CS 536 (Park) **Assignment VIII** **Due: Oct. 21 (Mon.), 1996**

PROBLEM 1

(a) Read Section 6.1 and 6.2 from Chapt. 6 of P & D.

(b) Write a one-page preliminary proposal of your potential project idea. Confer the project descriptions in my homepage for a menu of possibilities; you may come up with your own initiatives. You will be asked in 2-week's time to submit a finalized proposal; this one is for exploratory purposes and your final proposal may even be on a different topic. Submit the proposal on a separate page with your name and student ID.

PROBLEM 2

Implement an end-to-end stop-and-wait protocol for reliable transport on top of UDP. This entails both writing a `send` and `receive` module.

To make the problem more interesting, also write a `route` module which performs a routing function, thus acting as a gateway. Assuming xinu$i$ is running the `send` process, xinu$i+1$ the `route` process, and xinu$i+2$ the `receive` process (the numbering is modulo the number of machines in the Xinu Lab), `send` and `receive` are always to send their packets to the `route` process on xinu$i+1$, the default gateway. Let `route` listen on port $8000+i+1$; for this to work, you need to be physically in the lab sitting at xinu$i+1$. Assume the `receive` process listens for packets on UDP port $9000+i+2$; use port $7000+i$ for `send`.

The packet should have the following header format: $\langle$*source IP address*,*destination IP address*,*source port number*, *destination port number*, *packet sequence number*$\rangle$ for a total of $4+4+2+2+4=16$ bytes. `route` is to use the header information to craft the appropriate UDP packet to forward the payload of the received packet. Do not use the information contained in the UDP packet for this purpose. `route`, in this instance, is trivial in that it has no routing table to confer.

Before invoking `sendto`, with probability 7/8 force `route` to sleep for 1 second by invoking the `sleep` system call with argument 1. This emulates nontrivial network delay. With probability 1/8, "drop" the packet.

Test your protocol by making `send` transmit a 20kB file using a packet payload size of 1024 bytes (i.e., 1kB MTU); thus, ideally, only 20 send calls would be needed. You are to run your stop-and-wait scheme five times, with timeout values at `send` of 1 sec, 2 sec, 3 sec, 4 sec, and 5 sec, respectively, given as command-line arguments. To verify that your protocol is working correctly, at both `send` and `receive`, before sending a packet and after receiving a packet, print its sequence number followed by the letter 's' for *s*end and 'r' for *r*eceived. Set the ACK packet's MTU to 512 bytes.

Finally, at `send`, measure the reliable transmission completion time by logging the time before the first send, logging the time of the final ACK received, taking their difference and printing it. Plot the completion time as a function of timeout. Discuss what timeout value is the most "efficient" for the system at hand. Hand in the script files and source code files. Run all your processes, including `route` in the foreground; alternatively, make sure to clean up by terminating all your processes.