SSE2033

SUNG KYUN KWAN UNIVERSITY

Fall 2019

System Software Experiment 2

Pipes and FIFOs

Prof. Jinkyu Jeong (*jinkyu@skku.edu*)
TA – Gyusun Lee (*gyusun.lee@csi.skku.edu*)
TA – Jiwon Woo (*jiwon.woo@csi.skku.edu*)
Computer Systems and Intelligence Laboratory (*http://csi.skku.edu*)
Sungkyunkwan University

Contents

- IPC (Inter-Process Communication)
  - Representation of open files in kernel
  - I/O redirection
  - Anonymous Pipe
  - Named Pipe (FIFO)

- Exercise

Everything is a file

- Actually, “Everything is a file descriptor”

- Pros
  - Can reuse tools, APIs on a wide range of resources
- Cons
  - Not a fast or portable approach

- Communication using file interface?

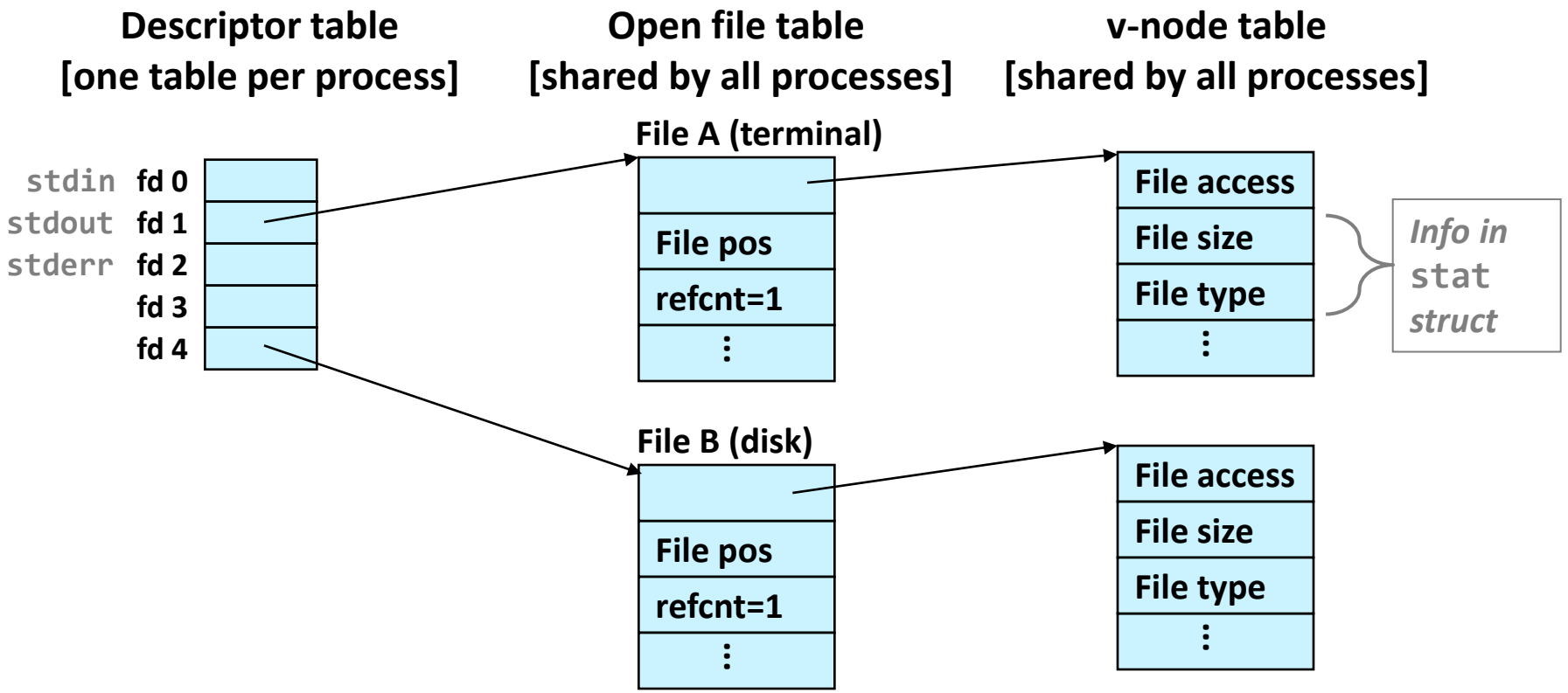
Open Files in Kernel

- How the Unix kernel represents open files?

- 3-levels
  - Descriptor table
    - 1 table per process
    - Pointer to entry in the “file table”
  - File table
    - Shared by all processes
    - Current file position, mode, reference count, pointer to entry in the “v-node table”
  - v-node table
    - Shared by all processes
    - Information about file itself (size, permission, …)

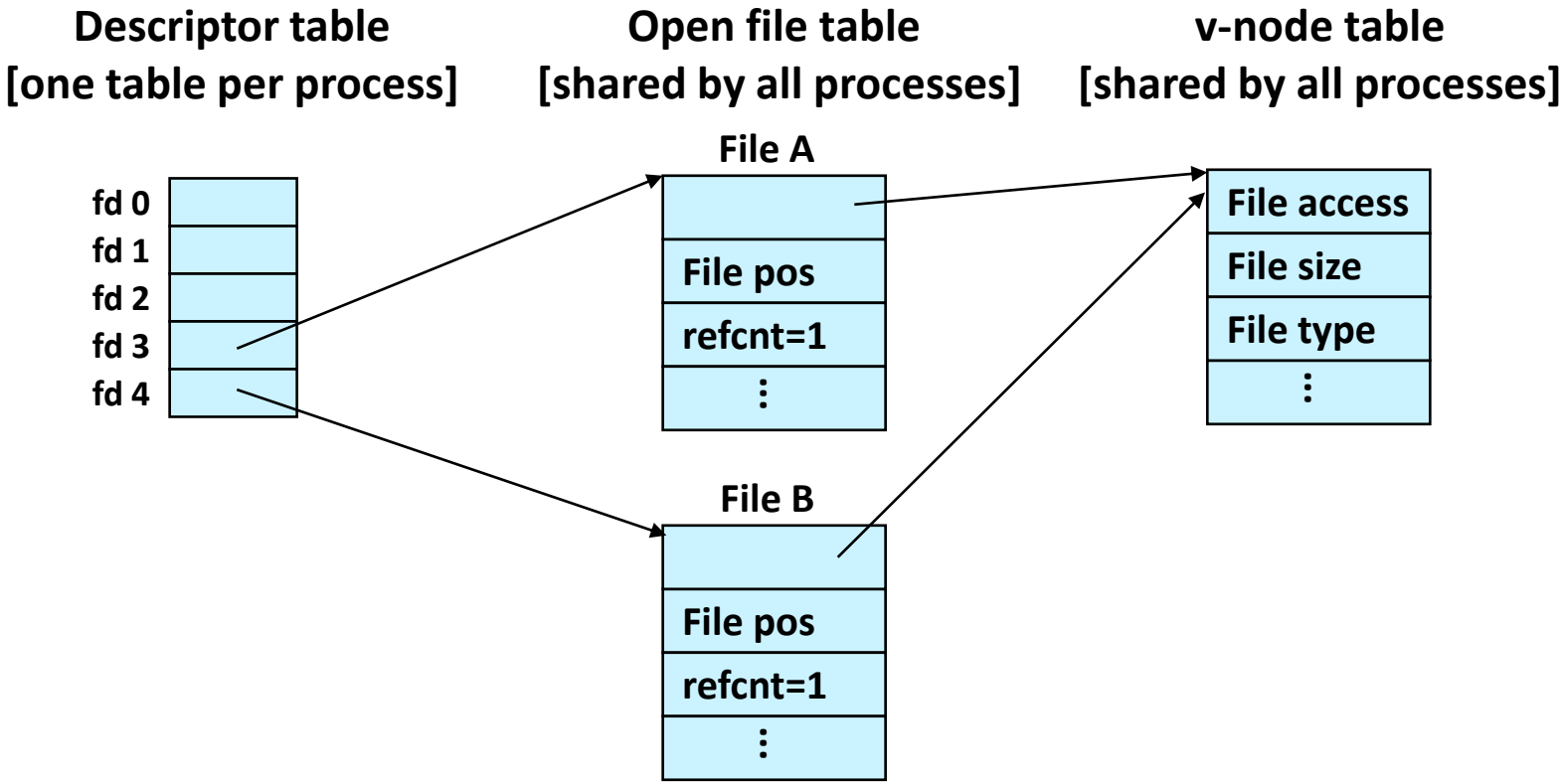
Open Files in Kernel (2)

- How the Unix kernel represents open files?



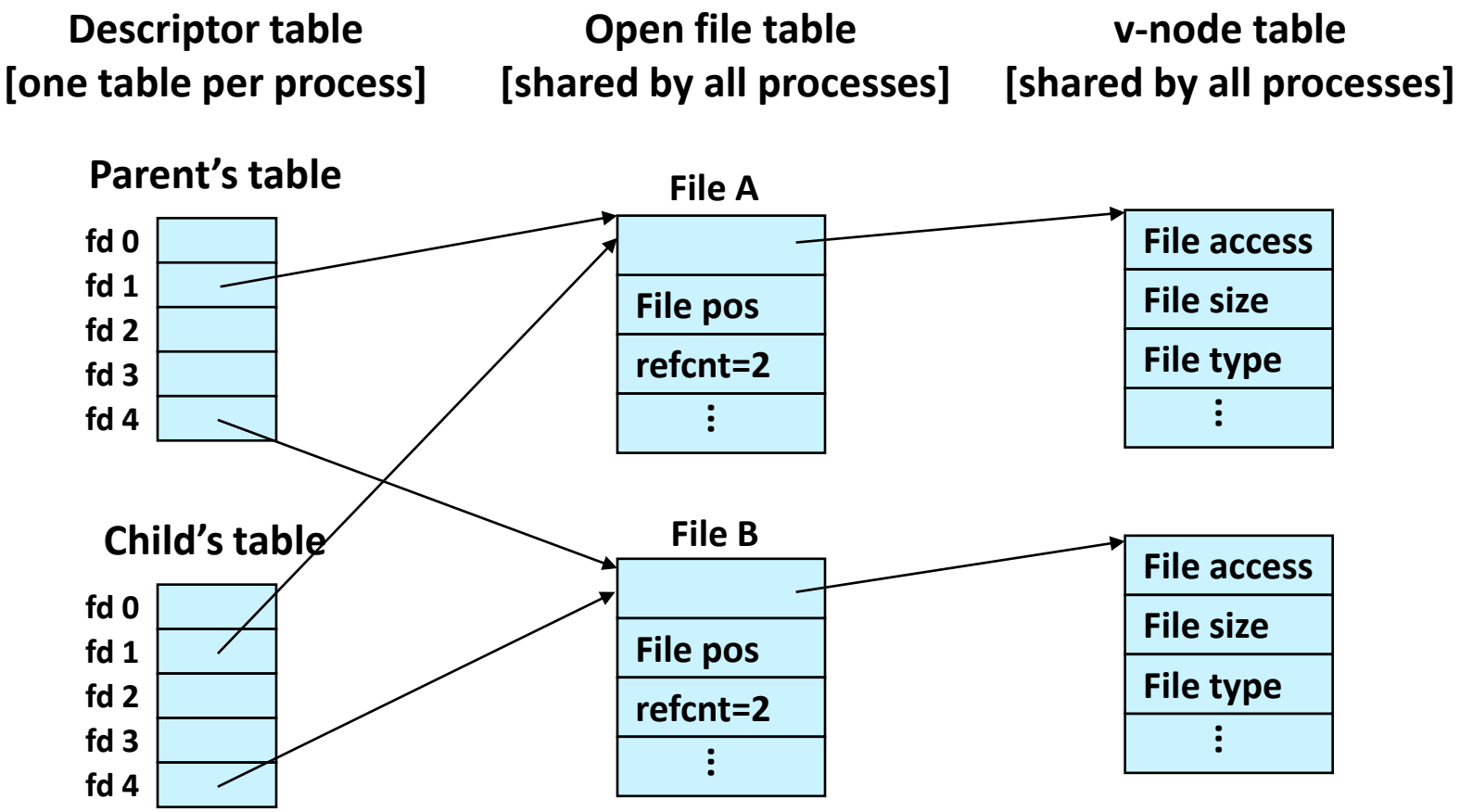
Open Files in Kernel (3)

- Calling open() twice with the same filename



Open Files in Kernel (4)

- Calling fork()



Open Files in Kernel (5)

- What will be the result?

```
int main(void)
{
   char buf[512];
   int fd = open("./tmp.txt", O_RDONLY);

   if (fork() == 0) {
      assert(read(fd, buf, 5) >= 0);
      exit(0);
   } else {
      wait(NULL);
      assert(read(fd, buf, 5) >= 0);
      assert(write(1, buf, 5) >= 0);
      assert(write(1, "\n", 1) >= 0);
   }

   return 0;
}
```

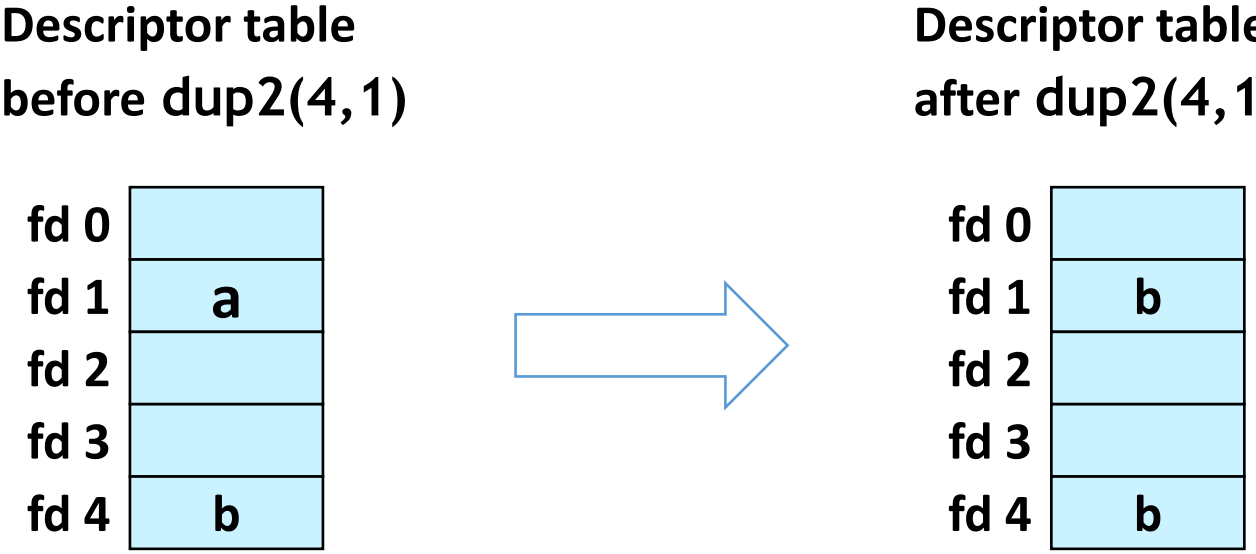
```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <assert.h>
```

I/O Redirection

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

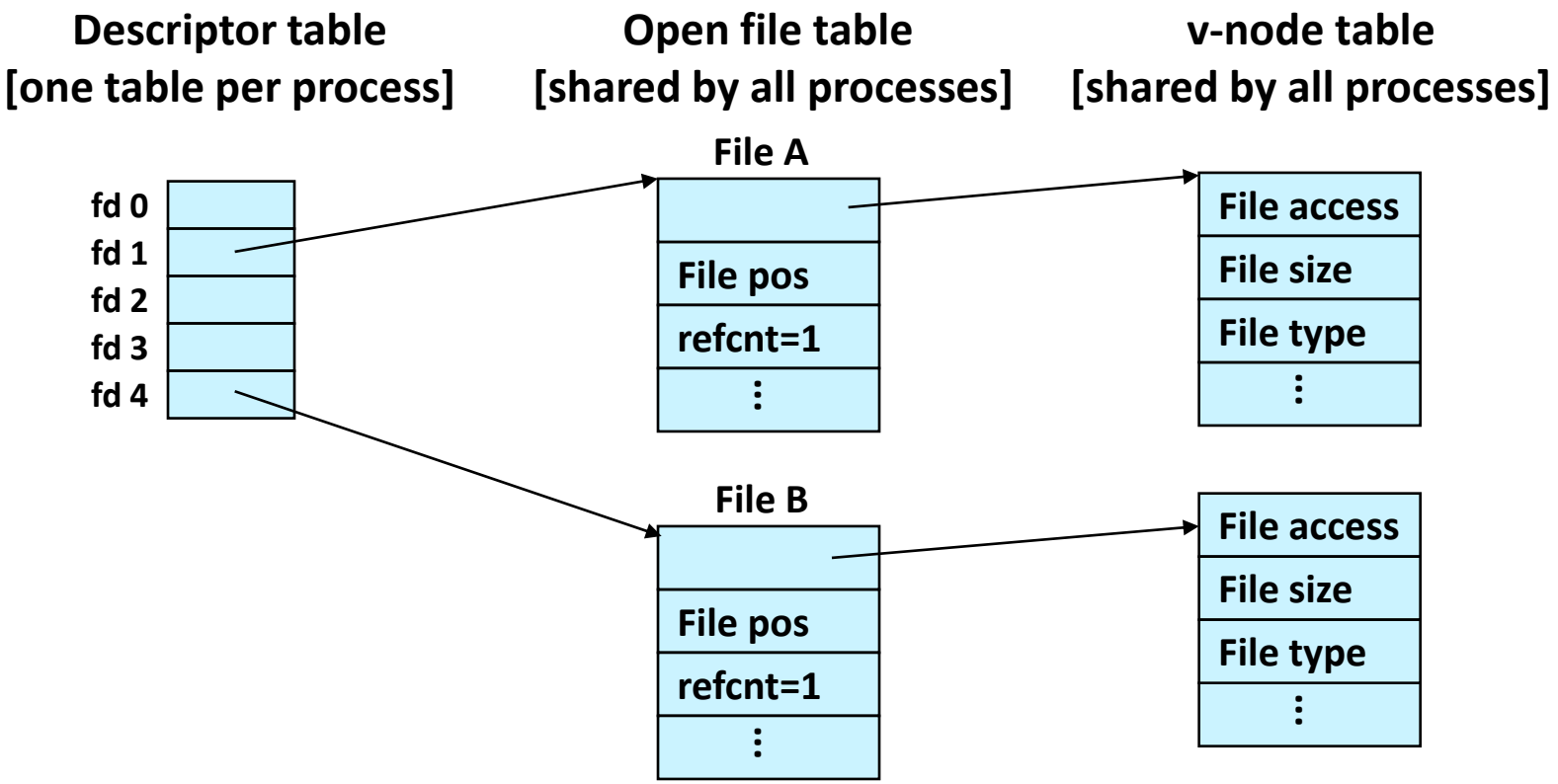
  $ ls > foo.txt

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



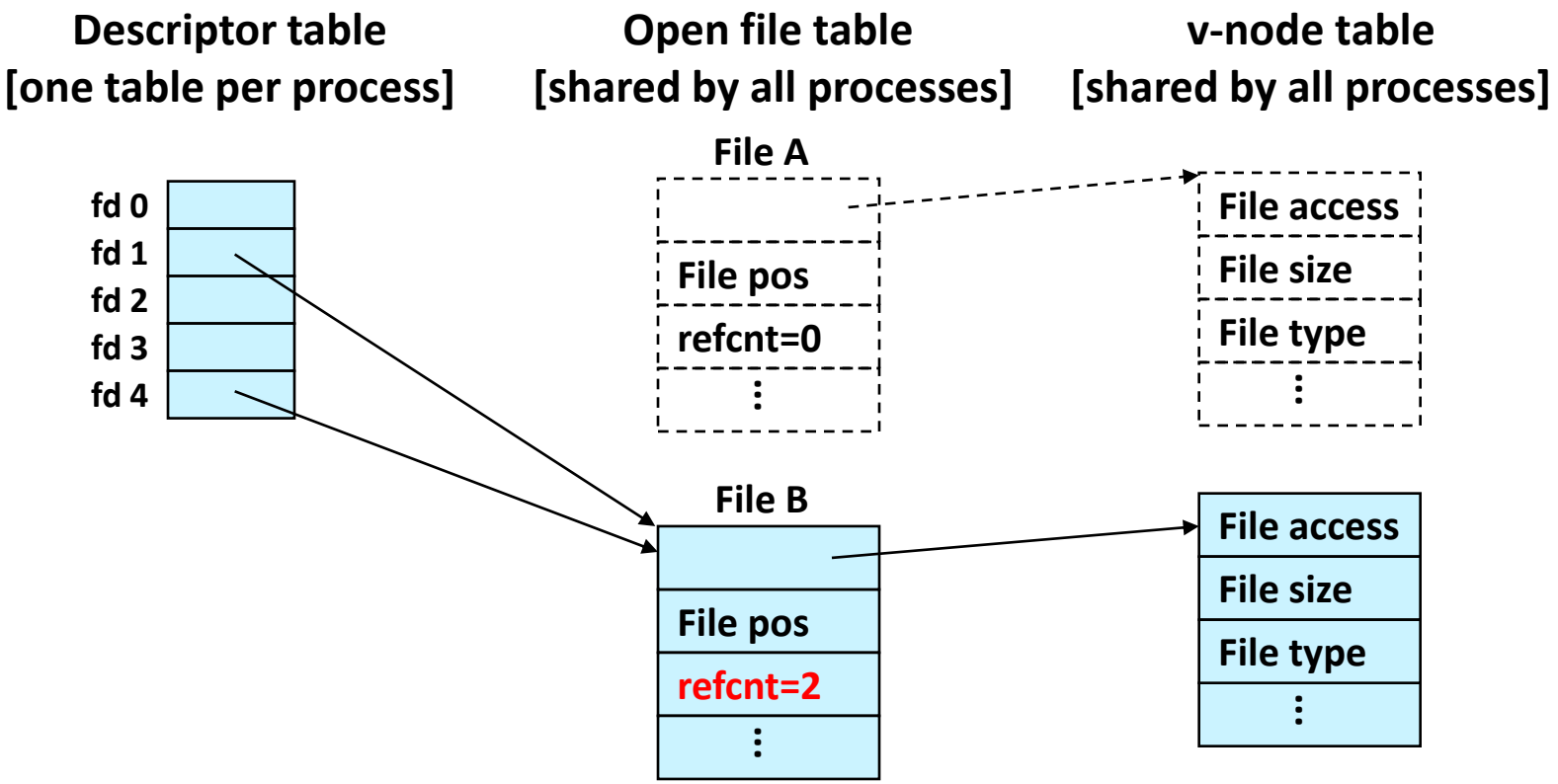
I/O Redirection Example (1)

- 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 not redirected to the disk file pointed at by descriptor 4.



Pipes

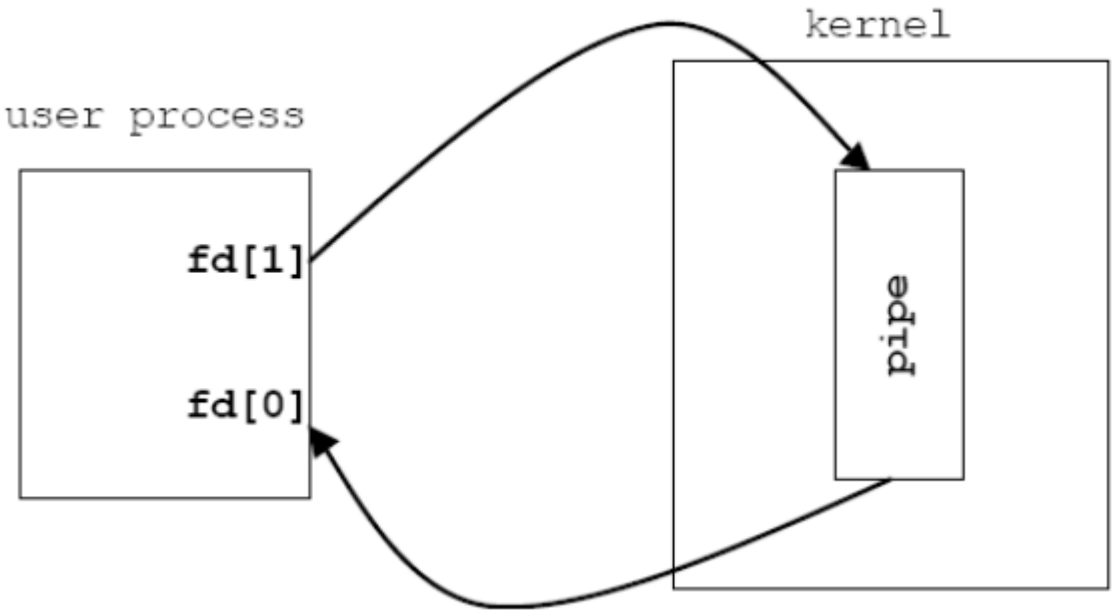
- Pipes
  - The oldest form of UNIX IPC (Inter-process Communication) and provide by all Unix systems.
  - IPC using ‘file interface’

- Limitations
  - Half-duplex: data flows only in one direction.
  - Data only can be read once.

- Two pipes
  - Anonymous pipe
    - No name
  - Named pipe
    - We can see it with a file-system

Anonymous Pipe (1)

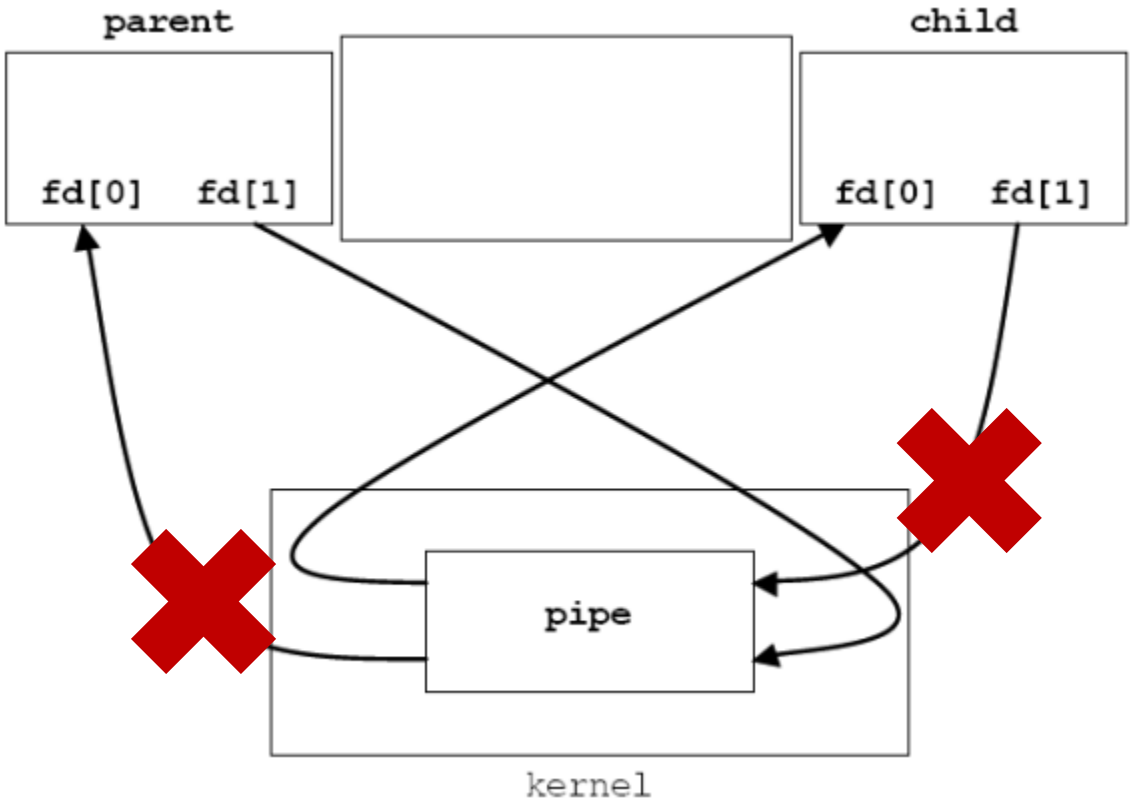
- int pipe (int fd[2]);
  - Two file descriptors are returned through the **fd** argument
    - **fd[0]**: open for reading
    - **fd[1]**: open for writing
  - The output of **fd[1]** is the input for **fd[0]**.



Anonymous Pipe (2)

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

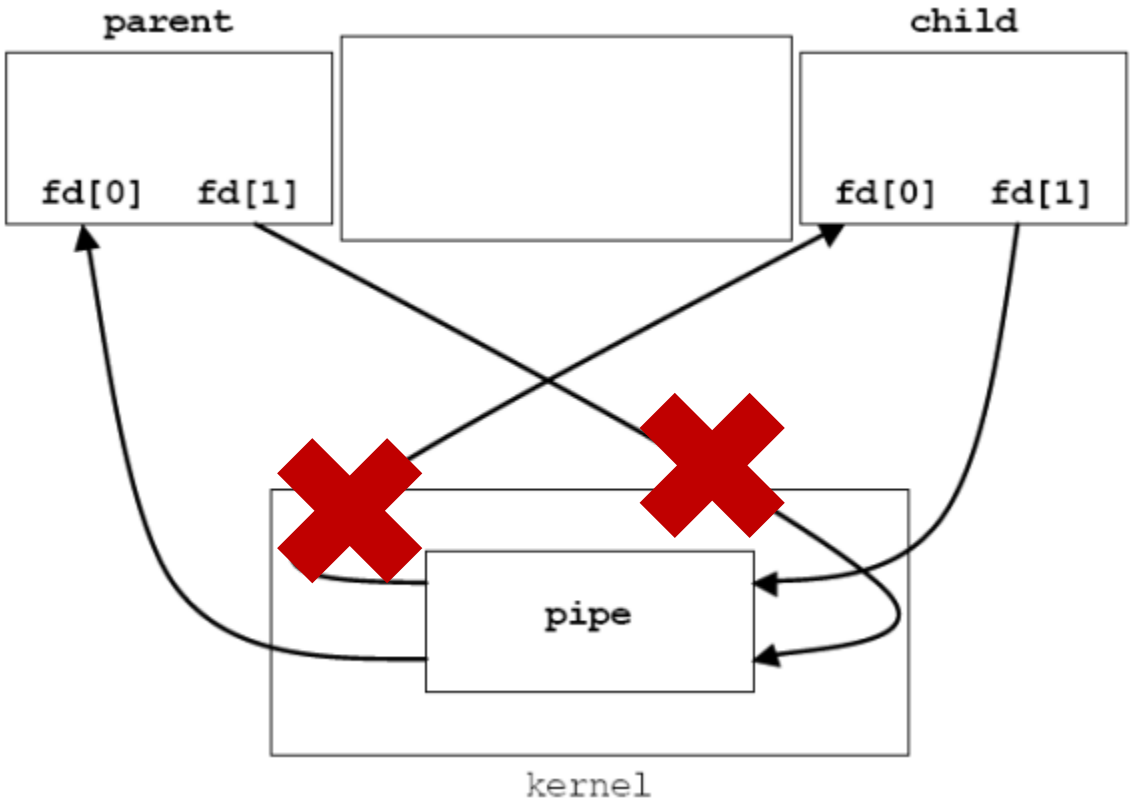
parent <= child:
parent closes fd[1];
child closes fd[0];



Anonymous Pipe (3)

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

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



Reading/Writing Pipe

- 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**).
  - **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 Anonymous Pipe

```
#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);

        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);
        }
        exit(0);
}
```

Named Pipe (FIFO)

- int mknod (const char *path, mode_t mode, dev_t dev)
  - mknod (“path”, S_IFIFO, 0);

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

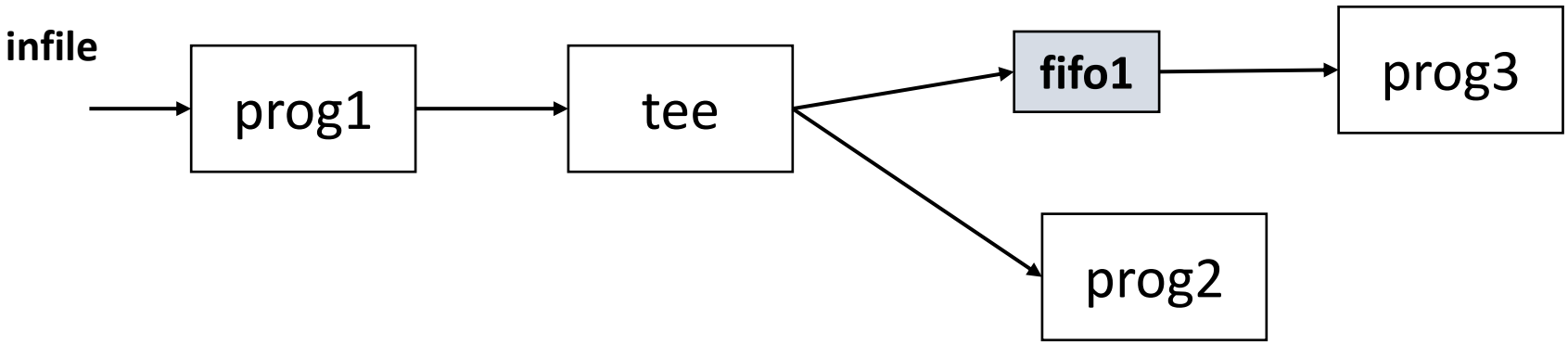
Using FIFOs

- 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)

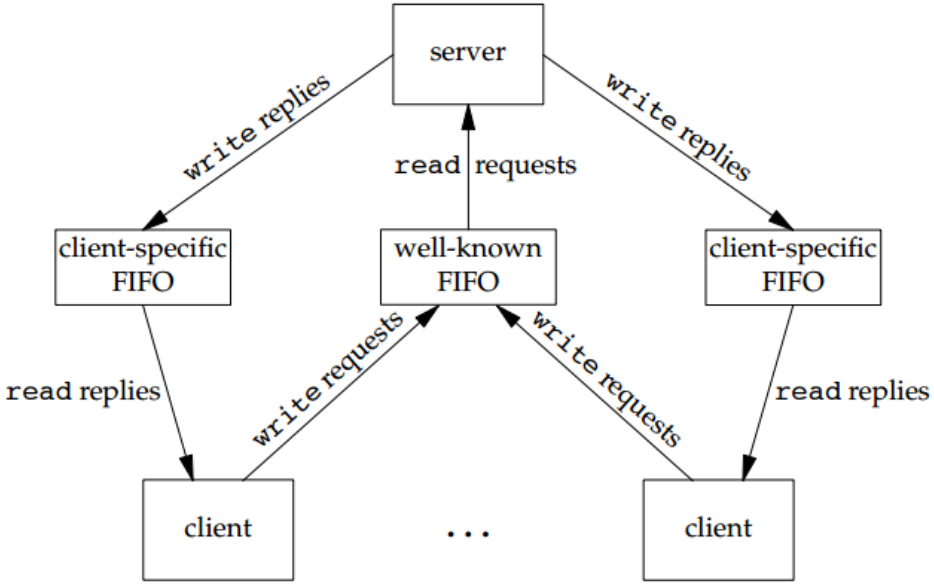
- Duplicating a Stream
  - Shell commands to pass data from one shell pipeline to another without creating intermediate temporary files

```
$ 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 client 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
  - …

Exercise

- Make C programs run the following tasks:

```
$ echo "124 * (42 + 3) % 17" | bc
```

- main -> pipe -> fork
  - dup2 -> exec family ➔ echo
  - dup2 -> exec family ➔ bc