SWE2024

SUNG KYUN KWAN UNIVERSITY

Pipe and Redirection

Prof. Joonwon Lee (joonwon@skku.edu)
TA – Jaehyung Park (jaeseanpark@gmail.com)
TA – Luke Albano (lukealbano@arcs.skku.edu)
Sungkyunkwan University

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 approach
- Communication using file interface?

IPC

- Inter-process Communication
  - Mechanism for communication between processes

- Methods in IPC
  - Signals
  - I/O redirection
  - Anonymous Pipe
  - Named Pipe (FIFO)
  - Shared Memory, Message Queue, Message Passing, etc.

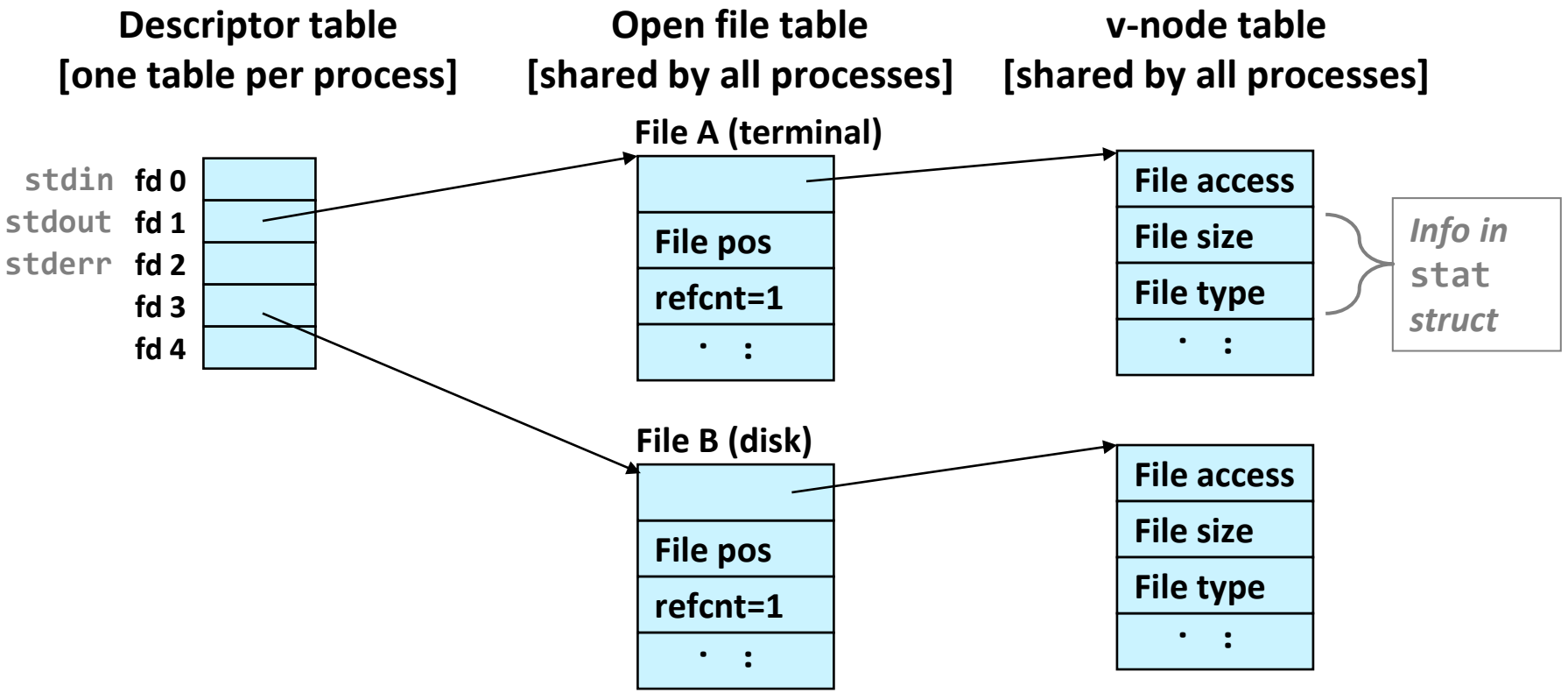
Open Files in Kernel (1)

- How the Unix kernel represents open files?

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

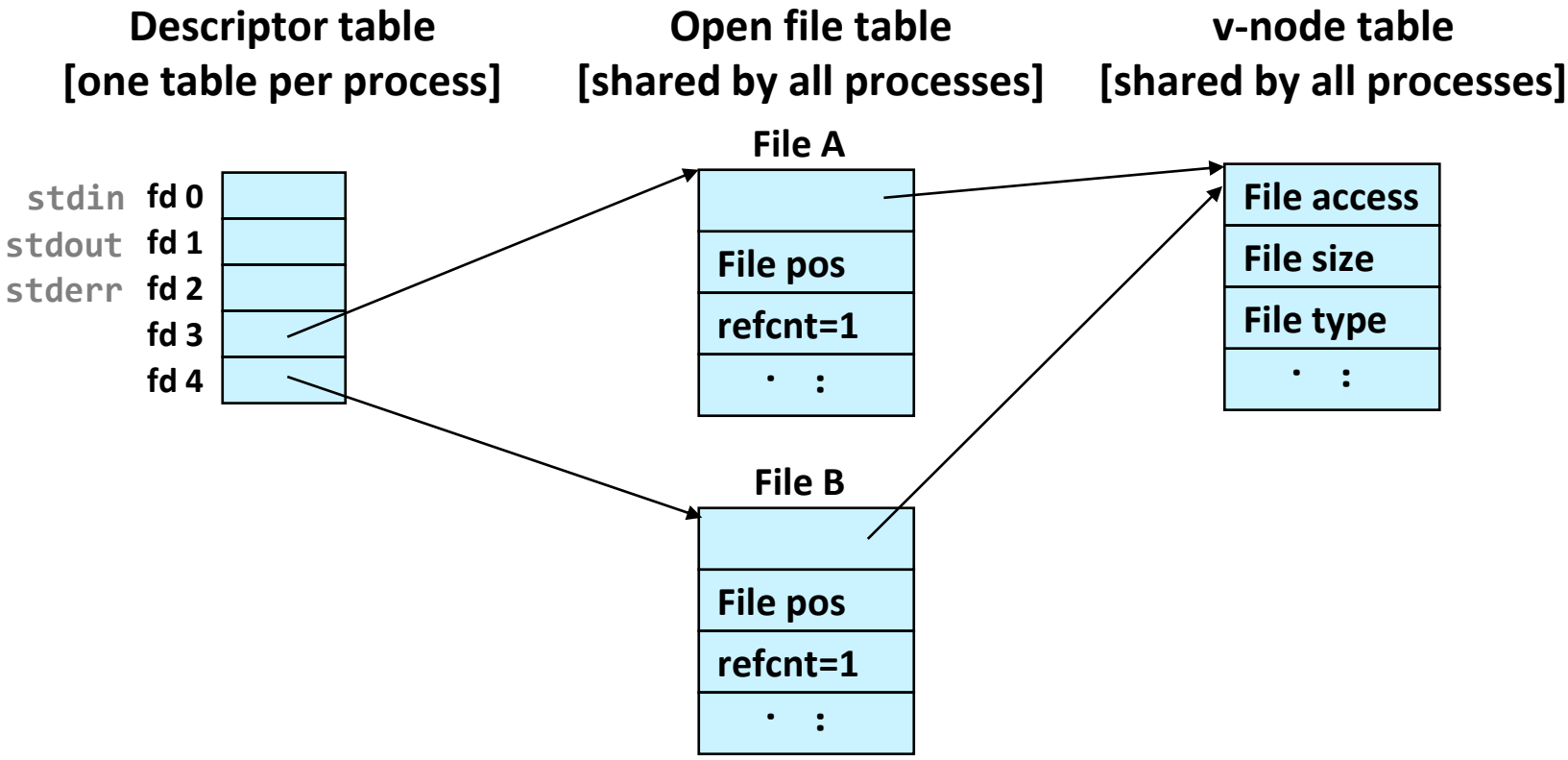
Open Files in Kernel (2)

- How the Unix kernel represents open files?



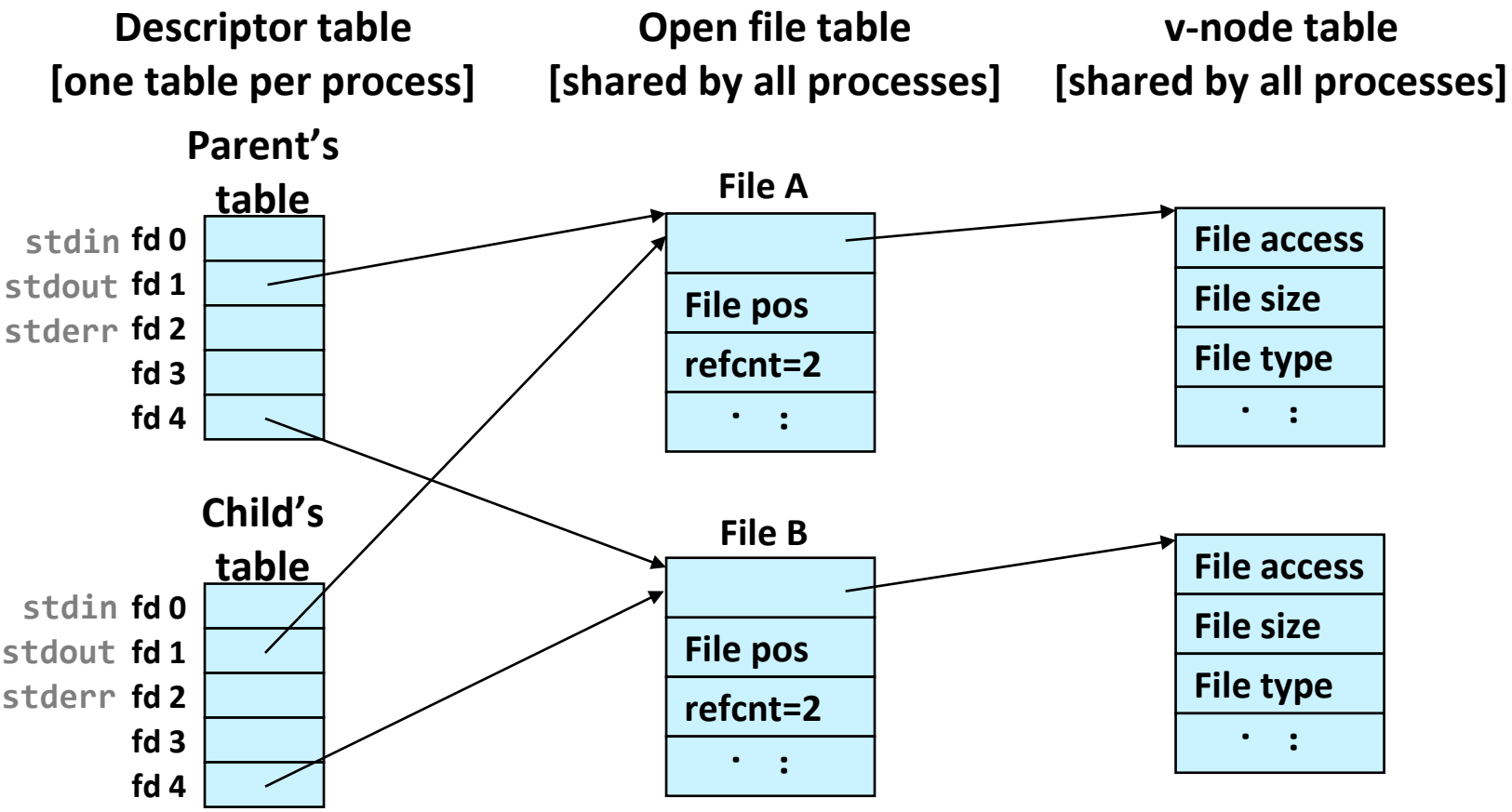
Open Files in Kernel (3)

- Calling open() twice with the same filename



Open Files in Kernel (4)

- Calling fork()



Open Files in Kernel (5)

- What will be the result?

```c
// ~swe2024-41_23s/2023s/w8/open_file.c

int main(void) {
    char c;
    int fd = open("temp.txt", O_RDONLY);

    if (fork() == 0) {  // child process
        read(fd, &c, 1); // reads 1 character
        exit(0); // child terminates
    } else {
        wait(NULL); // waits for child to exit
        read(fd, &c, 1); // reads 1 character
        printf("c=%c\n",c); // prints read character
    }

    return 0;
}
```

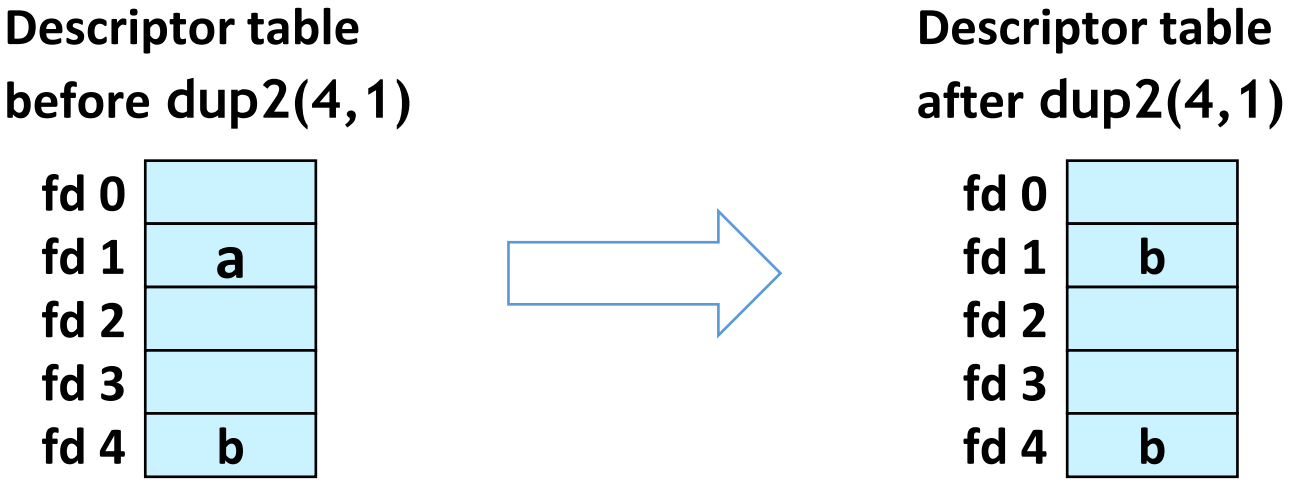
```c
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <assert.h>
#include <stdio.h>
```

temp.txt

sysexperiment

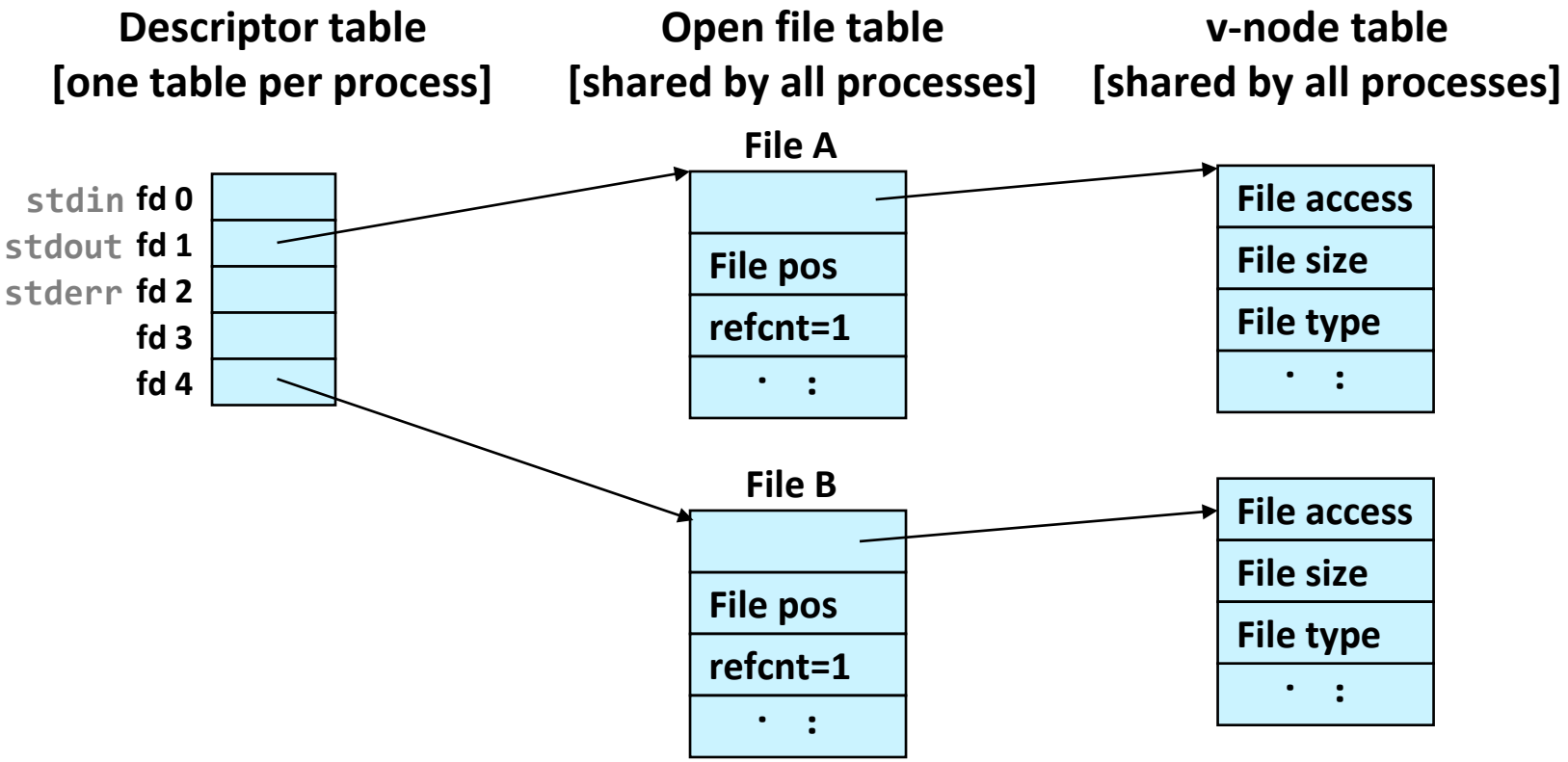
I/O Redirection

- 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**



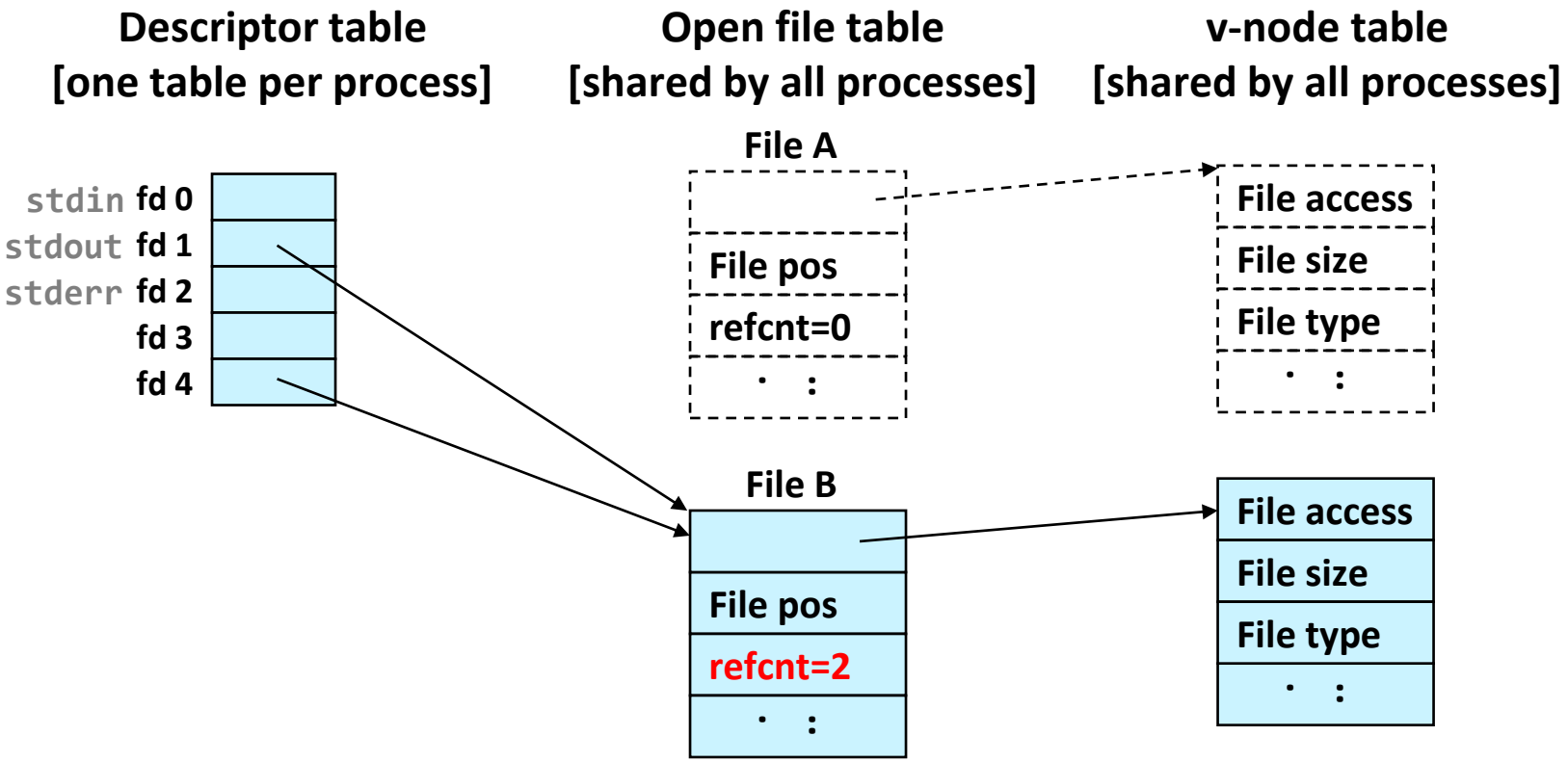
I/O Redirection Example (1)

- Before calling dup2(4,1)



I/O Redirection Example (2)

- After calling dup2(4,1)

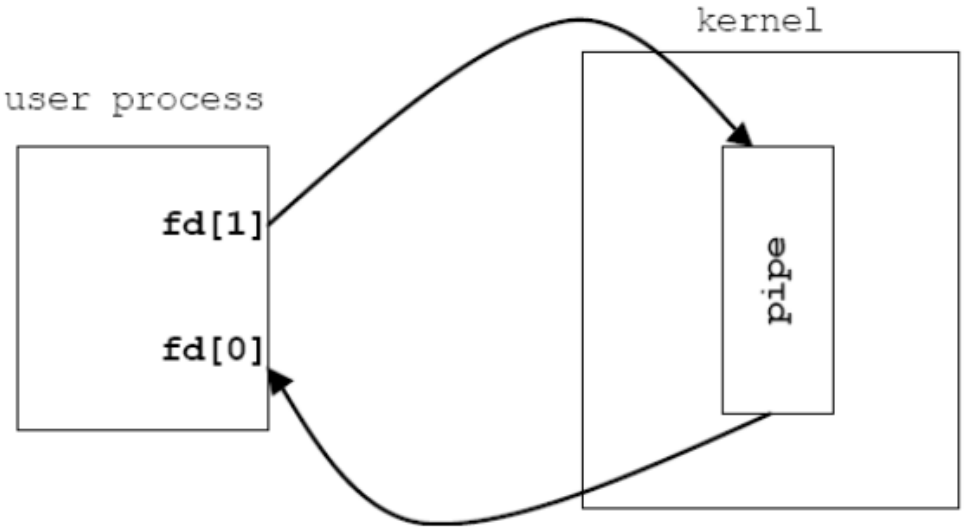


Pipes

- Pipes
  - Oldest mechanism for IPC in UNIX
  - Communicate with shared file descriptor of pipe
- Limitations
  - Half-duplex: data flows only in one direction
  - Data only can be read once
- Two pipes
  - Anonymous pipe
    - Temporary, between parent-child
  - Named pipe
    - Saved in file system, between arbitrary processes

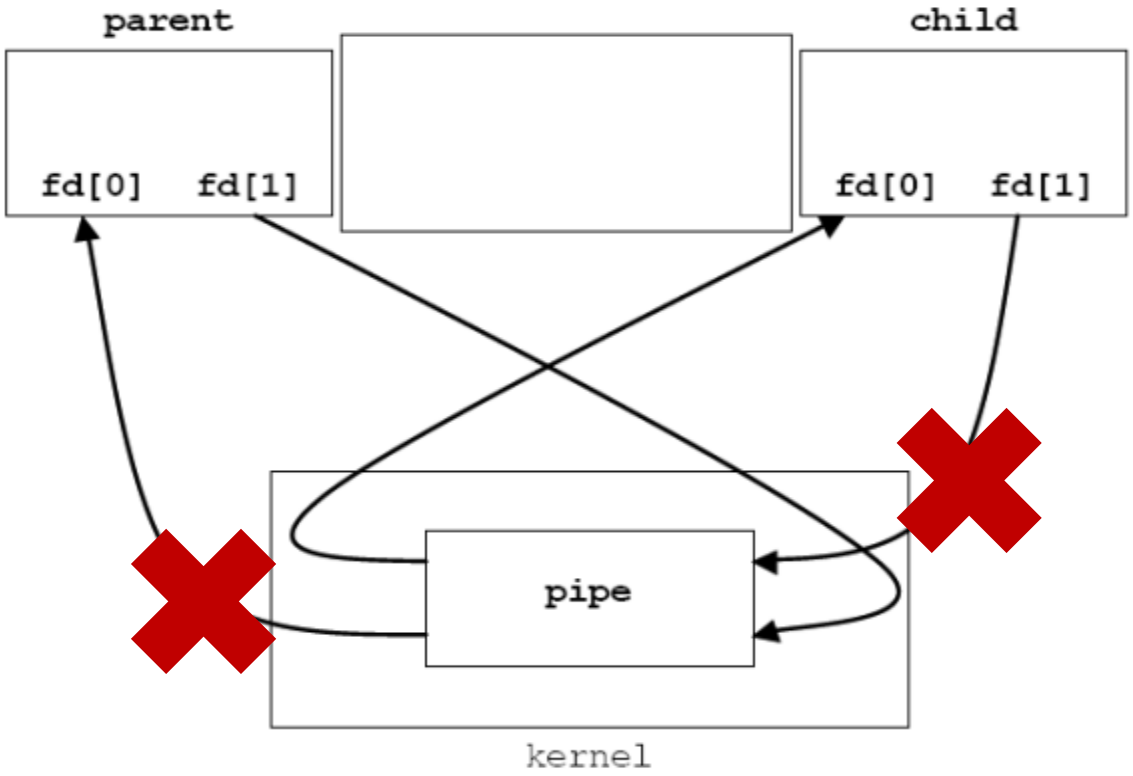
Anonymous Pipe (1)

- 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]**
  - On error, return -1



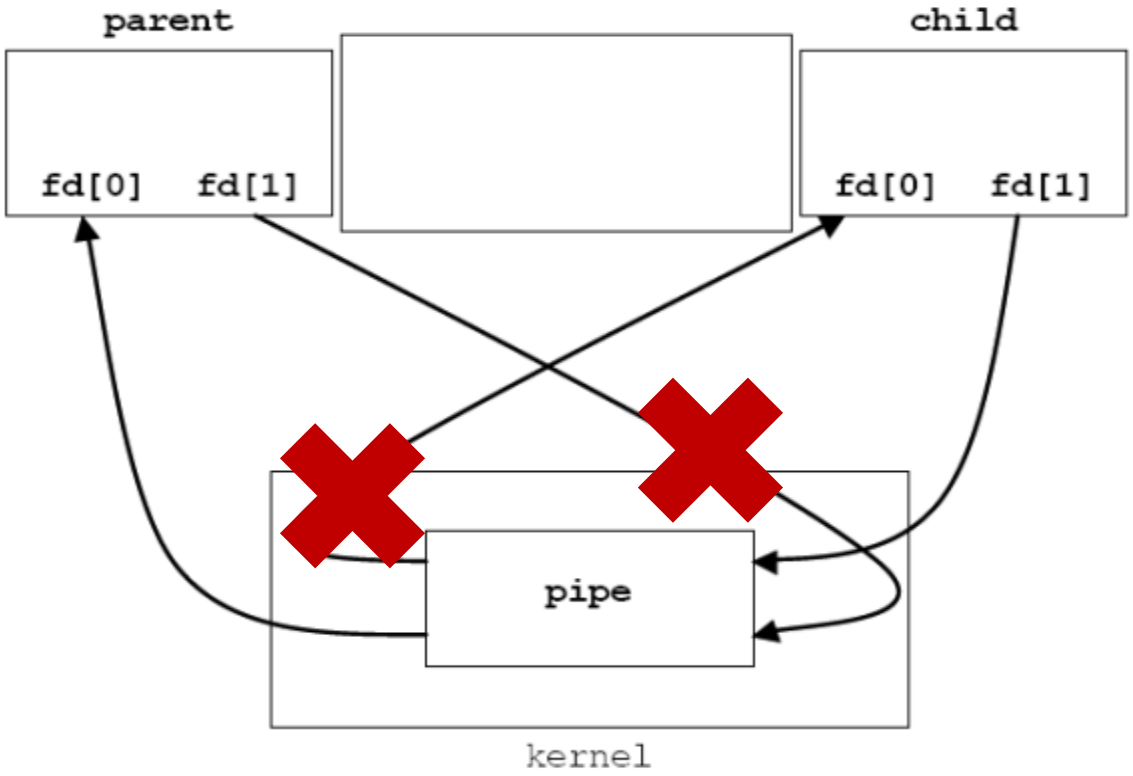
Anonymous Pipe (2)

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



Anonymous Pipe (3)

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



Using Anonymous Pipe

```c
// ~swe2024-41_23s/2023s/w8/anonymous_pipe.c
#include <unistd.h>
#include <stdlib.h>
#define MAXLINE 80

int main(void) {
    int n, fd[2]; // fd is used to receive the file descriptor of pipe
    pid_t pid;
    char line[MAXLINE]; // Contains a "string" of MAXLINE characters
    if(pipe(fd) < 0) exit(1); // Create an anonymos pipe, exit if failed
    if((pid = fork()) < 0) exit(2); // Fork to create child process, exit if
failed

    if (pid > 0) { // parent
        close(fd[0]); // Close as this fd will not be used by parent
        write(fd[1], "hello world\n", 12); // Writes "hello world" to fd[1]
    } else { // child
        close(fd[1]); // Close as this fd will not be used by child
        n = read(fd[0], line, MAXLINE); // Receives "hello world" from fd[0]
        write(1, line, n); // Writes "hello world" to fd 1 (STDOUT)
    }
    exit(0);
}
```

Named Pipe (FIFO)

- int mknod (const char ***path***, mode_t ***mode***,
  dev_t ***dev***)
  - Ex) mknod (“path”, S_IFIFO | 0666, 0);

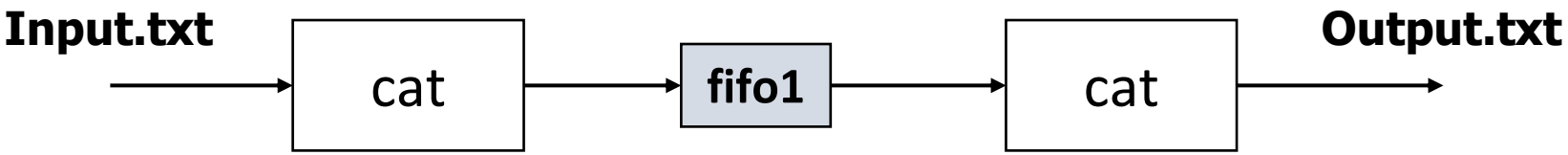
- int mkfifo (const char ***path***, mode_t ***mode***)
  - Ex) mkfifo (“path”, 0660);

- You can also make FIFOs on the command line by using ”mknod“ and ”mkfifo”

FIFO Example (1)

- Duplicating a stream
  - Passing data from one shell pipeline to another without creating intermediate temporary files

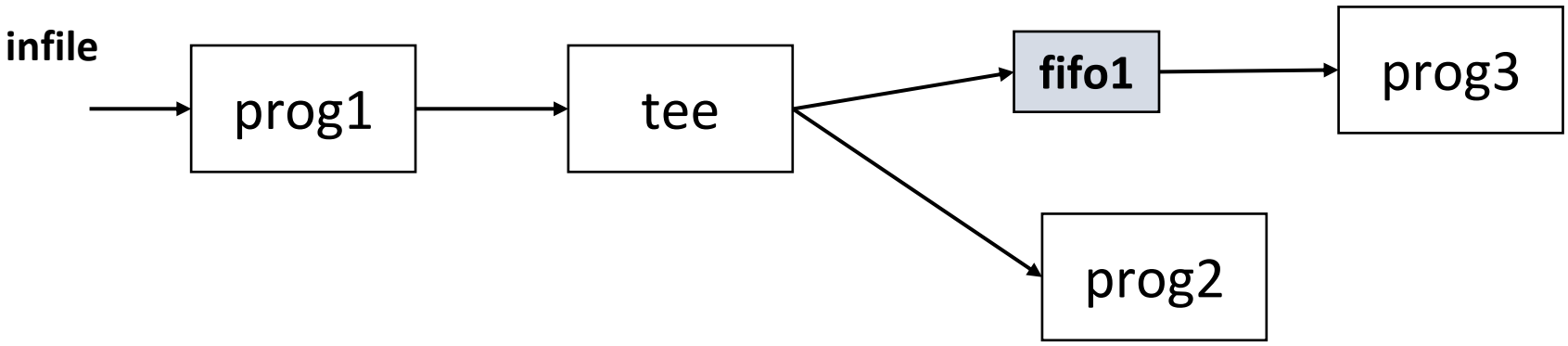
```
$ mkfifo fifo1
$ cat < fifo1 > output.txt &
$ cat < input.txt > fifo1
```



FIFO Example (1)

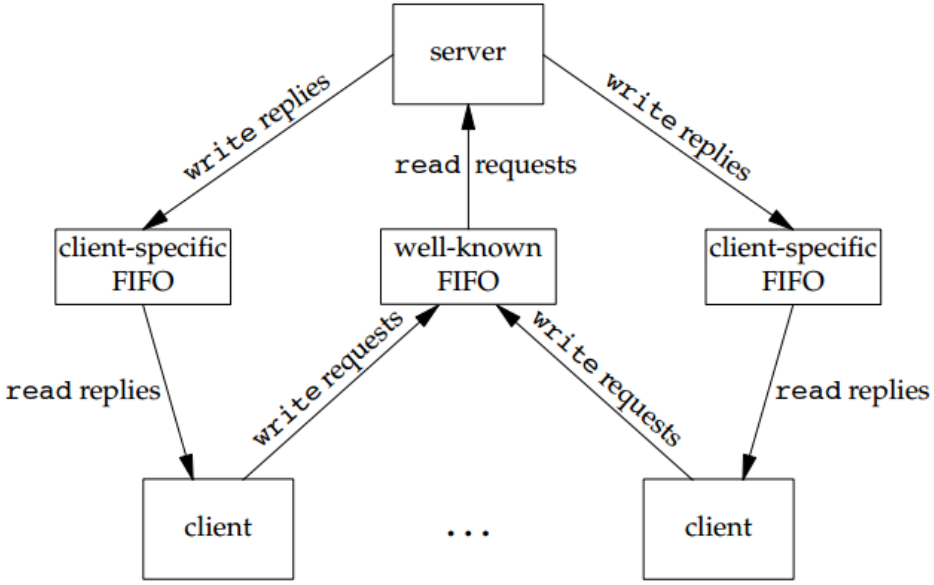
- Duplicating a stream
  - Passing data from one shell pipeline to another without creating intermediate temporary files

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



FIFO Example (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



Using FIFOs

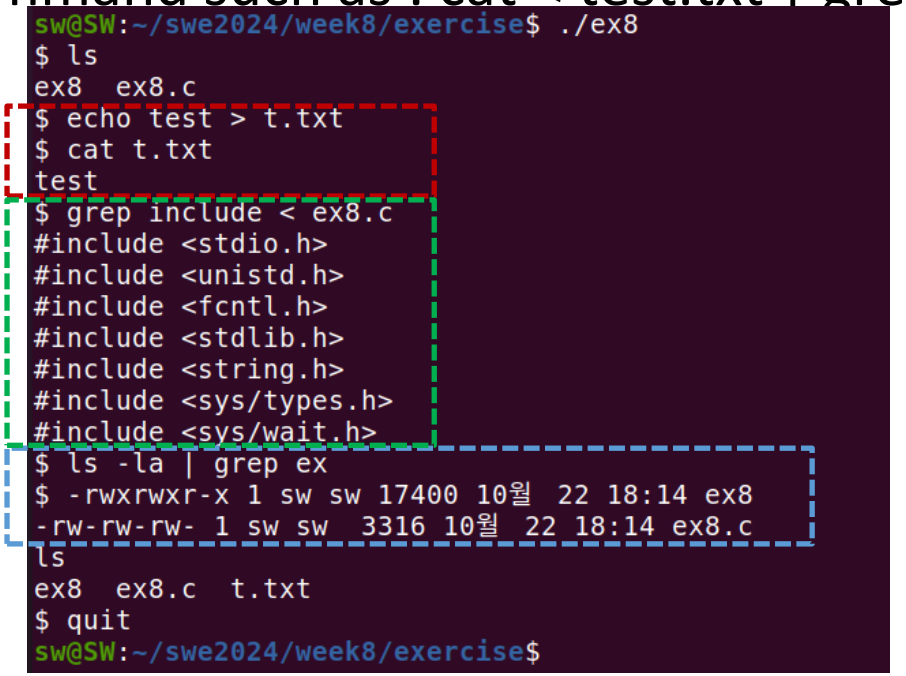
- Opening a FIFO
  - An open for read (write)-only blocks until some other process opens the FIFO for writing (reading)
- Reading / Writing a FIFO
  - Writing to a FIFO that no process has open for reading causes `SIGPIPE` to generate
  - When the last writer for a FIFO closes the FIFO, an end of file 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)

Lab Exercise

- Make *mini shell* which supports I/O redirection & pipes
  - This program should be under an infinite loop with conditional exit (“quit”)
  - When the command is entered, the command is executed using the child process
  - When the program is quitted, the parent process must wait for all the child processes to be done before exiting itself
  - The *mini shell* only executes programs under /bin directory
  - The *mini shell* must handle 2 types of redirections(>, <) and pipe(|)
  - There is no limit to the header file

Lab Exercise

- Skeleton code
  - cp ~swe2024-41_23s/2023s/p8_skeleton.c ./
- There will be only one IPC at one line
  - (e.g. No command such as : cat < test.txt | grep test)
- Example)



Exercise Submission

- Submit your exercise code and Makefile
  - InUiYeJi cluster
  - Submit the folder into p8
  - ~swe2024-41_23s/bin/submit p8 p8
  - Due date: Sunday, 23 April 2023, 23:59
  - We will compile by using command *make*
    - If compilation fails, your points for this exercise will be **zero**

Summary Report

- Summary report about man command result of
  - `pipe()`
  - `mkfifo()`

- Submission form
  - A4 size PDF format (No page limitation)
  - [SWE2024 Report-8] studentID_name
    e.g.) [SWE2024 Report-8] 2023XXXXXX_홍길동
  - Submit to iCampus
  - Due: until Sunday, 23 April 2023, 23:59