ECES 338 Assignment #5

Due: March 19, 1999

(100 or 200 points)

Spring 99; Ozsoyoglu, G.

You have two alternatives for this assignment.

(1) Solve, and code the searchers/inserters/deleters problem as given below, using Unix semaphores and shared memory. In this case,the grading will be out of 100 points.

(2) You can define a concurrent programming problem of your own (at a similar difficulty level--I will leave it up to you to judge what "similar difficulty level" means.), give its algorithmic solution **with binary/nonbinary semaphores**, and then code and solve it using Unix System V semaphores (this is a *must*) and Unix system V shared memory (another *must*), in which case the grading will be out of 200 points. And, in this case, you may win the honor of having (a possibly revised version of) your problem and your name chosen as next year's ECES 338 assignment/exam problem.

Searchers/Inserters/Deleters Problem: Three kinds of processes share access to a singly-linked list: searchers, inserters, and deleters. Searchers merely examine the list; hence they can execute concurrently with each other. Inserters add new items to the end of the list; insertions must be mutually exclusive to preclude two inserters from inserting new items at about the same time. However, one insert can proceed in parallel with any number of searches. Finally, deleters remove items from anywhere in the list. At most one deleter process can access the list at a time, and deletion must also be mutually exclusive with searches and insertions. Explicitly specify any additional assumptions you make about the model.

You are to write a program to synchronize searcher, inserter and deleter processes using semaphores. Your main program should fork three searchers, three inserters, and three deleters. The success/failure of each search, insert and delete operation should be printed. In addition, inserters and deleters should print the list after the modification. Searchers should search much more frequently than inserters and deleters; and inserters should insert more frequently than deleters. Implement the linked list with an array. (The array should be large enough so that you will not worry about the boundary condition of filling the array.) If you need to keep shared variables for your processes then you should also use Unix system V shared memory primitives. The system calls that you need to use are **fork**, **execl**, **wait**, and **exit** for concurrent process creation, binary replacement, wait and termination from a single process, **semop, semget** and **semctl** for semaphores, **shmat, shmget** and **shmctl** for shared memory variable acquisition and utilization (if you need any), and **sleep** to force a process to sleep.

Your main program will be very short; it will spawn (fork) nine processes, and exit after all child processes exit. For searcher, inserter, and deleter processes, you will write three C programs, compile them, and store the binaries, say searcher.bin, into a proper directory. When the main program forks out a process, say, searcher1, that process will execute "execl", which will exchange the executing binary by searcher.bin. When these system calls are successfully completed, you will have nine concurrently executing processes (in addition to the main process), each executing in their own logical address space.

To get a shared memory area accessible by all processes, you will need to get shared memory from the O.S.. For this, please study the system calls shmat, shmget, shmctl.

Test your program with a long run, script and turn in the output as well as your source code. It is possible that, during the debugging of your program, you may leave behind runaway processes. Locate them with "ps -aux", and "kill" them. Your processes should terminate after looping around for a while. And, finally, don't forget to clean up any stray semaphores or shared memory segments after each run.