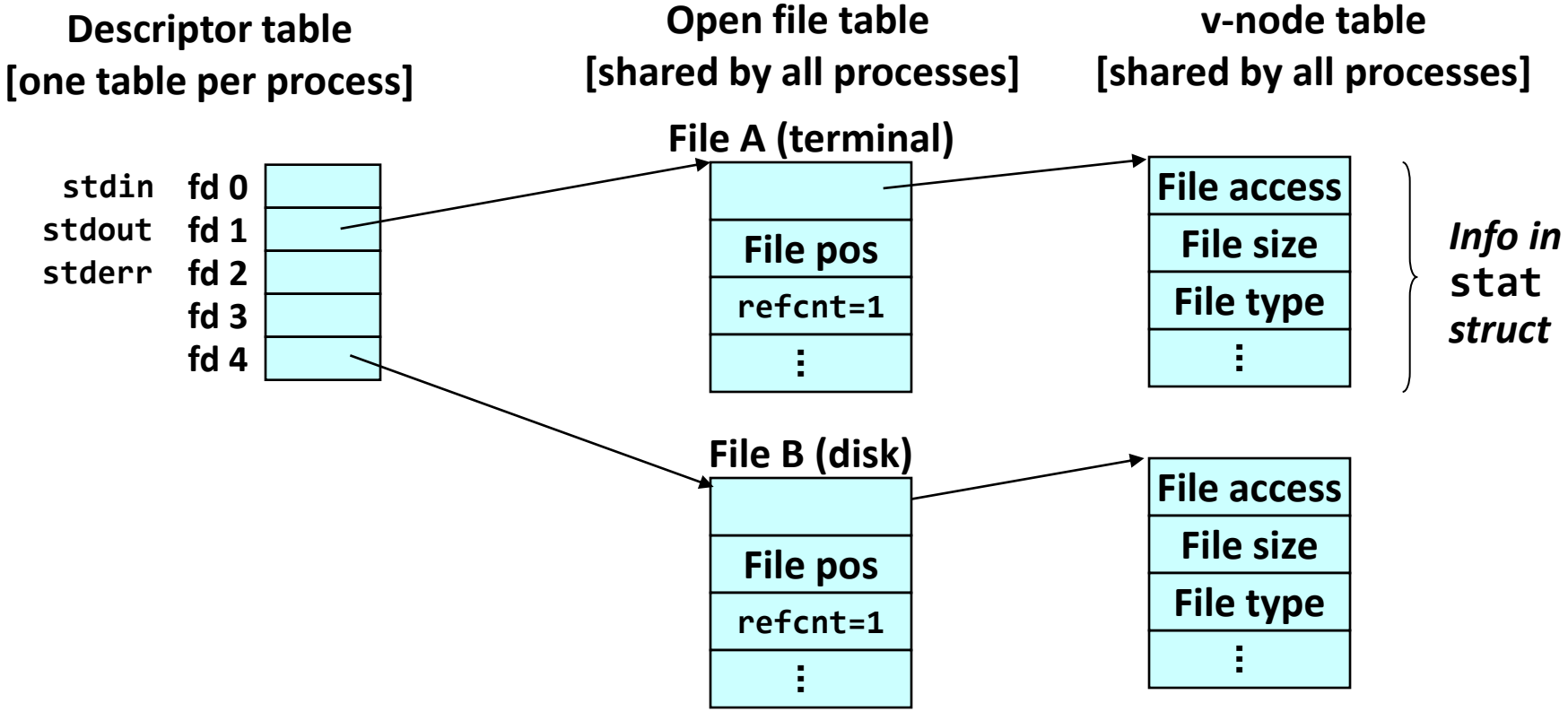
Pipes and FIFOs

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

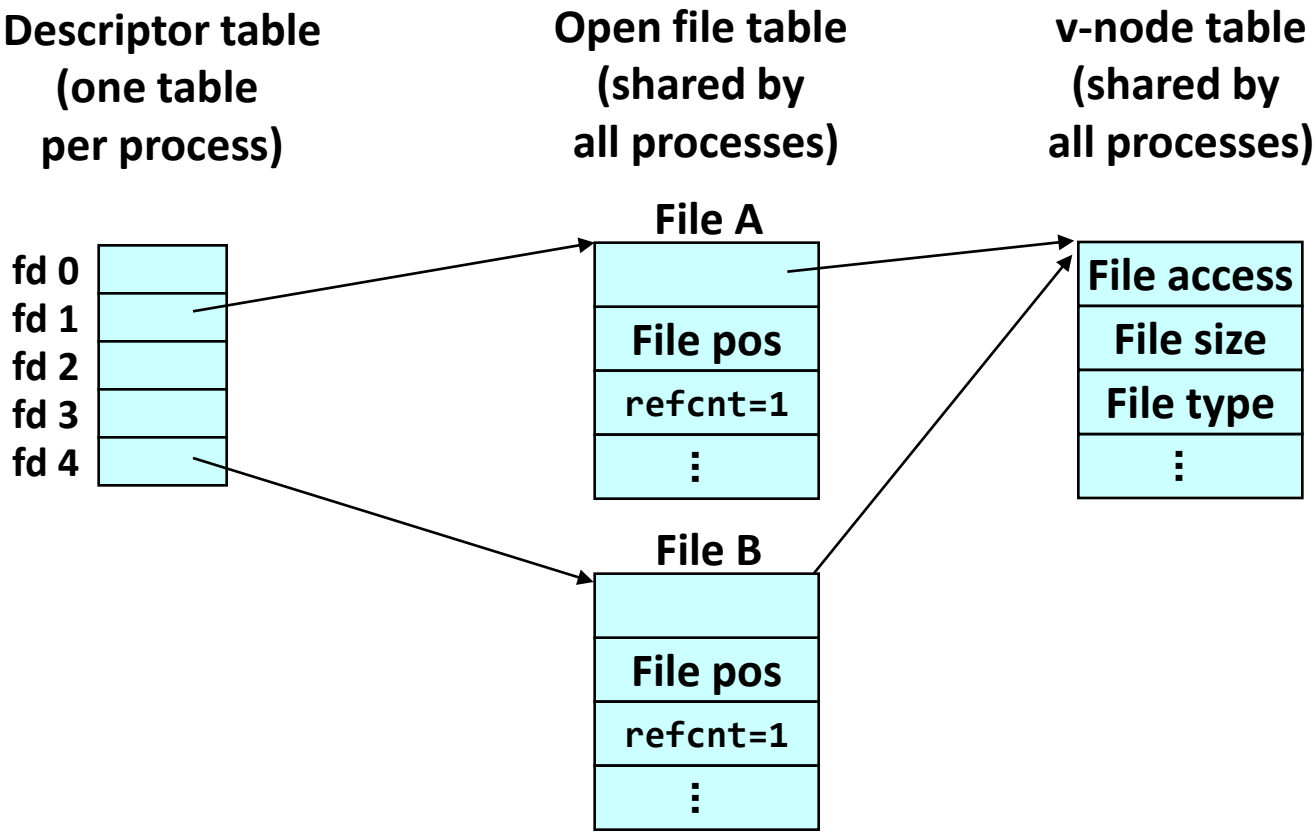
Open Files in Kernel

- **How the Unix kernel represents open files?**
  - Two descriptors referencing two distinct open disk files. Descriptor 1 (stdout) points to terminal, and descriptor 4 points to open disk file.



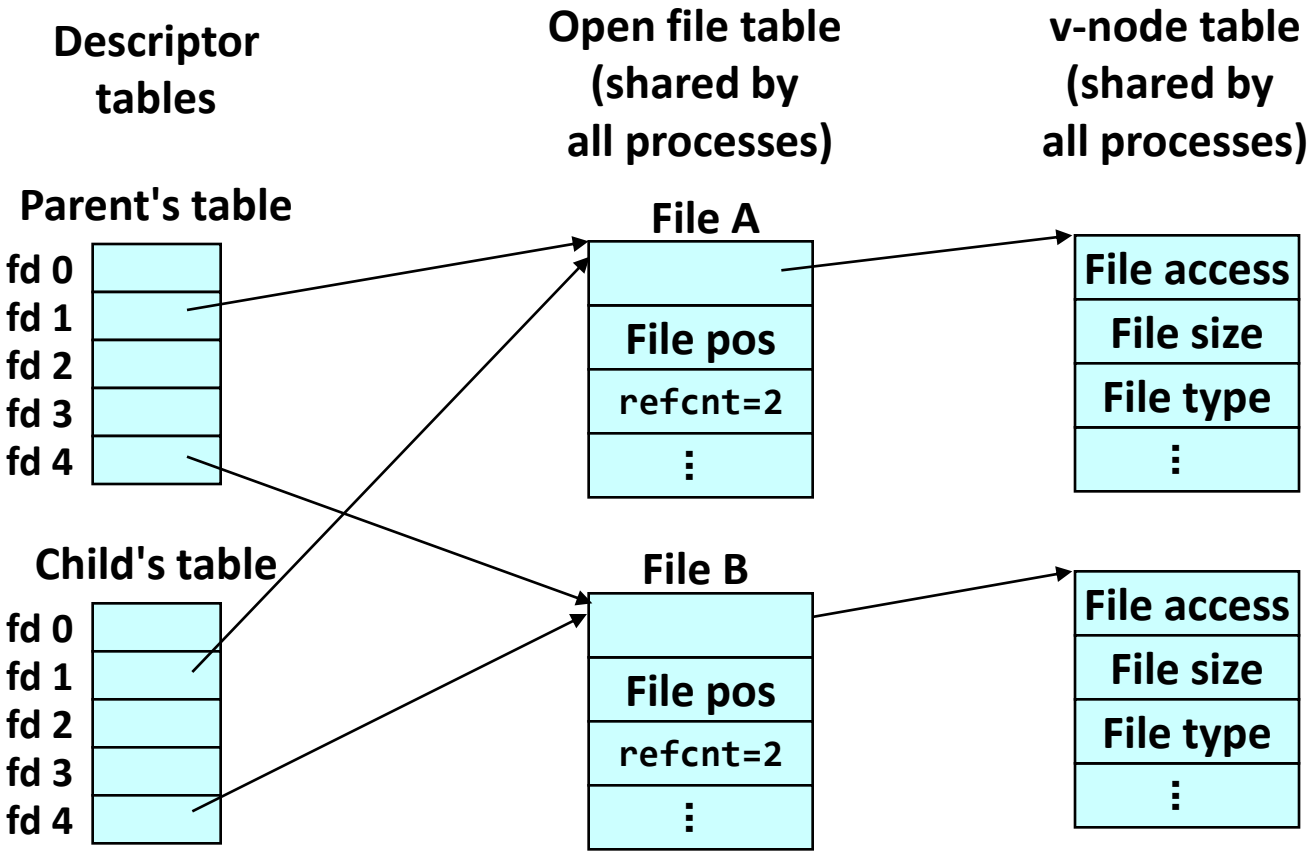
File Sharing

- **Two distinct descriptors sharing the same disk file through two distinct open file table entries**
  - E.g., Calling **open()** twice with the same **filename** argument



How Processes Share Files

- A child process inherits its parent's open files. Here is the situation immediately after a fork()



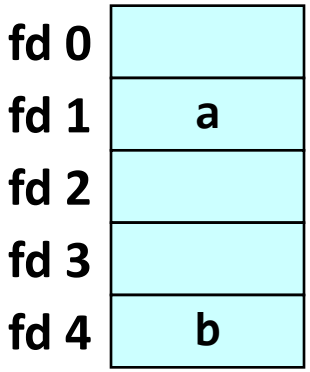
I/O Redirection

- **Q: How does a shell implement I/O redirection?**

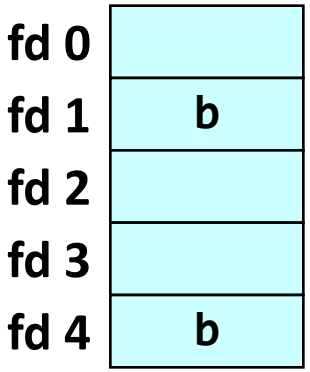
  $ ls > foo.txt

- **A: By calling the dup2(oldfd, newfd) function.**
  - Copies (per-process) descriptor table entry **oldfd** to entry **newfd**

Descriptor table before dup2(4,1)

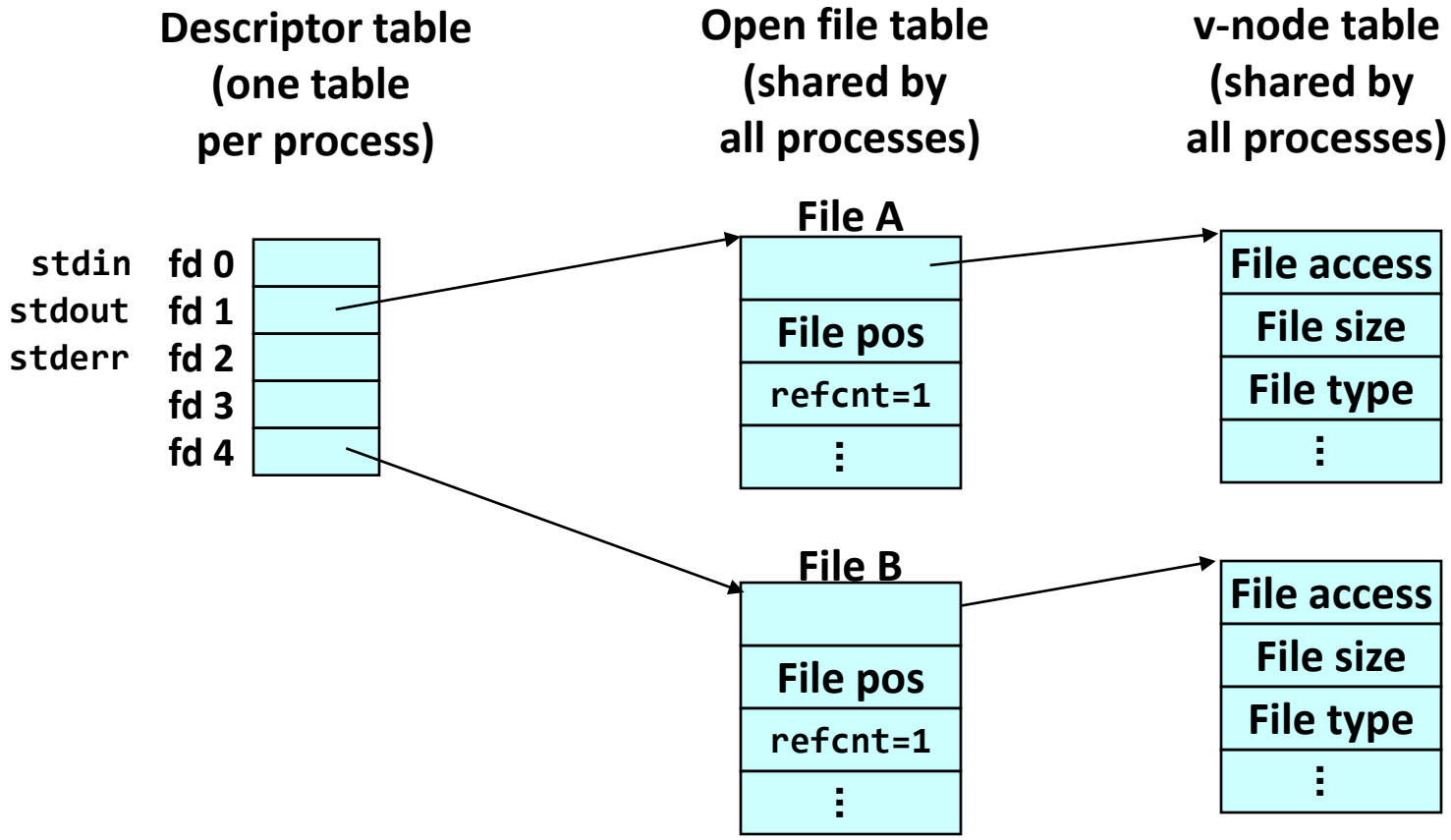


Descriptor table after dup2(4,1)



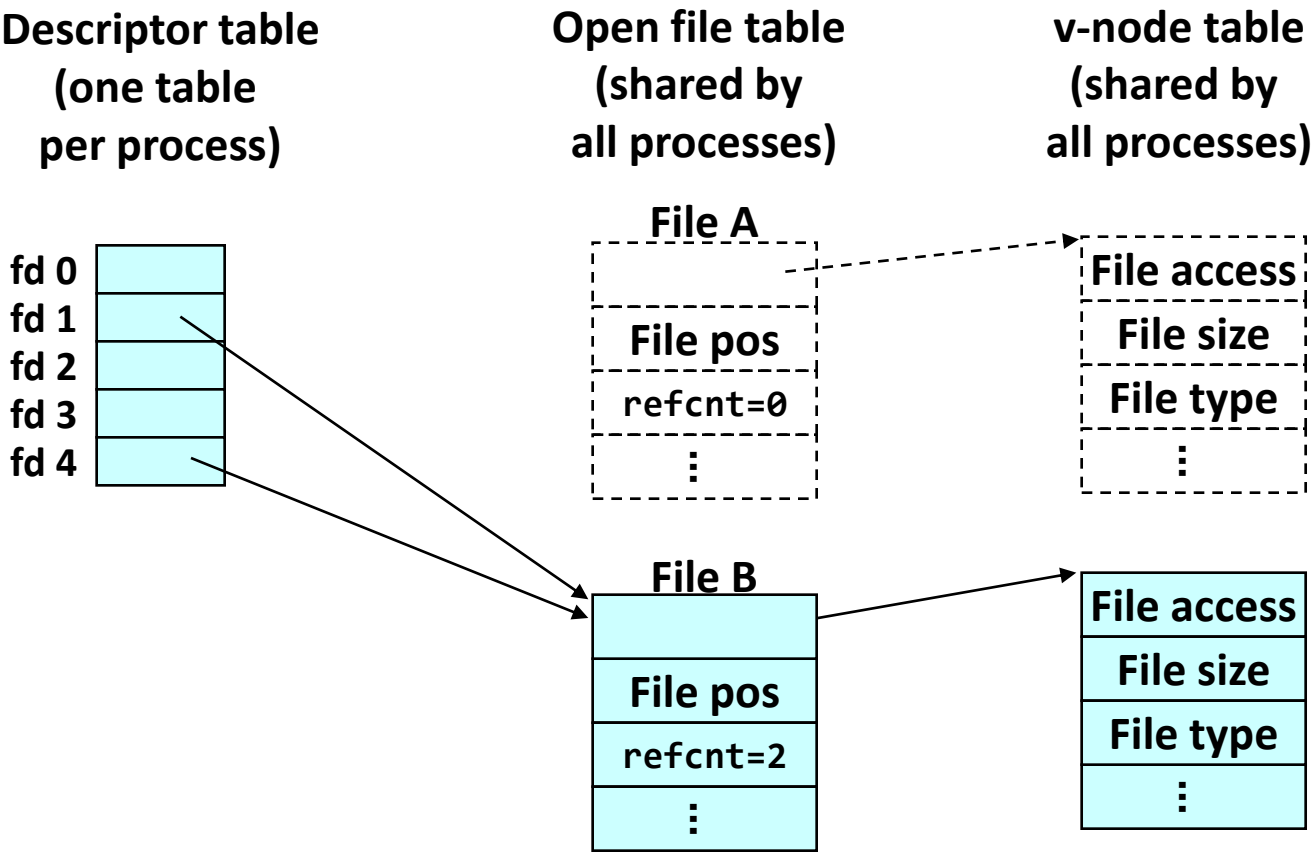
I/O Redirection Example (1)

- Before calling dup2(4,1), stdout (descriptor 1) points to a terminal and descriptor 4 points to an open disk file.



I/O Redirection Example (2)

- After calling `dup2(4,1)`, stdout is now redirected to the disk file pointed at by descriptor 4.



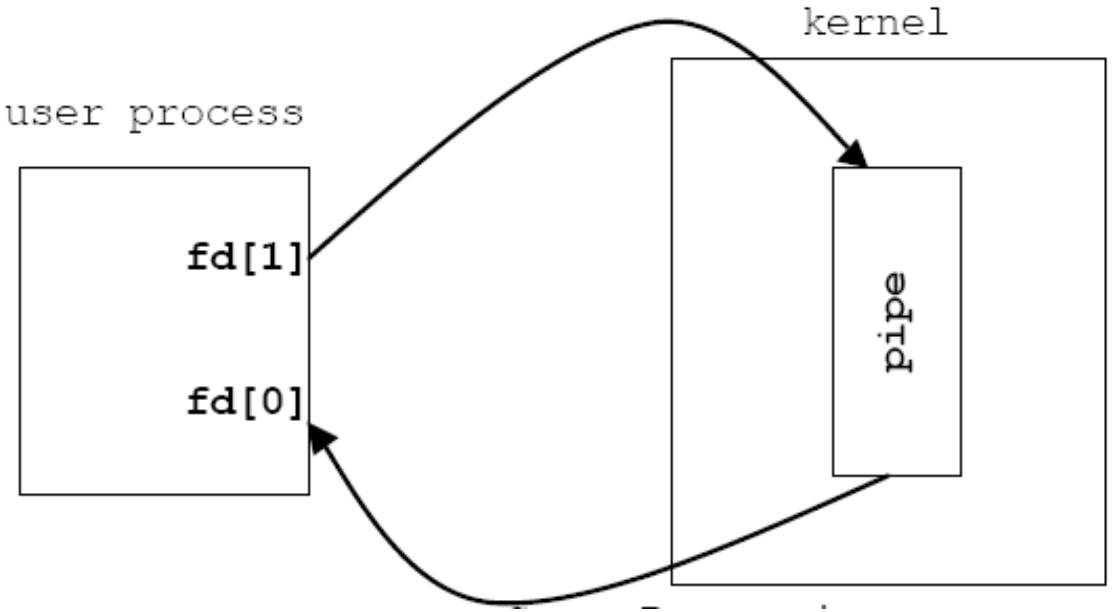
Pipes

- **Pipes**
  - The oldest form of Unix IPC (Interprocess Communication) and provided by all Unix systems.

- **Two limitations**
  - Half-duplex: data flows only in one direction.
  - Can be used only between processes that have a common ancestor.
    - Usually used between the parent and child processes.

Creating Pipes (1)

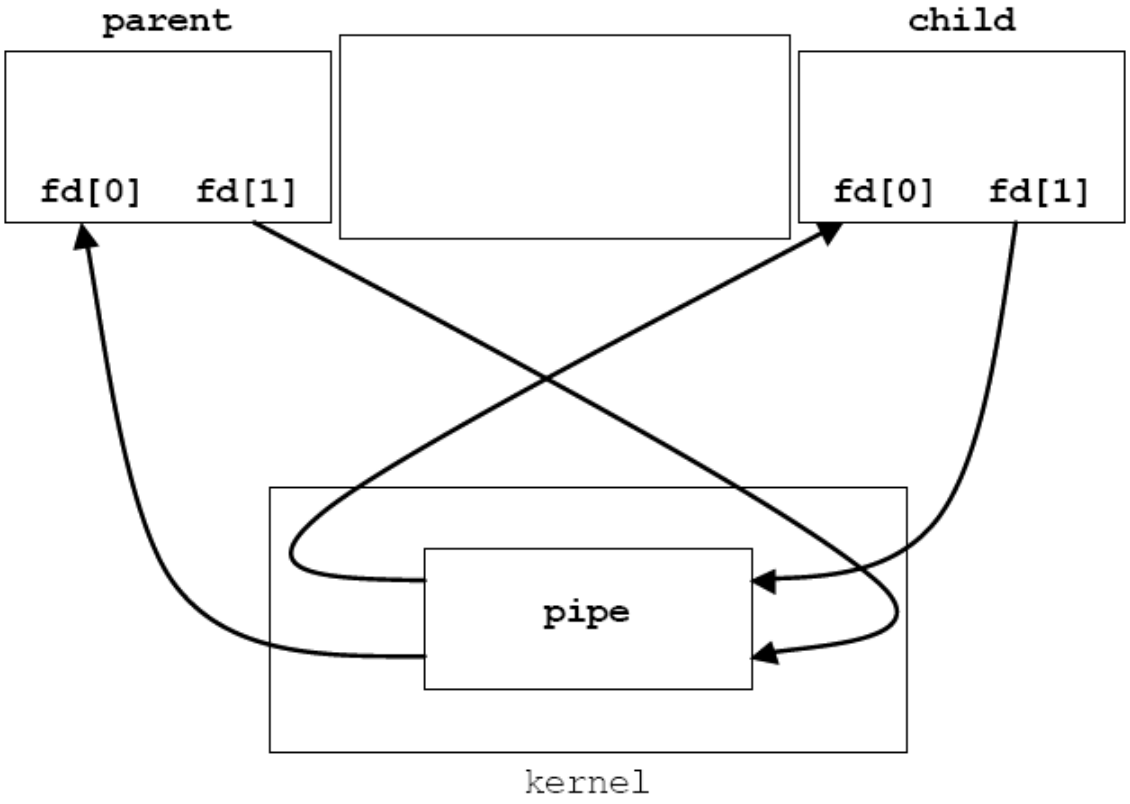
- **int pipe (int fd[2])**
  - Two file descriptors are returned through the **fd** argument
    - **fd[0]:** open for reading
    - **fd[1]:** open for writing
  - The output of **fd[1]** is the input for **fd[0].**



Creating Pipes (2)

```
parent => child:
parent closes fd[0];
child closes fd[1];
```

```
parent <= child:
parent closes fd[1];
child closes fd[0];
```



Reading/Writing Pipes

- **When one end of a pipe is closed,**
  - reading from a pipe returns an end of file.
  - writing to a pipe causes **SIGPIPE** is generated and the write returns an error (**EPIPE**).
  - A write of **PIPE_BUF** (kernel's pipe buffer size) bytes or less will not be interleaved with the writes from other processes.
  - **fstat** function returns a file type of FIFO for the pipe file descriptors (can be tested by **S_ISFIFO** macro)
- **You should close unused file descriptors!**

Using Pipes

```
#include <unistd.h>

#define MAXLINE   80

int main (void)
{
    int n, fd[2];
    pid_t pid;
    char line[MAXLINE];

    if (pipe(fd) < 0) exit (1);
    if ((pid = fork()) < 0) exit (2);
    else if (pid > 0) {          /* parent */
        close (fd[0]);
        write (fd[1], "hello world\n", 12);
    }
    else {                                  /* child */
        close (fd[1]);
        n = read(fd[0], line, MAXLINE);
        write (1, line, n);
    }
}
```

FIFOs

- **int mkfifo (const char *path, mode_t mode)**
  - Named pipes
  - Unrelated processes can exchange data, whereas pipes can be used only between related processes.
  - FIFO is a type of file: FIFO type (**S_ISFIFO** macro)
  - Once a FIFO created, the normal file I/O functions all work with FIFO.

- **/usr/bin/mkfifo program can be used to make FIFOs on the command line.**

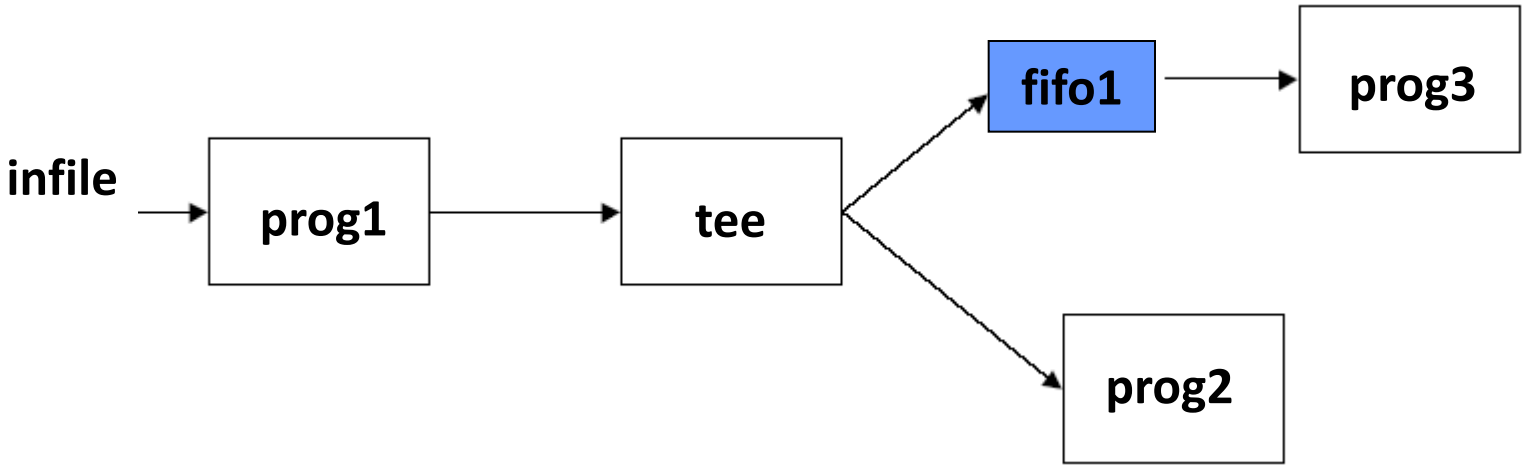
Using FIFOs

- **Opening a FIFO**
  - An open for read(write)-only blocks until some other process opens the FIFO for writing(reading).
- **Reading/Writing a FIFO**
  - Writing to a FIFO that no process has open for reading causes **SIGPIPE** to generate.
  - When the last writer for a FIFO closes the FIFO, an end of file is generated for the reader of the FIFO.
  - **PIPE_BUF**: the maximum amount of data that can be written atomically to a FIFO (without being interleaved among multiple writers).

Use of FIFOs (1)

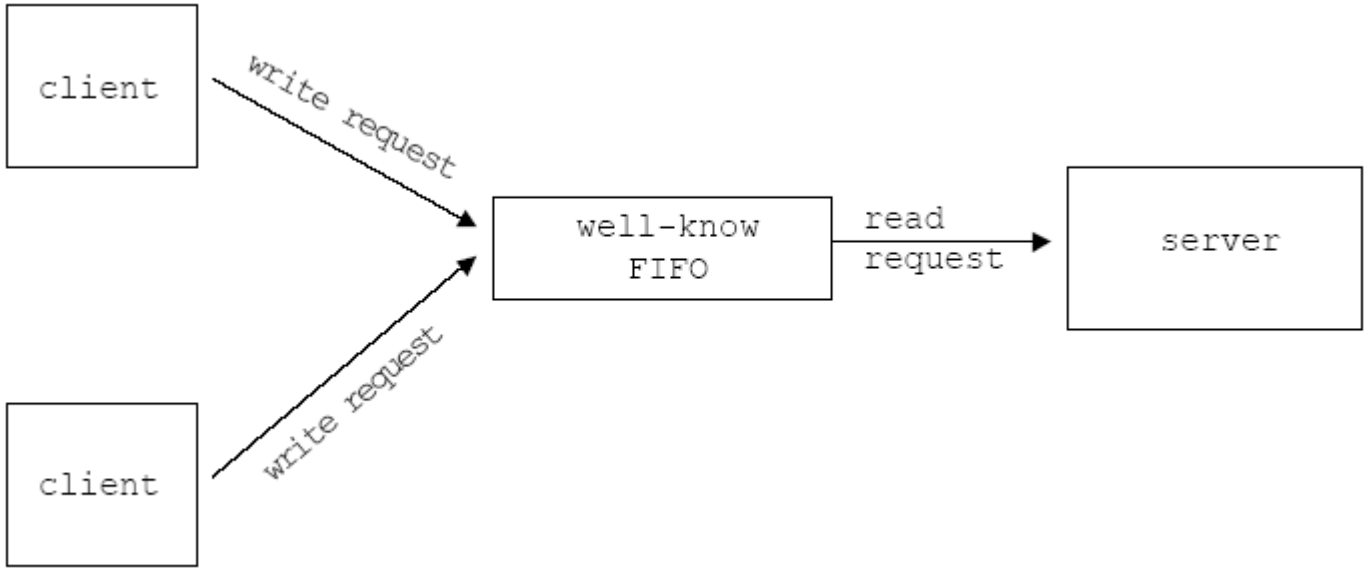
- **Duplicating a Stream**
  - Shell commands to pass data from one shell pipeline to another without creating intermediate temporary files

```
$ mkfifo fifo1
$ prog3 < fifo1 &
$ prog1 < infile | tee fifo1 | prog2
```



Use of FIFOs (2)

- **Client-server Communication**
  - A client-server application to pass data between the clients and server on the same machine.
    - Clients write to a "well-known" FIFO to send a request to the server.



Summary

- **IPC (Inter-Process Communication)**
  - Signal
  - Pipe
  - Named pipe (FIFO)

  - Shared memory
  - Semaphore
  - Sockets
  - …