Name: ______________________________

CSc 422, Spring 2005 — Examination 2

You may use up to *two* pages of notes for this exam, but otherwise it is closed book. Write your answers on these sheets, using additional sheets if necessary.

The exam is worth 60 points. There are 5 problems; each is worth 12 points. *You must show your work and/or explain your answers.* This is required for full credit and is helpful for partial credit.

1. A *reduction operation* takes an array of `N` values and combines them into a single value using some binary operator such as plus. Assume you have a program with `N` processes. Write a **monitor** that computes a plus-reduction of `N` values. Use the Signal and Continue discipline. The monitor has one operation with the following header (in MPD syntax):

```
op reduce (int value) returns int sum
```

The operation is to be called `N` times, once by each process. Return the sum of all `N` values to each process. Obviously, you are going to need some synchronization to make this happen!

2. Suppose you have a mean instructor who decides to assign students to two-person groups rather than letting each student pick their own partner. You are the TA for the course, and have been asked to write a **monitor** to create the groups. The monitor has one procedure with the following heading:

```
op formGroup (int userID) returns int partnerID
```

Each student has a unique userID. There are `N` students in the class; assume that `N` is even.

Each student calls the monitor procedure. For every pair of calls, `formGroup` returns to each student the identity of the other student. Do not have `formGroup` wait for all `N` calls. It should process two calls at a time. Use the Signal and Continue discipline.

3. In Homework 3 you developed monitors to solve the memory allocation problem. Now I want you to solve the problem using a server process. Clients are to communicate with the server using **asynchronous message passing.**

The server manages `P` pages of memory. Initially all pages are free. Clients send two kinds of messages to the server—one to request `amount` free pages of memory, and one to release `amount` free pages of memory. (Do not worry about the identities of pages.) Memory can be released in different amounts than it is requested. You may assume that a client does not release more memory than it has been allocated, and that it does not request more than `P` pages in total.

Declare the channels you need, and develop a server process to manage the memory. Your solution must be **fair**, meaning that every request is eventually satisfied. At the bottom of the page, briefly explain why your server is fair.

Channels:

Server process:

Fairness explanation:

4. Suppose that you are writing a distributed parallel program that employs `N` worker processes. There are places in your program where you need barrier synchronization.

Develop code that implements a dissemination barrier for `N` processes using **asynchronous message passing**. Every process should execute the same sequence of code, which should vary only in terms of the identities of the processes. Declare the channels you need, and give the code sequence and local variables for some Worker `i`.

Channels:

Local variables and code sequence for Worker i:

5. Assume that you have a distributed system containing `N` processes on `N` different machines. The collection of processes form a graph that is connected but not complete. In particular, each process knows the identity of—and hence can communicate with—a subset of the `N` processes. This is represented in each process by a set `Peers`. Assume that if `j` is in the `Peers` set of process `i`, then `i` is in the `Peers` set of process `j`.

Each process also has local copies of a collection of files. The names of the files stored on a process is represented as a set `Files` local to that process.

Develop a *heartbeat algorithm* that enables *every* process to learn the name of every file that is stored on at least one of the `N` processes. The processes are to communicate using **asynchronous message passing.** Each process can communicate only with its peers.

Declare the channels you need and give the code to be executed by each process `i`. You will need a loop, which should terminate as soon as you are certain that the process has acquired the entire set of files. Do not have each process automatically execute `N-1` heartbeat rounds.

Channels:

Local variables and code sequence for process i: