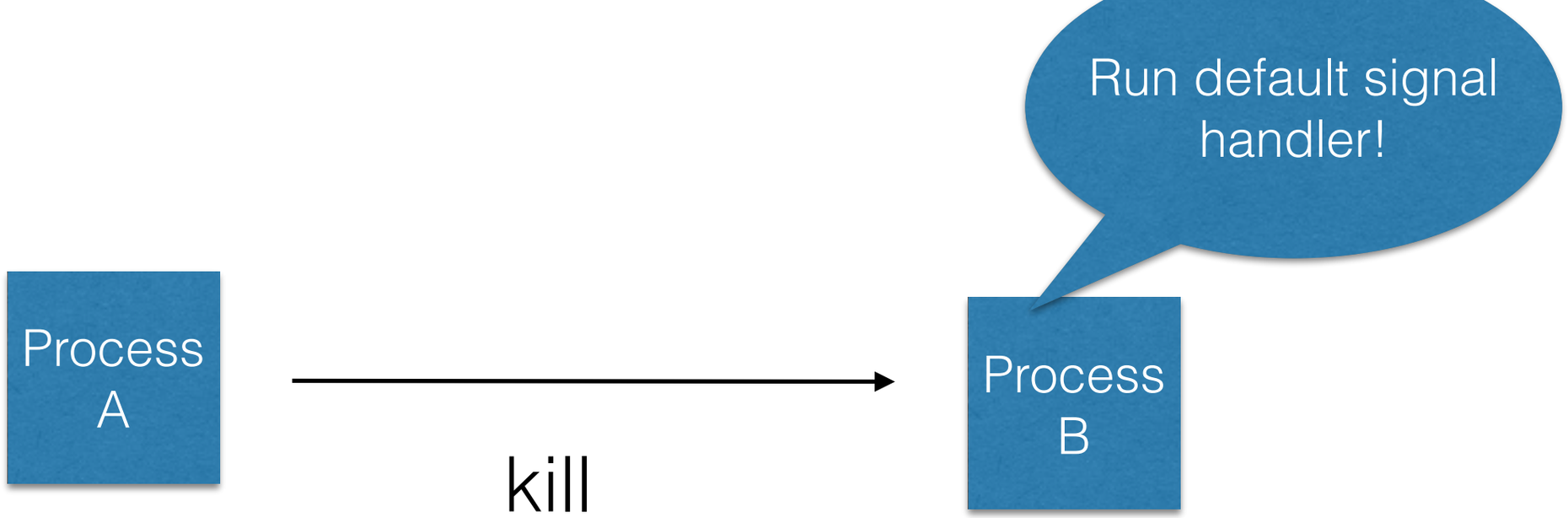
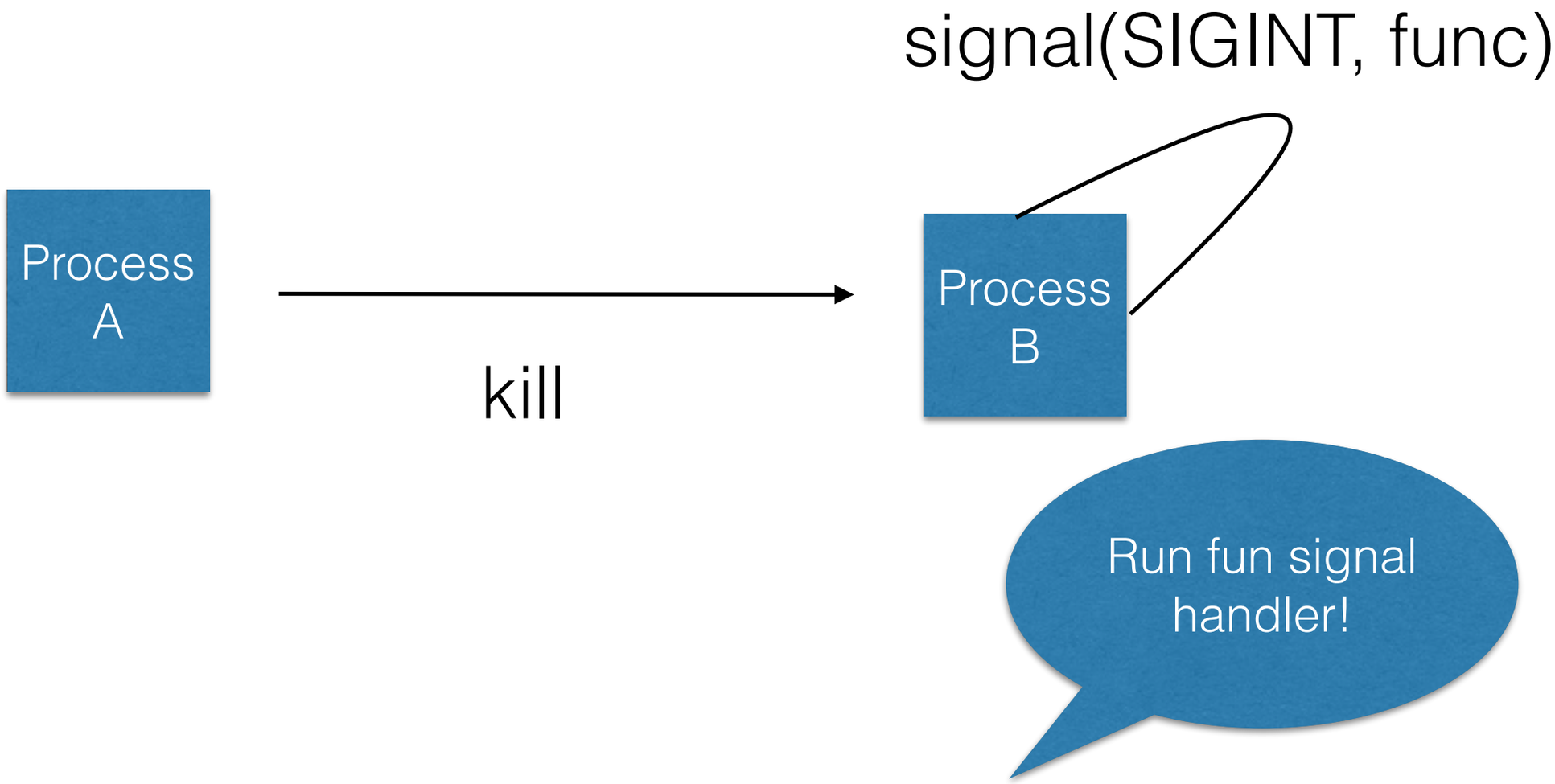
Process
A
kill
Process
B


Run default signal handler!
Process A
kill
Process B

signal(SIGINT, func)


signal(SIGINT, func)
Process A
kill
Process B
Run fun signal handler!

Terminate process

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

Generate Core Dump

Ignore

Stop

```
void sigint_handler(int sig)
{
    char c;
    signal(sig, SIG_IGN);
    printf(Ouch, you just hit Ctrl-C?. Do you really want to
quit [y/n]?);
    c = getchar();
    if (c == y || c = Y)
        exit(0);
    else
        signal(SIGINT, sigint_handler);
}
```

```
int
main(int argc, char **argv)
{
    int pid, status;
    int newfd;
    if ((newfd = open("output_file.txt", O_CREAT|O_TRUNC|O_WRONLY, 0644)) < 0) {
        exit(1);
    }

    printf("Luke, I am your...");
    dup2(newfd, 1);
    printf("father");
    exit(0);
}
```

```
int
main(int argc, char **argv)
{
    int pid, status;
    int newfd;
    char *cmd[] = { "/bin/ls", "-al", "/", 0 };
    if (argc != 2) {
        fprintf(stderr, "usage: %s output_file\n", argv[0]);
        exit(1);
    }
    if ((newfd = open(argv[1], O_CREAT|O_TRUNC|O_WRONLY, 0644)) < 0) {
        perror(argv[1]);    /* open failed */
        exit(1);

    }

    printf("writing output of the command %s to \"%s\"\n", cmd[0], argv[1]);
    dup2(newfd, 1);
    execvp(cmd[0], cmd);
    perror(cmd[0]);     /* execvp failed */
    exit(1);

}
```

int fd = open(“hello.txt”, FLAGS)

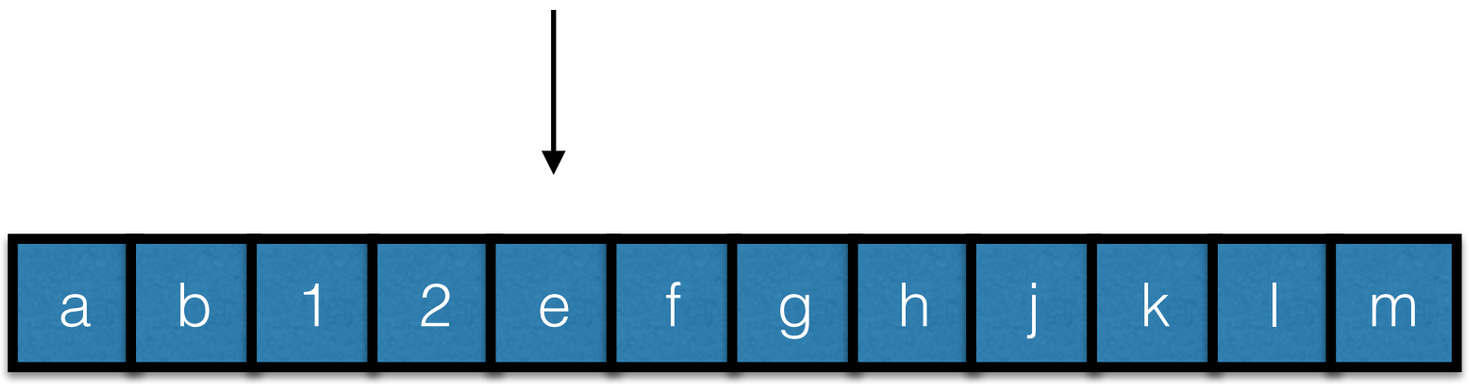
read(fd, buffer, 2)

read(fd, buffer, 2)

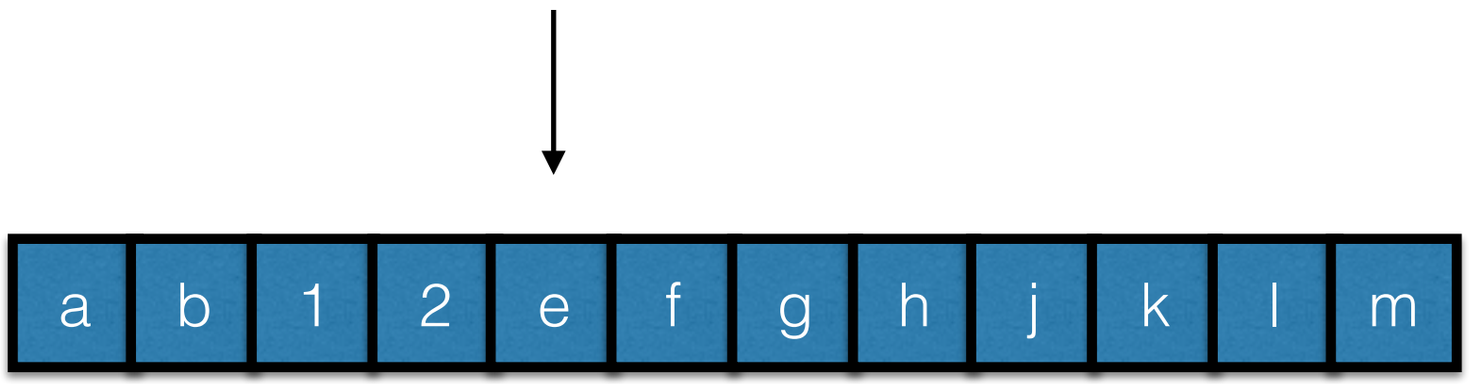
ab

write(fd, “12”, 2)

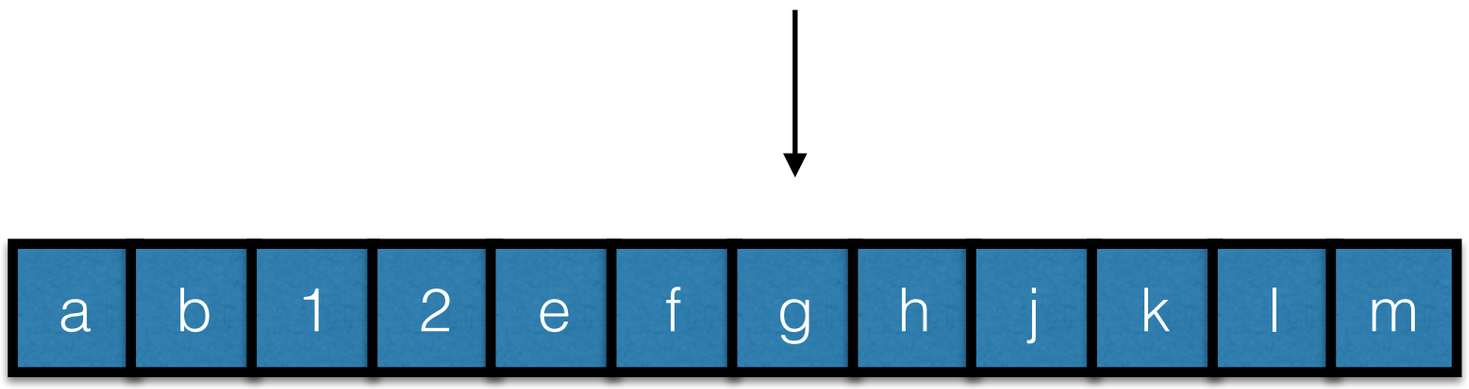
write(fd, “12”, 2)



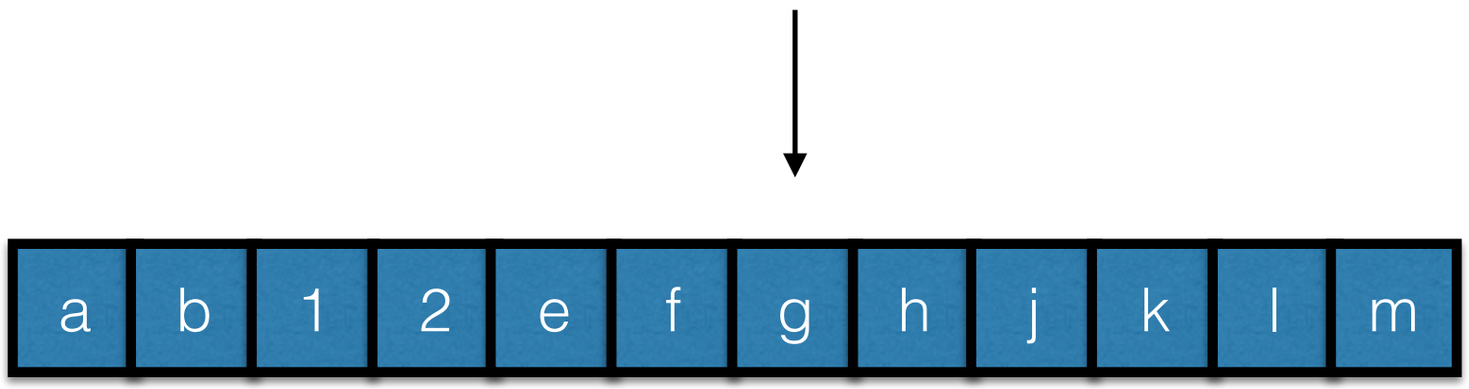
lseek(fd, 2, SEEK_CUR)



lseek(fd, 2, SEEK_CUR)



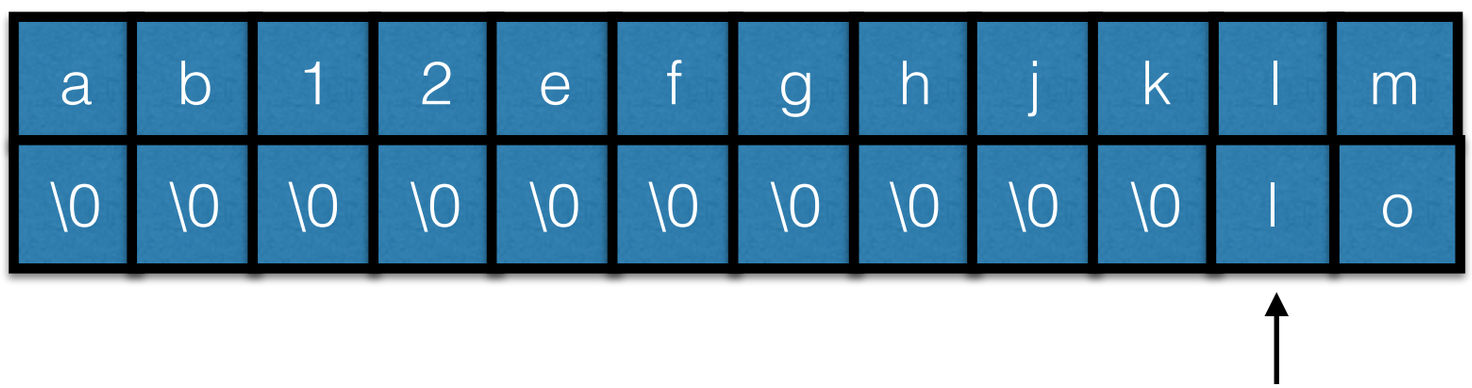
lseek(fd, 10, SEEK_END)



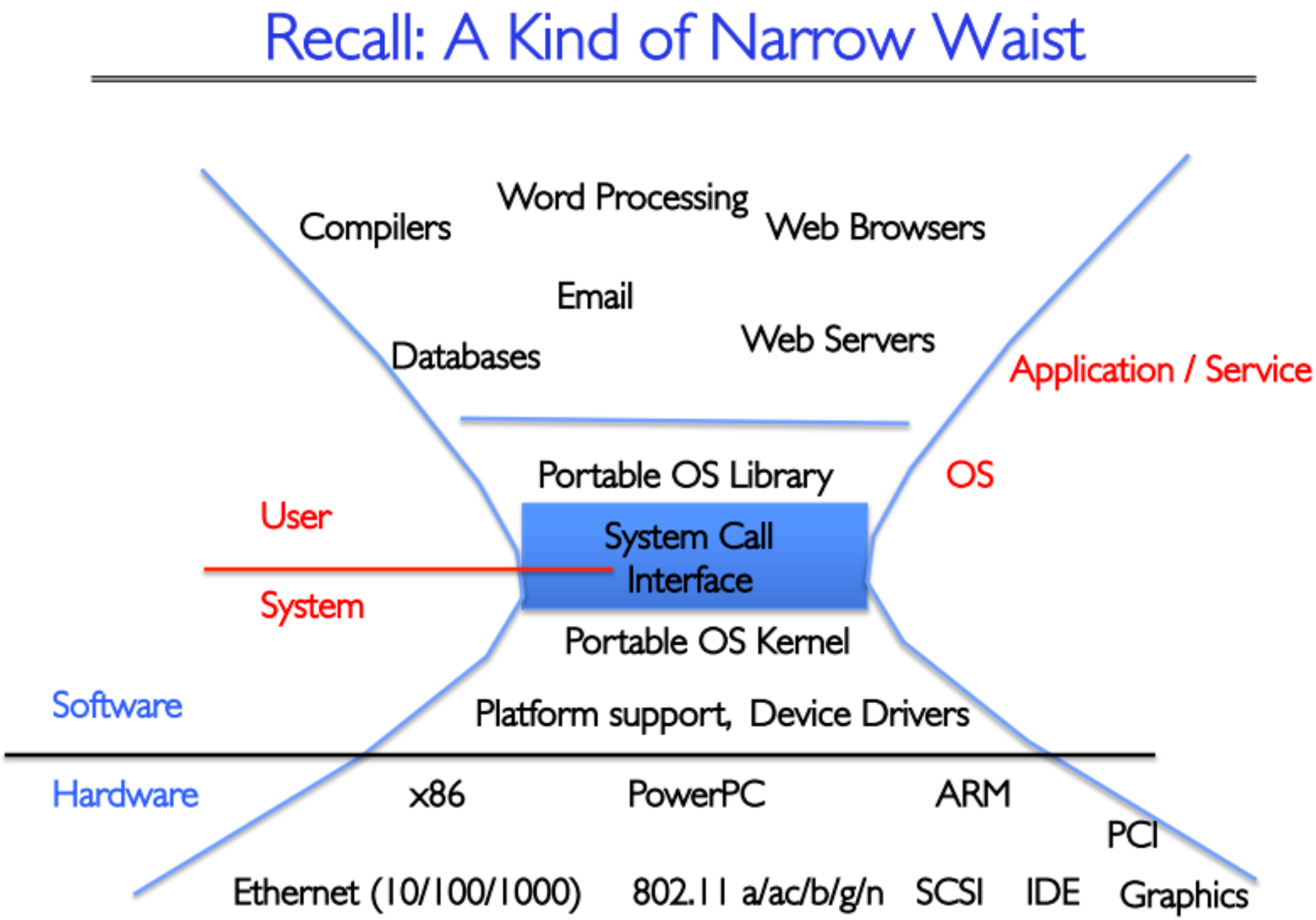
lseek(fd, 10, SEEK_END)

note that we don't write \0 in the middle until we make a write

write(fd, “lo”, 2)



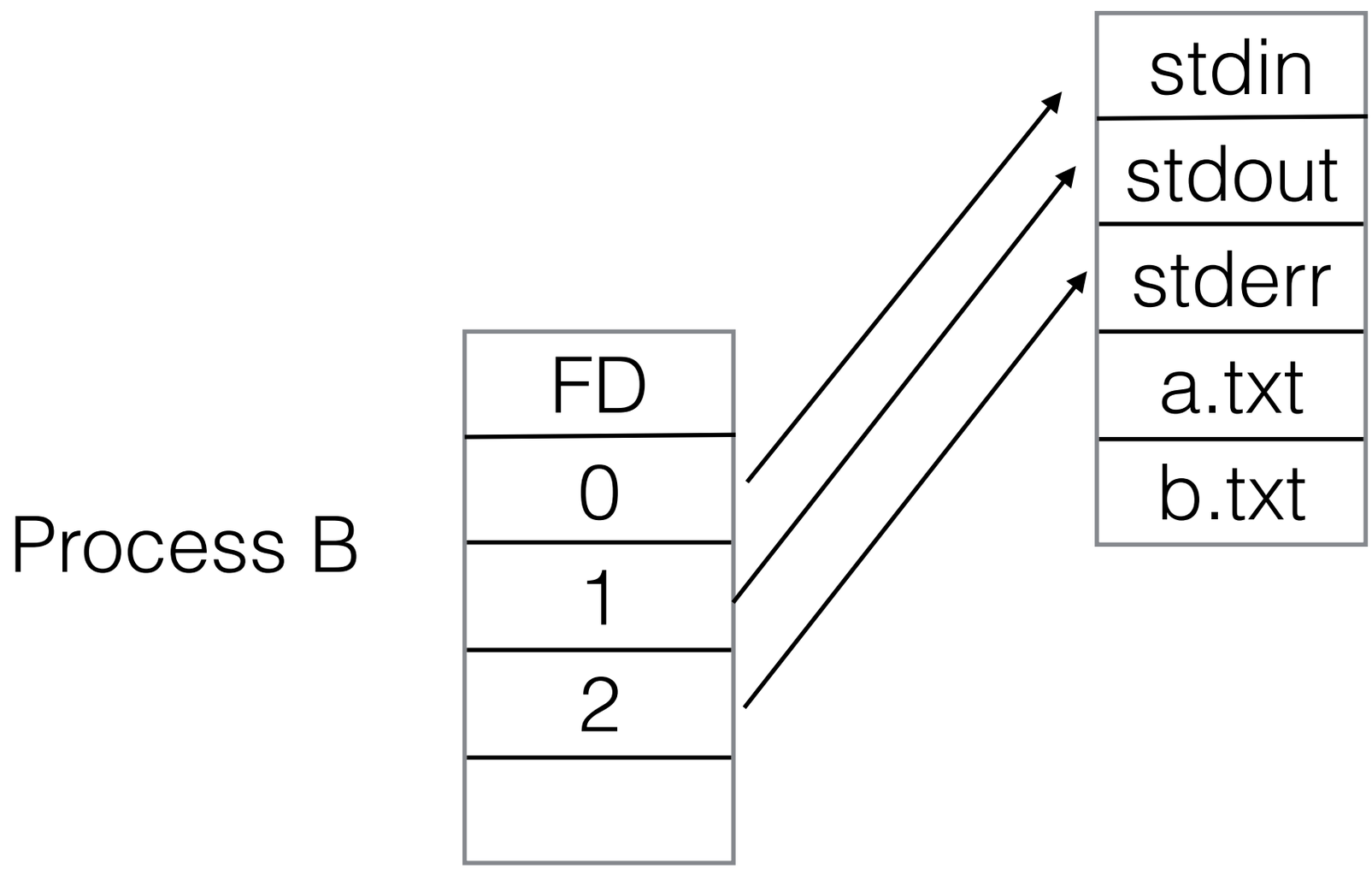
Files Vs File Descriptor



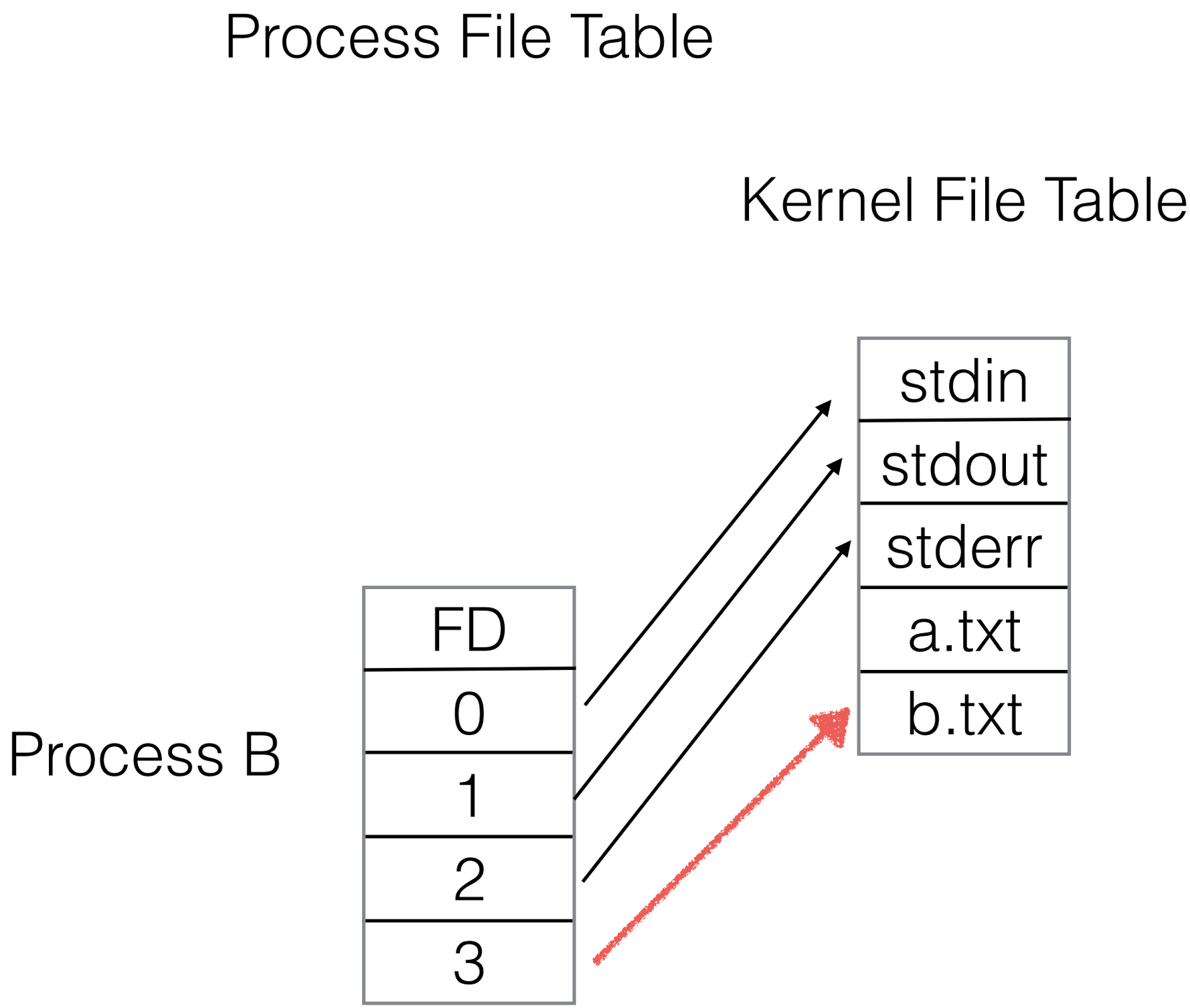
file descriptors

Process File Table

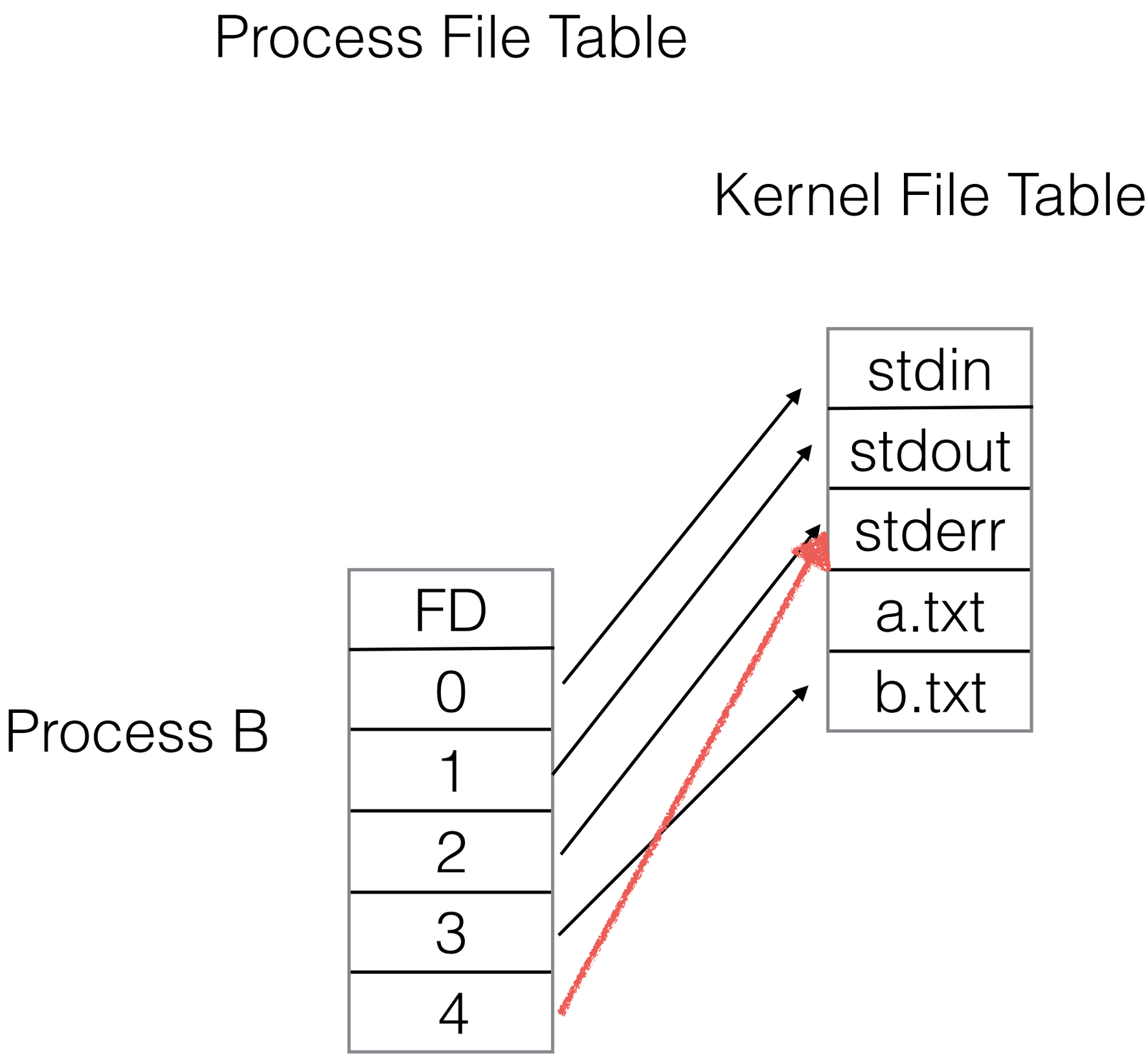
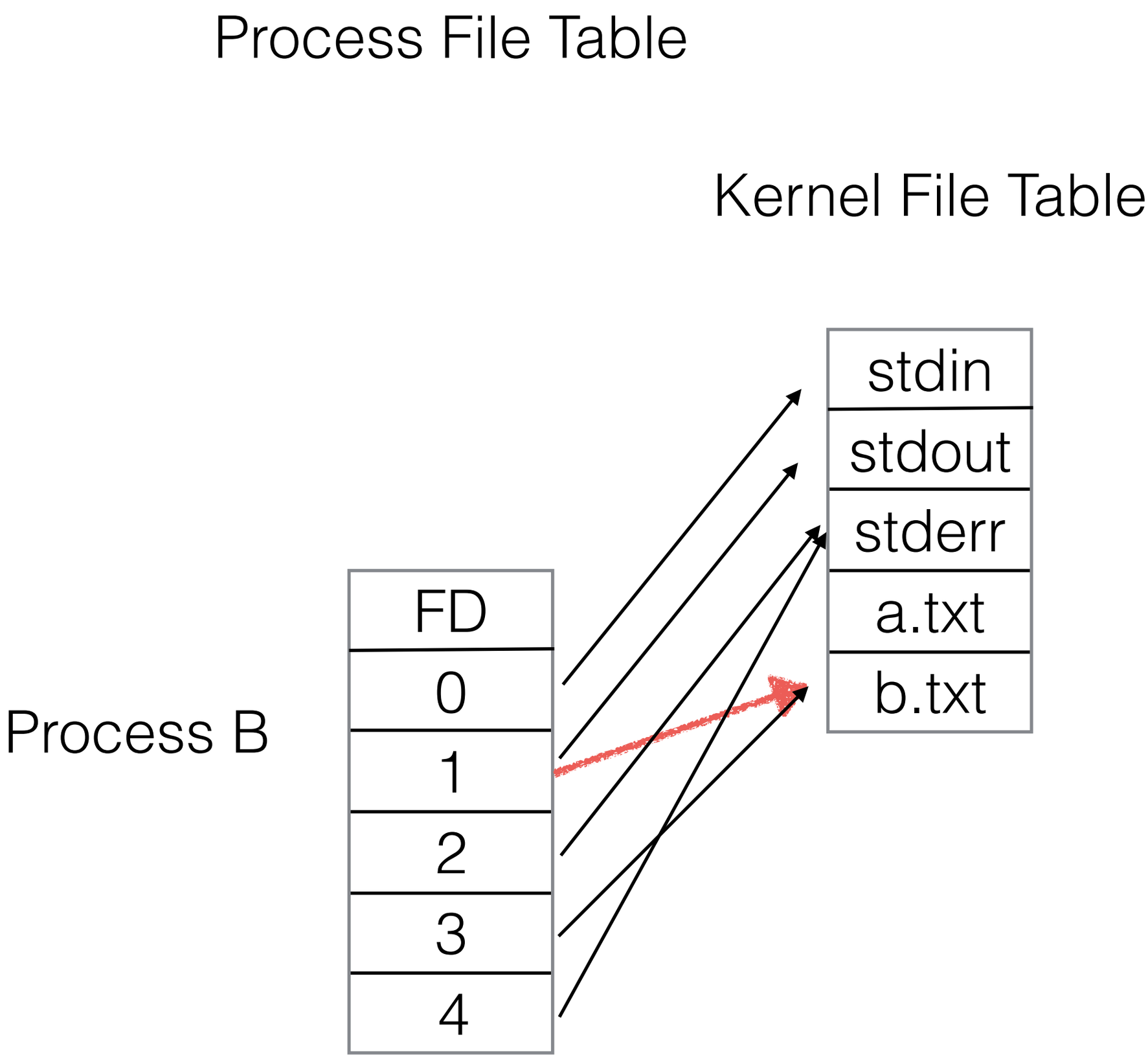
Kernel File Table



open(“b.txt”, O_CREAT|O_WR)



dup(2)


Process File Table
Kernel File Table
Process B
FD
0
1
2
3
4
stdin
stdout
stderr
a.txt
b.txt

dup2(3, STDOUT_FD)