**Threads Synchronization ( Mutex)**

Objective: When multiple threads are running they will invariably need to communicate with each other in order synchronize their execution. One main benefit of using threads is the ease of using synchronization facilities. Threads need to synchronize their activities to effectively interact. This includes: Implicit communication through the modification of shared data Explicit communication by informing each other of events that have occurred. This lab describes the synchronization types available with threads and discusses when and how to use synchronization. There are a few possible methods of synchronizing threads and here we will discuss: Mutual Exclusion (Mutex) Locks Condition Variables

Mutexes and Race Conditions: Mutual exclusion locks (mutexes) can prevent data inconsistencies due to race conditions. A race condition often occurs when two or more threads need to perform operations on the same memory area, but the results of computations depends on the order in which these operations are performed

Exercise 1. To check the race condition while running two threads without

Synchronization

Compile: gcc race.c-o race –lpthread

Run the executable race and observe the ouput.

we should see cnt equal to 2\*NITER at the end of the program.

But we see unexpected results. Find out the reason.

Exercise 2:

**Creating / Destroying Mutexes :**

*pthread\_mutex\_init ( pthread\_mutex\_t mutex, pthread\_mutexattr\_t attr)*

*pthread\_mutex\_destroy ( pthread\_mutex\_t mutex )*

The mutex is initially unlocked. Mutex variables must be of type *pthread\_mutex\_t.* The attr object is used to establish properties for the mutex object, and must be of type pthread\_mutexattr\_t if used (may be specified as NULL to accept defaults). If implemented, the *pthread\_mutexattr\_init( )* and *pthread\_mutexattr\_destroy( )* routines are used to create and destroy mutex attribute objects respectively. pthread\_mutex\_destroy( ) should be used to free a mutex object which is no longer needed.

Program Mutex.C, so that the program always produces the expected output (the value 2\*NITER).

.

To compile a program that uses pthreads *and* posix semaphores, use

gcc -o Mutex mutex.c –lpthread

2. Write C/C++ code to provide synchronization code to the below problem.

In a Company, goods are manufactured and stored in a sharable warehouse. Goods are distributed to the distributor by the company. Goods can't be manufactured when warehouse is full, and at the same time, Goods can't be distributed when warehouse is empty. Implement the above scenario using appropriate operating system concept.