# CCPS590 Lab 6 – Concurrency with Semaphores

## Preamble

This is the second lab where you will practice concurrent programming, this time using semaphores. Semaphores are a very common tool for implementing concurrency. If you understood last week's lab, this one should be very straightforward.

## Lab Description

1) Compile thread2.c (remember the -lpthread flag). Run it numerous times, and notice the output usually differs.

2) Why is the variable **`myglobal`** usually not 40? Be specific, be precise.

3) In what special case would **`myglobal`** be 40? Be specific, be precise.

4) Learn about POSIX semaphores using the man pages (or just Google). You could also use any web-based man pages if you prefer: i.e. http://linux.die.net/man/7/sem_overview

5) Copy thread2.c to threadSem.c, and modify threadSem.c to use an UNNAMED semaphore to synchronize the update of **`myglobal`**. You must use the semaphore efficiently, i.e., keep the critical sections *as small as possible*! Don't just wrap the entire thread in a semaphore. You may use functions:

```
sem_wait()
sem_post()
sem_init()
sem_destroy()
```

   Run threadSem numerous times to verify **`myglobal`** is always 40.

## Submission

For this lab you will submit only one file - your code threadSem.c. At the top of threadSem.c, include a block comment containing written answers to questions 2 and 3. Your code must compile and run out of the box even with the written answers included.

Labs are to be submitted ***individually***! Make sure your code and written answers are formatted cleanly and are easy to read.