System-Level I/O

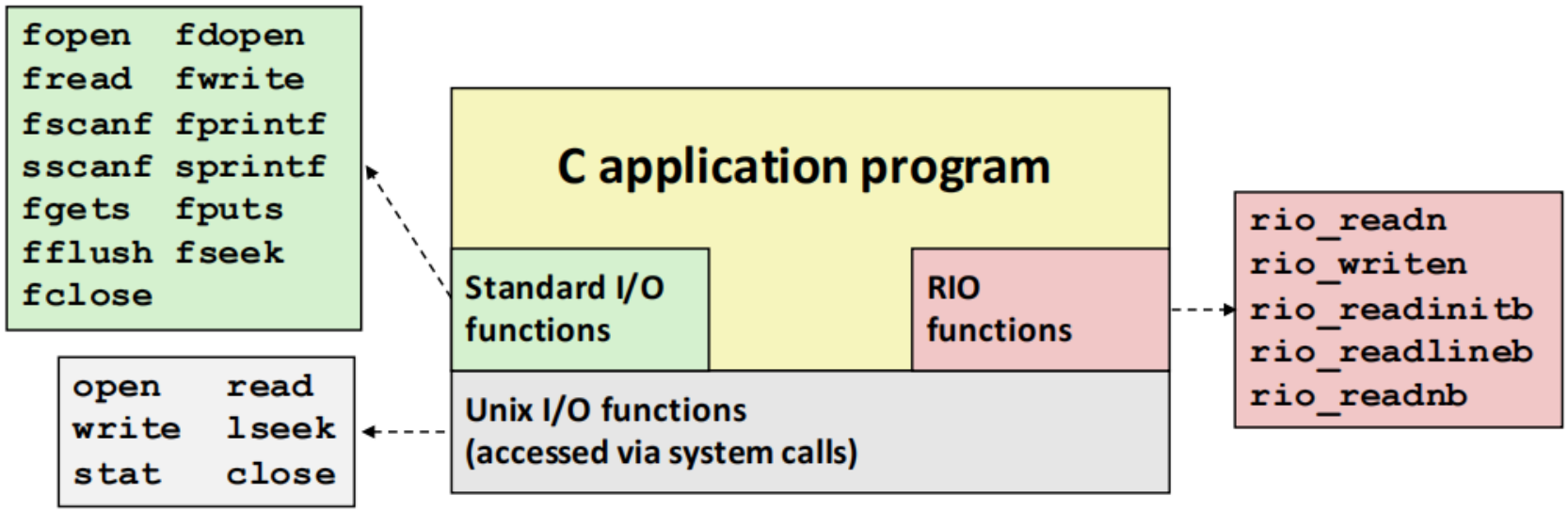
石元峰

2019年11月21日

Outline

不同的I/O库

- Unix I/O: Linux系统内核提供的系统级I/O函数,通过这些函数来实现较高级别I/O函数。
- Standard I/O: C语言定义的一组高级I/O函数，被称为标准I/O库
- Robust I/O(RIO): 弥补标准I/O库的漏洞，如不足值的健壮I/O包



文件

- 文件的抽象——一个Linux文件就是一个m字节的序列：$B_0, B_1, \cdots, B_k, \cdots, B_{m-1}$，而所有的I/O设备（不管是网络、磁盘还是终端,甚至内核）都会被模型化为文件。
- 每个Linux文件都有一个类型(type)来表明它在系统中的角色：普通文件、目录、套接字。
- 对于应用性普通文件，它又可被分为文本文件(只含有ASCII或Unicode字符)和二进制文件(所有其他文件)。但对内核而言两者没有区别。
- 目录：含一组链接，一个链接将一个文件名映射到一个文件。
- 套接字：用来和另一个进程进行跨进程网络通信的文件。

文件

- 内核打开一个文件便会返回一个小的非负整数，叫做描述符。而内核创建的每个进程开始时都有三个打开的文件：stdin(0)、stdout(1)和stderr(2)。
- 对于文本文件而言，每一个文本行的结束符，Linux和Mac OS是\n(0xa)，和ASCII换行符(LF)是一样的。
  而Windows和网络协议是\r\n(0xd 0xa)
- 任何一个目录均至少含有两个条目：.为到该目录自身链接，..为到其父目录的链接
- Linux命令：mkdir，ls，rmdir
- Linux内核将所有文件都组织成一个目录层次结构，/为根目录。路径名有绝对路径名(如/home/droh/hello.c)和相对路径名(如./hello.c)。

Unix I/O 函数

- int open(char *filename, int flags, mode_t mode)
- 若出错则返回-1，否则返回新文件描述符。
- flags参数决定如何访问文件：O_RDONLY(只读)，O_WRONLY(只写)，O_RDWR(可读可写)。它也可以为写提供一些提示：O_CREAT，O_TRUNC，O_APPEND
- mode参数指定的是新文件的访问权限位
- int close(int fd) fd为文件描述符，调用close函数来关闭该打开的文件。若关闭成功则返回0，否则返回-1

Unix I/O 函数

- ssize_t read(int fd, void *buf, size_t n)
- 成功则返回实际读的字节数，EOF返回0，若出错则返回-1
- ssize_t write(int fd, const void *buf, size_t n)
- 成功则返回实际写的字节数，若出错则返回-1
- ssize_t为有符号数(long)，size_t为无符号数(unsigned long)
- lseek( )函数显式地修改当前文件的位置

stat函数

- 通过调用stat函数和非Unix I/O的fstat函数，可以检索到文件的元数据
- stat函数以一个文件名为输入，但fstat函数以文件描述符为输入
- stat结构中st_size成员包含了文件的字节数大小
- st_mode成员：S_ISREG，S_ISDIR，S_ISSOCK

stat结构

```
/* Metadata returned by the stat and fstat functions */
struct stat {
    dev_t         st_dev;      /* Device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* Protection and file type */
    nlink_t       st_nlink;    /* Number of hard links */
    uid_t         st_uid;      /* User ID of owner */
    gid_t         st_gid;      /* Group ID of owner */
    dev_t         st_rdev;     /* Device type (if inode device) */
    off_t         st_size;     /* Total size, in bytes */
    unsigned long st_blksize;  /* Blocksize for filesystem I/O */
    unsigned long st_blocks;   /* Number of blocks allocated */
    time_t        st_atime;    /* Time of last access */
    time_t        st_mtime;    /* Time of last modification */
    time_t        st_ctime;    /* Time of last change */
};
```

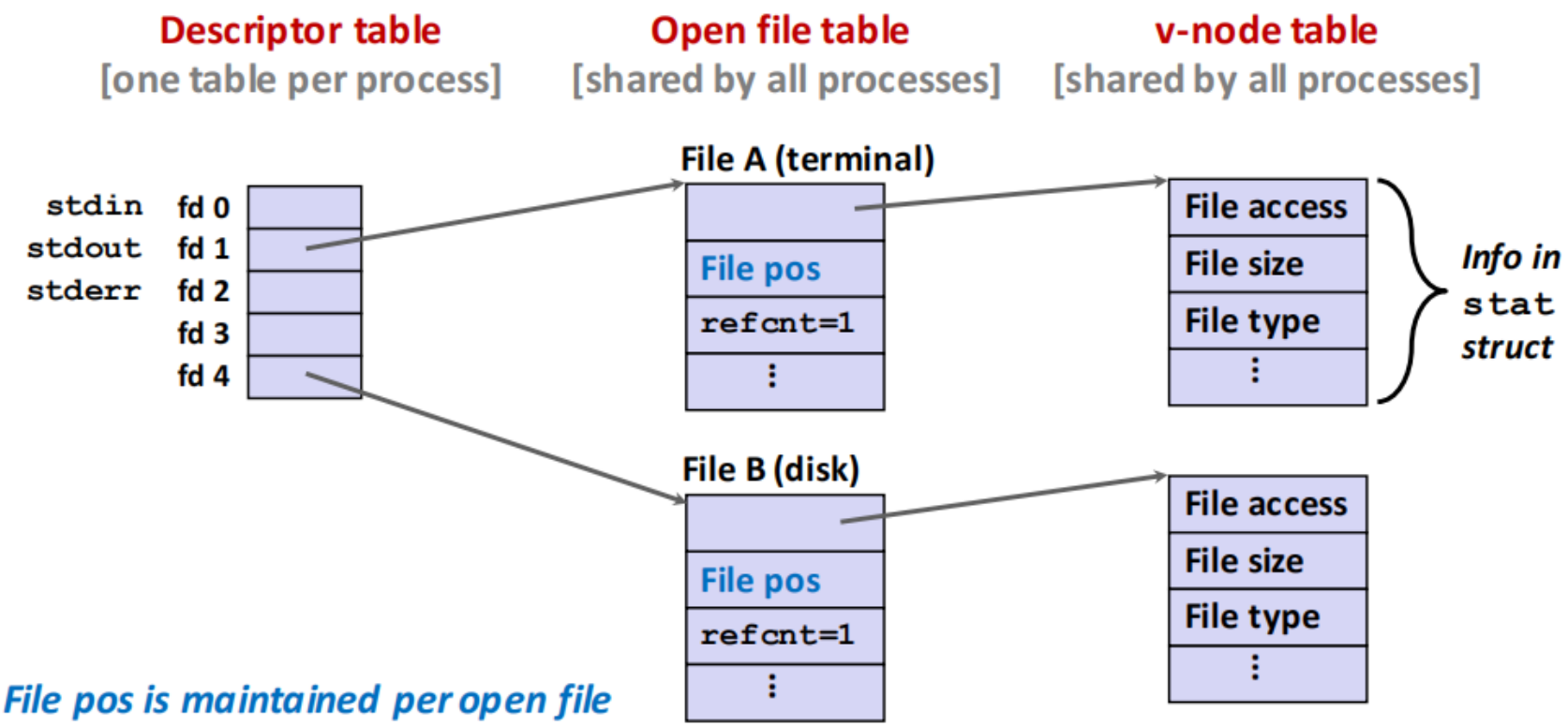
Unix I/O的不足值问题

- 在某些情况下，read和write传送的字节比应用程序要求的要少。但不足值并不是错误
  - 读时遇到EOF
  - 从终端(terminal)读文本行
  - 读和写网络套接字(socket)
- 但对磁盘文件进行读和写时，将不会遇到不足值。

打开文件的表示

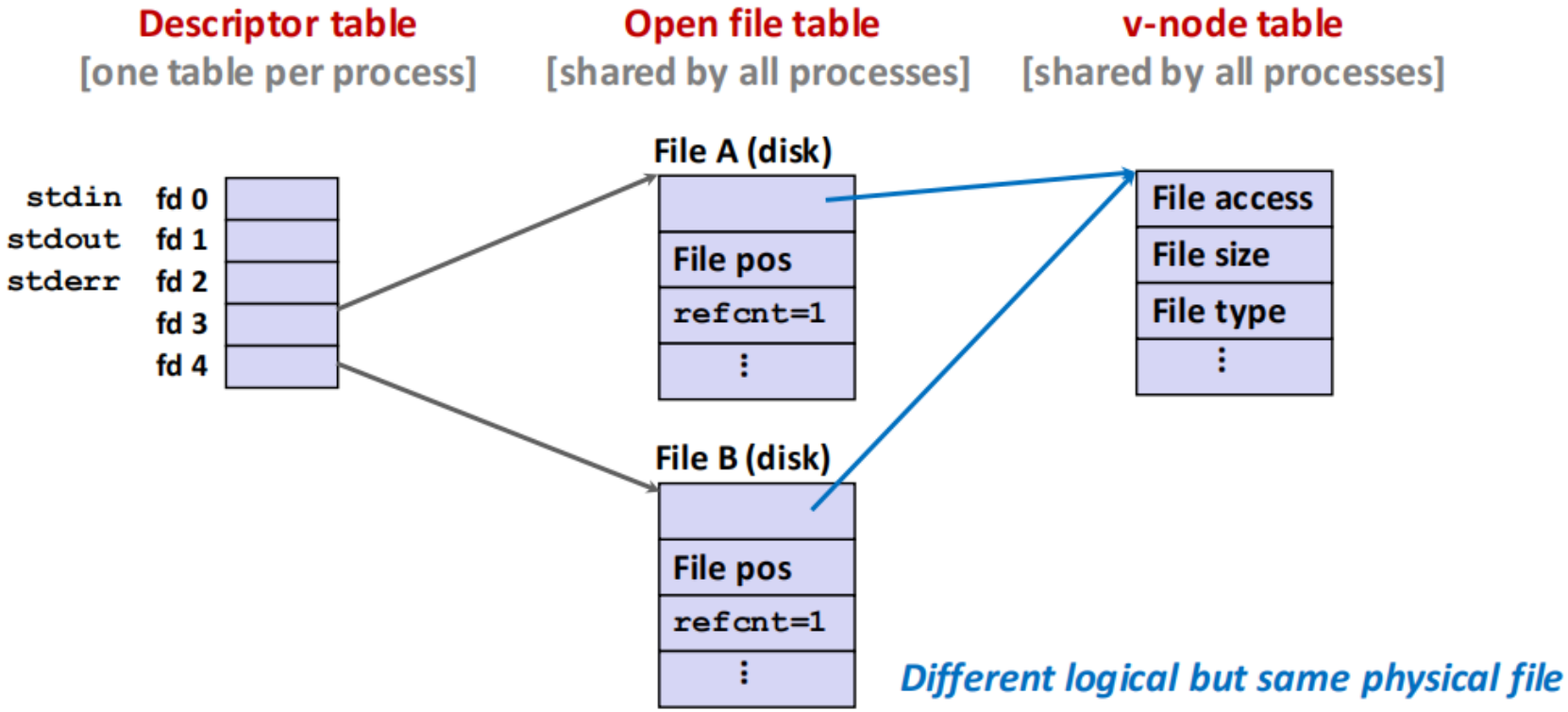
- 内核用三个相关的数据结构来表示打开的文件
  - 描述符表：每个进程都有独立的描述符表，其表项由进程打开的文件描述符来索引，每个打开的描述符表项指向文件表中一个表项
  - 文件表：所有的进程共享这张表。每个表项由当前文件位置、引用计数(当前指向该表项的描述符表项数)，以及一个指向v-node表中对应表项的指针
  - v-node表：所有进程共享这张表，每个表项包含stat结构中大多数信息

一般情况


Descriptor table
[one table per process]
Open file table
[shared by all processes]
v-node table
[shared by all processes]
stdin
stdout
stderr
fd 0
fd 1
fd 2
fd 3
fd 4
File A (terminal)
File pos
refcnt=1
File access
File size
File type
Info in
stat
struct
File B (disk)
File pos
refcnt=1
File access
File size
File type
File pos is maintained per open file

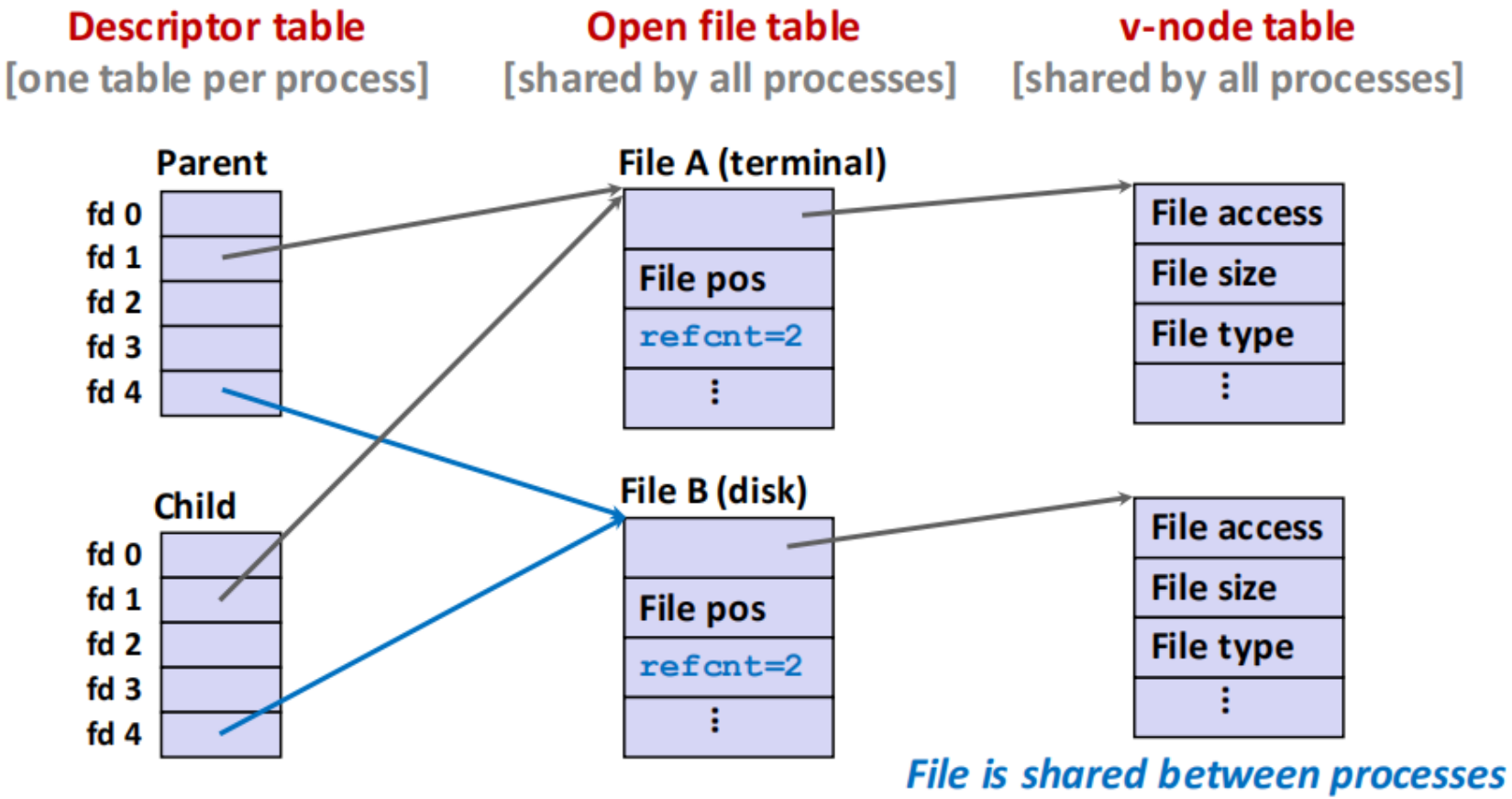
共享文件

若对同一个filename调用open函数两次，由于每个描述符都有它自己的文件位置，故会出现



共享文件

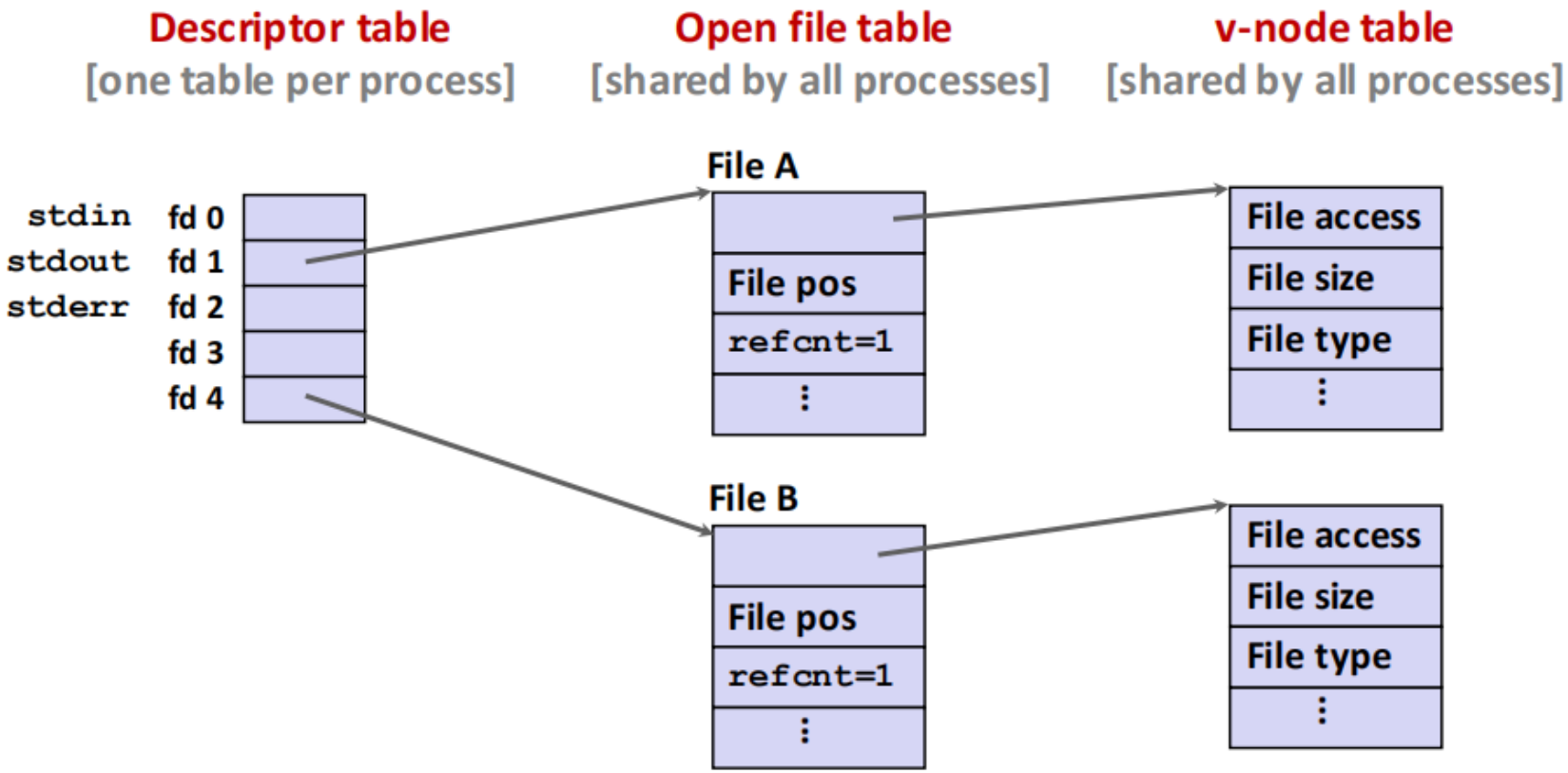
由于每个进程都有其独立的描述符表，在调用fork之后



且父子进程共享相同的文件位置

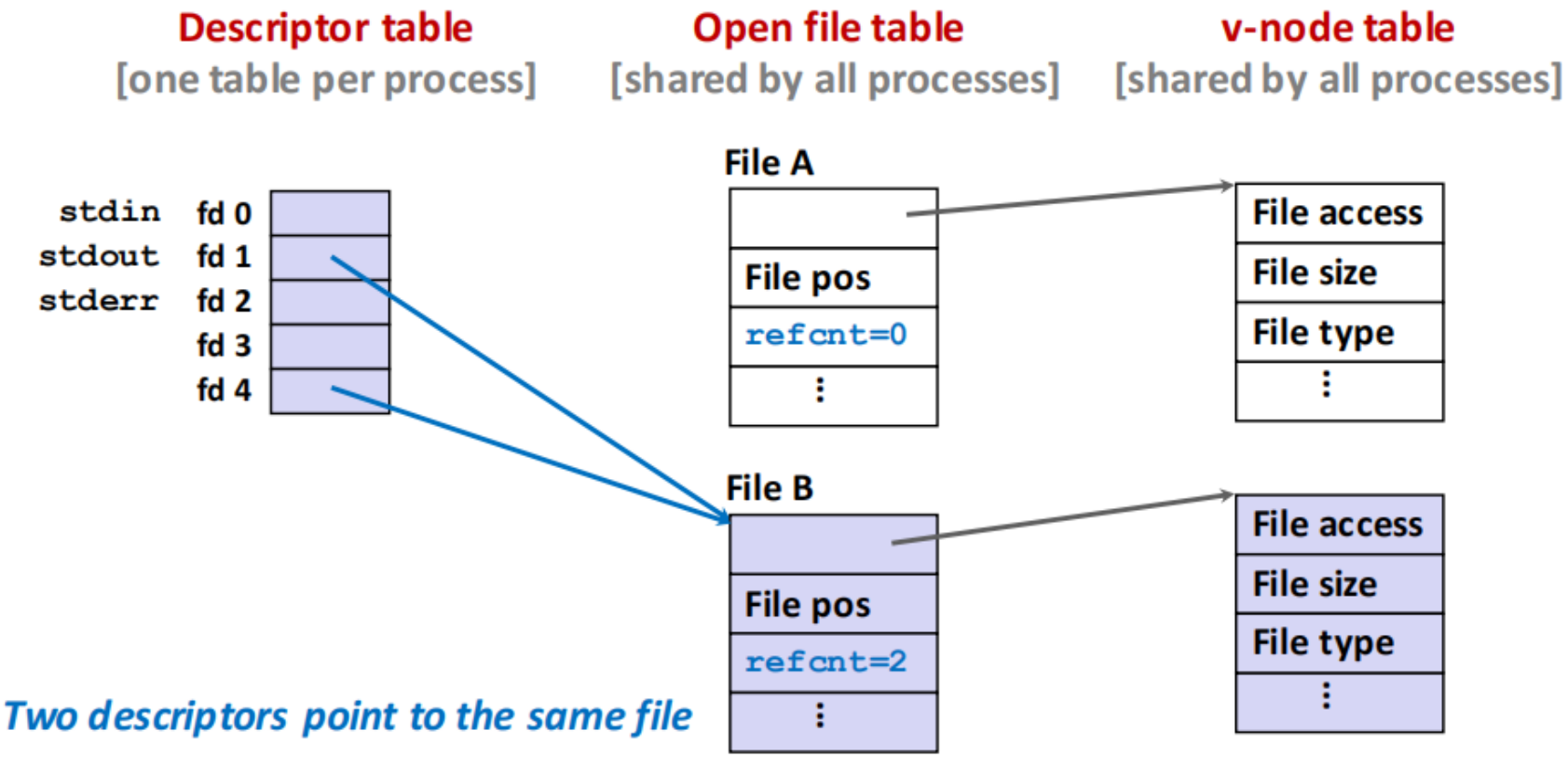
I/O重定向

- int dup2(int oldfd, int newfd);出错仍返回-1
- 复制描述表表项oldfd到newfd，且若newfd已经打开，dup2会先关闭newfd再复制



共享文件

调用dup2(1,4)后



标准I/O流

- 标准I/O库将一个打开的文件模型化为一个流(指向FILE类型的指针)。每个ANSI C程序开始时都有三个打开的流：stdin，stdout和stderr
- 流缓冲区：通过调用一次read函数填充流缓冲区，只要缓冲区中还有未读的字节，就会从缓冲区中读入
- 这种机制使得开销较高的Unix I/O系统调用的数量尽可能的小
- 然而对网络套接字而言，标准I/O会出现问题

RIO

- 本书实现的I/O包，可自动处理不足值问题，适用于网络程序(容易出现不足值)
- 两类函数：无缓冲输入输出函数(rio_readn,rio_writen)，带缓冲输入函数(rio_readlineb,rio_readnb)
- rio_readn出现不足值当且仅当EOF，而rio_writen不会出现不足值
- rio_readlineb从缓冲区读一行，rio_readnb为rio_readn带缓冲区的版本

不同I/O包的比较

- 只要有可能使用标准I/O就使用(对于磁盘和终端设备，标准I/O是首选)，但要检索文件的元数据需要用Unix I/O的stat函数。
- 标准I/O不是异步信号安全的，此时需要使用Unix I/O
- 对于网络程序，使用RIO包，避免使用标准I/O
- 不能使用scanf或rio_readlineb来读二进制文件，它们是专门用以读取文本文件，可用函数rio_readn或是rio_readnb代替