IPC Problems

We have looked at IPC applications using pipes and sockets. We have used pipes to pass data from one process to another. Using sockets, we have written a date/time server, a web server, a license server. In these three cases there is a single process that has info we want: the time, some files and directories, permission to run an application. There are many other sorts of reasons for programs to transmit data from one process to another. Today we consider three applications that require a different sort of communication.

A. A Notice Board - many sources, many destinations

Consider a TV news station that displays news stories as they come in from correspondents across the globe. For simplicity, consider each news article is written on an index card with 24 lines and 80 columns and is given a sequence number. Reporters across the planet can send in stories to the TV station. Viewers can tune in the news station and will see the stories appear on their screen in the order the stories are submitted to the station.

Now consider a computerized version of this system. One wants a system that allows users to submit messages to a notice system. People who 'tune in' to the notice channel will see these messages appear on their screen as they are submitted to the station.

A fancy version would allow you to pick up where you left off if you 'tune out' and later 'tune in.'

B. A Print Spooler - many sources, one destination

Consider a computer with one printer. All devices on a Unix machine appear as files in the `/dev` directory. If the file permission on `/dev/lp` allow it, any user could `open()` and `write()` data to that device. The problem is that is several users opened the device at the same time, the output would get intermixed.

The solution is to have one program that runs as a manager of the printer. When a user wants to print a file, he or she sends a request to a program called the print spooler. The spooler accepts requests in the order of submission and sends the files to the printer.

C. Copy from Disk to Tape - simultaneous reads and writes

Early on in the course, we looked at how to copy a file. It was easy, one calls `open()` to open the source file for reading and then calls `open()` to open the destination file for writing. Then, the program `read()`s a buffer of data from the source file, then `write()`s the buffer to the destination file. This `read()-write()` loop is repeated until the contents of the file are transferred.

Note that the program alternates between reading and writing. When it is reading, it is not writing, and vice versa. This is simplistic, because write really transfers data to kernel buffers and schedules writing. Nonetheless, getting kernel buffers could take a while. Or, the program could be reading from a disk file and writing to a socket.

How can one devise a program that can read and write at the same time?