Topics: Pipes

Approach: Build up to "who | sort" and bc

Today's System Calls: pipe(int pfd[2]), dup(int fd), fcntl(fd, F_DUPFD, choice)

Outline

I. A Real-World use for i/o redirection and pipes: watch.sh

Shows ease of shell programming vs C programming
flexibility of simple software tools
use and value of i/o redirection and pipes

II. Facts about Standard I/O and Redirection

Every process gets file descriptors 0,1, and 2 preopened
Most software tools write to stdandard output (fd=1)
The shell redirects the output, the program does *not* redirect it

III. How to Attach stdin to a File

Fact: open uses the lowest number available slot in the fd array
Method 1: close(0), open(filename,0)
Method 2: fd = open(file); close(0); dup(fd); close(fd);
Method 3: fd = open(file); dup2(fd,0); close(fd);
Puzzler: how could you attach a file to fd#10?

IV. Redirecting I/O for Another Program writing who > userlist

redirect output (as shown in previous section)
then exec() the program.
Note: Open files are copied across an exec.
Recall: code and data are replaced,
but environment and files are copied to new program space

V. Introduction to Pipes

implementing: `who | sort`

1. goal: attach fd 1 of `who` to fd 0 of `sort`
2. method: create a pipe, fork, redirect, exec

examples

1. pipedemo.c: shows creation and simple use of pipe
2. pipedemo2.c: shows pipes across forks, multiple writers
   Question: what about multiple readers?
3. pipe.c: shows how to connect processes
   Question: what about `ls | head`

technical details

1. writes are atomic
2. pipe has fixed capacity
3. what if all copies of writing end are closed?
4. what if all copies of reading end are closed?
5. exercise: monitor pipe as in `who | tee file1 | sort`