# Homework 6: Semaphores

**cs341**
**Spring 2002**

**Allen B. Downey**
**Computer Science Department**

Due: Thursday 11 April

The purpose of this assignment is to write an implementation of a semaphore using the mutex and condition variable provided by the POSIX thread library, and then to use semaphores to solve a common synchronization problem.

Before starting you should read the rest of Chapter 3 from *POSIX Threads* and Section 8.3.2 in Nutt.

## Get the code

1. Grab the files from

   `http://rocky.wellesley.edu/cs341/code/hw06/`

   and look them over.

2. The files `lock.c` and `lock.h` are a veneer over the pthread mutex. The files `cond.c` and `cond.h` are a veneer over the pthread condition variable. Notice that the two veneers are implemented slightly differently. A lock *contains* a mutex, but a cond *is* a condition variable. Also notice that every time I call a pthread function I check the return value and print error messages.

   In your future life, if you find yourself working with C or C++, I highly recommend writing veneers like this for the libraries you work with. They improve the readability of the rest of your program, which makes it more likely to be correct.

3. I have also provided three versions of `main`, in three files. `main.c` contains a very simple test of the semaphore. You should be able to compile it, but if you run it, it might seg fault.

   `array.c` contains my array-checking code from the last assignment. For now, you should look it over to see what I did to test the lock. You will use this code to test your semaphore implementation.

   `coke.c` contains the skeleton of a coke-delivery simulation. You will fill in the missing code.

4. Finally, take a look at `semaphore.c` and `semaphore.h`. They contain a skeleton of a semaphore implementation.

## Fill in some code

1. Fill in `semaphore.c` and `semaphore.h` with an implementation of a semaphore using a lock and a condition variable.

2. Compile and run `main`.

3. Compile and run `array`.

4. Once you get those working, how confident do you feel that your implementation of semaphores is correct? Which gives you more confidence, examination of the code or testing?

## Coke machine

1. Compile and run `coke`. If you run it a couple of times it is likely that the counter will sometimes be negative, indicating that one of the the synchronization constraints has been violated.

2. The constant `TIME_BETWEEN_COKES` controls the average interarrival time for consumers; the constant `TIME_BETWEEN_REFILLS` is the average interarrival time for producers. Adjust these values so that the producer comes often enough to keep the machine full (or overfull).

3. Add synchronization code to the producer and consumer to enforce exclusive access to the variable `cokes` and to enforce the constraint that the number of the cokes is never negative or greater than the capacity of the machine.

4. Compile and run `coke`.

## What to turn in

Unlike the empirical experiments we have been doing, there is not much on this assignment to measure. It's mostly about the implementation of synchronization mechanisms (and a healthy dose of C).

I would like you to write a report that explains what each part of the assignment is and what you were trying to accomplish. Please present your code in a way that demonstrates it's correctness, as in the *Little Book of Semaphores*. You can leave out details like header files, type definitions, constant definitions, etc. If your program compiles, then those things have to be right. Just present the essential parts of the code, and make a supporting argument that demonstrates their correctness. Also, please test each piece of code carefully and report the results of your tests.