# MUTEX

Lab 07

اللهم علمنا ما ينفعنا،،، وانفعنا بما علمتنا،،، وزدنا علماً

# Lab Objective

- To practice Mutual Exclusion in threads using Mutexes.

# Mutexes

- **`pthreads`** includes support for MUTual Exclusion primitives.

- A mutex is useful for protecting shared data structures from concurrent modifications, and implementing critical sections.

- The idea is to lock the critical section of the code before accessing global variables and to unlock as soon as you are done.

# Mutex Declaration

- A global variable of type **pthread_mutex_t** is required and it's defined as the following:

```
pthread_mutex_t Count_mutex = PTHREAD_MUTEX_INITIALIZER;
```

# Mutex States

- A mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread).

- A mutex can never be owned by two different threads simultaneously.

- A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.

- To lock use:

```
pthread_mutex_lock(&Count_mutex);
```

- To unlock use:

```
pthread_mutex_unlock(&Count_mutex);
```

# Practice

- In the following program, the main process creates two threads of the function `doit`.

- That function has a loop to increment the global variable `counter` by 1 for 10 times.

- The mutex is defined in the program but not utilized around the critical section..

- Write, compile and run the program in Linux then answer the questions in the check-off section.

# Steps

1. Defining and initializing global Mutex (global)
2. Destroying the Mutex (end of main)
3. **Identifying the critical section.**
4. Locking the mutex variable (entry section)
5. Unlocking the mutex variable (exit section)

```cpp
#include <iostream>  #include<stdlib.h> #include<unistd.h>
#include "pthread.h"
using namespace std; //Output a new line
#define NLOOP 10 //Constant value

pthread_mutex_t Count_mutex = PTHREAD_MUTEX_INITIALIZER;

int counter = 0;
void * doit(void *);
int main()
{
       pthread_t tidA, tidB, tidC;
       pthread_create(&tidA, NULL, doit, NULL);
       pthread_create(&tidB, NULL, doit, NULL);
       pthread_create(&tidC, NULL, doit, NULL);

       pthread_join(tidA, NULL);
       pthread_join(tidB, NULL);
       pthread_join(tidC, NULL);

//Leaving a mutex without destorying it may affect system
performance
       pthread_mutex_destroy(&Count_mutex);
       exit(0);
}//end main
```

**Defining and initializing global Mutex**
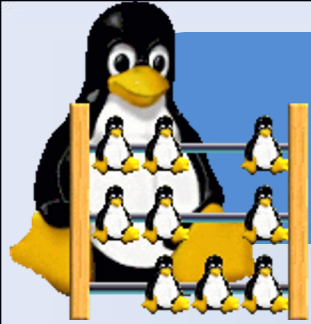
**Destroying the Mutex**

```cpp
void * doit(void *vprt)
{
        int i;
        for( i = 0; i<NLOOP; i++)
        {
                cout<<pthread_self()<<""<<dec<<counter<<endl;
                sleep(1);
                counter ++;
        }
        return (NULL);
} //end doit function
```

# Check Off

1) Compile and run the above program as shown then record the output.

2) Add the required lock and unlock statements around the critical section. Re-compile and run the program then record the new results.

3) Explain the difference between both results.

**Extra:** Change the code so each thread can increment the global variable once then pass it to the next thread and so on, the output should be something as the following:

<div align="center">

tidA 1

tidB 2

tidA 3

tidB 4 …

</div>

??? ANY QUESTIONS ???
☺