Last lecture in OS :(

Files and File Descriptors

files are an abstraction, just like a process

process abstracts processors

files abstracts storage
- I open/read/write/close files
- OS: manage bit locations on hardware

lseek → indicate offset in a file (DVD chapters)

Open file

open in read/write modes

```
fd = open( file, read )
```

fd < 0 ? → error

fd is unique id of open operation

```
retval = close( fd )
```
close file descriptor

Read file

```
nbytes = read( fd, buf, 1024 )
```
read 1024 bytes, put onto buf

nbytes < 0 ? → error

read more? do read() more times
→ it will change position of file pointer

Write file

```
nbytes = write( fd, buf, 1024 )
```
write 1024 bytes from buf onto file

nbytes < 0 ? → error

downloading 1MB file w/ 1 KB buffer? call write() 1024 times

map through FD table, maintained per process

The first 3 file descriptors

Each process begins life with three open files associated with a terminal

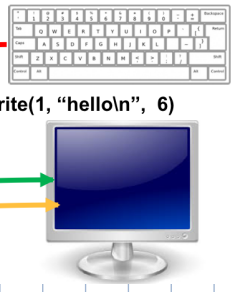
0: standard input (default: keyboard)

1: standard output (default: screen)

2: standard error (default: screen)

Open a new file

Starts from fd = 3



File Descriptors

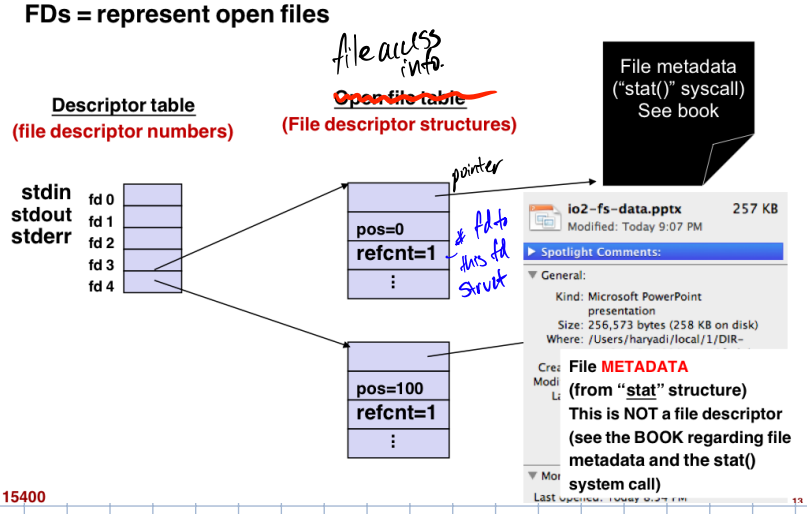
fd is just a number

to OS, it represents a pointer to file descriptor structure

gives file access, next read/write position, ...

fd is not a struct

FDs = represent open files



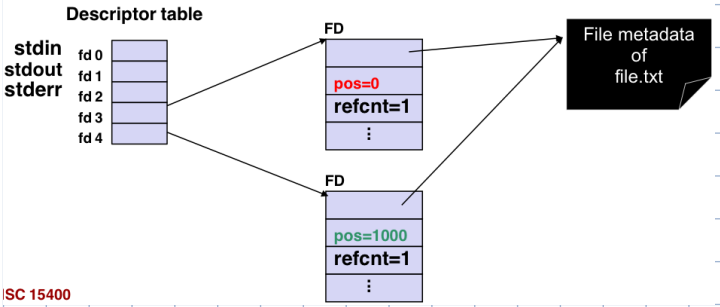
multiple fd's can point to same fd struct

→ can open same file multiple times → multiple file instances, point to same metadata

Two distinct FDs sharing the same disk file

E.g., Calling **open** twice with the same **filename** argument

```
fd3=open("file.txt"),
fd4=open("file.txt"),
read(fd4, buf, 1000)
```



Redirection

in shell: `ls > foo.txt` print ls into foo.txt

done using `dup2( fd_src, fd_dst )` change pointer of fd

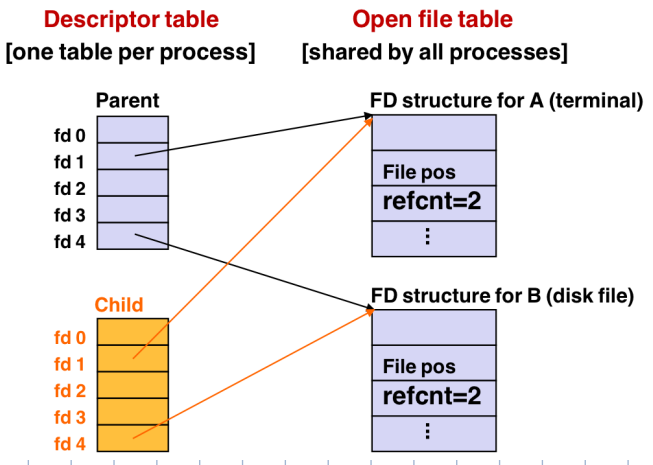
`read( )` affects fd struct.

17:00-20:00 p2 → cool animation

child inherits parents open files

After fork():

Child's table same as parent's, and +1 to each refcnt



2:30 animation

O_APPEND → start position @ end

Buffering

rio_read( ) → read much more per syscall
↳ library fn call, not syscall
much more efficient

still need to specify num bytes

don't read from disk, read from memory
prepares its own buffer
uses mem copy

new syscall only when buffer needs to be refreshed