4- file abstraction

virtual filesystem (VFS) interface = file is the abstraction used for I/O

↳ used to access disks, CDs, DVDs, USBs, serial devices, network → /dev/usb1, /dev/mouse, /dev/kbd (keyboard)

C standard I/O = bufferring between program and actual files

↳ fopen, fread, fscanf, fprintf → not system calls by themselves

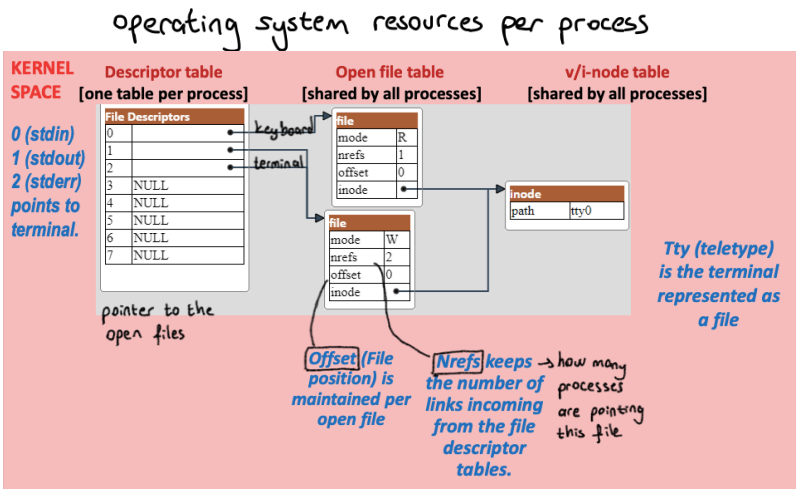
UNIX I/O = open, read, write, stat, close, lseek → changing the current file position (offset in file)

- file is a sequence of bytes → all I/O devices represented as files (even kernel)

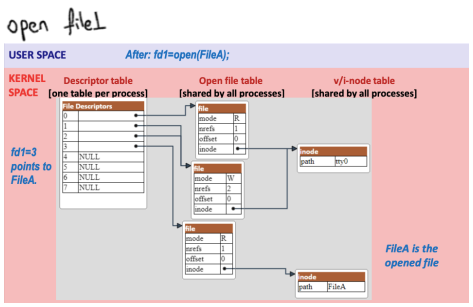
Opening files = int file_descriptor = open("/etc/hosts" (file), O_RONLY (mode: read only)) if == -1 → error

* every process has three special files already open = stdin, stdout, stderr → standard error

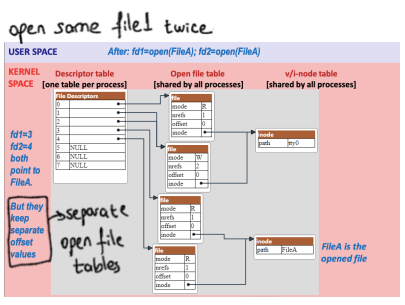
operating system resources per process



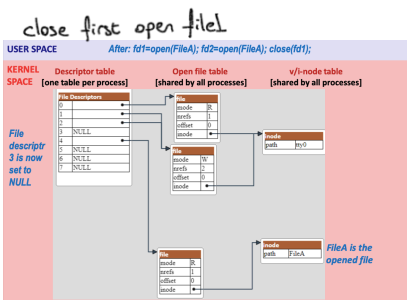
open file1



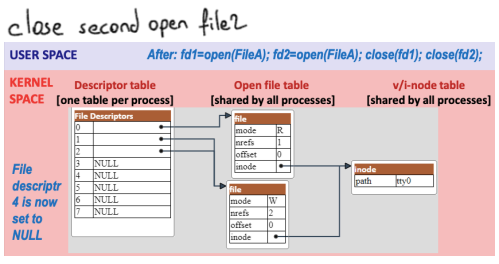
open same file1 twice

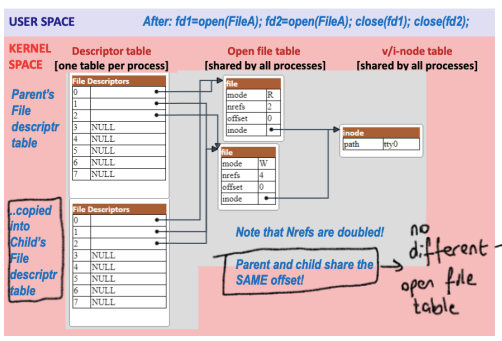


close first open file1



close second open file2





after fork():

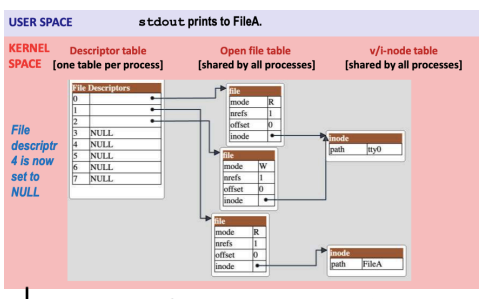
→ way to communicate between child and parent

shell redirection

> ./myprogram > somefile.txt → writes output to somefile (connects stdout to somefile.txt)

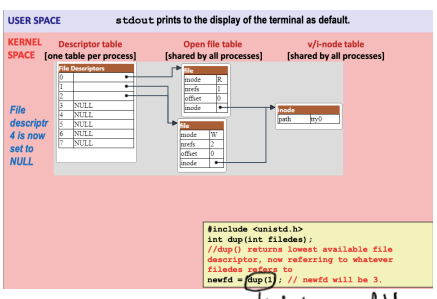
> ./myprogram < input.txt > somefile.txt → connects stdin to input.txt, stdout to somefile.txt

> ./myprogram 2> error.txt → connects stderr to error.txt



↳ connect stdout to file A, can't be done directly because it is kernel space

- system calls are needed

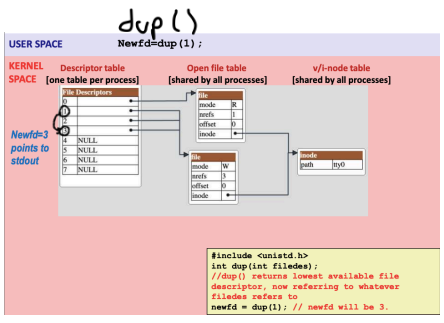


dup() system call

↳ duplicates fd 1

↳ returns new index

dup()

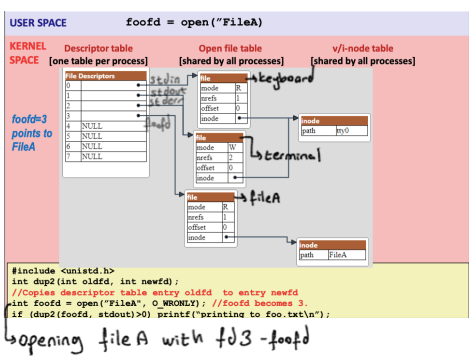


dup (fd)

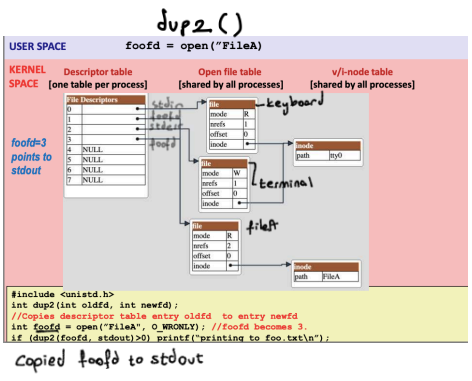
```
//Descriptor table
void *DT[maxFd];

int dup(int oldfd){
   //get the lowest available
   //file descriptor
   newfd = lowestFd(DT);
   DT(newfd)=DT(oldfd);
   return(newfd);
}
```

→ copies the pointer of file table

dup2()



↳ opening file A with fd 3 - foofd



Copied foofd to stdout

dup2(oldfd, newfd)

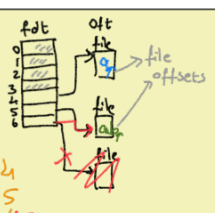
```
//Descriptor table
void *DT[maxFd];

int dup2(int oldfd, int newfd){
   DP[newfd]=DP[oldfd];
   return(newfd);
}
```

copies descriptor table entry oldfd to entry newfd

- if oldfd not valid → faid
  ↳ newfd still open

Example-1

```
#include "csapp.h"
int main(int argc, char *argv[])
{
    int fd1, fd2, fd3;
    char c1, c2, c3;
    char *fname = argv[1];
    fd1 = open(fname, O_RDONLY, 0);=4
    fd2 = open(fname, O_RDONLY, 0);=5
    fd3 = open(fname, O_RDONLY, 0);=65
    dup2(fd2, fd3);(fd2)→(fd3) → they share same file offset
    read(fd1, &c1, 1);a
    read(fd2, &c2, 1);a
    read(fd3, &c3, 1);b
    printf("c1 = %c, c2 = %c, c3 = %c\n", c1, c2, c3);
    return 0;
}
```



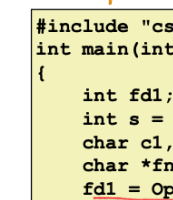
a a b

file contains "abcde"

ffiles1.c

Example-2

```
#include "csapp.h"
int main(int argc, char *argv[])
{
    int fd1;
    int s = getpid() & 0x1;
    char c1, c2;
    char *fname = argv[1];
    fd1 = Open(fname, O_RDONLY, 0);
    Read(fd1, &c1, 1);→a
    if (fork()) { /* Parent */
        sleep(s);
        Read(fd1, &c2, 1);
        printf("Parent: c1 = %c, c2 = %c\n", c1, c2);
    } else { /* Child */
        sleep(1-s);
        Read(fd1, &c2, 1);
        printf("Child: c1 = %c, c2 = %c\n", c1, c2);
    }
    return 0;
}
```



s=0
- parent a b
  child a c

s=1
- child a b
  parent a c

if s=0 parent executes first → a, b/c

if s=1 child executes first → a, c/b

file contains "abcde"

ffiles2.c

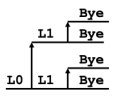
bufferring in standar I/O

buf → hello\n ... → output if "\n" or fflush() is seen, no separate system calls (print(h), print(e), print(l), ⋮, print(\n))

strace ./hello → prints a list of system calls made by the program, 1 write syscall

```
void fork2()
{
    printf("L0\n");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```

1-20



→

```
void fork2a()
{
    printf("L0");
    fork();
    printf("L1\n");
    fork();
    printf("Bye\n");
}
```

2-20

- L0 printed twice, because it is buffered without "\n" and buffer is duplicated to child