EX.NO:5	IMPLEMENTATION OF THREADING
DATE :	

AIM :

To write a c program to implement the concept of Threads

ALGORITHM:

1. pthread_create() creates a new thread which starts to execute thread_function
2. pthread_join() makes the main function wait until the newly created thread finishes its execution
3. call pthread_create() function a value if passed to the thread by passing that value as the fourth parameter of the pthread_create() function.
4. main function prints “Inside Main program” and executes the loop from 20-24.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<string.h>
void *thread_function(void *arg);
 int i,n,j;
 int main() {
 char *m="5";
 pthread_t a_thread;  //thread declaration
 void *result;
 pthread_create(&a_thread, NULL, thread_function, m); //thread is created
 pthread_join(a_thread, &result);
 printf("Thread joined\n");
 for(j=20;j<25;j++)
 {
 printf("%d\n",j);
 sleep(1);
 }
 printf("thread returned %s\n",(char *)result);
 }
 void *thread_function(void *arg) {
 int sum=0;
 n=atoi(arg);
for(i=0;i<n;i++)
 {
 printf("%d\n",i);
 sleep(1);
 }
 pthread_exit("Done"); // Thread returns "Done"
 }
```

OUTPUT:

```
 0

 1

 2

 3

 4

 Thread joined

 20

 21

 22

 23
```

24

thread returned Done

RESULT:

Thus the concept of Threads was implemented successfully and verified by using c program.

EX.NO:5	IMPLEMENTATION OF SYNCHRONIZATION APPLICATIONS
DATE :	

AIM:

To implement process synchronization using locks

ALGORITHM:

1. pthread_create() creates a new thread which starts to execute thread_function
2. pthread_join() makes the main function wait until the newly created thread finishes its execution
3. call pthread_create() function a value if passed to the thread by passing that value as the fourth parameter of the pthread_create() function.
4. Create two threads: one to increment the value of a shared variable and second to decrement the value of shared variable
5. The threads acquires the lock and is making changes to shared variable the other thread (even if it preempts the running thread) is not able to acquire the lock

PROGRAM:

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
 int shared=1; //shared variable
 pthread_mutex_t l; //mutex lock
 int main()
 {
 pthread_mutex_init(&l, NULL); //initializing mutex locks
 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;
    printf("Thread1 trying to acquire lock\n");
     pthread_mutex_lock(&l); //thread one acquires the lock. Now thread 2 will not be able to acquire the lock //until it is unlocked by thread 1
   printf("Thread1 acquired lock\n");
    x=shared;//thread one reads value of shared variable
    printf("Thread1 reads the value of shared variable as %d\n",x);
    x++;  //thread one increments its value
    printf("Local updation by Thread1: %d\n",x);
    sleep(1);  //thread one 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);
    pthread_mutex_unlock(&l);
    printf("Thread1 released the lock\n");
 }
void *fun2()
 {
    int y;
    printf("Thread2 trying to acquire lock\n");
    pthread_mutex_lock(&l);
    printf("Thread2 acquired lock\n");
    y=shared;//thread two reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread two increments its value
    printf("Local updation by Thread2: %d\n",y);
    sleep(1); //thread two is preempted by thread 1
    shared=y; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    pthread_mutex_unlock(&l);
    printf("Thread2 released the lock\n");
```

```
}
```

OUTPUT:

```
Thread2 trying to acquire lock
Thread2 acquired lock
Thread2 reads the value as 1
Local updation by Thread2: 0
Thread1 trying to acquire lock
Value of shared variable updated by Thread2 is: 0
Thread2 released the lock
Thread1 acquired lock
Thread1 reads the value of shared variable as 0
Local updation by Thread1: 1
Value of shared variable updated by Thread1 is: 1
Thread1 released the lock
Final value of shared is 1
```

RESULT:

Thus the above c program to implement process synchronization using locks has been executed and verified.