**CSC 338 Parallel and Distributed Computing**

**Exercise No. 11, November 14, 2017**

**Pthreads and Critical Sections with Mutual Exclusion**

**Goal**

Learn how to handle critical sections with pthreads and mutexes

**Background**

Busy-waiting has the advantage that we can control the order in which threads access a critical section. Threads can burn CPU cycles, though, while waiting for their turn. A *mutex* can be more efficient. Also, an optimizing compiler might not realize that a do-nothing while loop can't be moved to a different place in the code.

**Procedure**

1. The exercise folder has versions of the busy-waiting programs from the last exercise. *pth\_pi\_busy1.c* does busy-waiting inside the loop that adds terms; *pth\_pi\_busy2.c* uses a local sum for each thread, moving the busy-wait loop until after the term-sum loop. Compile and run these programs and compare the run times. Note the use of the timer macro, *timer.h.*
2. Now modify *pth\_pi\_busy2.c* to use a mutex instead of busy-waiting. You won't need the variable *flag* anymore—replace it with a mutual exclusion lock:

pthread\_mutex\_t mutex;

This is a shared variable so it should be declared globally. Don't forget to delete all statements using *flag.*

Before you can use a mutex you have to create and initialize it:

pthread\_mutex\_init(&mutex, NULL);

A good place for this call is right before or after allocating memory for the thread handles.

When you're finished with a mutex you should release the resources:

pthread\_mutex\_destroy(&mutex);

A good place for this is right before or after freeing the thread\_handles memory.

Now use the mutex in the Thread\_sum() function. Remove the do-nothing while loop; replace it with a statement to acquire the lock:

pthread\_mutex\_lock(&mutex);

After the critical section (updating the sum), release the lock:

pthread\_mutex\_unlock(&mutex);

1. Compile and execute your mutex program and compare the timing to the busy-wait programs. Try all three programs with several different numbers of threads and terms. How do the times compare when you have no more threads than cores? Is there a difference when the number of threads exceeds the number of cores?