dup() and friends

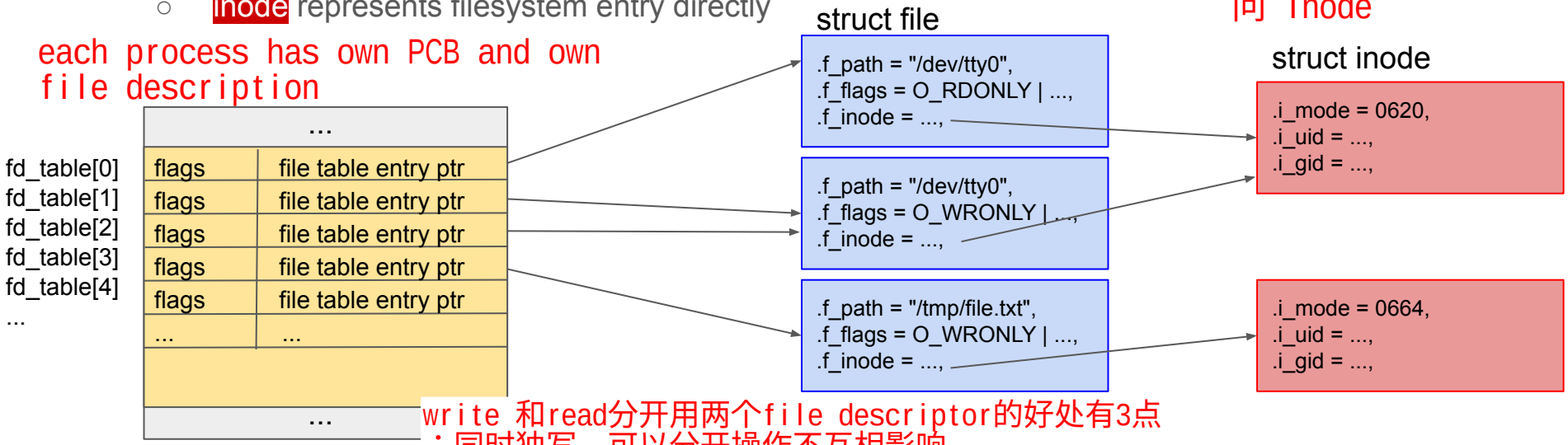
duplicate function

How does the OS track file descriptors?

- File descriptors are process-specific
  - Tracked in PCB, in an array mapping fds to {flags, file table entry pointer}s
  - File table entries track flags for the file, the current position in the file, and a pointer to the corresponding inode (conceptually, an open file)
  - inode represents filesystem entry directly

each process has own PCB and own file description

file system 可以访问 inode



write 和read分开用两个file descriptor的好处有3点：同时独写，可以分开操作不互相影响

How can a process change the file table in its PCB?

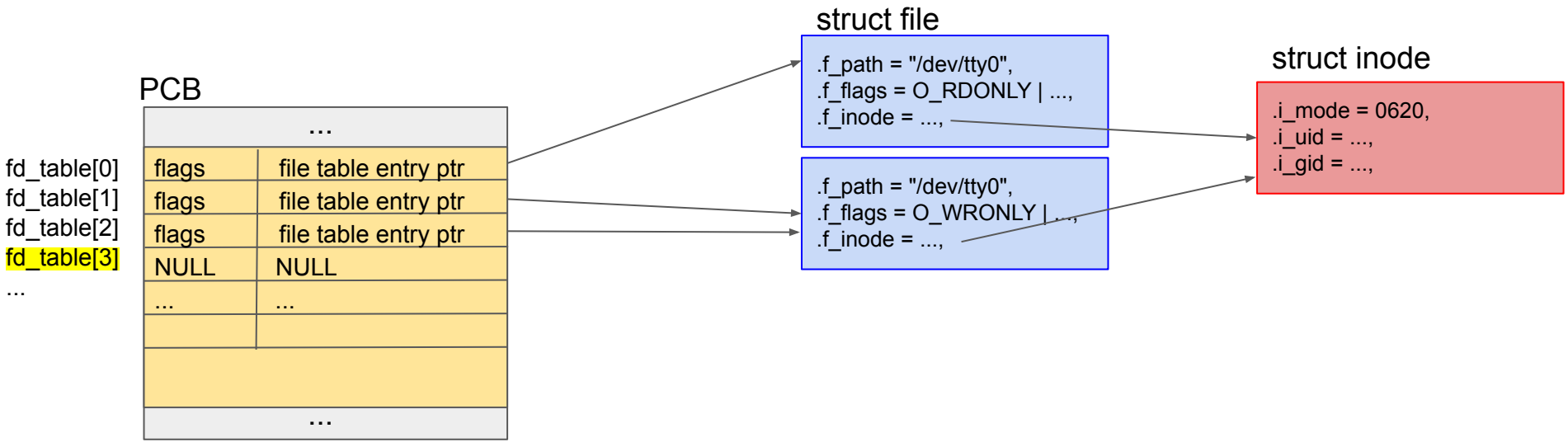
- Remember that we don't have direct access to our PCB, and that we must ask the kernel to read/modify it thru system calls.
  - fd = open(path, flags, mode) - allocate a new "struct file" in the kernel & make a new file descriptor for our process which points to it. return an integer, 是file desriptor ID
  - close(fd) - remove the entry corresponding to fd in our table, if no pointers left to the "struct file", the kernel will free the "struct file" as well
  - dup(fd) - create a new file descriptor in our table, which is an exact copy of fd from our table. fd passed to the function and the returned fd will point to the same "struct file" after this operation. 但是copy后creat a new，会有一个新的descriptor ID
  - dup2(fd1, fd2) - the entry at fd2 in our PCB will be overwritten with the contents at fd1, so that fd1 and fd2 now both point to the same "struct file" 不会creat a new， 是用fd1去raplace fd2
    - if fd2 was last pointer to that "struct file", then the corresponding struct will be freed

Example: `open("/tmp/file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666)`

如果file不存在，
会creat file，
存在就忽略creat

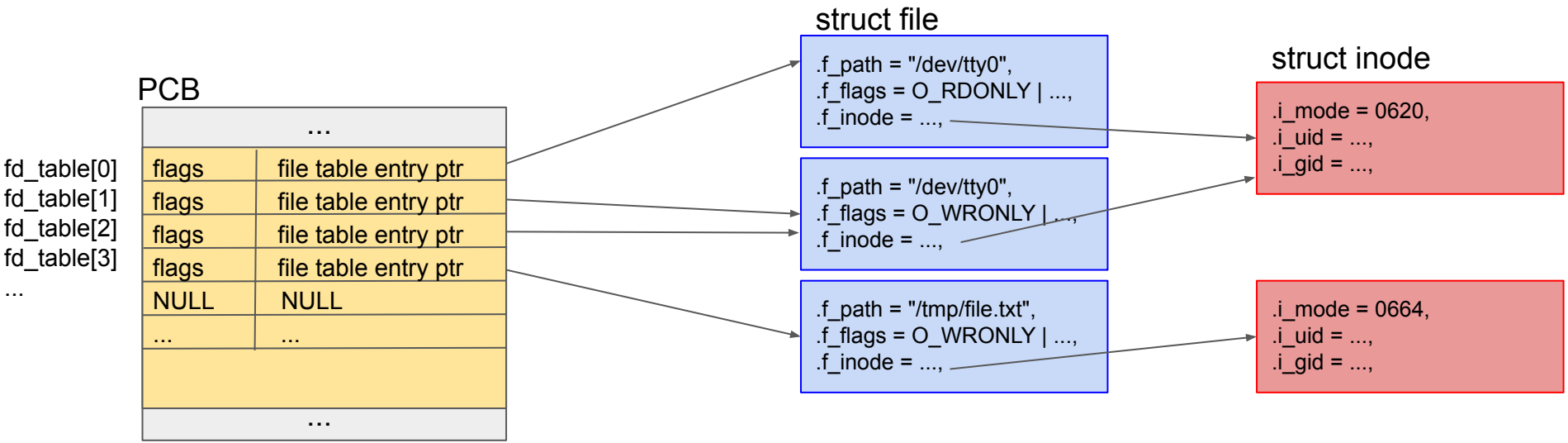
十六进制的数
permintion control

BEFORE



Example: `open("/tmp/file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666)`

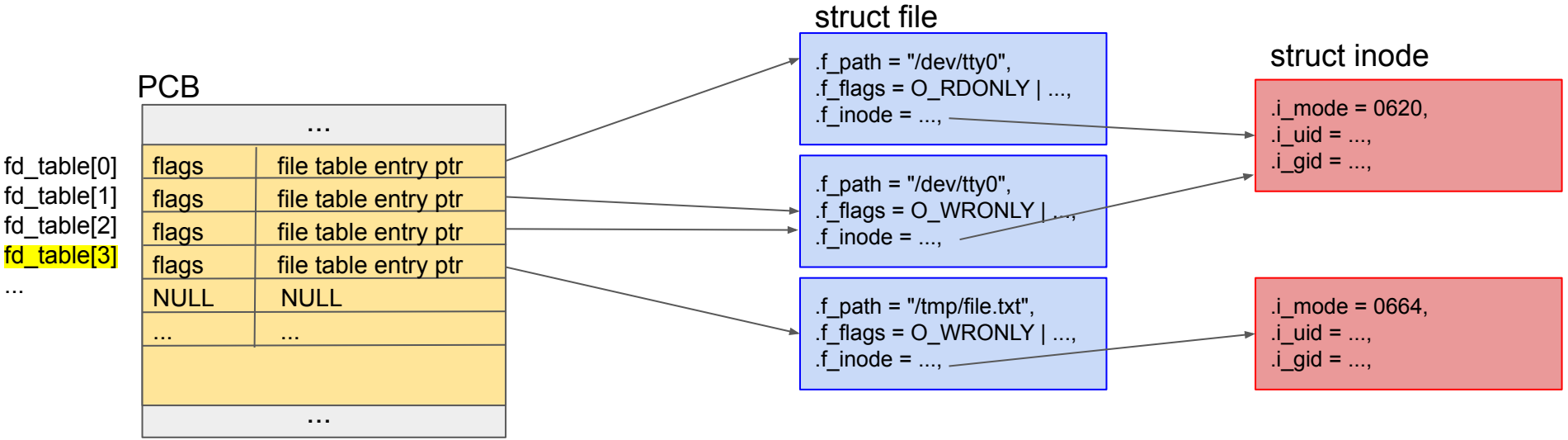
AFTER



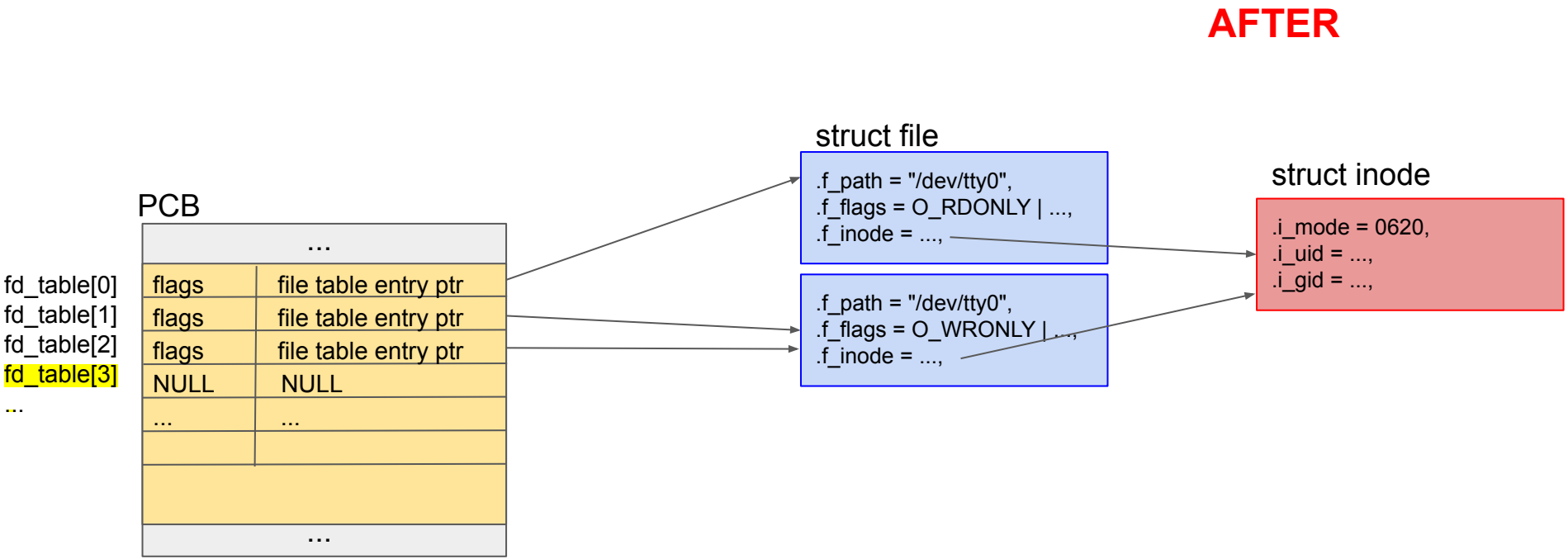
Example: `close(3)`

在project中，对所有creat的file都需要close

BEFORE



Example: `close(3)`

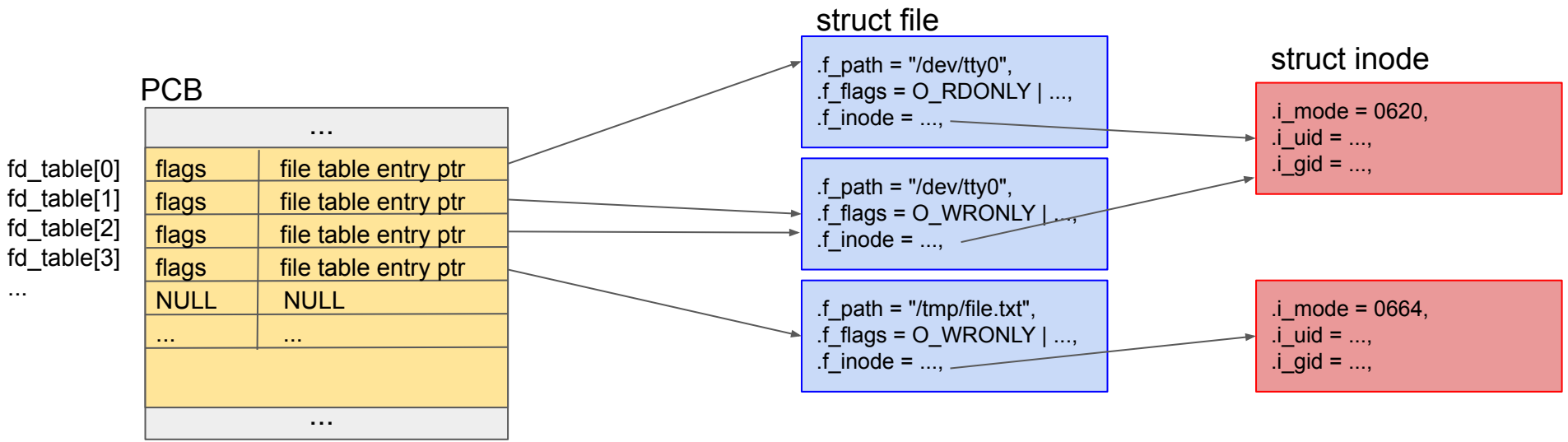


Example: dup(3)

```
text = "Hello World"
fd2 = dup (fd1)
write (fd1)
write (fd2, text, strlen () )
输出的是两行hello word

dup2 (fd1, fd2)
write (fd2, text, strlen () )
```

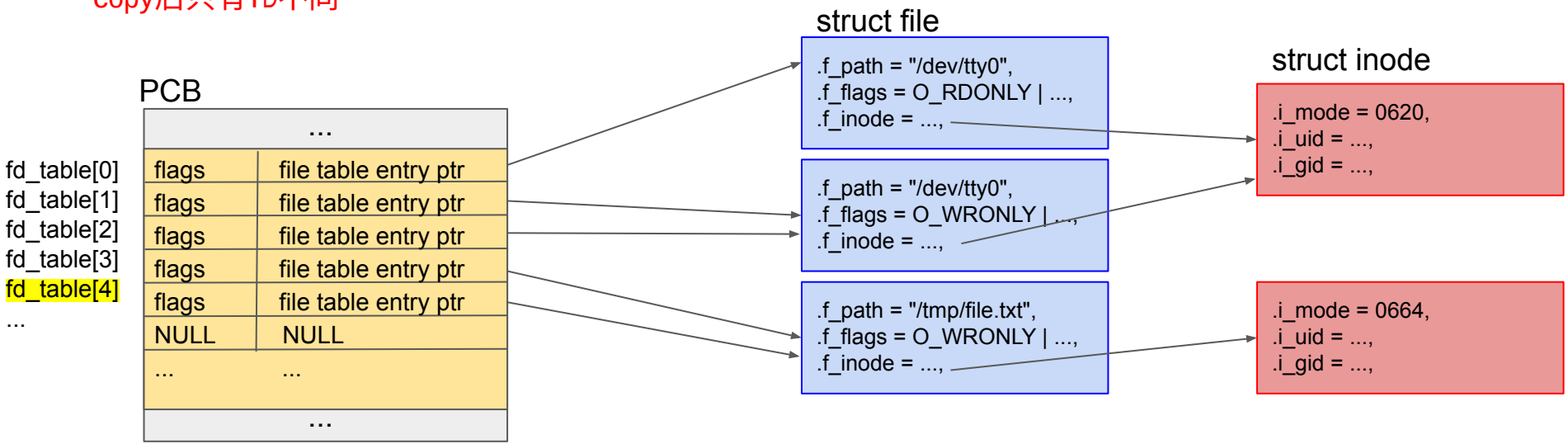
BEFORE



Example: dup(3) - returns 4

AFTER

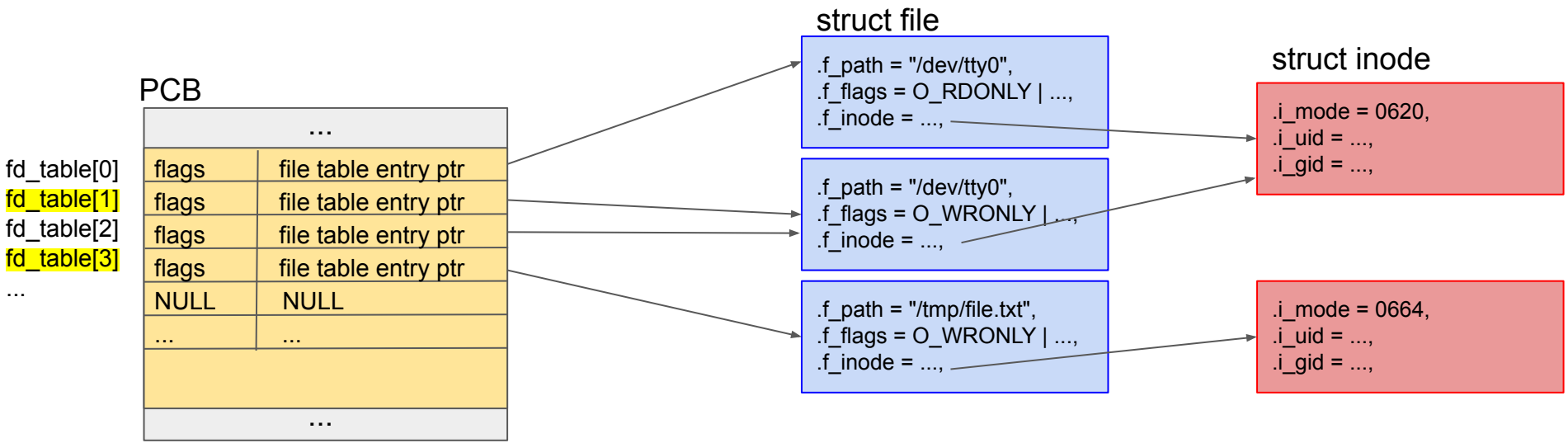
Table4是table3的copy
copy后只有ID不同



Example: dup2(3, 1)

用table 3的内容去替换table 1的内容

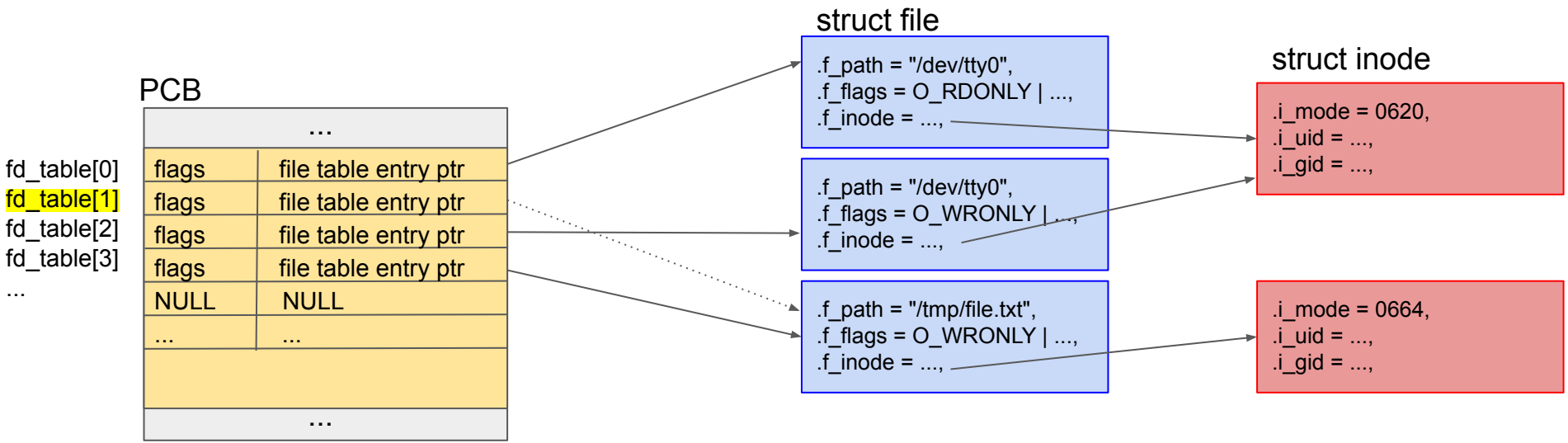
BEFORE



Example: dup2(3, 1)

table 1被替换后，内容和Table 1
完全一样，指向的是同1个

AFTER

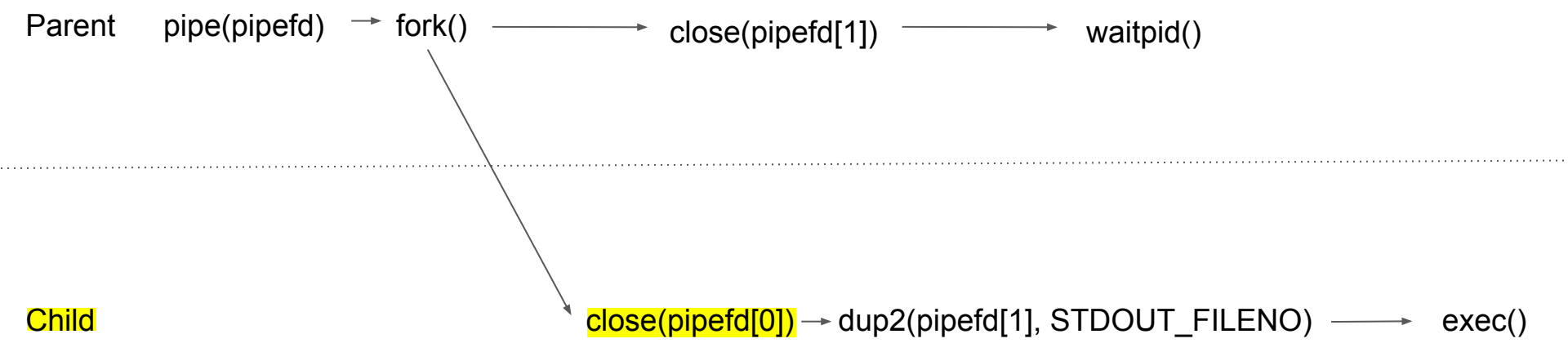


The syscalls you need for P2D2

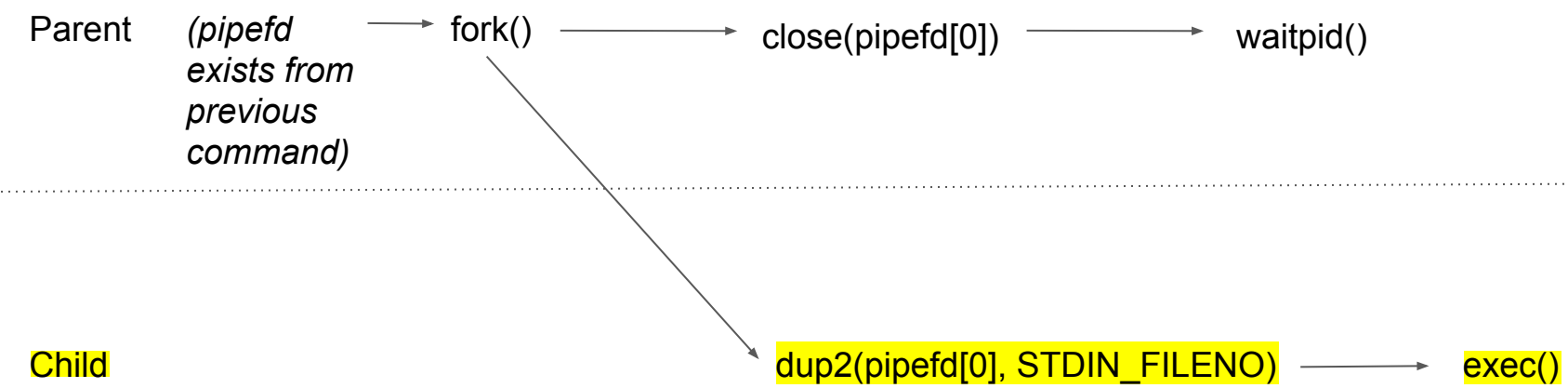
- File Redirection
  - open() - to open the input and output files
  - dup2() - to point stdin/stdout to the file descriptors you've opened
- Piping
  - pipe() - to create new pipes in the parent shell process
  - dup2() - to point stdin/stdout to the read/write end of the pipes you've made
  - close() - to close the read end of the pipe in the child process (for piping out), and to close the read/write end of the pipe in the parent process

对parent和child， colse的顺序不一样
老师上课演视的代码

Recommended flow for piping into another command



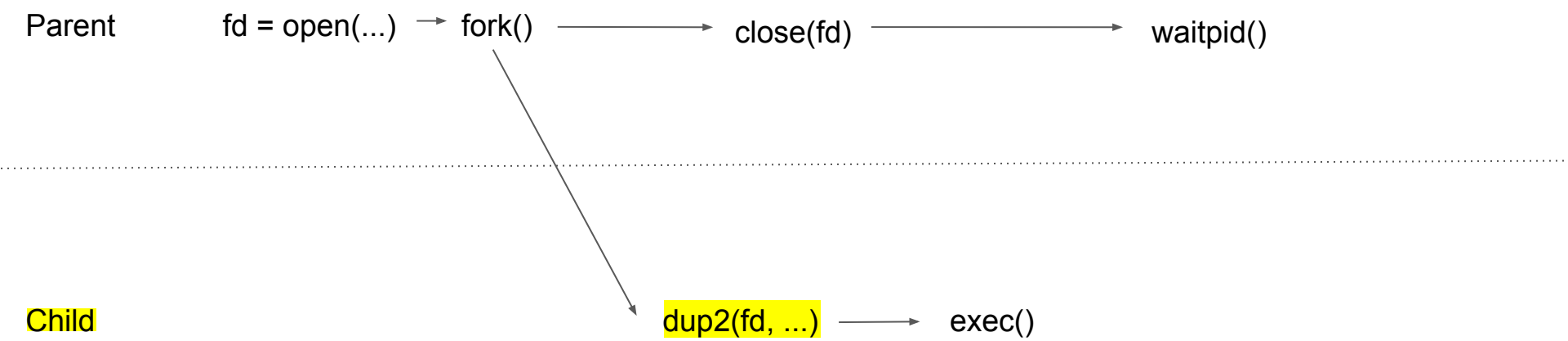
Recommended flow for receiving pipe from previous command in pipeline



Note: we don't want to close pipefd[1] at this point, as it was already closed in parent process

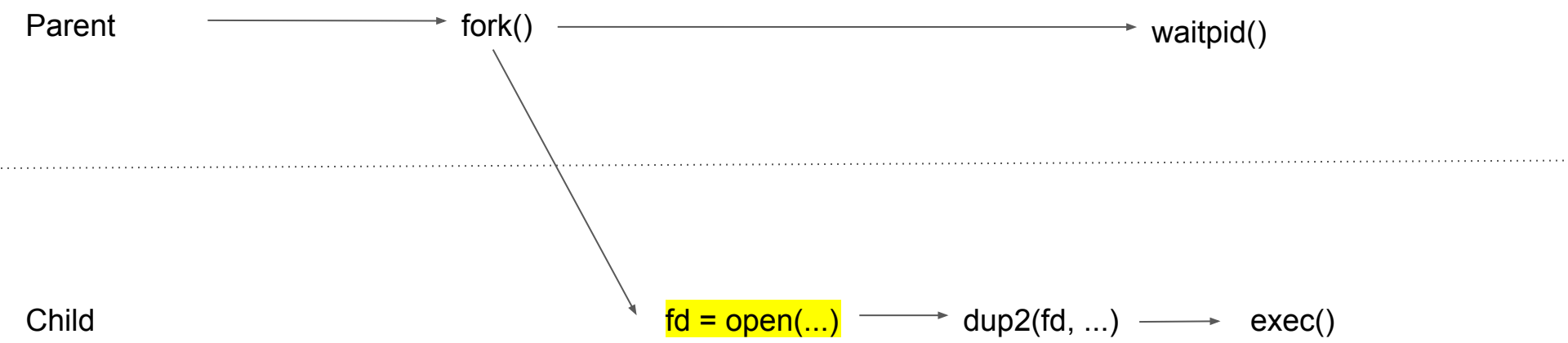
Recommended flow for file redirect

(Option A: open in parent process)



Recommended flow for file redirect

(Option B: open in child process)



Combining it all together

- Think about how to handle commands which have both a pipe input and a pipe output (Hint: we need to keep track of *two* pipefd arrays, one for input, one for output)
- Keep in mind that an input file can appear in the first command of a pipeline, and an output file on the last command of a pipeline
- For a pipeline of length 1 (single command), we can have both an input and output file, potentially
- ***Try it:*** *writing pseudocode before real C can be an effective strategy to comprehend this problem.*