CSCI 4061: Pipes and FIFOs

Chris Kauffman

Last Updated:
Mon Mar 25 21:53:00 CDT 2019

Logistics

Reading

- Stevens/Rago
  Ch 15.1-5
- OR Robbins and Robbins
  Ch 6.1-6.5

Goals

- `select()`: Multiplexing I/O
- Pipes (Unnamed)
- FIFOs (Named pipe)

Lab07: select(), signals

How did it go?

Project 2

- Kauffman not happy with delay
- You will be happier with result

Exercise: Warm-up

Lab08 `select()`

- What is `select()` used for? Identify some "real world" situations in which you might need to use it in code.
- Describe some macros/data types that are associated with using `select()`
- What kinds of arguments does `select()` take?
  - First peculiar argument
  - Middle 3 arguments
  - Final argument

Recall: Pipes

- What's a pipe?
- How does one set up a pipe in C?
- How does one set up a pipe on the command line shell?

Answers: Warm-up

Lab08 `select()`

- What is `select()` used for? Identify some "real world" situations in which you might need to use it in code.
  Used to determine which inputs are ready for reading to avoid blocking on read(). Useful when input sources have differing rates such as multiple children or a server with multiple clients.
- Describe some macros/data types that are associated with using `select()`
  Uses `fd_set` type, set of file descriptors, use macros like `FD_ZERO(fds)`, `FD_SET(num,fds)` and `FD_ISSET(num,fds)`
- What kinds of arguments does `select()` take?
  1. Maximum file descriptor + 1
  2-4. `fd_sets` to check for Read, Write, Errors
  5. `struct timeval *` describing time to wait on fd_{sets}

Recall: Pipes

- What's a pipe?
  Communication buffer to allow programs to talk to one another, typically output of one program becomes input to another
- How does one set up a pipe in C?

```
int pipe_fds[2];
pipe(pipe_fds);   // 2 file descriptors for read/write now in array
```

- How does one set up a pipe on the command line shell?

```
$> cmd1 | cmd2
```

Pipes and Pipelines

- Have discussed pipes considerably
- Unix **pipelines** allow simple programs to combine to solve new problems

Count the files in a directory

- Solution 1: write a C program using `readdir()` in a counting loop
- Solution 2: `ls`, then count by hand
- Solution 3: `ls > tmp.txt`, count lines in file
- **Pipe Solution**

  ```
  > ls | wc -l
  ```

  `wc -l file` counts lines from file / stdin

History

McIlroy noticed that much of the time command shells passed the output file from one program as input to another. His ideas were implemented in 1973 when ("in one feverish night", wrote McIlroy) Ken Thompson added the pipe() system call and pipes to the shell and several utilities in Version 3 Unix. "The next day", McIlroy continued, "saw an unforgettable orgy of one-liners as everybody joined in the excitement of plumbing."
– Wikipedia: Unix Pipes

- Pipe solutions alleviate need for temporary files

A historical note

"Programming Pearls" by Jon Bentley, CACM 1986 with special guests

- Donald Knuth, godfather of CS
- Doug McIlroy, inventor of Unix pipes

Problem statement: Top-K words

Given a text file and an integer K, print the K most common words in the file (and the number of their occurrences) in decreasing frequency.

Knuth's Solution:

- ~8 pages of text and pseudo-code / Pascal
- Demonstration of "literate programming" [1] so may be a bit more verbose than needed

McIlroy's Solution?

[1] Literate Programming is a Knuth invention involving writing code interspersed detailed, formatted comments describing it. A program is then used to extract and compile the code.

Pipeline for Top-K Words

McIlroy's Solution (Roughly)

```
#!/bin/bash
#
# usage: topk.sh <K> <file>
K=$1                        # arg1 is K value
file=$2                     # arg2 is file to search

cat $file                 | # Feed input \
tr -sc 'A-Za-z' '\n'      | # Translate non-alpha to newline \
tr 'A-Z' 'a-z'            | # Upper to lower case \
sort                      | # Duh \
uniq -c                   | # Merge repeated, add counts \
sort -rn                  | # Sort in reverse numerical order \
head -n $K                  # Print only top 10 lines
```

- 9 lines of shell script / piped Unix commands
- Original was not a script so was only 6 lines long

Exercise: Tool Familiarity

- It is not possible to write complex pipelines unless you are somewhat familiar with each component
- Getting basic familiarity with available Unix tools can save you TONs of work
- Note: solutions don't necessarily involve pipelines

Diff between `DirA` and `DirB`

- Have two directories `DirA` and `DirB` with about 250 of mostly identical files
- Some files exist in only one directory, some files differ between them
- Want the *difference* between the directories

Find Phone Numbers

We have 50,000 HTML files in a Unix directory tree, under a directory called `/website`. We have 2 days to get a list of file paths to the editorial staff. You need to give me a list of the .html files in this directory tree that appear to contain phone numbers in the following two formats: (xxx) xxx-xxxx and xxx-xxx-xxxx.

From: The Five Essential Phone-Screen Questions, by Steve Yegge

Answers: Tool Familiarity

Diff between `DirA` and `DirB`

```
> find lectures/ | wc -l        # 247 files in lectures/
    247     247    9149
> find lectures-copy/ | wc -l   # 246 files in lectures-copy
    246     246   15001
> diff -rq lectures/ ~/tmp/lectures-copy
Files lectures/09-pipes-fifos.org and lectures-copy/09-pipes-fifos.org differ
Files lectures/09-pipes-fifos.pdf and lectures-copy/09-pipes-fifos.pdf differ
Files lectures/09-pipes-fifos.tex and lectures-copy/09-pipes-fifos.tex differ
Only in lectures/: new-file.txt
```

Find Phone Numbers

Here's one of many possible solutions to the problem:

```
grep -l -R \
  --perl-regexp "\b(\(\d{3}\)\s*|\d{3}-)\d{3}-\d{4}\b" * \
  > output.txt
```

But I don't even expect candidates to get that far, really. If they say, after hearing the question, "Um... grep?" then they're probably OK.

– Steve Yegge

Exercise: Pipes have a limited size

In Linux, the size of the buffer is 65536 bytes (64KB).
– Wikipedia: Unix Pipes

- Examine the program `fill_pipe.c`
- Observe the behavior of programs as pipes fill up
- Relate this to **a major flaw** in Project 1 `commando`
  Hint: when did `cmd_fetch_output()` get called...

Answer: Pipes have a limited size

- `commando` set up child processes to write into pipes for their standard output
- `commando` used calls to `waitpid()` to wait until a child was finished, THEN read all child output from the pipe
- Children would call `write()` to generate output going into pipes
- If the pipe filled up, the child's `write()` would block
- `commando` would be waiting on blocked child but never empty the pipe to allow it to proceed
- End result: child never finishes

This is an example of deadlock: protocol used by cooperating entities ends with both getting stuck waiting for the other

- Resolutions for `commando`?

Convenience Functions for Pipes

C standard library gives some convenience functions for use with FILE* for pipes. Demoed in pager_demo.c

```
#include <stdio.h>

FILE *popen(const char *cmdstring, const char *type);
// Does a fork and exec to execute the cmdstring and returns a
// standard I/O file pointer.
// If type is "r", the file pointer is connected to the standard
// output of cmdstring.
// If type is "w", the file pointer is connected to the standard input
// of cmdstring.
// Returns: file pointer if OK, NULL on error

int pclose(FILE *fp);
// The pclose function closes the standard I/O stream, waits for the
// command to terminate, and returns the termination status of the
// shell.
```

Figures below from Stevens/Rago

Figure 15.9 Result of `fp = popen(cmdstring, "r")`

Figure 15.10 Result of `fp = popen(cmdstring, "w")`

FIFO: Named Pipe

- Major limitation of pipes is that they must be created by a parent and shared with a child
- No way for two unrelated processes to share a pipe...
  Or is there?

First In First Out

- A Unix **FIFO** or **named pipe** is a pipe which has a place in the file system
- Can be created with either a shell command or via C calls

Command/Call	Effect
`mkfifo filename`	Create a FIFO on the command shell
`int mkfifo(char *path, mode_t perms)`	System call to create a FIFO

Working with Fifos

A FIFO looks like a normal file but it is not

```
> mkfifo my.fifo                                  # Create a FIFO
> ls -l my.fifo
  prw-rw---- 1 kauffman kauffman 0 Oct 24 12:05 my.fifo
# ^ it's a 'p' for Pipe

> echo 'Hello there!' > my.fifo                   # write to pipe
# hung  C-c

> echo 'Hello there!' > my.fifo &                 # write to pipe in background job
[1] 1797
> cat my.fifo                                     # read from pipe
Hello there!                                      # got what was written in
[1]+  Done    echo 'Hello there!' > my.fifo       # writer finished

> cat my.fifo                                     # read from pipe (nothing there)
# hung  C-c

> cat my.fifo &                                   # read from pipe in background job
[1] 1933
> echo 'Hello there!' > my.fifo                   # write to pipe
Hello there!
>
[1]+  Done    cat my.fifo                         # reader finished
```

A Few Oddities for FIFOs

> *In the normal case (without `O_NONBLOCK`), an `open()` for read-only blocks until some other process opens the FIFO for writing. Similarly, an `open()` for write-only blocks until some other process opens the FIFO for reading.*
> *– Stevens/Rago pg 553 (15.5 on FIFOs)*

- Explains why following hangs

  ```
  > echo 'Hello there!' > my.fifo                # write to
  ```

- No other process is reading from the FIFO yet
- Much harder to set up non-blocking I/O in terminals and likely not worth it
- Also requires **care** to make sure processes writing to FIFOs don't hang because no reader exists

Exercise: Differences Between Pipes/FIFOs and Files

- OS manages position for read/write in both Files and FIFOs but in subtly different ways
- `multiple_writes.c` forks a child, both parent and child write different messages into a File or FIFO

Scenarios: Predict what happens

1. Process opens normal file, forks, Parent / Child write.

   ```
   > multiple_writes prefork file tmp.txt 20
   ```

2. Process forks, opens file, Parent / Child write.

   ```
   > multiple_writes postfork file tmp.txt 20
   ```

3. Process opens a FIFO, forks, Parent / Child write.

   ```
   > multiple_writes prefork fifo tmp.fifo 20
   ```

4. Proccess forks, opens FIFO, Parent / Child write.

   ```
   > multiple_writes postfork fifo tmp.fifo 20
   ```

Answers: Differences Between Pipes/FIFOs and Files

Scenarios

1. Process opens normal file, forks, Parent / Child write. File position in file is shared.

   ```
   > multiple_writes prefork file tmp.txt 20
   ```

2. Process forks, opens file, Parent / Child write. File position is NOT shared so will overwrite each other in file.

   ```
   > multiple_writes postfork file tmp.txt 20
   ```

3. Process opens a FIFO, forks, Parent / Child write. Both hang until something reads the pipe but all data is present.

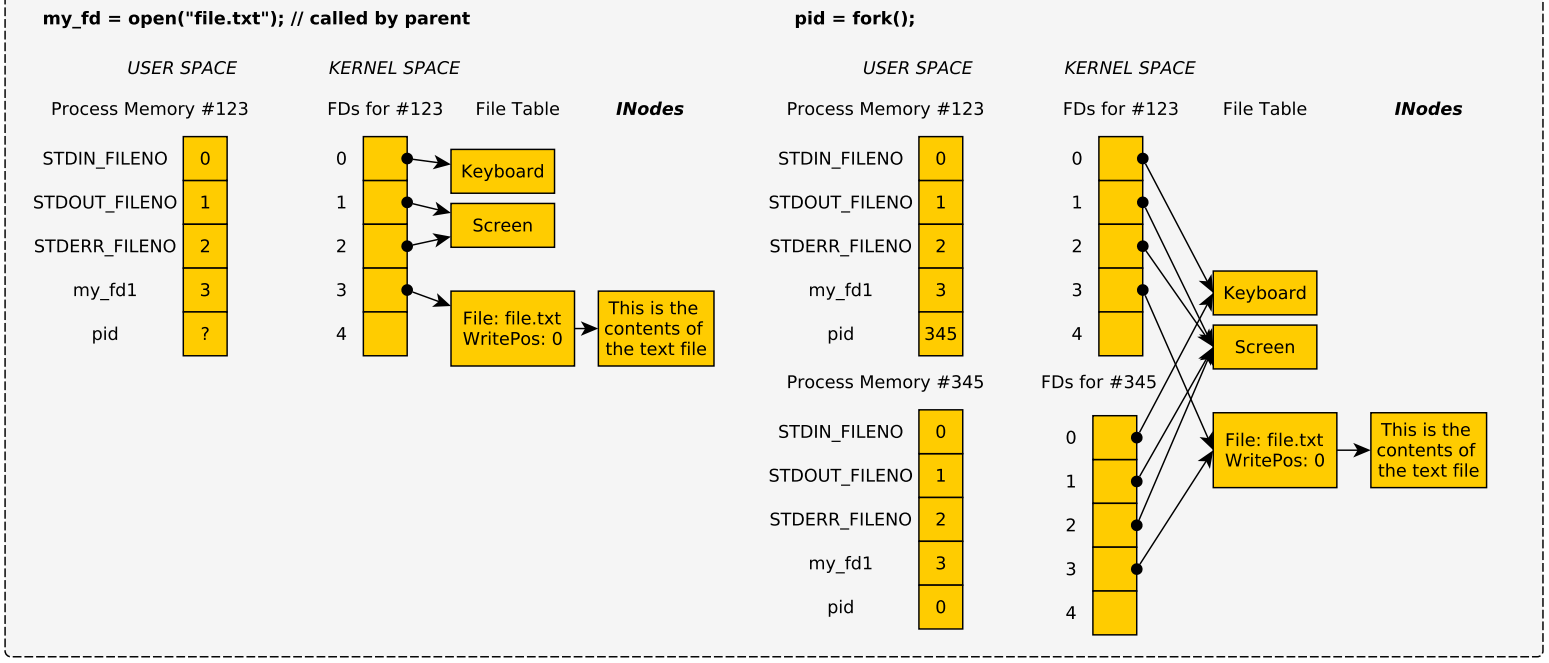
   ```
   > multiple_writes prefork fifo tmp.fifo 20
   ```

4. Proccess forks, opens FIFO, Parent / Child write. Both hang until something reads the pipe but all data is present.

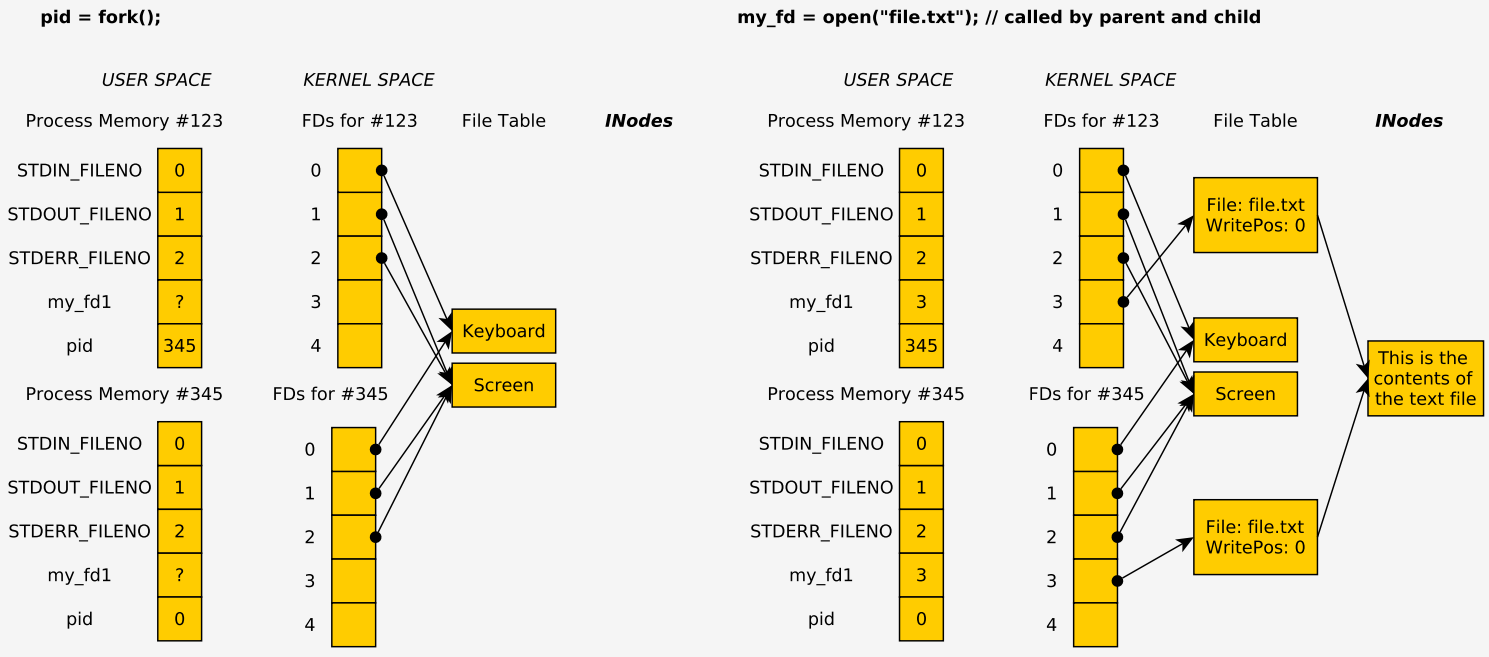
   ```
   > multiple_writes postfork fifo tmp.fifo 20
   ```

Draw some pictures of the internal FD table, Open file table, and INodes to support these.

open() normal file then call fork()

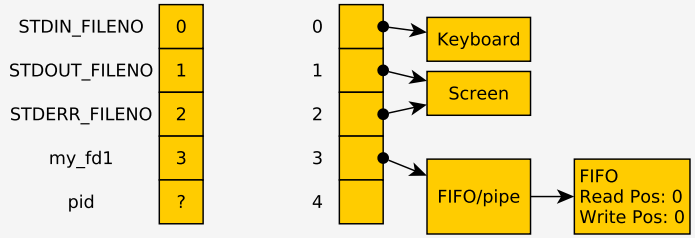


fork() then call open() normal file

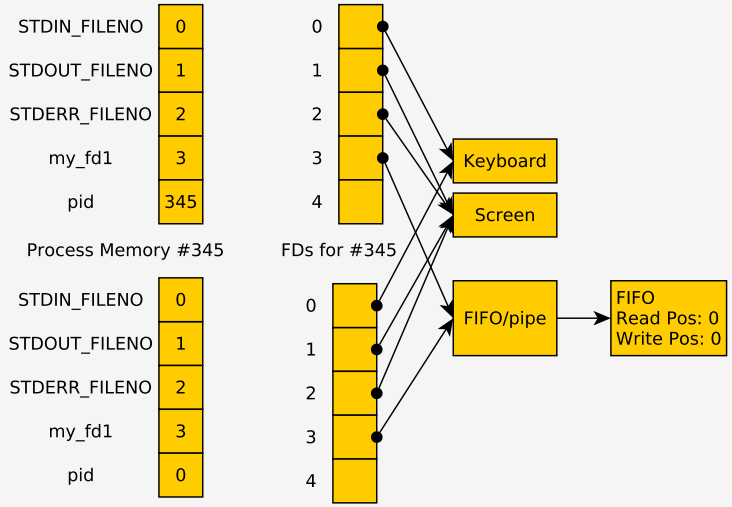


open() FIFO then call fork()

my_fd = open("my.fifo"); // called by parent

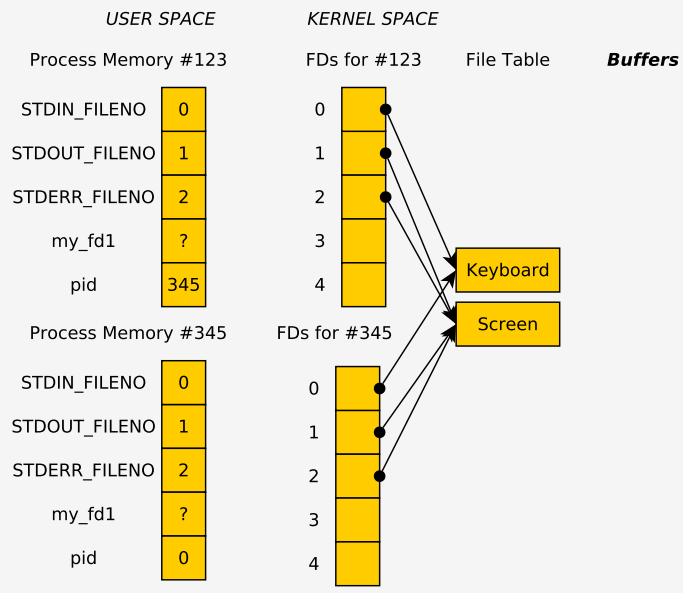


pid = fork();

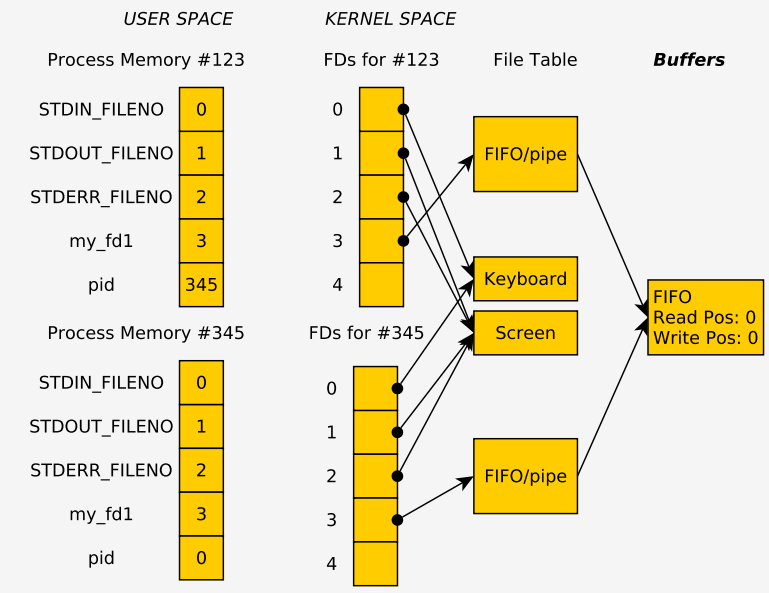


fork() then call open() FIFO

pid = fork();



my_fd = open("my.fifo"); // called by parent and child



Lessons on OS Treatment of Files/Pipes

File Descriptor Table

- One per process but stored in kernel space
- Each numbered entry refers to system wide File Table

INodes

Contains actual file and contents, corresponds to physical storage

Buffers for Pipes / Fifos

Internal kernel storage, Read/Write positions managed by kernel

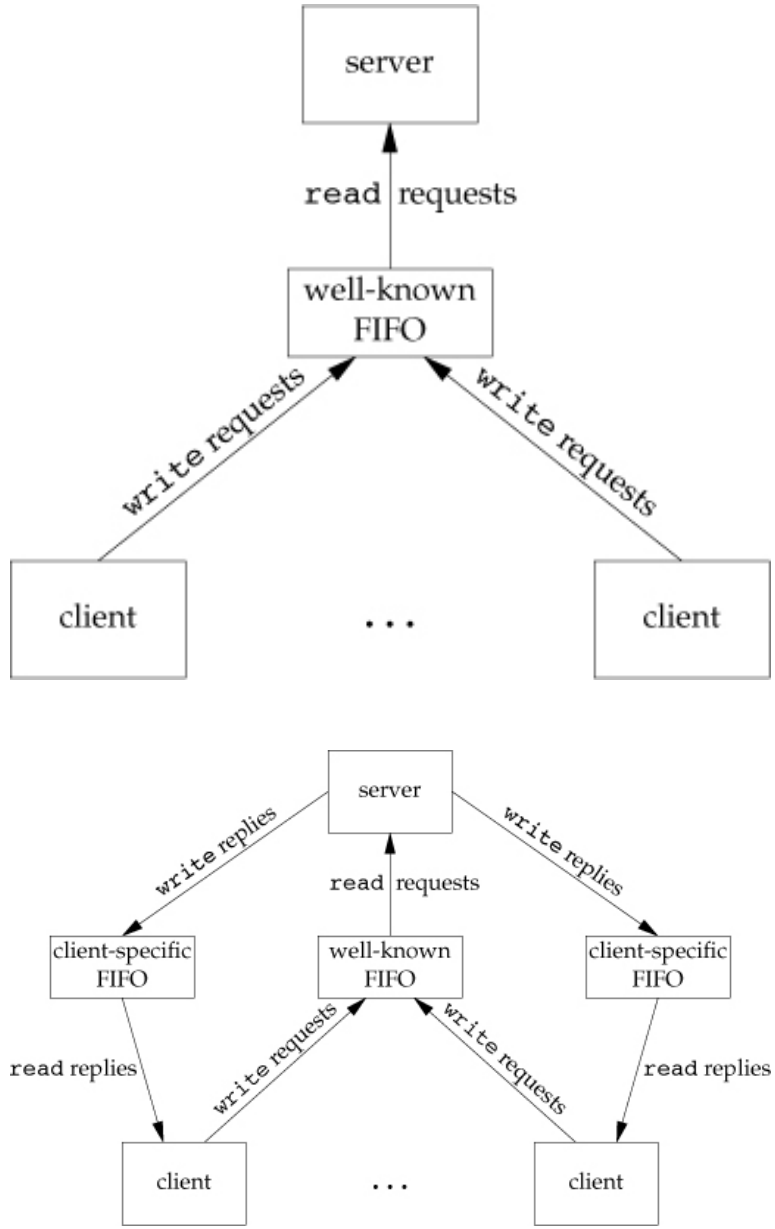
System File Table

- Shared by entire system, managed by the OS
- Each entry corresponds to open "thing" in use by a proc
- May have multiple file table entries per "real" file
- Each File Table Entry has its own Read/Write positions
- Connects File Descriptor Table to INodes, Buffers

Servers/Clients with FIFOs

- Create simple communication protocols
- Server which has names/email addresses
- Clients which have names, want email addresses
- Server are Daemon always running
- Client uses FIFOs to make requests to server and coordinate
- Basics of message passing between processes

Upcoming lab will discuss this, will be used for a project later in the semester



Source: Stevens and Rago Ch 15.5