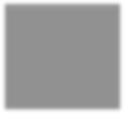
AIR UNIVERSITY, ISLAMABAD



# Department of Computer Sciences

FACULTYOF COMPUTING AND ARTIFICIAL INTELLIGENCE

Operating Systems Lab-CS225L Lab Manual: 09

Topic: Thread Mutex

Lab Instructor:

**Instructions:**

**Plagiarism:** Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in this assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

**Deadline:** Late submission with marks deduction will be accepted according to the course policy shared earlier. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

**Tip:** For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

**Objectives:**

In this lab, you will learn:

* Synchronization
* Mutex
* Shared Variables
* Mutex System Calls
* How to Use Mutex

# Multithreading and Synchronization

**How do the processescommunicate?**

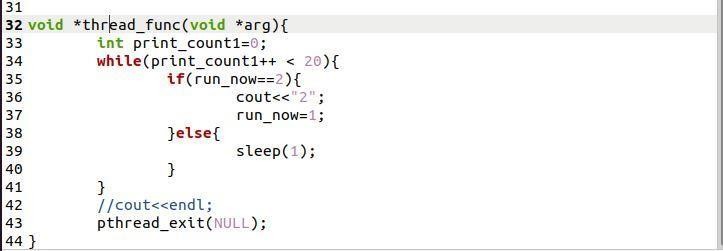
One process should not get into the way of another process when doing critical activities

Proper sequence of execution when dependencies are present

* A produces data, B prints it
* Before printing B should wait while A is producing data
* B is dependent on A

**Example: Synchronization**





**Mutex:**

**What is a Mutex:**

* It is a special variable which act as a mutual exclusion device to protect sections of code (hence the name mutex).
* Simplest and most efficient thread synchronization mechanism

1. It can be either in

* locked state: a distinguished thread that holds or owns the mutex; or
* unlocked state: no thread holds the mutex

1. When several threads compete for a mutex, one wins. The rest block at that call
   * The mutex also has a queue of threads that are waiting to hold the mutex.
2. POSIX does not require that this queue be accessed FIFO

## Mutex and Shared Variables

* + Mutex locks are usually used to protect access to a shared variable.
  + The idea:

o lock the mutex

o critical section

o unlock the mutex

## Mutex System Calls

* + - **pthread\_mutex\_init()**
    - **pthread\_mutex\_destroy()**
    - **pthread\_mutex\_lock()**
    - **pthread\_mutex\_trylock()**
    - **pthread\_mutex\_unlock()**

**Mutex System Calls**

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

Initialize a mutex. The attrargument specifies optional creation attributes.A null value of attrinitializes mutex with default attributes.

0 is returned for success, and on failure an error code is returned

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

Destroy a mutex that you no longer need.

**Int pthread\_mutex\_lock(pthread\_mutex\_t\* mutex)**

Lock a mutex. If the mutex is currently locked, the calling thread is blocked until mutex is unlocked. On return, the thread owns the mutex until it calls pthread\_mutex\_unlock.

**Int pthread\_mutex\_unlock(pthread\_mutex\_t\* mutex);**

UnLock a mutex. The mutex becomes available. If threads are waiting for the mutex, the mutex is assigned to any thread by the operating system.

**int pthread\_mutex\_trylock(pthread\_mutex\_t\* mutex);**

Lock a mutex. If the mutex is currently locked, returns immediately with EBUSY. Otherwise, calling thread becomes owner until it unlocks.

## Steps in Using Mutex

Declare an object of type pthread\_mutex\_t.

* Initialize the object by calling pthread\_mutex\_init().
* Call pthread\_mutex\_lock() to gain exclusive access to the shared data object.
* Call pthread\_mutex\_unlock() to release the exclusive access and allow another thread to use the shared data object.
* Get rid of the object by calling pthread\_mutex\_destroy().

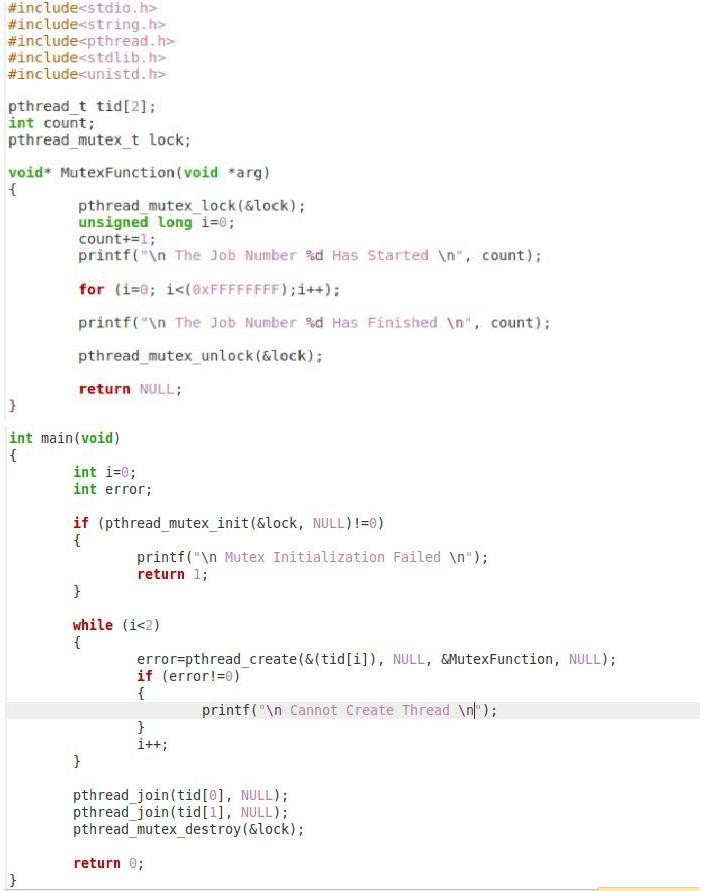
## Things to note while using Mutex

No thread should attempt to lock or unlock a mutex that has not been initialized.

* The thread that locks a mutex must be the thread that unlocks it.
* No thread should have the mutex locked when you destroy the mutex.
* Any mutex that is initialized should eventually be destroyed, but only after any thread that uses it has either terminated or is no longer interesting in using it.

**Example**

**Creating a Program Demonstrating the Usage of Posix Mutex with C Programming**



**Your Task:**

Create a program which have two thread functions, one thread increament in a variable and the other decrement in the same variable after increment. Use Mutex lock and unlock for syncronization of increamenting and decrementing in a variable.