Section Handout

RSS News Feed Madness

You're writing a concurrent C program to simulate the chaos at Terman the night that Assignment 6: News Feed Aggregator comes due. All 10 TAs are there, ready to answer questions through the night—in fact, the TAs love CS107 students so much that they all stay until the very last student leaves. All 115 CS107 students come to Terman with what they assume is just one bug. Each student waits for one of the 20 computers to become available (and all computers are initially available), and once one becomes available, he logs in, debugs for a while, and then searches for an available TA (all of whom are initially available) to look at his code. The TA informs him of the number of bugs in his code. If there are no bugs, the student rejoices, logs out, and leaves. If his code is still buggy, then the student goes back to his computer and debugs a little more, and repeats the process. If the TA ever reports that the number of bugs is 10 or more, the student gives up, logs out and leaves without finishing.

Each of the 10 TAs sleeps until a student gets her attention. The student shows her his code, she studies the code for a while, reports the number of bugs back the student, reads some of her email, and then goes back to sleep. The very last student needs to wake up all the TAs to tell them that they can all go home. Here is the starting **`main`** function for the Terman Cluster simulation. You cannot change any of this code:

```
#define NUM_TAS         10
#define NUM_STUDENTS    115
#define NUM_MACHINES    20

void main(void)
{
   int i;
   InitThreadPackage(false);
   for (i = 0; i < NUM_TAS; i++) ThreadNew("TA", TA, 1, i);
   for (i = 0; i < NUM_STUDENTS; i++) ThreadNew("Student", Student, 0);
   RunAllThreads();
}

// these simulation functions don't do anything, just "fake"
static int Examine(void);     // for TA, studies student code, returns bug count
static void ReadEmail(void);  // for TA, after helping student, before sleeping
static void Debug(void)       // for Student, to simulate debugging
static void Rejoice(void)     // for Student, after logging out
```

Assume the above helpers are already written and are thread safe, you can just call them when you need to. Your job will be to write the **`TA`** and **`Student`** functions to properly synchronize the different activities and efficiently share the common resources. During section, you'll all agree what global variables and **`Semaphore`**s are needed, and then you'll continue to write the **`TA`** and **`Student`** functions.