**Systems Programming**

**Week 5 – Lab 10**

**Shared data**

pthread_mutex_init(3p) - Linux manual page - man7.org

pthread_mutex_lock(3p) - Linux manual page - man7.org

pthread_mutex_destroy(3p) - Linux manual page - man7.org

The Linux Programming Interface – Sections 30.1

In this laboratory students will start to use locks/mutex to guarantee that the concurrent access to the same variable by multiple threads renders always a correct value.

The supplied code loops over an array with random numbers and counts how many are prime. Each thread iterates over a different set of values and increments a local variable. The main collects the partial counts of each thread (with the **pthread_join**) and sums them to print the total count.

In the following exercises students will implement different versions of this solution where the threads need to access and update shared shared data.

## 1 Global counter

Modify the provided code so that every thread also increments two additional global variables:

- **global_primes** is incremented every time a thread finds a prime number of the array.
- **global_not_prime** is incremented when the evaluated number is not prime

The **main** should print these variables at the end.

**Do not use any mutex to guard this variable.**

Run the code and compare the sum of the partial results with this new value.

## 2 Race condition

Experiment running the code with different data configuration, array length and number of thread:

- delete the comment on line **//rand_num_array[i] = i;**
- increase the array length
- increase number of threads

Try to find an execution where the various sums of number of primes and not primes is note correct, as in the next example with an array of of integers from 0 to 100000:

```
main - total primes 9594
main - Global primes 9574 not primes - 90101
```

## 3 Critical region / Mutual exclusion

Correct the previous code by defining a critical region for each of the global variables.

To guarantee that the access to the critical region does not affect the final result of the program execution, implement mutual exclusion using **pthread_mutex**:

- **pthread_mutex_t** – data type corresponding a mutex
- **pthread_mutex_init** – creation and initialization of a mutex
- **pthread_mutex_lock** – function to call before accessing the critical region
- **pthread_mutex_unlock** – function after leaving the critical region
- **pthread_mutex_destroy** – to destroy the mutex before exiting the program

## 4 Storage of results

Modify the previous program so that each thread stores the prime numbers in a shared array:

- declare a global variable (called **prime_array**) of the same length of **rand_num_array**

- every time a prime number is found, store it in the new variable and increment the global counter of prime numbers.

Use all the necessary mutexes to guarantee that all prime number are correctly stored.

## 5 Random retrieving

Modify the previous program so that the numbers to be processed by each thread are not defined by the programmer/algorithm but are retrieved by each thread after concluding the processing of the previous value.

The program should have a new variable that stores the index of the next number to be processed (**next_random_index**) and all the treads access such value, and process the corresponding random number from the array.

Each thread should also increment this index so that another thread processes the next number.