**Problem Statement**

You will use a mutex lock and a condition variable from the pthread library to provide the appropriate synchronization for a program in C that implements resource allocation. The skeleton source code for the program, "resource.c", can be found in the Canvas folders for the course.

**Discussion**

Although the program is written in C, it is designed with an object-based interface for resources:

* The equivalent of a constructor method is resource\_init().
* The equivalent of a destructor method is resource\_reclaim().
* The public methods for a resource are resource\_allocate(), resource\_release(), and resource\_print().

Note that accesses to resource state variables and to synchronization variables should only made in the above methods, and not in the code that uses resource objects. In particular, do not add or change any code in the worker and observer functions.  Instead, search for the keyword ADD in the comments.

Your final version of the program should be consistent with the best practices listed in section 5.5.2 of the textbook.

* A lock is used to synchronize all accesses to the state variables, and a condition variable is used to cause threads to wait on resource availability.
* The lock is acquired at the beginning of methods and released before returns.
* The lock is held whenever a condition variable operation (wait or signal) is called.
* The wait operation on a condition variable is done inside a loop.
* sleep() is not used in the object methods for synchronization.

The program must run on the School of Computing Linux servers (e.g., the titans) to be graded. Compile the program using a command line such as:

gcc -Wall resource.c -pthread

You can use the helgrind synchronization checker using a command line such as:

valgrind --tool=helgrind ./a.out

Please see the course syllabus web page for links to pthread tutorials and the helgrind manual.

Turn in the completed source file named "resource.c" using handin.cs.clemson.edu. Do not submit a compressed source file, and do not submit any other source files.

**Guidelines**

The code should be written totally by yourself.

You may discuss the project requirements and the concepts with me or with anyone in the class.

However, you should not send code to anyone or receive code from anyone, whether by email, printed listings, photos, visual display on a computer/laptop/cell-phone/etc. screen, or any other method of communication.

Do not post the assignment, or a request for help, or your code on any web sites.

The key idea is that you shouldn't short-circuit the learning process for others once you know the answer. (And you shouldn't burden anyone else with inappropriate requests for code or "answers" and thus short-circuit your own learning process.)

**Example Expected Output**

The output should look something like the following. However, note that  
each time you run the program you may see a slightly different order of  
execution and a different mapping of resource id to thread id.

-- resource table for type 1 --  
resource #0: 0,-1  
resource #1: 0,-1  
resource #2: 0,-1  
resource #3: 0,-1  
-------------------------------  
thread #3 uses resource #0  
thread #5 uses resource #1  
thread #4 uses resource #2  
thread #6 uses resource #3  
thread #2 uses resource #0  
thread #7 uses resource #1  
thread #1 uses resource #2  
thread #8 uses resource #3  
thread #9 uses resource #0  
-- resource table for type 1 --  
resource #0: 1,9  
resource #1: 1,7  
resource #2: 1,1  
resource #3: 1,8  
-------------------------------  
thread #11 uses resource #1  
thread #10 uses resource #2  
thread #12 uses resource #3  
thread #13 uses resource #0  
thread #14 uses resource #1  
thread #0 uses resource #2  
thread #15 uses resource #3  
thread #16 uses resource #0  
-- resource table for type 1 --  
resource #0: 1,16  
resource #1: 1,14  
resource #2: 1,0  
resource #3: 1,15  
-------------------------------  
thread #17 uses resource #1  
thread #18 uses resource #2  
thread #19 uses resource #3  
-- resource table for type 1 --  
resource #0: 0,-1  
resource #1: 0,-1  
resource #2: 0,-1  
resource #3: 0,-1  
-------------------------------

When you run the program with helgrind, the last line printed should have an error summary with zero errors, ignoring the count of suppressed errors. E.g.,

==8775==  
==8775== For counts of detected and suppressed errors, rerun with: -v  
==8775== Use --history-level=approx or =none to gain increased speed, at  
==8775== the cost of reduced accuracy of conflicting-access information  
==8775== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1624 from 81)