CS39002: Operating Systems Lab
Spring 2013

Assignment 3
Due: February 11, 2013, 1 pm

In this assignment, you will learn to use *threads*, also sometimes called *lightweight processes*. The major difference between threads and processes that you have to remember for this assignment is that while two processes do not share their memory by default, two threads created by a process share the same memory. Thus, while for processes, you need to create separate shared memory to share data, for threads of the same process, global variable declarations are enough, they will be shared by all the threads of the process (changes made by one are visible to the others). Although you can use semaphores to synchronize between threads (in the same way as processes do), there are thread-specific calls available for mutual exclusion and event ordering.

In this assignment, we will do almost the same thing as in Assignment 2, but X and Y, instead of being two processes started from two terminals, will be two threads started from the same process. There will be **exactly one copy** of X and **exactly one copy** of Y. Also, you will **not** use any shared memory or semaphores. As mentioned, there is no need of shared memory for threads in the same process. For synchronization, you will only use thread-specific calls for mutual exclusion and event ordering.

Your program will first declare all necessary global variables (for sharing data), and create and initialize (if needed) all synchronization tools that you may need. After that, it will create two threads, one to run the code for X and the other to run the code for Y.

The behavior of Y is the same as earlier, except for a small change as follows. It will increment a counter *C* every time it makes an update to any record. Thus, *C* will contain the number of updates that are not written back to the file at any time. Otherwise, the behavior of Y will be exactly the same as in Assignment 2. Note that if Y runs before X can load the data from the file, it should wait as in Assignment 2.

The behavior of X will be as follows. It will first load the records from the file in the global variables declared. It will then wait for either one of the two conditions to be true:

(1) There are two or more updates that are not written back to the file (that is, $C > 1$).
(2) Five seconds have elapsed.

When X wakes up, it will check *C*. If *C* is 0, X will go to wait on the above conditions again. Otherwise, X will write all the records back to the file, reset *C* back to 0, and wait.

Write a single C program *access.c*. The file with all the records must be named *infile.txt* and must be a text file (not binary). The record for each student is kept in one line in the file and contains the following, separated by one or more spaces: First_name (ascii string, one word with no spaces in between, max. 20 characters), Last_Name (ascii string, one word with no spaces in between, max 20 charaters), Roll_No (integer), CGPA (float). Roll no. is unique for a student, but others may be the same for two students.

Submit the file *access.c*.