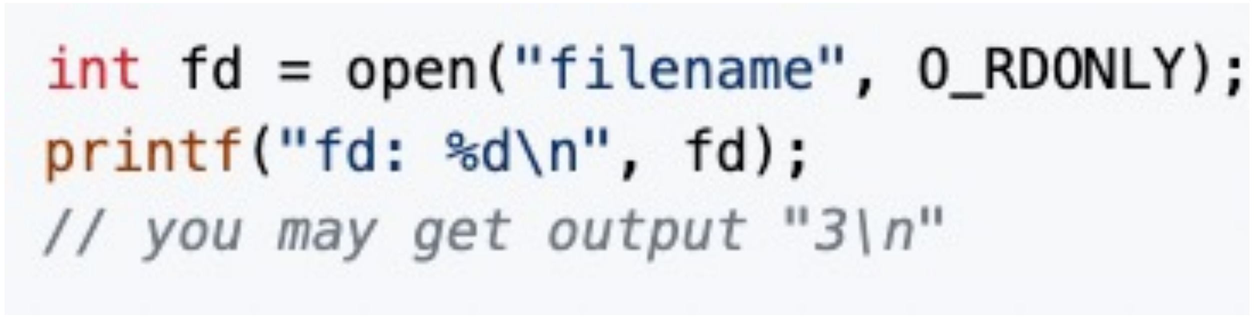
More about p2a

Jinlang Wang

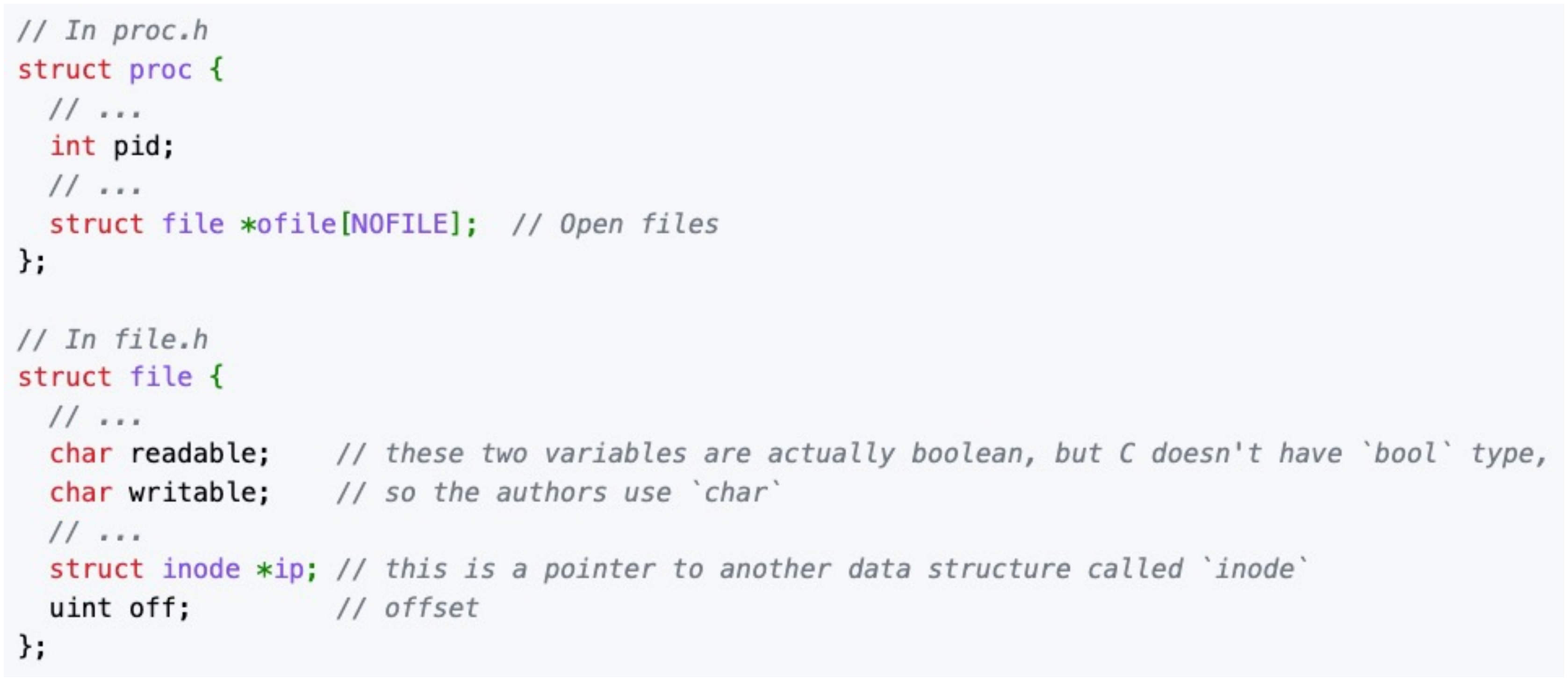
File Descriptor: How It Works

- If you have event printed out a file descriptor, you may notice it is just an integer



```c
int fd = open("filename", O_RDONLY);
printf("fd: %d\n", fd);
// you may get output "3\n"
```

Fd in xv6



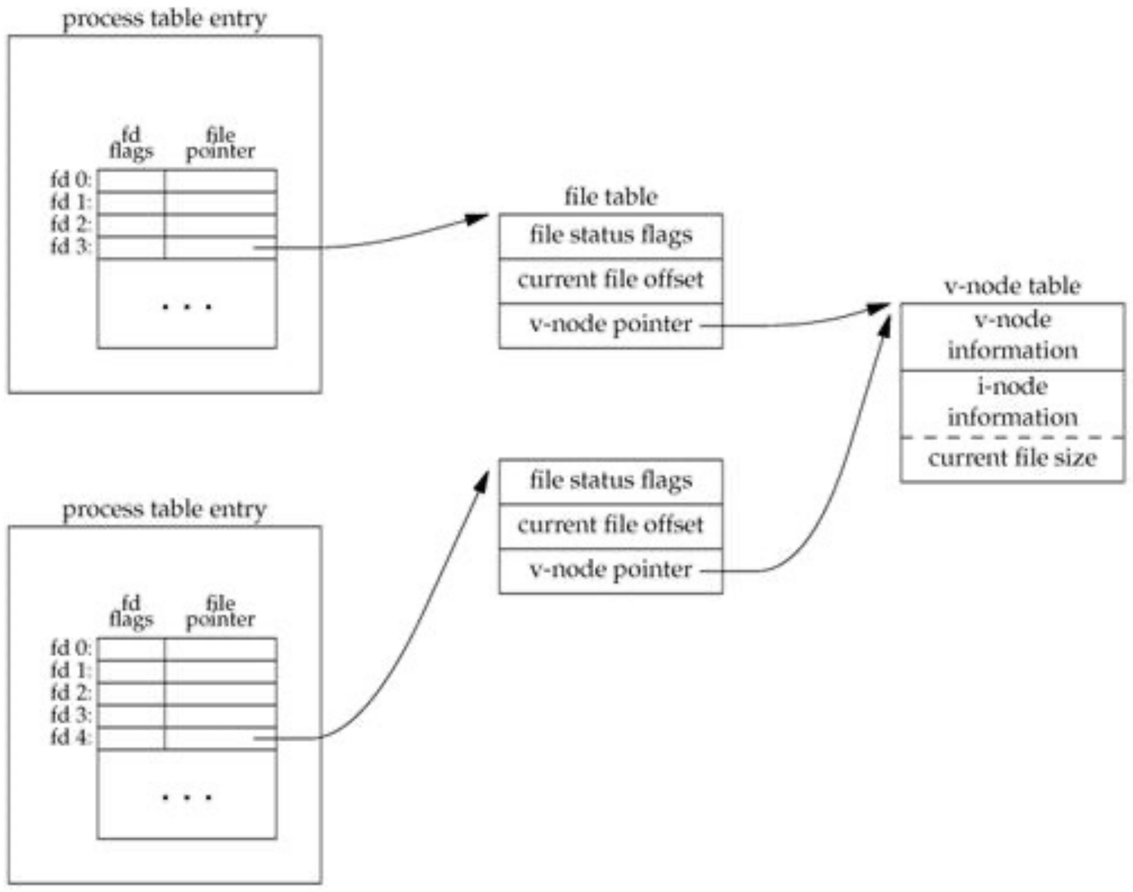
```c
// In proc.h
struct proc {
  // ...
  int pid;
  // ...
  struct file *ofile[NOFILE];  // Open files
};

// In file.h
struct file {
  // ...
  char readable;    // these two variables are actually boolean, but C doesn't have `bool` type,
  char writable;    // so the authors use `char`
  // ...
  struct inode *ip; // this is a pointer to another data structure called `inode`
  uint off;         // offset
};
```

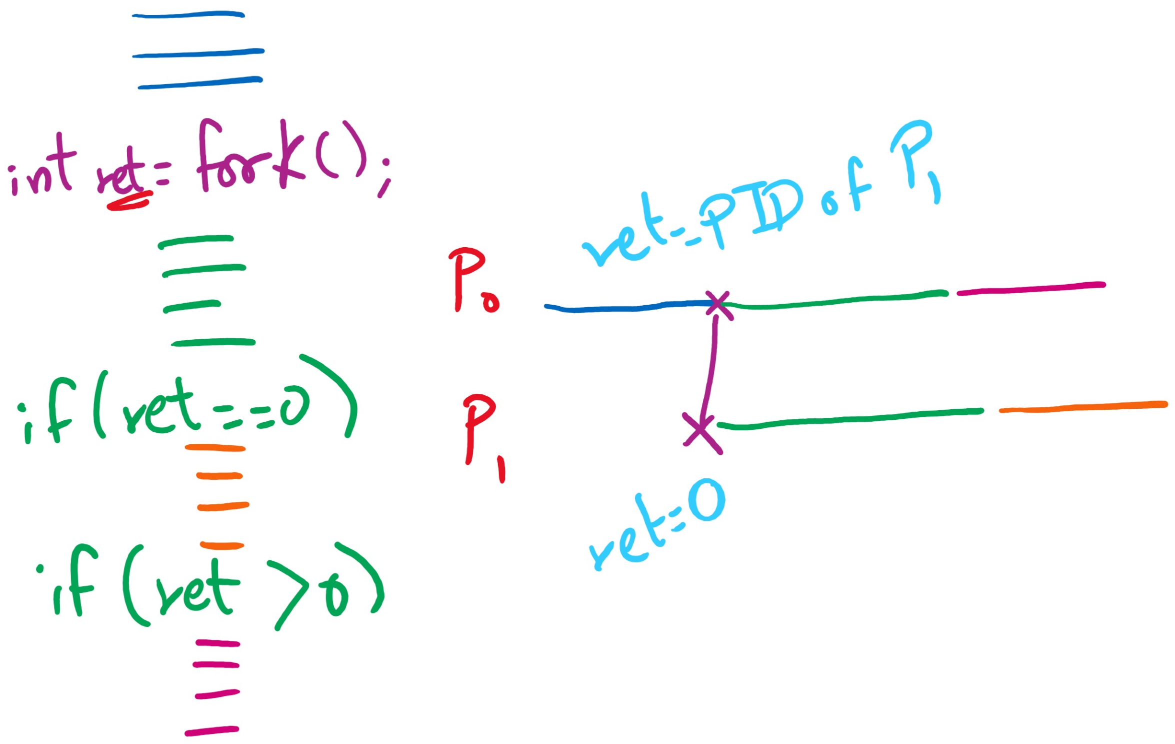
Fd

- File descriptors 0, 1, 2 are reserved for stdin, stdout, and stderr.
- File descriptor ranges from 0 to OPEN_MAX. File descriptor max value can be obtained with ulimit -n. For more information, go through 3rd chapter of APUE Book.

Figure 3.7. Two independent processes with the same file open



File Descriptors after fork()



```
int ret = fork();
if (ret == 0)
if (ret > 0)
```

File Descriptors after fork()

- During fork(), the new child process will copy proc.ofile (i.e. copying the pointers to struct file), but not struct file themselves. In other words, after fork(), both parent and child will share struct file. If the parent changes the offset, the change will also be visible to the child.

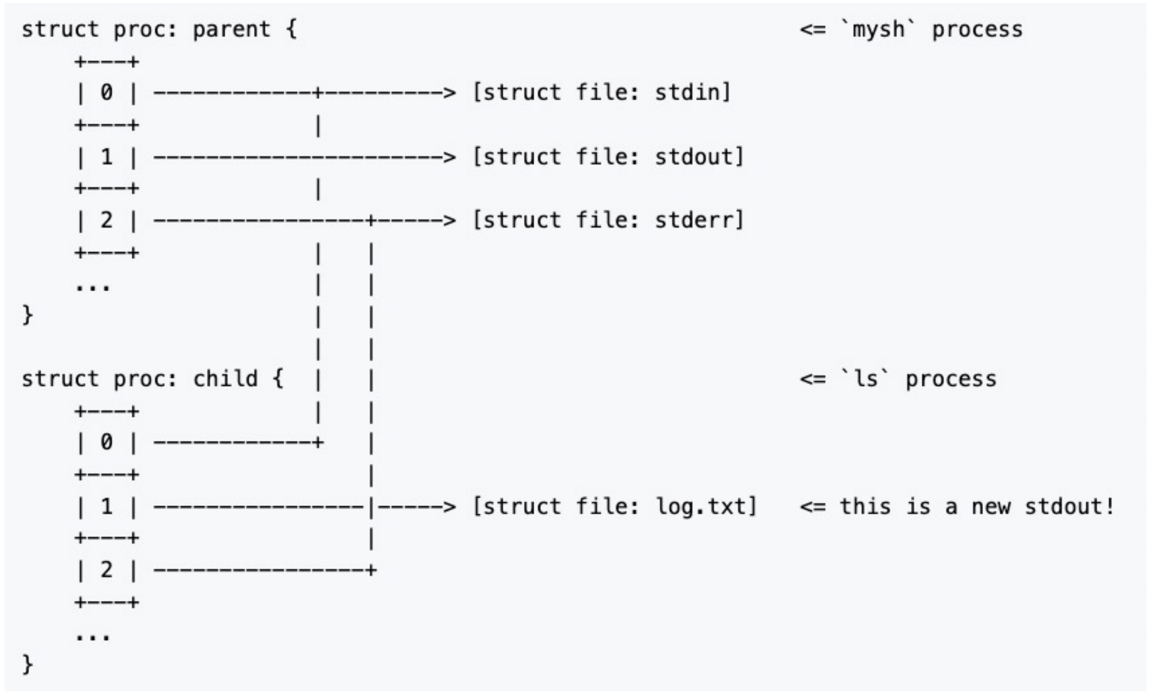
```
struct proc: parent {
    +---+
    | 0 | ------------+---------> [struct file: stdin]
    +---+             |
    | 1 | --------------+-------> [struct file: stdout]
    +---+             | |
    | 2 | ----------------+-----> [struct file: stderr]
    +---+             | | |
    ...               | | |
}                     | | |
                      | | |
struct proc: child {  | | |
    +---+             | | |
    | 0 | ------------+ | |
    +---+               | |
    | 1 | --------------+ |
    +---+                 |
    | 2 | ----------------+
    +---+
    ...
}
```

Redirection

- When a process writes to stdout, what it actually does it is writing data to the file that is associated with the file descriptor 1. So the trick of redirection is, we could replace the struct file pointer at proc.ofile[1] with another struct file.

e.x.

- When handling the shell command ls > log.txt, what the file descriptors eventually should look like:



```
struct proc: parent {                                      <= `mysh` process
    +---+
    | 0 | ------------+---------> [struct file: stdin]
    +---+             |
    | 1 | ----------------------> [struct file: stdout]
    +---+             |
    | 2 | ----------------+-----> [struct file: stderr]
    +---+             |   |
    ...               |   |
}                     |   |
                      |   |
struct proc: child {  |   |                                <= `ls` process
    +---+             |   |
    | 0 | ------------+   |
    +---+                 |
    | 1 | ----------------|-----> [struct file: log.txt]   <= this is a new stdout!
    +---+                 |
    | 2 | ----------------+
    +---+
    ...
}
```

dup2()

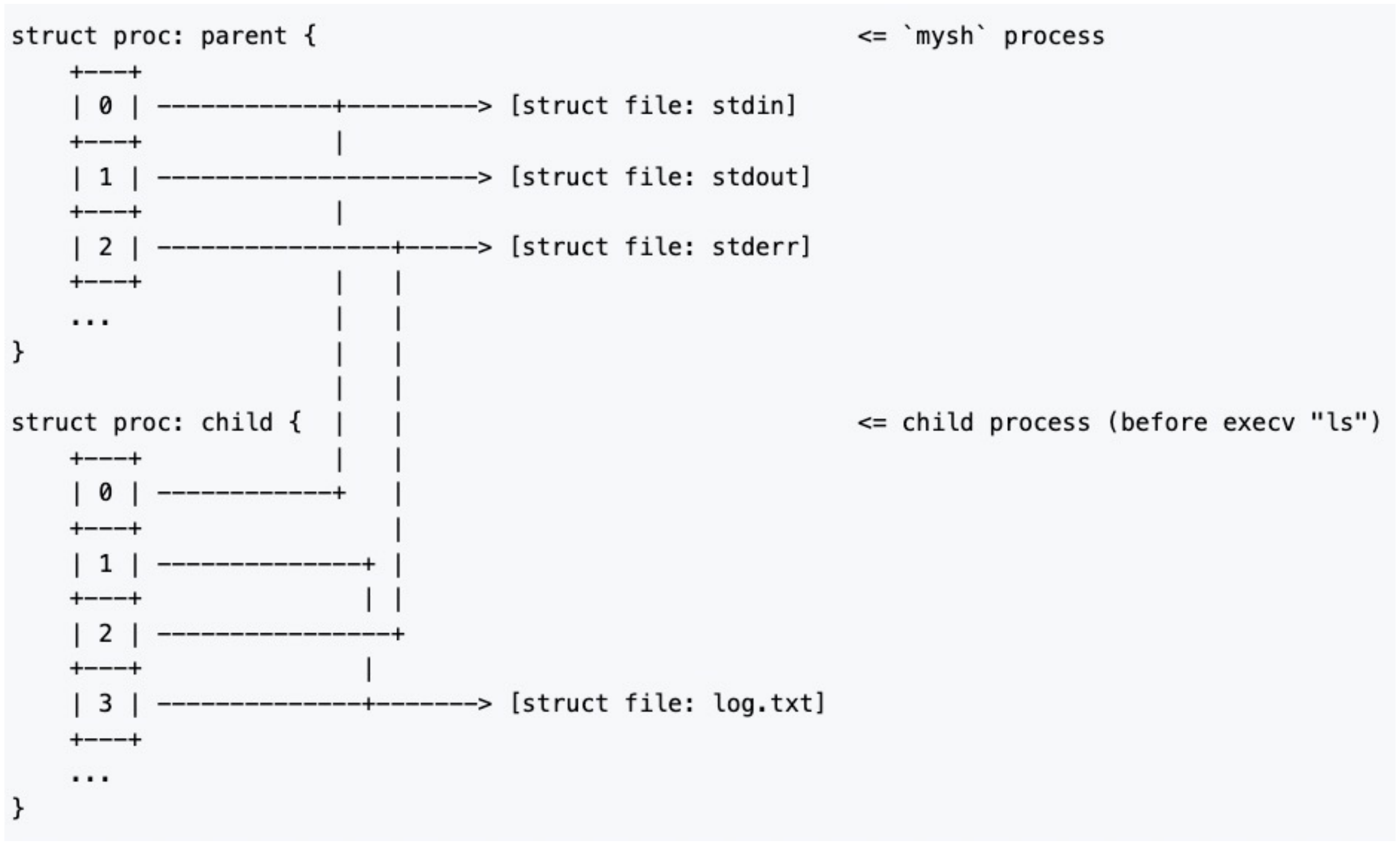
- The trick to implement the redirection is the syscall dup2. This syscall performs the task of "duplicating a file descriptor. "
- int dup2(int oldfd, int newfd);

```c
int pid = fork();
if (pid == 0) { // child;
    int fd = open("log.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644); // [1]
    dup2(fd, fileno(stdout));                                     // [2]
    // execv(...)
}
```

After [1]

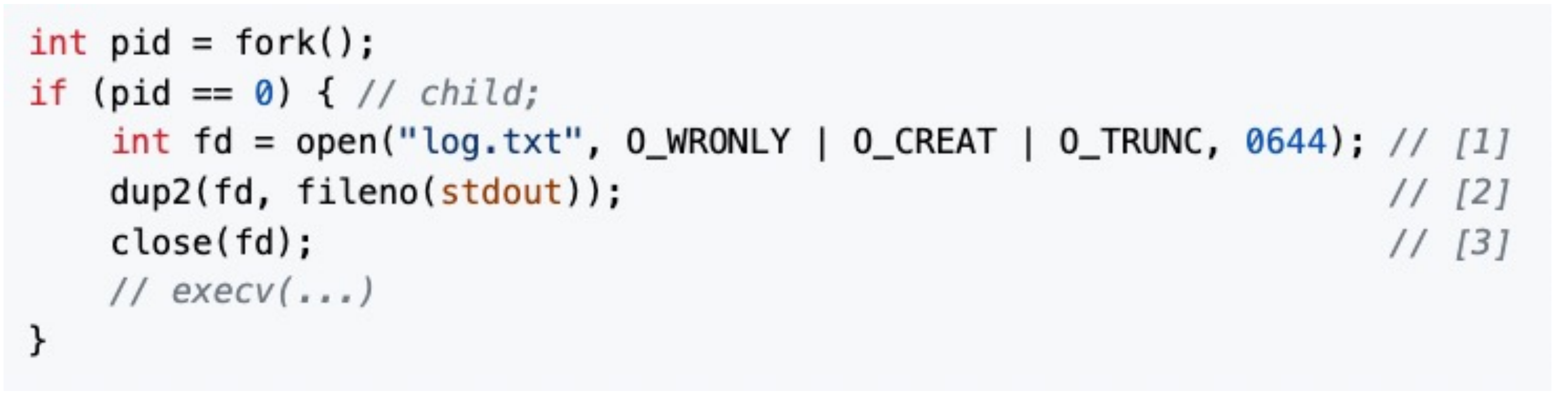
```
struct proc: parent {                                     <= `mysh` process
    +---+
    | 0 | ------------+---------> [struct file: stdin]
    +---+             |
    | 1 | --------------+-------> [struct file: stdout]
    +---+             | |
    | 2 | ----------------+-----> [struct file: stderr]
    +---+             | | |
    ...               | | |
}                     | | |
                      | | |
struct proc: child {  | | |                               <= child process (before execv "ls")
    +---+             | | |
    | 0 | ------------+ | |
    +---+               | |
    | 1 | --------------+ |
    +---+                 |
    | 2 | ----------------+
    +---+
    | 3 | ----------------------> [struct file: log.txt]  <= open a file "log.txt"
    +---+
    ...
}
```

After [2]



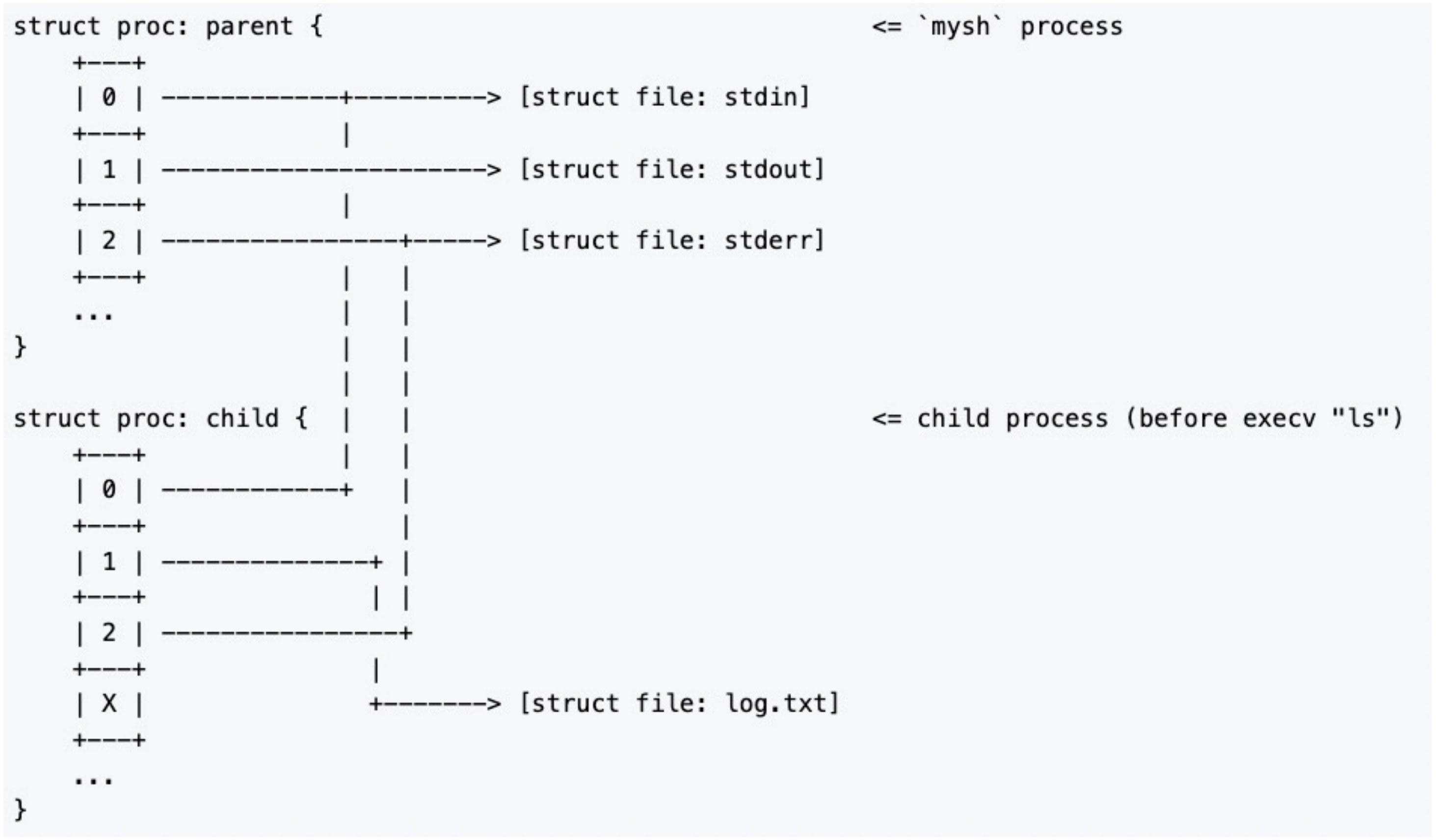
```
struct proc: parent {                                     <= `mysh` process
    +---+
    | 0 | ------------+---------> [struct file: stdin]
    +---+             |
    | 1 | ----------------------> [struct file: stdout]
    +---+             |
    | 2 | ----------------+-----> [struct file: stderr]
    +---+             |   |
    ...               |   |
}                     |   |
                      |   |
struct proc: child {  |   |                               <= child process (before execv "ls")
    +---+             |   |
    | 0 | ------------+   |
    +---+                 |
    | 1 | --------------+ |
    +---+               | |
    | 2 | ----------------+
    +---+               |
    | 3 | --------------+-------> [struct file: log.txt]
    +---+
    ...
}
```

Remember to close fd



```c
int pid = fork();
if (pid == 0) { // child;
    int fd = open("log.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644); // [1]
    dup2(fd, fileno(stdout));                                      // [2]
    close(fd);                                                     // [3]
    // execv(...)
}
```

After [3]



```
struct proc: parent {                                     <= `mysh` process
    +---+
    | 0 | ------------+---------> [struct file: stdin]
    +---+             |
    | 1 | ----------------------> [struct file: stdout]
    +---+             |
    | 2 | ----------------+-----> [struct file: stderr]
    +---+             |   |
    ...               |   |
}                     |   |
                      |   |
struct proc: child {  |   |                               <= child process (before execv "ls")
    +---+             |   |
    | 0 | ------------+   |
    +---+                 |
    | 1 | --------------+ |
    +---+               | |
    | 2 | ----------------+
    +---+               |
    | X |               +-------> [struct file: log.txt]
    +---+
    ...
}
```

Document of dup2()

- https://man7.org/linux/man-pages/man2/dup2.2.html

Difference Between FILE* and File Descriptors

- FILE *fopen(const char *pathname, const char *mode);
- The return value of fopen is a FILE pointer, not an integer (file descriptor).
- Note that, fopen() is not a syscall, but a library function (in stdio.h. It is a wrapper built on top of open(). Similarly, FILE is also defined in stdio.h, which implements some buffering for IO.

If Statement

```
if      one              ==       1        then      hello         fi
  (condition cmd)       (op)  (CONSTANT)        (EXECUTIIVE CMD)
```

$$\begin{tikzcd}
0 \arrow[r, "\text{if}"] & 1 \arrow[r, "\text{token}"] & 2 \arrow[loop above, "\text{token}"] \arrow[r, "==/!="] & 3 \arrow[r, "\text{constant}"] & 4 \arrow[r, "\text{then}"] & 5 \arrow[r, "\text{token}"] \arrow[rr, bend right, "\text{fi}"'] & 6 \arrow[loop above, "\text{token}"] \arrow[r, "\text{fi}"] & 7
\end{tikzcd}$$