```
	printf(">\n");
	sem_post(&s);
}
void main()
{
	sem_init(&s, 0, 1);
	pthread_t t[5];
	for(int i=0;i<5;i++)
	{
		int status = pthread_create(&t[i], NULL, fun, NULL);
	}
	for(int i=0;i<5;i++)
	{
		pthread_join(t[i],NULL);
	}
}
```

OUTPUT:



```
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-8$ gcc task1.c -pthread
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/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>
birva@LAPTOP-TJ5C014G:/mnt/c/Users/Admin/Documents/OS/LAB-8$
```

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

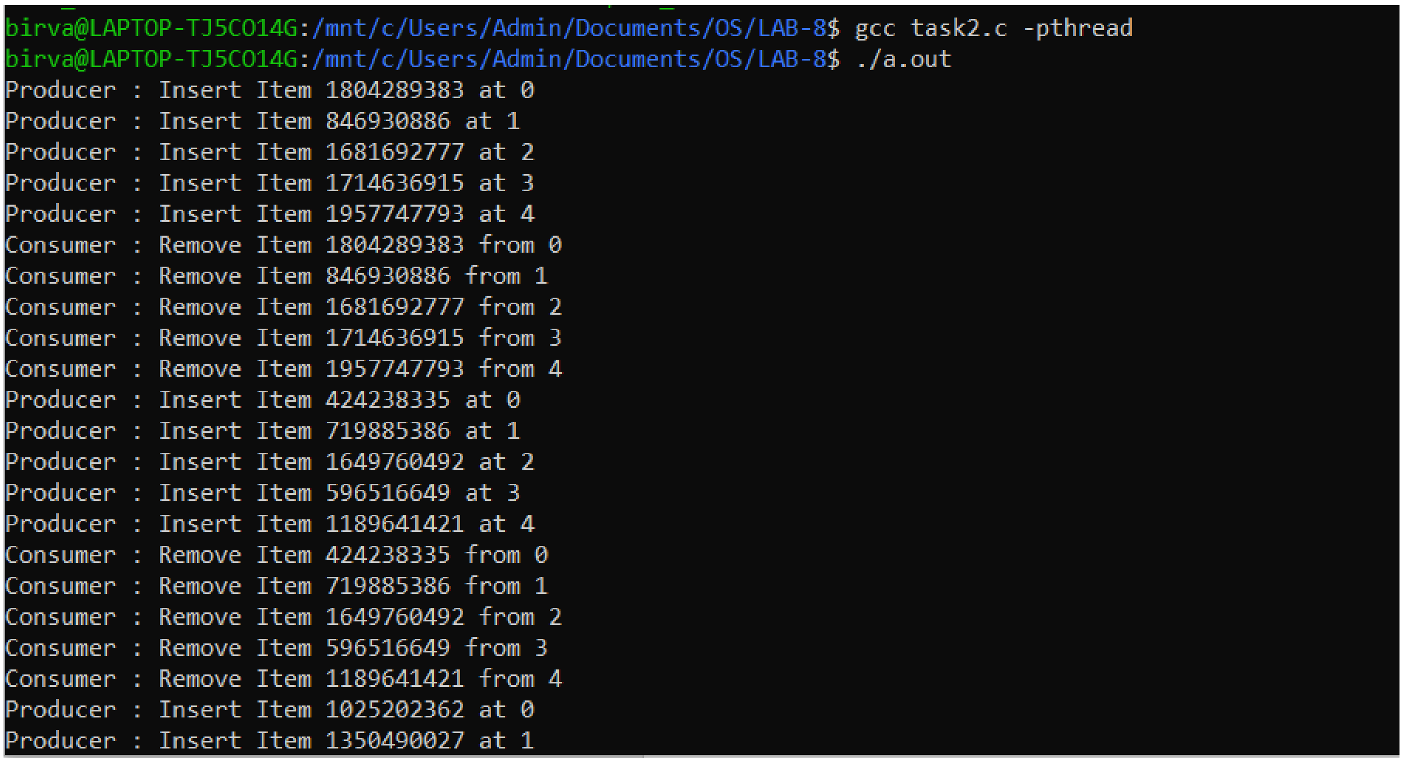
CODE:

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

```
#define MaxItems 5
#define BufferSize 5
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
    int item;
    while (1)
    {
           item = rand();
           sem_wait(&empty);
           pthread_mutex_lock(&mutex);
           buffer[in] = item;
           printf("Producer : Insert Item %d at %d\n", buffer[in], in);
           in = (in + 1) % BufferSize;
           sleep(1);
           pthread_mutex_unlock(&mutex);
           sem_post(&full);
     }
}
void *consumer(void *cno)
{
     while (1)
     {
           sem_wait(&full);
           pthread_mutex_lock(&mutex);
           int item = buffer[out];
           printf("Consumer : Remove Item %d from %d\n", item, out);
           out = (out + 1) % BufferSize;
           sleep(1);
```

```
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
      }
}
int main()
{
      pthread_t pro, con;
      pthread_mutex_init(&mutex, NULL);
      sem_init(&empty, 0, BufferSize);
      sem_init(&full, 0, 0);
      pthread_create(&pro, NULL, (void *)producer, NULL);
      pthread_create(&con, NULL, (void *)consumer, NULL);
      pthread_join(pro, NULL);
      pthread_join(con, NULL);
      pthread_mutex_destroy(&mutex);
      sem_destroy(&empty);
      sem_destroy(&full);
      return 0;
}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-8$ gcc task2.c -pthread
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/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
Producer : Insert Item 1350490027 at 1
```