**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++)

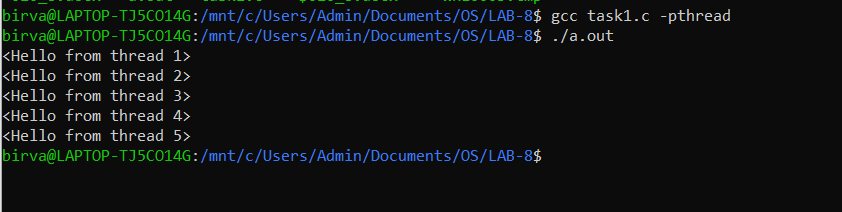
{

pthread\_join(t[i],NULL);

}

}

**OUTPUT:**

****

**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);

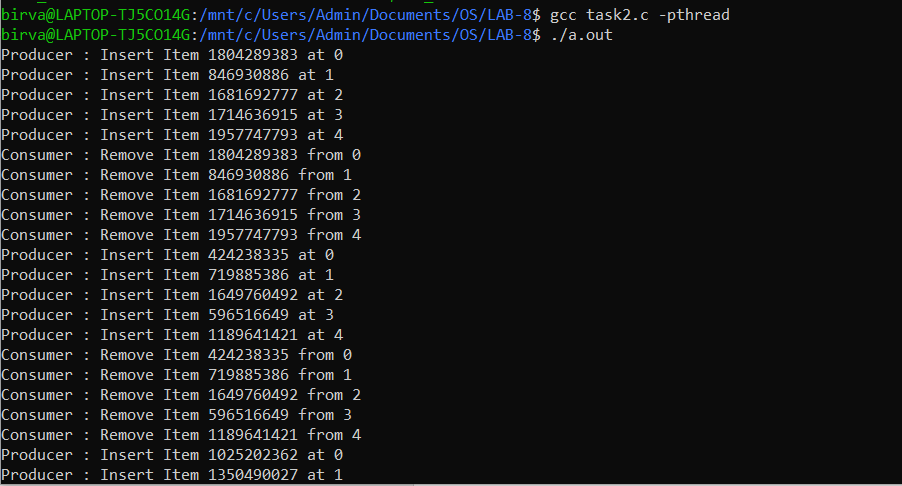
sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

**OUTPUT:**

****