

## EECS 338 Assignment #5: Semaphores and Shared Memory

Due: Tuesday, April 3<sup>rd</sup>, 2007

Spring 2007, G. Ozsoyoglu

In this assignment you will use the semaphore and shared memory constructs of UNIX system V to implement an adaptation of the fair readers-writers problem. You have seen the algorithmic solution to this problem in midterm 1 of this year.

**Fair Readers-Writers Problem.** Give a semaphore-based solution to the **Fair Readers-Writers Problem where NOBODY STARVES:** When there are readers and writers waiting to read and write (due to a writer or reader in its CS), the service of the system becomes first-come-first-serve (FIFO) with the provision that consecutive FIFO readers are still allowed to read concurrently. You may assume that you know the total number of readers and writers in the system.

Your solution consists of seven processes, four for readers, and three for writers. Arrange two execution scenarios, one in which readers and writers arrive randomly to read and write, respectively, and another one in a carefully controlled manner to demonstrate the correctness of your implementation. For each action, print a message to the screen containing the pid of the involved process, the action, the current time, and the state of the queue.

Also, do not forget to conform to the assignment grading policy requirements listed at <http://art.cwru.edu/338.S07/grpolicy.html>.

### Requirements and Hints:

- Make sure that your shared variables are mutually exclusively modified and/or accessed. You can use the wait() and signal() implementations via systems V semaphore commands, as listed at <http://art.case.edu/338.S07/example.semaphore.html>.
- You will need the rand() and srand() functions (seed srand() with the current time).

Run your program in the script environment, just like in the previous assignments.

On the due date (March 29<sup>th</sup>), submit a hard copy printout of your code and output in the class, and upload your assignment (source and printout) to the blackboard.

Remember to error-check all system calls: check return values for success, and use *perror* when possible on failure.