**CS 422 (Park)** **Assignment III** **Due: Mar. 3 (Wed.), 2004**

*Submission instructions: Please type your answers and submit electronic copies using* `turnin` *by 11:59pm on the due date. You may use any number of word processing software (e.g., Framemaker, Word, LaTeX), but the final output must be in pdf or ps format that uses standard fonts (a practical test is to check if the pdf/ps file prints on a CS Department printer without problem). For experiments and programming assignments that involve output to terminal, please use* `script` *to record the output and submit the output file. Use* `gnuplot` *to plot graphs. Use* `ps2gif` *to convert a eps/ps plot to gif format (e.g., for inclusion in Word).*

**PROBLEM 1** (15 pts)

Read Chapters 8, 9, and 10 from Comer. Pick one problem from Chapter 8 and solve it. Solve Problems 9.6 (related to PROBLEM 2 below) and 10.4.

**PROBLEM 2** (20 pts)

The purpose of this problem is to introduce "sniffing" raw data from the "wire" on the private LAN consisting of two Xinu machines, two laptops, and Ethernet hub. Execute the following command on a Xinu machine to capture a packet from the network:

% `/usr/local/etc/sudo /usr/local/etc/snoopwrap-elxl0 -c1 -o /var/tmp/login-ID`

The command will capture a packet from the Ethernet LAN and save it in the file `/var/tmp/login-ID`, where `login-ID` is your login ID. Generate your own traffic on the private network by doing telnet to communicate with a machine on the private net (`telnet 10.0.0.X`). Submit the log file in octal format. Using the Ethernet header format(s) discussed in class, decode from the octal dump what the values for the fields in the Ethernet header are. Use `/sbin/ifconfig` to compare the Ethernet addresses that you have snooped with those printed out by `ifconfig`. From the value of the type/length field, can you determine whether the Ethernet frame is a DIX- or IEEE 802.3-compliant frame? How long is the payload of the Ethernet frame? Can you relate the payload length and payload content to the telnet session that you've been conducting (the default IP and TCP header sizes are 20 bytes each)? Noting that the first four bits of an IP header indicate its version number (4 or 6), locate the version number bits in the Ethernet payload and identify the IP version running on your test machine. Noting that the last 8 bytes of a 20-byte IP header represent the 4-byte IP source address and 4-byte IP destination address, respectively, locate the IP source/destination address bits in the payload and compare their values to those output by `ifconfig`.

**PROBLEM 3** ($50 = 30 + 20$ pts)

**(a)** As a continuation of Problem 6, Assignment II, extend the client/server application such that any UNIX command can be requested by the client to be executed by the server and its output returned to the client. The request format should be of the form

*# process-ID # command # argument-*1 *# argument-*2 *# ⋯ # argument-n #*

where *argument-1, argument-2, ..., argument-n* represent command-line arguments (possibly empty), and "#" is a delimiter symbol. Test your program by first running the server application in the background, then executing the client `client.bin` multiple times with `date`, `host`, `ps -l`, `ps -l -a`, `ls`, `ls -a -l -i`, `servus`. Use `script` to record the run-time session. Submit your code and output. You must provide adequate documentation in your code. (*Remark: Make use of string processing library functions to minimize the parsing effort.*)

**(b)** As a continuation of the problem above, modify the client code such that its first argument is a real number (`float` type) representing a timeout parameter, *mytimeout*, followed by the UNIX command to be requested. The client, before sending out the service request, sets a timer using `ualarm`() to *mytimeout*. If the response from the server does not arrive before the timer expires (i.e., the SIGALRM signal is raised), the request is retransmitted. Use `sigaction`() to register a SIGALRM signal handler that performs the retransmission. After 7 tries it should give up and terminate with the message "no response from server". There is also a change to the server. First modify the

message format so that the timeout value is transmitted with the command payload:

$$\#\ \textit{process-ID}\ \#\ \textit{mytimeout}\ \#\ \textit{command}\ \#\ \textit{argument-1}\ \#\ \textit{argument-2}\ \#\ \cdots\ \#\ \textit{argument-n}\ \#$$

The server process, upon receiving the message request, hands the timeout value along with the command requested to its child process. The child process, after forking, sleeps for *mytimeout* seconds using `usleep`() before executing the requested command. Test your program by first running the server application in the background, then executing `client.bin` multiple times with `1.0 date`, `3.0 date`, `5.0 date`, `0.1 date`, `0.05 date`, `0.01 date`, and `0.001 date` as arguments. Use `script` to record your output. What do you observe and why?

**PROBLEM 4** ($40 = 30 + 10$ pts)

**(a)** 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—purely in software—that emulates a token ring. A token is passed around a logical (or virtual) ring and only a host in possession of the token is allowed to send a single packet after which the token is passed on. First, a logical ring needs to be set up and the token assigned to one of the hosts. How would you go about building such software? Give a detailed sketch of a design and explain how it is able to function correctly as a token ring. *Constraints: Your inter-host communication must use CSMA/CD Ethernet, i.e., speaking IP or other higher layer communication protocols is not allowed. Separate your design into two parts: one residing in the kernel and the other living in user space. Provide a system call interface between the two subsystems.*

**(b)** Answer the following questions about PROBLEM 4(a) and your specific design. Explain your reasoning. (i) In your design, can the token get lost? (ii) Can a collision ever occur once the token ring emulation system has been set up and is ready to go? (iii) When, if ever, is emulating a token ring over Ethernet a good idea from a throughput perspective? (iv) Assuming the underlying Ethernet is 10 Mbps (IEEE 802.3 10BaseT), there are 10 hosts connected to the same Ethernet LAN, and you need to transmit a 10 MB file using Ethernet frames with payload size 1500 B, how long will it take to complete the file transfer in your system (both in the best case and worst case)?