**Experiment-10**

**Aim-**Write a program to simulate the concept of semaphores.

**Description-**

The term "semaphore" refers to an integer variable that is shared by several threads. In a multiprocessing context, this variable is utilised to address the critical section problem and establish process synchronisation.

That is, Semaphore is an integer variable which is used in mutually exclusive manner by various concurrent cooperative processes in order to achieve synchronization.

Semaphores are of two types:

1. Binary Semaphore
2. Counting Semaphore
3. **Binary Semaphore -** It can only have two possible values: 0 and 1. Its value is set to one at the start. It's used to implement several procedures to solve critical section problems.
4. **Counting Semaphore-** Its value can range over an unrestricted domain ie. -∞ to +∞

To solve the critical section problem utilising two atomic operations for process synchronisation, wait and signal.

Wait operation is also called sleep, or down operation, and P and Signal operation is also called signal, wake-up, or up operation and V.

Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.

A critical section is surrounded by both operations to implement process synchronization. The critical section of Process P is in between P and V operation.

**Source Code-**

#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

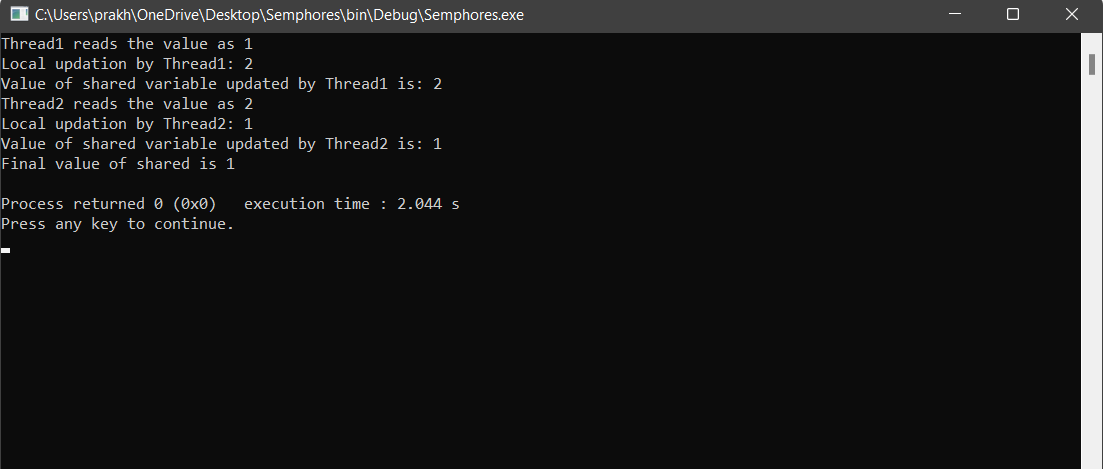
shared=y; //thread2 updates the value of shared variable

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

sem\_post(&s);

}

**Output-**

****

**…………………..**