Fall 2015 **EECS 338 Assignment 2** **Due:** Oct. 1st, 2015

G. Ozsoyoglu *Concurrent Programming with Semaphores; 140 points*

(100 pts) **1. Priority-based Searchers/Inserters/Deleters Problem without starvation.** Three types of processes, namely, *searchers, inserters*, and *deleters* share access to a singly linked list L, and perform search, insert, or delete operations, respectively. The list L does not have duplicate values.

a) *Searchers* merely search the list L, and report success (i.e., item searched is in L) or no-success (i.e., item searched is not in L) to a log file. Hence they can execute concurrently with each other.

b) *Inserters* add new items to the end of the list L, and report success (i.e., item is not in L, and successfully inserted into L) or no-success (i.e., item is already in L, and no insertion takes place) to a log file. 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.

c) *Deleters* remove items from anywhere in the list, and report success (i.e., the item is found in L and deleted) or no-success (i.e., item is not in L, and could not be deleted) to a log file. At most one *deleter* can access the list L at a time, and the deletion must be mutually exclusive with searches and insertions.

d) ***Initial start.*** *Searcher, inserter*, and *deleter* processes are initially launched as follows. A user process that needs a search/insertion/deletion operation to the list L first *forks* a process, and then, in the forked process, performs an *execv* into a *searcher/ inserter/deleter* process.

e) ***Log maintenance.*** Upon start, each *searcher/inserter/deleter* writes to a log file, recording the time of insertion, process id, process type (i.e., *searcher, inserter*, or *deleter*), and the item that is being searched/inserted/deleted.

f) ***Termination.*** Upon successful or unsuccessful completion, each *searcher/inserter/deleter* writes to the same log file, recording the time and the result of its execution.

g) ***Priority-based service between three types.*** *Searchers, inserters*, and *deleters* perform their search, insert, delete operations, respectively, ***on a priority basis*** (not on a first-come-first-serve (FCFS) basis) between separate process types (i.e., *searchers, inserters, deleters)* as follows. *Searchers* search with the highest priority; *inserters* insert with the second highest priority (**except that** one inserter can proceed in parallel with any number of searchers*)*, and *deleters* delete with the lowest priority.

h) ***FCFS service within a single type.*** Processes of the same type are serviced FCFS. As an example, among multiple *inserters*, the order of insertions into L is FCFS. Similarly, among multiple *deleters*, the order of deletions into L is FCFS. Note that, among *searchers*, while the start of search among searchers is FCFS, due to concurrent *searcher* execution, the completions of multiple searchers may not be FCFS.

i) ***Starvation avoidance.*** In addition to the above priority-based search/insert/delete operations, the following ***starvation-avoidance rule*** is enforced.

   - ***After 10 consecutive searchers search the list L***, if there is at least one waiting *inserter* or *deleter* then newly arriving *searchers* are blocked until (a) all waiting *inserters* are first serviced FCFS, and, then (b) all waiting *deleters* are serviced FCFS. Then, both the standard priority-based service between process types and the FCFS service within a process type resume.

You are to specify a semaphore-based algorithm to synchronize *searcher, inserter* and *deleter* processes.

Note:

- Explain your algorithm.
- Make sure to state any assumptions you make in your solution.
- Specify the initial states of your variables and semaphores.
- Specify whether your semaphores are binary or nonbinary.
- Do not bother specifying algorithms for sequential tasks: simply specify a well-defined function/procedure (i.e., one with well-defined input/output/functional specification).

(40 pts) **2. Four-of-a-Kind Problem** is defined as follows.

- There is a deck of 24 cards, split into 6 different kinds, 4 cards of each kind.
- There are 4 players (i.e., processes) $P_i, 0 \leq i \leq 3$; each player can hold 4 cards.
- Between each pair of adjacent (i.e., seated next to each other) players, there is a pile of cards.
- The game begins by
  - someone dealing four cards to each player, and putting two cards on the pile between each pair of adjacent players, and
  - P_0 starting the game. If P_0 has four-of-a-kind, P_0 wins. Whoever gets four-of-a-kind first wins.
- Players take turns to play clockwise. That is, P_0 plays, P_1 plays, P_2 plays, P_3 plays, P_0 plays, etc.
- Each player behaves as follows.
  - So long as no one has won, keep playing.
  - If it is my turn and no one has won:
    - Check for Four-of-a-Kind. If yes, claim victory. Otherwise discard a card into the pile on the right; pick up a card from the pile on the left; and, check again: If Four-of-a-Kind, claim victory; otherwise revise turn so that the next player plays and wait for your turn.

- There are no ties; when a player has claimed victory, all other players stop (when their turns to play come up).

You are to specify a semaphore-based algorithm to the Four-of-a-Kind problem.

Note:

- Explain your algorithm.
- Make sure to state any assumptions you make in your solution.
- Specify the initial states of your variables and semaphores.
- Specify whether your semaphores are binary or nonbinary.
- Do not bother specifying algorithms for sequential tasks: simply specify a well-defined function/procedure (i.e., one with well-defined input/output/functional specification).