**Exercise 1: Implementation Critical Section Problem by Semaphore**

#include<pthread.h>

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

void \*fun1();

void \*fun2();

int shared=1; //shared variable

sem\_t s; //semaphore variable

int main()

{

sem\_init(&s,0,1); //initialize semaphore variable - 1st argument is address

of variable, 2nd is number of processes sharing semaphore, 3rd argument is the

initial value of semaphore variable

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("Final value of shared is %d\n",shared); //prints the last updated

value of shared variable

}

void \*fun1()

{

int x;

sem\_wait(&s); //executes wait operation on s

x=shared;//thread1 reads value of shared variable

printf("Thread1 reads the value as %d\n",x);

x++; //thread1 increments its value

printf("Local updation by Thread1: %d\n",x);

sleep(1); //thread1 is preempted by thread 2

shared=x; //thread one updates the value of shared variable

printf("Value of shared variable updated by Thread1 is: %d\n",shared);

sem\_post(&s);

}

void \*fun2()

{

int y;

sem\_wait(&s);

y=shared;//thread2 reads value of shared variable

printf("Thread2 reads the value as %d\n",y);

y--; //thread2 increments its value

printf("Local updation by Thread2: %d\n",y);

sleep(1); //thread2 is preempted by thread 1

shared=y; //thread2 updates the value of shared variable

printf("Value of shared variable updated by Thread2 is: %d\n",shared);

sem\_post(&s);

}

**2.Program to simulate Deadlock Using C in Linux using Mutex Locks and threads**

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

void \*function1();

void \*function2();

pthread\_mutex\_t first\_mutex; //mutex lock

pthread\_mutex\_t second\_mutex;

int main() {

pthread\_mutex\_init(&first\_mutex,NULL); //initialize the lock

pthread\_mutex\_init(&second\_mutex,NULL);

pthread\_t one, two;

pthread\_create(&one, NULL, function1, NULL); // create thread

pthread\_create(&two, NULL, function2, NULL);

pthread\_join(one, NULL);

pthread\_join(two, NULL);

printf("Thread joined\n");

}

void \*function1( ) {

pthread\_mutex\_lock(&first\_mutex); // to acquire the resource/mutex lock

printf("Thread ONE acquired first\_mutex\n");

sleep(1);

pthread\_mutex\_lock(&second\_mutex);

printf("Thread ONE acquired second\_mutex\n");

pthread\_mutex\_unlock(&second\_mutex); // to release the resource

printf("Thread ONE released second\_mutex\n");

pthread\_mutex\_unlock(&first\_mutex);

printf("Thread ONE released first\_mutex\n");

}

void \*function2( ) {

pthread\_mutex\_lock(&second\_mutex);

printf("Thread TWO acquired second\_mutex\n");

sleep(1);

pthread\_mutex\_lock(&first\_mutex);

printf("Thread TWO acquired first\_mutex\n");

pthread\_mutex\_unlock(&first\_mutex);

printf("Thread TWO released first\_mutex\n");

pthread\_mutex\_unlock(&second\_mutex);

printf("Thread TWO released second\_mutex\n");

}