COSC 3360/6310-Operating System Fundamentals
Assignment #3: Post Offices and Semaphores-
(due August 4, 2016 at 11:59:59 PM)

OBJECTIVE

You are to learn how to use Pthreads, POSIX semaphores and Pthread mutexes.

THE PROBLEM

You are to simulate the behavior of customers in a post office. Each of your customers will be simulated by a separate thread created by your main program. Arriving customers will do a `P()` operation on the semaphore representing the number of clerks working in the post office and release that semaphore by doing a `V()` operation on that semaphore when they are done.

Your post office should have a single FIFO queue that you will represent by a single semaphore.

YOUR PROGRAM

All your program inputs parameters will be read from the—redirected—standard input.

The first input line will specify the number of clerks in the post office. Each remaining input line will describe a customer arriving at the post office and will contain three integers representing:

a) A customer serial number,

b) The number of seconds elapsed since the arrival of the previous customer (it will be equal to zero for the first customer); and

c) The number of seconds the customer will take to get processed by the clerk.

One possible set of input could be:

```
5
1  0  10
2  3  5
3  4  8
4  2 75
```

Your program should print out a descriptive message including the customer serial number every time a customer:

a) Arrives at the post office;

b) Starts getting helped; and

c) Leaves the post office.

At the end, your program should display:

a) The total number of customers that got helped;

b) The number of customers that did not have to wait; and

c) The number customers that had to wait.

You will notice that there is no direct way for a customer thread to know whether it actually had to wait before getting helped. Your program should thus keep a count of the number of busy clerks as well as a count of the processes that did not have to wait. It should store these two values in shared `static` variables so all your processes could access them. You should use a separate Pthread mutex to ensure mutual exclusion for all operations accessing the two shared variables.

PTHREADS

1. Don't forget the Pthread include:

   ```
   #include <pthread.h>
   ```

2. All variables that will be shared by all threads must be declared `static` as in:

   ```
   static int nbusyclerks, nzerowaits;
   ```

3. If you want to pass an integer value to your thread function, you should declare it as in:

   ```
   void *customer(void *arg) {
       int serial;
       serial = (int) arg;
       …
   } // customer
   ```

4. To start a thread that will execute the customer function and pass to it an integer value use:

   ```
   pthread_t tid;
   int i;
   …
   pthread_create(&tid, NULL,
          customer, (void *) i);
   ```

5. To wait for the completion of a specific thread, use:

   ```
   pthread_join(tid, NULL);
   ```

Note that the Pthread library has no way to let you wait for an unspecified thread and do the equivalent of:

```
for (i = 0; i < nchildren; i++)
    wait(0);
```

Your main thread will have to keep track of the thread id's of all the threads of all the threads it has created:

```
int child[maxchildren];
for (i = 0; i < nchildren; i++)
    pthread_join(child[i], NULL);
```

POSIX SEMAPHORES

1. Don't forget the following includes:

   ```
   #include <semaphore.h>
   ```

2. To create an ***unnamed*** POSIX semaphore, use:

   ```
   static sem_t mysem;
   int sem_init(sem_t *mysem, 0,
   unsigned initial_value);
   ```

   Note that the semaphore `mysem` must be declared to be `static`, so it can remains visible to all threads. We set the second argument to zero because the semaphore will only be shared among Pthreads.

3. To do a `P()` operation on a semaphore, use:

   ```
   sem_wait(mysem);
   ```

4. To do a `V()` operation on a semaphore, use:

   ```
   sem_post(mysem);
   ```

5. To destroy an unnamed semaphore, use:

   ```
   sem_t mysem;
   sem_destroy(mysem);
   ```

PTHREAD MUTEXES

1. Don't forget the Pthread include:

   ```
   #include <pthread.h>
   ```

2. To be accessible from all threads Pthread mutexes must be declared `static`:

   ```
   static pthread_mutex_t mylock;
   ```

   Your mutex will be automatically initialized to one.

3. To acquire the lock for a given resource, do:

   ```
   pthread_mutex_lock(&mylock);
   ```

4. To release your lock on the resource, do:

   ```
   pthread_mutex_unlock(&mylock);
   ```

All programs using Pthreads or POSIX semaphores must be compiled with the library flag `-lpthread` ***after the list of source code modules as in***

```
g++ postoffice.cpp -lpthread
```

These specifications were modified last on Tuesday, July 19, 2016.