CS 320: Assign 3
Concurrency

For this activity, you will solve three classic concurrency problems using semaphores. You should study goodcnt.c as an example of how to implement and manipulate the semaphores in C. Specifically:

#include <semaphore.h>	Include the semaphore library in your program
sem_t mutex;	Defines a variable named mutex using the sem_t struct;
sem_init(&mutex, 0, 1);	Initializes the semaphore. The parameters are the name (mutex); How it is shared (0 means share within the threads of a process); and the value
sem_wait(&mutex)	Tell the specified semaphore to wait. This will decrement the value of the semaphore by 1 and block if the value is negative
Sem_post(&mutex)	Signals the specified semaphore. This will increment the semaphore by one and alert the first blocked thread (if any).

Specifications:

1. **Producer- Consumer:**

   The producer-consumer problem assumes that there are multiple threads attempting to update the same data structure. The data structure has a limited size (this problem is also sometimes referred to as the "bounded-buffer" problem. Each thread has a specific role in updating the structure. Some add elements to the structure (Producers) and some remove elements from the structure (Consumers). We have a few things that need our attention from a concurrency perspective.

   - To ensure the integrity of the data structure – updates need to be mutally exclusive
   - Consumers cannot remove data that isn't there; if the structure is empty, they need to wait.
   - Since the buffer is bounded; the structure can also get full. This means that the producers will sometimes need to wait.

   The file badpc.c contains an attempt to solve this problem. But as the name implies --- it isn't doing a good job.

   a. Study this code to get familiar with its operations. Remember when you compile, use the –pthread flag

      ```
      gcc –o badpc  badpc.c  -pthread
      ```

   b. Run the code and convince yourself that there are concurrency problems.

   c. Update the code to use semaphores to enforce the concurrency constraints outlined above. Note that the semaphores should be used to help eliminate any busy-waiting

2. **Readers-Writers:**

   The readers-writers problem allows one class of threads to access the data (Readers) and another class of threads to alter the data (Writers). Since the readers don't change the data, there is no limit to how many can access the shared resource simultaneously. However, only one writer can access the data at a time. Moreover, while the data is being written, we don't want readers to report inconsistent data. Therefore a writer must always have exclusive access to the critical section.

   Create a sample program that demonstrates the Readers-Writers scenario. Your shared data should consist of three integer values: x, y, and z.

   a. Create one writer thread that wants to assign odd values to x, y, and z. Create a second writer thread that wants to assign even values. Demonstrate that the readers always display all even or all odd values.

   b. Allow multiple readers to access the data simultaneously. Demonstrate that the readers do not cause starvation in the writers.

3. **Roller-Coaster:**

   Imagine an amusement park where patrons line up to enjoy a ride on a roller coaster. Suppose there are n passenger threads and a car thread. The passengers repeatedly wait to take rides in the car, which can hold C passengers, where C < n. The car can go around the tracks only when it is full.

   Here are some additional details:

   - Passengers actions include board and unboard; the car should be able to load, run and unload. You can think of these as function calls or methods --- but it may be easier just to treat them as print statements.
   - Passengers cannot board until the car has invoked load
   - The car cannot depart until C passengers have boarded.
   - Passengers cannot unboard until the car has invoked unload.

   Write code for the passenger and car threads that enforces these constraints.

Note: If you are stuck the solutions to all of these problems can be found online.
In fact, one great resource is the little book of Semaphores: http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf. This was the source for the description of the Roller Coaster Problem.

If you do use an outside resource:

a. It should be as a last resort after you have worked on the problem for a while on your own.
b. Remember to document a link to the resources you consulted and describe how you used the resource.
c. It won't hurt your grade, but make sure that you *completely* understand any code you submit.