CS 536 (Park) **Assignment IV** **Due: Sep. 19 (Thu.), 1996**

PROBLEM 1

(a) Read "Ethernet: Distributed packet switching for local computer networks" by R. Metcalf and D. Boggs. *Communications of the ACM*, 19(7):395–403, 1976. Give a 1-page summary and critique (required) of the paper. (Due in two weeks.)

(b) Give a 3-page summary overview of the following aspects of UNIX system programming: signals (`sigaction`, `signal`, `kill`, `pause`) and interprocess communication (`pipe`, `mkfifo`, `semget`, `shmctl`). Organize your description around the main system calls (a subset is indicated in parenthesis). Make sure to explain the notions *atomicity*, *blocking/non-blocking*, *synchronous/asynchronous I/O*, and *interrupts*, and in which system calls they play a role.

PROBLEM 2

(a) You will find a client/server application under park/pub/cs536; the client program is `client.c` and the server program is `server.c`. Reverse-engineer the code and explain, in a high-level way, what the server and client are doing. Then, give an "almost" line-by-line explanation of the code—you may group several lines into one unit, as you see fit, and give a modular description. The important criterion is that your answer makes clear that you understand all aspects of the code. If you find inefficiencies, indicate what and why, and suggest possible changes. Include in your answer a clear explanation of what trick is being used to direct the output of the server back to the client and why this is nontrivial.

(b) Extend the client/server application such that any UNIX command (you may ignore argument options) can be requested by the client to be executed on the server and its output returned to the client. The request format should be of the form

process-ID # command #

Your client program is to take as its command-line argument the name of the UNIX command to be executed. 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`, `ps`, `time`, and `hostname`, respectively.

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.

PROBLEM 3

(a) In the approximate analysis of Ethernet's CSMA/CD protocol shown in class, what simplifying assumptions were made to keep the analysis tractable? In what sense were these assumptions *conservative* and enabled us to use the conclusions with some grain of validity?

(b) Consider a CSMA/CD protocol which uses *linear* backoff instead of the exponential backoff mechanism discussed in class. Give reasons why the latter may be preferable to the former and describes situations where this may hold.

PROBLEM 3

(a) Consider a binary message source which pumps out a continuous stream of bits to be sent to the destination at the end of a noisy channel with signal-to-noise ratio Y dB and channel bandwidth W Hz. If digital transmission using NRZ code is employed, what is the maximum achievable data rate (bps)?

Let us further characterize the source by assuming that it is i.i.d. with output 1 occurring with probability p and 0 with probability $1-p$. Define the *entropy* (or "uncertainty" or "information content") of the source as $H(p) = p \log 1/p + (1-p) \log 1/(1-p)$. As a function p, when is $H(p)$ maximized? When is it minimized? Give an intuitive explanation of what maximum and minimum entropy mean in practical terms and relate it to the "compressibility" of data.

Given the channel characteristics above, is it possible that the maximum data rate is a function of p? If so, what is its qualitative relation?

(b) Assume you have a telephone line with specs. given by $X = 30$ dB and $W = 3000$ Hz. What is the maximum achievable data rate, in general, i.e., without considering potential peculiarities of the source? If the message size is denoted by m and $p = 1$, then for fixed X and W, what is the maximum data rate achievable as $m \to \infty$? How would you go about achieving this data rate in practice?

PROBLEM 5

(a) Give a concise pseudo-code description of the ARQ algorithm discussed in class, one for the sender and one for the receiver.

How would you extend it to include a "pre-processing" step where a connection is first established and the receiver is made aware of the sender's first sequence number (assume it need not be 0)? Beware that the link is unreliable. How would you extend ARQ to include a "post-processing" step where the channel is torn down?

(b) Problem 3.12, 3.14 from P & D.