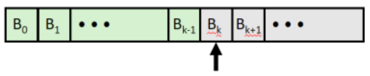
Unix I/O Overview

- Mapping of devices to files allows kernel to export simple interface called *Unix I/O:*
  - Opening and closing files
    - **open()** and **close()**
  - Reading and writing a file
    - **read()** and **write()**
  - Changing the ***current file position*** (seek)
    - indicates next offset into file to read or write
    - **lseek()**



Current file position = k

File Types

- Each file has a *type* indicating its role in the system
  - *Regular file:* Contains arbitrary data
  - *Directory:* Index for a related group of files
  - *Socket:* For communicating with a process on another machine
- We ignore the other file types (beyond our scope)
  - *Named pipes (FIFOs)*
  - *Symbolic links*
  - *Character and block devices*

Opening Files

- When you open a file →
  Informs the kernel that you are ready to access that file

```
int fd;   /* file descriptor */

if ((fd = open("/etc/hosts", O_RDONLY)) < 0) {
   perror("open");
   exit(1);
}
```

- Returns an identifying integer *file descriptor*
  - **fd == -1** indicates that an error occurred
- Each process created by a Linux shell begins life with three open files associated with a terminal:
  - 0: standard input (stdin)
  - 1: standard output (stdout)
  - 2: standard error (stderr)

Closing Files

- When you closing a file →
  informs the kernel that you have finished accessing that file

```
int fd;     /* file descriptor */
int retval; /* return value */

if ((retval = close(fd)) < 0) {
   perror("close");
   exit(1);
}
```

Reading Files

- Reading a file → copies bytes from the current file position to memory, and then updates file position

```
char buf[512];
int fd;       /* file descriptor */
int nbytes;   /* number of bytes read */

/* Open file fd ...  */
/* Then read up to 512 bytes from file fd */
if ((nbytes = read(fd, buf, sizeof(buf))) < 0) {
   perror("read");
   exit(1);
}
```

- Returns number of bytes read from file `fd` into `buf`
  - Return type **ssize_t** is signed integer
  - **nbytes < 0** indicates that an error occurred

Writing Files

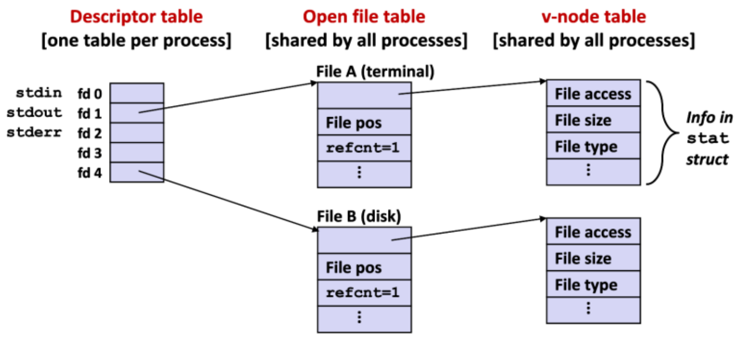
- Writing a file → copies bytes from memory to the current file position, and then updates current file position

```
char buf[512];
int fd;       /* file descriptor */
int nbytes;   /* number of bytes read */

/* Open the file fd ... */
/* Then write up to 512 bytes from buf to file fd */
if ((nbytes = write(fd, buf, sizeof(buf)) < 0) {
   perror("write");
   exit(1);
}
```

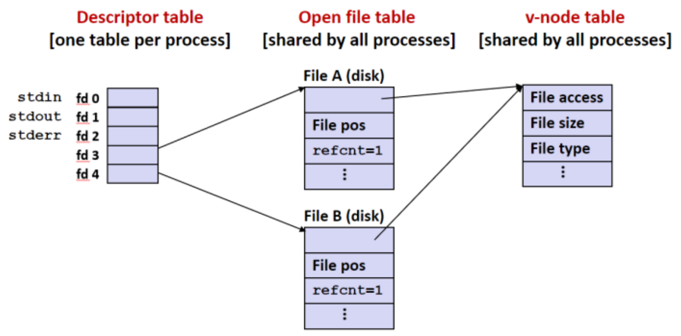
- Returns number of bytes written from `buf` to file `fd`
  - **nbytes < 0** indicates that an error occurred

- Two descriptors referencing two distinct open files
  - Descriptor 1 (stdout) points to terminal
  - Descriptor 4 points to open disk file



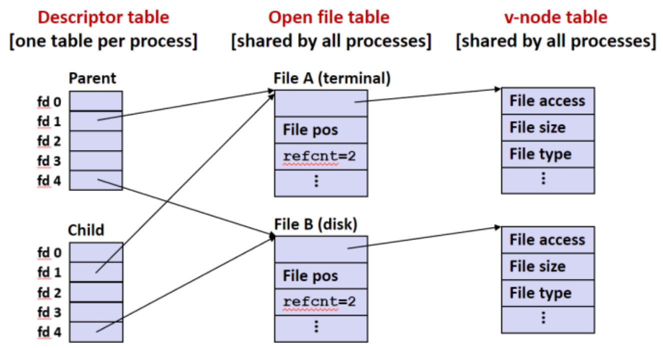
File Sharing

- Two distinct descriptors sharing the same disk file through two distinct open file table entries
  - E.g., Calling `open` twice with the same `filename` argument



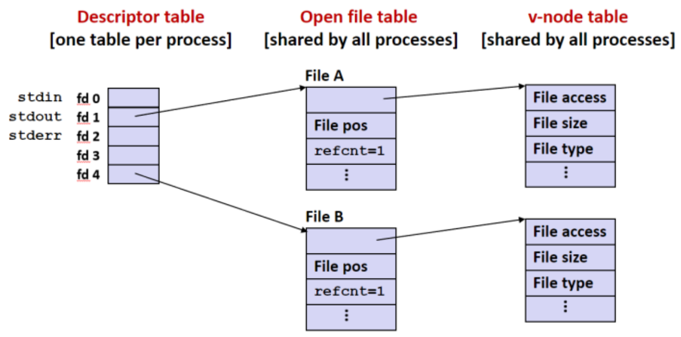
How Processes Share Files: `fork`

- A child process inherits its parent's open files
- *After* `fork`:
  - Child's table same as parent's, and +1 to each refcnt



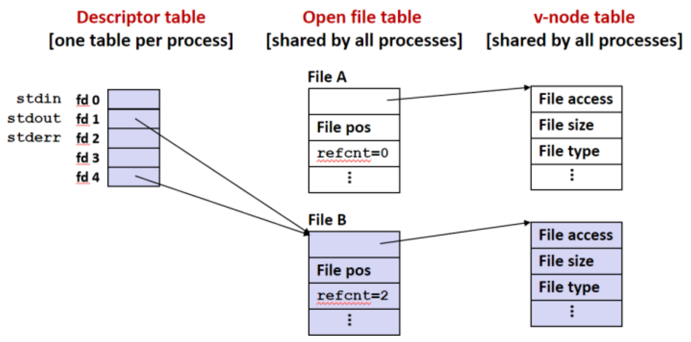
I/O Redirection Example

- Step #1: open file to which stdout should be redirected
  - Happens in child executing shell code, before `exec`



I/O Redirection Example (cont.)

- Step #2: call `dup2(4,1)`
  - cause fd=1 (stdout) to refer to disk file pointed at by fd=4



Descriptor table [one table per process]
Open file table [shared by all processes]
v-node table [shared by all processes]
stdin fd 0
stdout fd 1
stderr fd 2
fd 3
fd 4
File A
File pos
refcnt=0
File B
File pos
refcnt=2
File access
File size
File type
File access
File size
File type