

CSIS0230A Principles of Operating Systems(Class A)

Tutorial 10

Reader-Writer with Writer Priority

In the computer stores a solution to the tutorial 9, with the idle time, read time, write time and write fraction modified slightly so that most of the time there is some process wanting to read, and some of the processes do no write at all. If you run the program you will notice when one of them wants to write, it has to wait until some of the processes finishes. As a result the program is not very efficient in using the shared resource.

Modify the program to implement the reader-writer problem with writer priority. Here are some hints.

- You need 3 shared counters, to hold the number of readers in progress (the *active reader count*), the number of waiting readers (the *pending reader count*), and the number of writers that are either in progress or waiting (the *writer count*).
- One semaphore, say *mutex*, can be used to protect all these counters. You should make sure that whenever any counter is used, *mutex* is locked.
- We need another semaphore (called “*wrt*” in tutorial 9) to make sure that at most 1 writer or a set of readers can be working on the shared data.
- The writer should increment the writer count, and lock *wrt* before writing. At the end it should unlock *wrt* and decrement the writer count.
- The reader should check whether the writer count is zero. If so, it should increment the active reader count and start reading. Like the solution of tutorial 9, it needs to lock *wrt* if it is the first reader.
- If the reader find that some writer is in progress, it should start waiting, after incrementing the pending reader count. When a writer finishes, it should check whether it is the last writer. If so, and the pending reader count is non-zero, it should resume all pending readers, and transfer *wrt* to them.
- Try to make sure that *mutex* (the semaphore protecting the counts) will never be held when a process can potentially wait. If you can guarantee that, it is likely that your program will never deadlock.
- Design and argue for correctness **before** actually writing your code!