Named Pipes

A pipe is a data channel in the kernel with two ends. It is created by a process and may be shared with children of that process. A pipe cannot be attached to an unrelated process.

A *named pipe* or *FIFO* is a pipe that exists independently of any processes and may be connected to by any two processes that want to communicate. Consider a piece of garden hose lying on the lawn. Anyone can walk up to this hose and place his ear to one end. Another person can walk up to the hose and place his mouth to the other end. These two unrelated people may communicate via this pipe. If there are several pipes lying around, it is useful to give a unique name to each pipe. That way, if you want to chat with someone you can arrange to chat using a particular pipe. That is the idea behind named pipes.

Creating A FIFO

The Unix command `mkfifo` creates a named pipe. You simply type: `mkfifo name-of-pipe` where 'name-of-pipe' is any filename. You can delete a pipe with the standard `rm` command.

Putting Your Ear to a FIFO

To listen at a FIFO, call `open()` as you would for any file. Open it for reading if you plan to put your ear to the fifo. `open()` will block until another process opens the fifo for writing.

Putting Your Mouth to a FIFO

To speak into a FIFO, call `open()` as you would for any file. Open it for writing. The process blocked on open for reading will proceed. You may now use `write()` to send data into the fifo, and the listening will use `read()` to receive data from the fifo.

Use `close()` when Done

Since these are normal file descriptors, you use the standard `close()` system call to indicate you are done speaking. The reading end will receive the end of file indication. The reader can then close the reading end.

A Time/Date Server Using a FIFO

The following pair of shell scripts shows how easy it is to implement the old 'at the tone, the time will be' client-server pair.

```
#!/bin/sh
# time server
    while true ; do
        rm -f /tmp/time_fifo
        mkfifo /tmp/time_fifo
        date > /tmp/time_fifo
    done

#!/bin/sh
# time client
    cat /tmp/time_fifo
```

The Server Could be the Reading End

The example above shows that the server is the writing end. It blocks until some client opens the fifo for reading. In other applications, the server could open the fifo for reading and the client could send stuff into the fifo. Can you think of an example where that might be useful?

Programming with FIFO's: One Small Trick

In the server example, the script deletes the fifo and creates a new each time through the loop. The description says that closing the fifo is enough to send an end of file stataus to the reader. If the reader is not reading or does not close the fifo quickly enough, the writer will not block, but will instead be connected again to the same reader. Unlinking and remaking the fifo solves the problem.