CS 536 (Park) **Assignment V** **Due: Sep. 26 (Thu.), 1996**

PROBLEM 1

(a) Read "Measured capacity of an Ethernet" by D. Boggs, J. Mogul, and C. Kent. *Proc. ACM SIGCOMM '88*, pages 222-234, 1988. Give a one-page summary and critique of the paper. Do you agree with their conclusions?

(b) Give a one-page summary of Prof. Grama's guest lecture. Include your view on what is common to the communication problem in parallel computing vis-a-vis distributed/network computing, and what is dissimilar.

PROBLEM 2

Extend the client/server application of Assignment IV such that command-line arguments are supported. Note that redirection (<, >) and pipes (|) are shell features, and need not be implemented. Clearly indicate your new request format. Test your program by first running the server application in the background, then executing four copies of the client in the background "simultaneously" with arguments `finger` *your-hostname*, `ps` *-a -l*, and two more commands of your choice with options.

Use the UNIX `script` facility to record the run-time session. Hand in the output of the script as well as the client and server code. You must provide adequate documentation in your code. (*This hand-in format is the default requirement unless otherwise specified.*)

PROBLEM 3

Implement a *barrier* mechanism using the notion of a monitor, in our case, implemented as an iterative server. The *barrier server* is to have a well-known request FIFO to which clients trying to synchronize register by issueing a `barrier` function call. This should enqueue the client's process ID into the FIFO. After *all* clients have issued `barrier` calls, the server should send a suitable *go-ahead* message back to the blocked clients, allowing them to continue. This feedback should also be implemented as a FIFO, similarly to the client/server example of Assignment IV.

The driver portion of the client code should consist of a `for`-loop iterating 10 times, at each iteration outputting its process ID followed by the `barrier` call. `barrier` is the only function besides the driver codes for the server and client, and it has type definition

void `barrier`(*void*)

First run the barrier server in the background, then start up five copies of the client in the background. Assume, for now, that the number "five" is hardcoded into the server. Assuming the process IDs of the five processes are 100, 200, 300, 400, and 500, respectively, the output of a sample run may look as follows:

```
100 200 300 400 500
200 100 300 400 500
100 200 500 400 300
100 300 200 400 500
200 100 300 400 500
200 100 400 300 500
100 200 300 400 500
100 200 300 400 500
200 500 300 400 100
200 100 300 500 400
```

Indicate possible ways by which you could remove the hardcoded feature of the number of processes participating in the synchronization.

PROBLEM 4

This is an extension of Problem 4, Assignment III. Assume your `send` and `receive` modules in *mysim* are driven, in turn, by higher level modules, call them `source` and `destination`. Assume communication is one-way (from

`source` to `destination`) and `source` has the property of sleeping for time X ms where is X is a random variable with probability distribution $p_X(X)$, then waking up to send Y fixed-size frames where Y has distribution $p_Y(Y)$. Then `source` again becomes idle for X ms repeating this process ad infinitum. Give a pseudo-code specification of *mysim*'s driver program which achieves this outer structure. You need not give the pseudo-code of the various components such as `source`, `destination`, `send`, `receive`, and other modules from your earlier design. Note that the crux of the problem lies in showing how you keep track of the *simulated time* w.r.t. the relevant events occurring in it.