CS 536 (Park) **Assignment V** **Due: Oct. 3 (Fri.), 1997**

PROBLEM 1

(a) Finish reading Chapt. 3 of P & D. Read Section 4.1 of P & D.

(b) 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?

PROBLEM 2

(a) 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*, `/usr/ucb/ps` *-a -l*, and three more commands of your choice with options.

If you haven't done so already, add error checking capabilities to your server such that "Command not found" and other errors can be reported.

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.*)

(b) If you had to implement redirection (<, >) and pipes (|) as part of the server capabilities, how would you go about doing so? That is, what are the technical problems to be solved and what are possible solutions? Be precise in your descriptions so that coding is only a stone throw away.

PROBLEM 3

(a) 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 and iteration count 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(1) 200(1) 300(1) 400(1) 500(1)
200(2) 100(2) 300(2) 400(2) 500(2)
100(3) 200(3) 500(3) 400(3) 300(3)
100(4) 300(4) 200(4) 400(4) 500(4)
200(5) 100(5) 300(5) 400(5) 500(5)
200(6) 100(6) 400(6) 300(6) 500(6)
100(7) 200(7) 300(7) 400(7) 500(7)
100(8) 200(8) 300(8) 400(8) 500(8)
200(9) 500(9) 300(9) 400(9) 100(9)
200(10) 100(10) 300(10) 500(10) 400(10)
```

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

In what applications would a barrier server with the `barrier` call be useful? Give two possible application scenarios.

PROBLEM 4

Assume our goal is to turn a CSMA/CD Ethernet into a token ring. That is, while preserving Ethernet's existing MAC, we want to build a system (in software) that emulates a token ring-based access mechanism with its associated deterministic properties. How would you go about doing that? Give a detailed sketch of a design and explain how it is able to function as a token ring. Given an infinite-source model with 10 hosts attached to a 10 Mb/s Ethernet segment, estimate the throughput achievable in your particular system.