Tutorial 9 – Threads

1. Copy the following program into a cpp file and compile it (using "g++ -pthread –o outputName source.cpp"). Modify it so that SleepTime is called on a separate thread. **Ensure you add your join in the correct place.** Your first implementation will probably not work as expected (it should count in one second intervals from 0 to 4). Use a mutex so it behaves as expected.

```
#include<cstdlib>
#include<iostream>
#include<unistd.h>
#include<pthread.h>
#include<sstream>


using namespace std;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

using namespace std;

void *SleepTime(void *timeToSleep)
{
	(*((int*)timeToSleep))++;
	sleep(0.1); //simulates race condition
	int duration = *((int*)timeToSleep);
sleep(duration);
	cout << "Slept for " << duration << " seconds." << endl;

	return NULL;
}

int main()
{
	const int NUM_ITTS = 5;
int time_sleep = 0;

	for (int i = 0; i< NUM_ITTS; i++)
	{
		SleepTime((void*)& time_sleep);
	}

	return 0;
}
```

2. Make sure you understand the conditional variable example from blackboard (CondVar.cpp).
3. A) Write a program that allows a user to enter a number that will then be processed to see if it is prime or not. To begin with write a single threaded version.

   Pseudo code is as follows:

```
For (int i=2; i<=test_num/2; i++) //test_num is number that is being tested.
{
        If(test_num%i == 0)
                Return NOT_PRIME
}
Return PRIME.
```

b) Update your code so that multiple threads will attempt to solve this. You will need to think about how you will activate your threads when a new number is given to be processed. You will need to think about how each thread will know what number it is responsible to test is a divisor or not. Is your method faster than the single threaded version?