Name: _____ Student #: _____

Engineering 8893
Concurrent Programming
Final Exam

Dr. D. K. Peters

April 15, 2002

Instructions: Answer all questions. Write your answers on this paper. This is an **closed** book test, no textbooks, notes, calculators or other aids are permitted.
Total points: 100

---

1. [12 points] Consider the following java classes, which implement a proposed solution to the critical section problem. You may assume that `main` creates one instance of `SharedVars` and `n` `Threads`, each of which runs an instance of `Proc`, all of which reference the same `SharedVars`. The `criticalSection` and `nonCriticalSection` functions are defined elsewhere and may require a different amount of time for different processes and different iterations. (The questions are on the following pages.)

```
class SharedVars
{
  public boolean[] flag;
  public int turn;
  public SharedVars(int n) { flag = new boolean[n]; turn = 0; }
}

class Proc implements Runnable
{
  private SharedVars v; private int id; private int n;
/*****************************************************************
 * @param _v shared variables
 * @param i my id
 * @param _n number of processes
 *****************************************************************/
  Proc(SharedVars _v, int i, int _n)
    { v = _v; id = i; n = _n; }

  public void run()
  {
    while (true) {
      nonCriticalSection();
      v.flag[id] = true;
      v.turn = (id+1)%n;
      while (v.turn != id && contention()) skip();
      criticalSection();
```

```
      v.flag[id] = false;
    }
  }

  private boolean contention()
  {
    boolean result = false;
    for (int i = 0; i < n && !result; i++) {
      if (i != id) {
        result = v.flag[i];
      }
    }
    return result;
  }
  private void skip()
    { try { Thread.sleep(50); } catch (InterruptedException e) {} }
}
```

For each of the following essential properties of a critical section solution, state if the above solution satisfies the property or not, and justify your answer.

a) [3 points] Mutual exclusion.

b) [3 points] Absence of deadlock.

c) [3 points] Absence of unnecessary delays.

d) [3 points] Eventual entry.

2. [20 points] *The Roller Coaster Problem.* Suppose there are `n` passenger processes and one car process. Passengers repeatedly wait and take rides in the car, which holds `C` passengers, `C` < `n`. The car will only go around the tracks when it is full.

   You are to develop a monitor to simulate the synchronization of this system. (Assume that the passenger and car processes are defined elsewhere and obey the protocol described below.) The monitor should have public methods as follows:

   `void takeRide(int id)` which is called by a passenger process to indicate that it is ready for another ride. It blocks until the passenger has been loaded into a car and the (full) car has completed its trip.

   `void load()` which is called by the car process to indicate that it is ready to begin loading passengers. It should block until the car is full.

`void unload()` which is called by the car process to indicate that it has completed the trip (and passengers can now disembark).

a) [5 points] Give the class private variables and initial values or constructor implementation.

b) [15 points] Give the (pseudo-Java) implementations for the three methods.

3. [25 points] Throughout this course I have insisted that you not make any assumptions about the fairness of a semaphore. This is what is known as a *blocked-set* semaphore—if more than one process is waiting (i.e., blocked at a call to `P`), then, when it is released (i.e., some process calls `V`), one of the waiting processes will be allowed to proceed, but we don't know which one. Another form of semaphore, known as a *blocked-queue* semaphore, releases processes in FIFO order—the process that has been waiting the longest will be released first. Fortunately, if we know how many processes will share the semaphore, it is possible to implement a blocked-queue binary semaphore using several blocked-set semaphores (as defined in the `Semaphore` class, with methods `P` and `V`) and the technique of 'passing the baton'.

   a) [10 points] Give the pseudo-Java private variable declarations and constructor implementation for the blocked queue semaphore class, using the (blocked-set) `Semaphore` class and the technique of 'passing the baton.' (*Hint:* The maximum number of processes that will share an instance of `FairSemaphore` should be a parameter to the constructor.)

```
class FairSemaphore {
```

   b) [15 points] Give a pseudo-Java implementation of the class methods `P` and `V`, and any private methods you choose to use. (*Hint:* The processes may pass their process Id as a parameter to one or both of these. You may assume that the process Ids are in a reasonable range.)

4. [10 points] Assume there are three symmetric processes, each having a local array of integers, each of which is sorted in non-decreasing order. Assume also that at least one value is common to all three arrays. We would like to find the *smallest* value that is in *all* three arrays. The processes are arranged in a ring, and initialized so that the local variables `left` and `right` are bi-directional channels for asynchronous message passing (using methods "`void send(int m)`" and "`int receive()`") to each neighbour. Each message can be only a single integer.

   Give a pseudo-java implementation of the processes (i.e., the `run` method) so that they interact until each has determined the *smallest* common value. You may neglect `Channel` set-up and closing.

```
class Looker implements Runnable
{
  private Channel left;  // AMP with left neighbour
  private Channel right; // AMP with right neighbour
  private int[] numbers; // the values to search,
                         // (A)i.0 <= i < N-1 -> numbers[i+1] >= numbers[i]
  private int N;         // Number of values in numbers

  // constructor omited

  public void run()
  {
```

5. [15 points] Consider the problem of modeling the motion of N perfectly elastic balls bouncing about in a perfectly elastic box in a zero gravity vacuum (i.e., no friction). The balls have some initial velocity and position, and change velocity when they collide with the box walls or each other, so a sequential algorithm would be as follows:

```
1: loop
2:   for each ball do
3:     if in collision with other ball or wall then
4:       Calculate new velocity
5:     end if
6:     Calculate position at t + δt
7:   end for
8: end loop
```

   Describe the structure (i.e., the distribution of computations and data over processes and the communications and/or synchronization between processes) of at least two possible distributed solutions to this problem. What are the advantages and disadvantages of each?

6. [8 points] Consider the following set of periodic tasks, which share a single processor, but otherwise do not interact. The deadline for completion of each request is the next request for the same task.

Task	Period	Duration
1	120	30
2	200	20
3	30	10

   a) [4 points] What relative priority assignment will be given to these processes under

rate monotonic scheduling?

b) [4 points] Is this a feasible schedule? Justify your answer.