CS69001: Computing Lab – I
Autumn 2009

Assignment 6

Synchronization and mutual exclusion using semaphores

Due: October 30, 2009 (Friday)

Part 1 **(20)**

Suppose that you want to implement a search engine for web pages. In order to simplify matters, restrict to a list of movies. A file of 1000+ movies is supplied to you. Each movie record contains a name, a director, a year of release, and a short description. Read information from the file, and store the records in a shared memory segment.

Each movie has a popularity rating not provided in the database file. Generate the rating values randomly as integers between 0 and 100. Store the rating information too in shared memory.

Part 2 **(30)**

A set of processes concurrently search in the movie database for key words. Given a keyword (like surreal), a search process reads all the movie records, identifies only those movies whose descriptions contain the keyword, and prints these movies (names, directors, and release dates only), sorted in the decreasing order of the popularity ratings.

In order that the database server does not become over-loaded by many search processes, allow at most five processes at any instant. Use a counting semaphore to implement this restriction. When a sixth (or seventh or . . .) process attempts to make a search, it has to wait until one (or more) running search processes finish working with the database and signal the semaphore.

Part 3 **(50)**

Now, suppose that there is also an update process which occasionally changes the popularity ratings of one or more movies. Since popularity ratings are usually computed from viewer feedback, these changes are likely to occur in any movie database (although not very frequently, perhaps). An update in the database during an ongoing search is expected to produce inconsistent and/or erroneous results. In view of that, the update process waits until the running search processes finish working with the shared database.

Assume also that the update process has higher priority than the search processes. This means that during the tenure of the update procedure, no new or waiting search process is granted access to the movie database. When updating is complete, the search processes proceed as in Part 2.

Implement this idea using semaphores.

Note

All the relevant tasks (initial database creation, search, and update) may be handled by child processes forked by a single parent process. For example, the parent process may spawn ten search processes which keep on searching in an infinite loop.

Alternatively, you may use different processes from different terminals to carry out different jobs. In that case, all the processes must agree upon the shared memory and semaphore keys. Using `IPC_PRIVATE` during the creation of shared memory segments and/or semaphores impairs this agreement possibility. You should instead use specific key values based upon your roll number (like 09CS6068 will use keys 9680, 9681, 9682, and so on). Another possibility is to use `ftok()` based upon your home directory.

Submit a single C/C++ file solving all the above parts. The file must contain your name and roll number.