CSCI 473: Network Programming

Final Exam -- May 7, 1992

Write your answers on your own paper. This exam is to be turned in by 4:20PM.

Problem 1. (15 points)

Outline how the following process can be implemented in Unix. Don't write real C, but write a pseudo-C that shows how you would use the Unix system calls.

Process *Interleave* reads characters on file descriptors 5 and 6. As soon as a character is available on either on these input file descriptors, *Interleave* reads that character and writes it on file descriptor 7. *Interleave* terminates when input is no long available on either input file descriptor.

Problem 2. (15 points)

How are two pipes very similar to two sockets?

How is one pipe very different from one socket?

Problem 3. (10 points)

Sam Sna says that Unix socket programming is a pain, because you have to know the machine and port numbers of remote processes. Write a paragraph or two to educate him!

Problem 4. (15 points)

Vivian Vaxcluster says that Unix socket programming is a pain and that all networking programming problems could be solved by use of a universal network file system. If you want to send mail to `turing` on the machine `ivy`, open the network file `/ivy/usr/spool/mail/turing` and write your message. The network file system does the rest. Write a paragraph to two to educate her!

Problem 5. (15 points) --

This semester we have seen at least eight ways processes can communicate: files, signals, pipes, sockets, pseudo terminals, Sun RPC, shared memory, and ports. For each of these eight IPC mechanism, state one task for which it is better suited than any of the rest.

Problem 6. (15 points)

The pseudo-terminal interface in Unix provides a mechanism for ``fooling" a process into thinking it is connected to a physical terminal. In standard Unix distributions, all master pseudo-terminals are protected so that any user can read or write to them; however, the kernel does not allow a master pseudo-terminal to be opened more than once simultaneously. For example, if two processes attempt to open **/dev/ptyp5** at nearly the same time, the first will succeed and the second will fail due to an ``I/O error."

Subproblem 6A:

It's a general rule that no Unix file or device should be universally writable, yet an exception is made for master pseudo-terminals. Why is it necessary to leave the master pseudo-terminals so unprotected?

Subproblem 6B:

What mischief could you perform if you were able to open and write to master pseudo-terminals being used by other processes?

Problem 7. (15 points)

Protection in Unix is often based on user ids. For example, processes and files have user ids and, usually, a process can open a file for writing if the user ids of the two match.

Consider extending this style of protection to sockets of the Internet domain. That is, think of the possibilities and problems that would occur if sockets had user ids that reflected the identities of their ``owners."

First, discuss how the security of network applications could be enhanced by user owned sockets. Would such a scheme make Kerberos unnecessary?

Next, outline an implementation of user-owned sockets. How would you face the problem of assigning unique user id's? Or, would you face the problem? How would you authenticate user id's across machines?

Finally, how would a network application running under a significant different operating system, such as MS/DOS, access a user-owned socket on a Unix machine?