Advanced Programming in the UNIX Environment

Week 08, Segment 3:
Pipes and FIFOs

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

pipe(2)

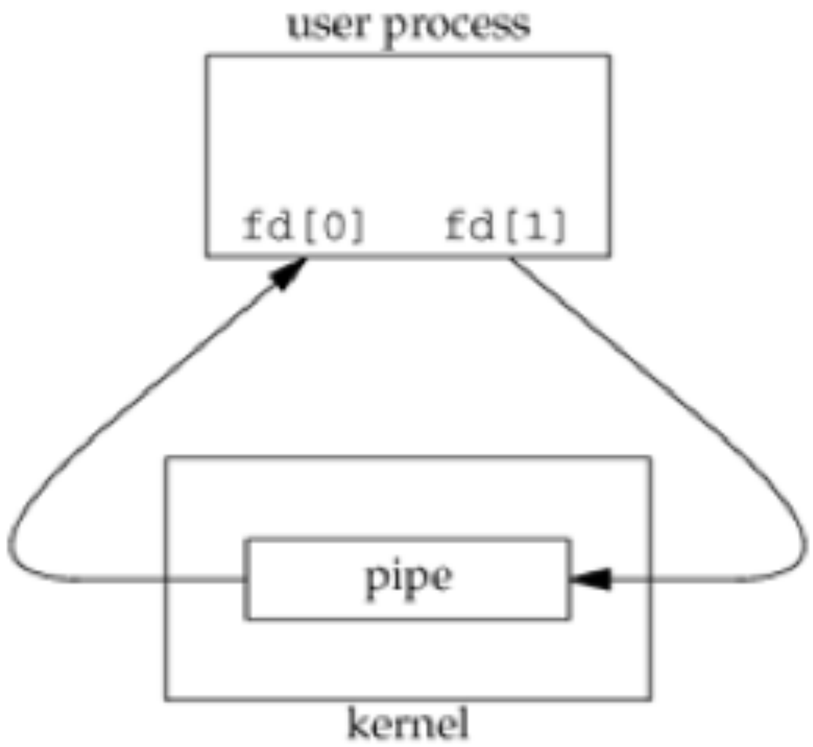
```
#include <unistd.h>
int pipe(int filedes[2]);
```

Returns: 0 ok, -1 otherwise

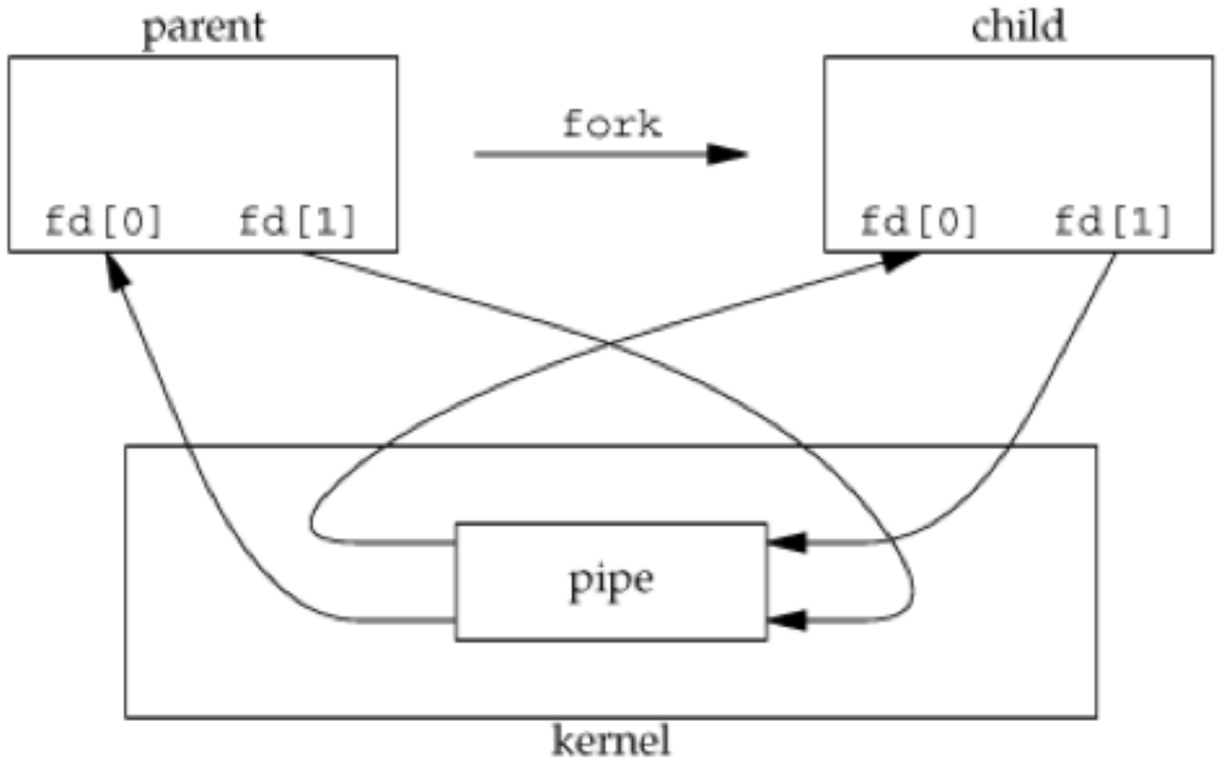
```
$ proc1 | proc2
```

- oldest and most common form of IPC
- usually unidirectional / half-duplex

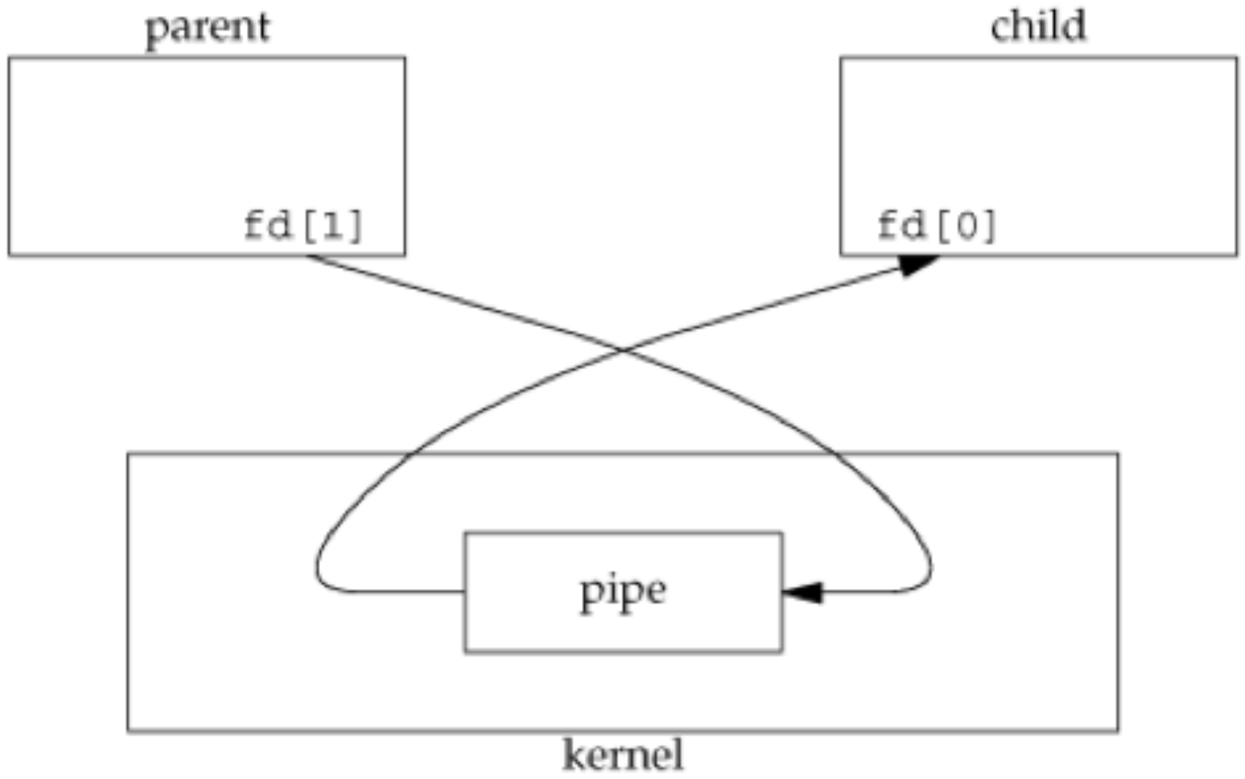
pipe(2)



pipe(2)



pipe(2)



```
jschauma@apue$ echo $$
41
jschauma@apue$ ./a.out | cat
Hello child!  I'm your parent, pid 19553!
C=> Child process with pid 19987 (and its ppid 19553).
C=> Reading a message from the parent (pid 19553):
P=> Parent process with pid 19553 (and its ppid 41).
P=> Sending a message to the child process (pid 19987).
jschauma@apue$ vim pipe1.c
jschauma@apue$ cc -Wall -Werror -Wextra pipe1.c
jschauma@apue$ ./a.out
P=> Parent process with pid 23766 (and its ppid 41).
C=> Child process with pid 16451 (and its ppid 23766).
P=> Sending a message to the child process (pid 16451).
C=> Reading a message from the parent (pid 23766):
jschauma@apue$ Hello child!  I'm your parent, pid 23766!

jschauma@apue$ ./a.out | cat
P=> Parent process with pid 20061 (and its ppid 41).
P=> Sending a message to the child process (pid 22902).
Hello child!  I'm your parent, pid 20061!
C=> Child process with pid 22902 (and its ppid 1).
C=> Reading a message from the parent (pid 1):
jschauma@apue$
```

```
        /* NOTREACHED */
}
jschauma@apue$ export PAGER=/usr/bin/wc
jschauma@apue$ ./a.out pipe2.c
      102     309    2180
jschauma@apue$ unset PAGER
jschauma@apue$ vim pipe2.c
jschauma@apue$ cc -Wall -Werror -Wextra pipe2.c
jschauma@apue$ ./a.out pipe2.c
[1] + Stopped                 ./a.out pipe2.c
jschauma@apue$ ps -o pid,ppid,stat,com
ps: com: keyword not found
  PID  PPID STAT
   41   701 Ss
19864 20415 T
20415    41 T
23863    41 O+
jschauma@apue$ ps -o pid,ppid,stat,comm
  PID  PPID STAT COMMAND
   41   701 Ss   -sh
19864 20415 T    tar
20415    41 T    ./a.out
24739    41 O+   ps
jschauma@apue$
```

popen(3)

```
#include <stdio.h>

FILE *popen(const char *cmd, const char *type);
```

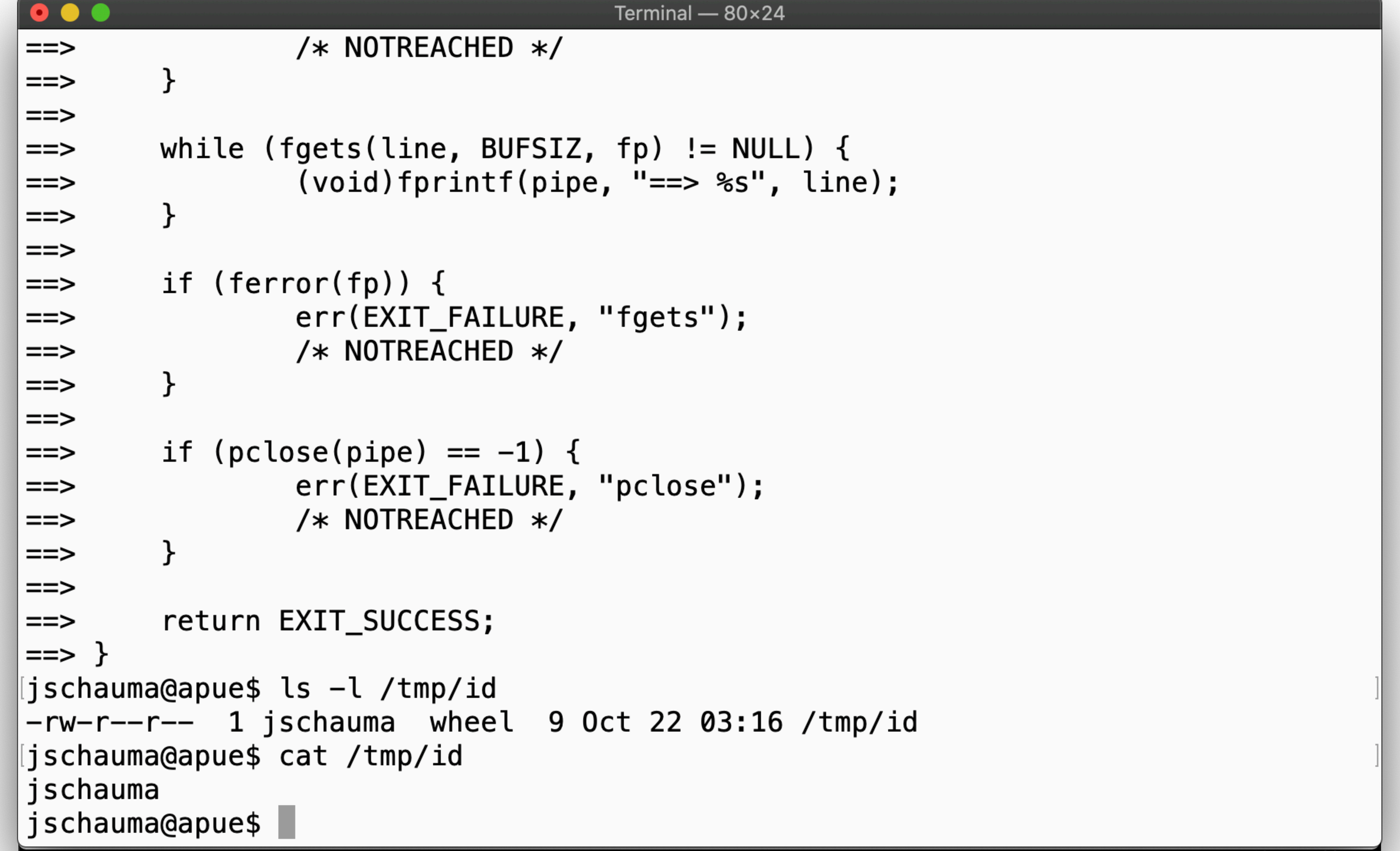
Returns: file stream if ok, NULL otherwise

- historically implemented using unidirectional pipe (nowadays frequently implemented using e.g. sockets)
- type one of “r” or “w” (or “r+” for bi-directional communication, if available)
- cmd passed to /bin/sh -c

popen(3)

```
#include <stdio.h>
FILE *popen(const char *cmd, const char *type);
FILE *popenve(const char *path, char * const *argv,
                        char * const  *envp, const char *type);
                                    Returns: file stream if ok, NULL otherwise
```

- historically implemented using unidirectional pipe (nowadays frequently implemented using e.g. sockets)
- type one of “r” or “w” (or “r+” for bi-directional communication, if available)
- cmd passed to /bin/sh -c
- popenve(3) non-standard


Terminal — 80×24
==>   /* NOTREACHED */
==>   }
==>
==>   while (fgets(line, BUFSIZ, fp) != NULL) {
==>   (void)fprintf(pipe, "==> %s", line);
==>   }
==>
==>   if (ferror(fp)) {
==>   err(EXIT_FAILURE, "fgets");
==>   /* NOTREACHED */
==>   }
==>
==>   if (pclose(pipe) == -1) {
==>   err(EXIT_FAILURE, "pclose");
==>   /* NOTREACHED */
==>   }
==>
==>   return EXIT_SUCCESS;
==> }
jschauma@apue$ ls -l /tmp/id
-rw-r--r-- 1 jschauma wheel 9 Oct 22 03:16 /tmp/id
jschauma@apue$ cat /tmp/id
jschauma
jschauma@apue$

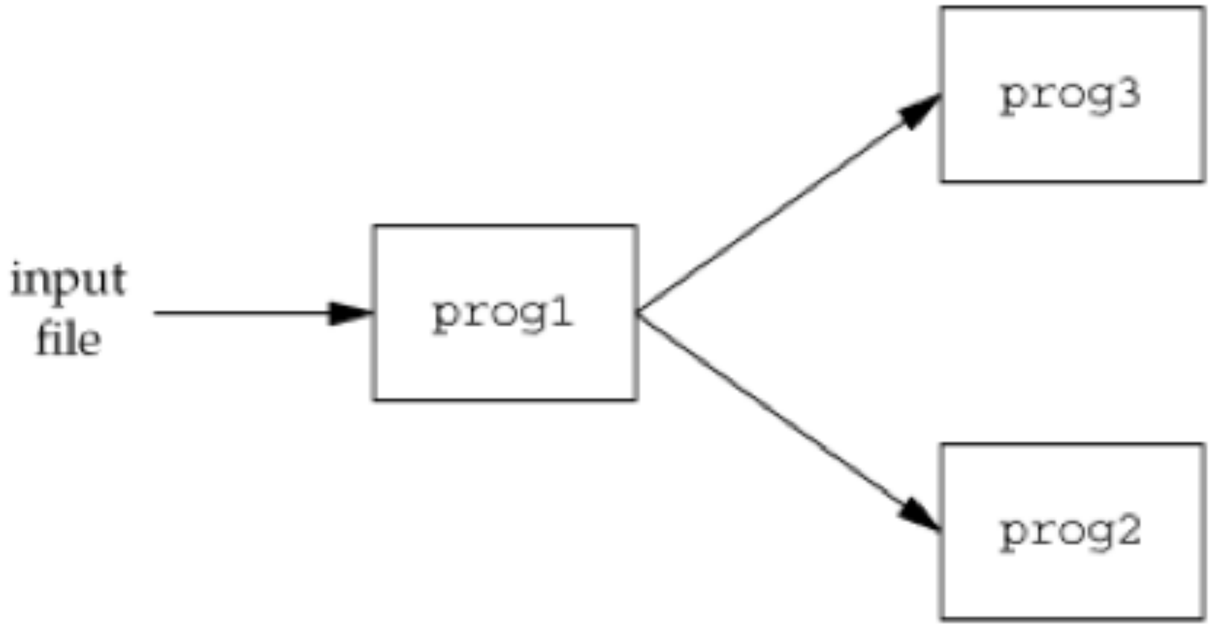
mkfifo(2)

```
#include <sys/stat.h>
#include <fcntl.h>
int mkfifo(const char *path, mode_t mode);
int mkfifoat(int fd, const char *path, mode_t mode);
```

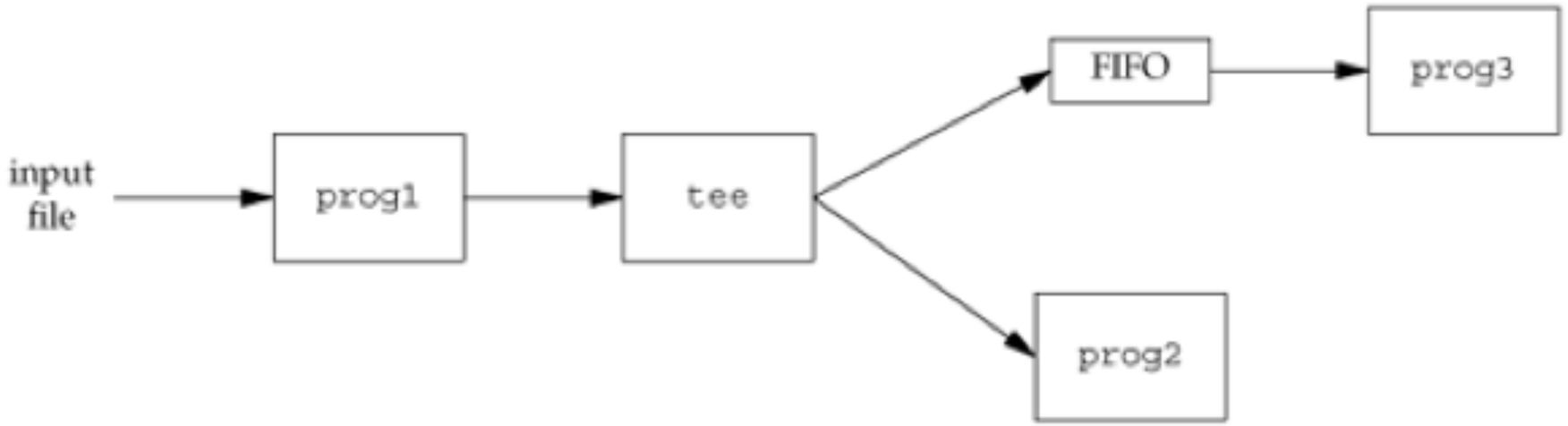
Returns: 0 ok, -1 otherwise

- aka “named pipes”
- allows unrelated processes to communicate
- just a type of file – test for using S_ISFIFO(st mode)
- mode same as for open(2)
- use regular I/O operations (i.e. open(2), read(2), write(2), unlink(2) etc.)
- used by shell commands to pass data from one shell pipeline to another without creating intermediate temporary files

tee(1)



tee(1)



```
Terminal — 80×24
apue$ bzip2 -d -c *.bz2 | grep -v " 200 " > not-ok
apue$ wc -l *ok
   17492 not-ok
   23848 ok
   41340 total
apue$ mkfifo fifo
apue$ ls -l fifo
prw-r--r--  1 jschauma  wheel  0 Oct 22 13:46 fifo
apue$ grep " 200 " fifo > ok &
apue$ ps
  PID TTY   STAT    TIME COMMAND
  597 pts/0 Ss   0:00.09 -sh
  803 pts/0 S    0:00.00 grep  200  fifo
 1008 pts/0 O+   0:00.00 ps
apue$ bzip2 -d -c *.bz2 | tee fifo | grep -v " 200 " > not-ok
apue$ ps
PID TTY   STAT    TIME COMMAND
418 pts/0 O+   0:00.00 ps
597 pts/0 Ss   0:00.10 -sh
apue$ wc -l *ok
   17492 not-ok
   23848 ok
   41340 total
apue$
```

pipe(2) and FIFOs

- basis of the Unix Philosophy of building filters and operating on text streams
- pipes require a common ancestor, FIFOs do not
- data written into a pipe is no longer line buffered
- can have multiple readers/writers (PIPE_BUF bytes are guaranteed to not be interleaved)

Behavior after closing one end:

- read(2) from a pipe whose write end has been closed returns 0 after all data has been read
- write(2) to a pipe whose read end has been closed generates SIGPIPE; if caught or ignored, write(2) returns an error and sets errno to EPIPE.

Additional Reading

- HW2: https://stevens.netmeister.org/631/f20-hw2.html

- http://blog.petersobot.com/pipes-and-filters
- https://blog.jessfraz.com/post/for-the-love-of-pipes/
- https://www.pixelbeat.org/programming/stdio_buffering/

- Useful manual pages: mkfifo(1), mkfifo(2), pipe(2), setbuf(3), tee(1)