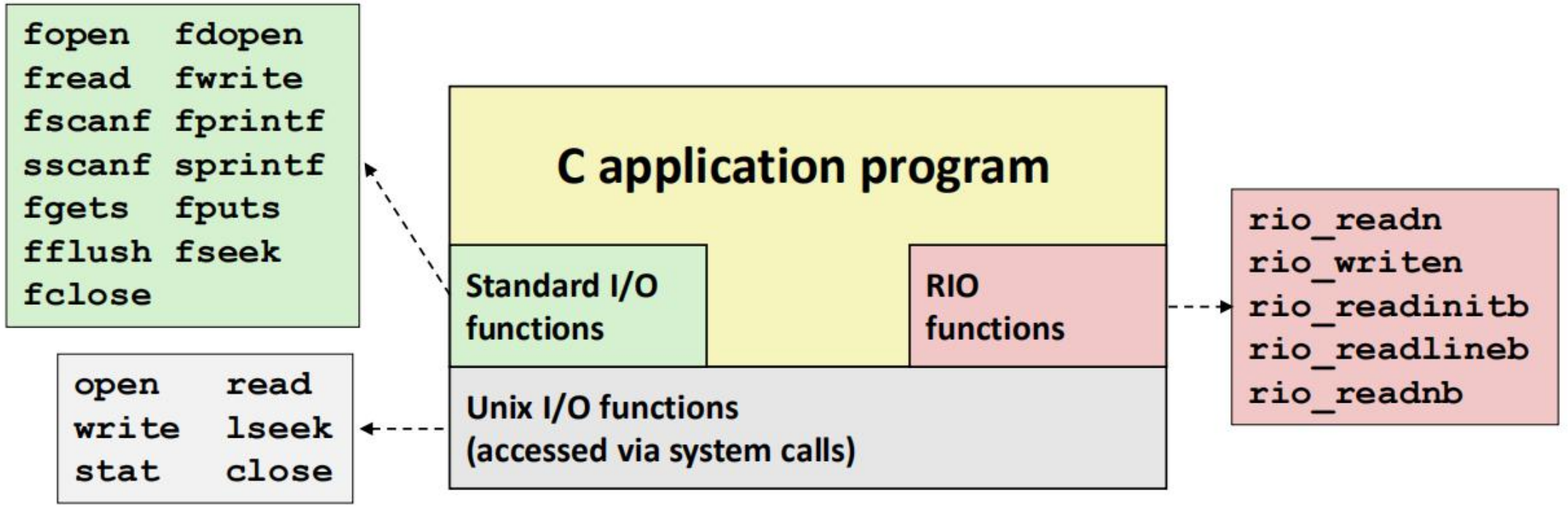
# System-Level I/O

12.3 彭亦男

# 目录

CONTENTS

# I/O


fopen fdopen
fread fwrite
fscanf fprintf
sscanf sprintf
fgets fputs
fflush fseek
fclose
C application program
Standard I/O functions
RIO functions
Unix I/O functions (accessed via system calls)
open read
write lseek
stat close
rio_readn
rio_writen
rio_readinitb
rio_readlineb
rio_readnb

- C语言
- 系统调用

# 文件和Unix I/O

- linux文件：m字节序列

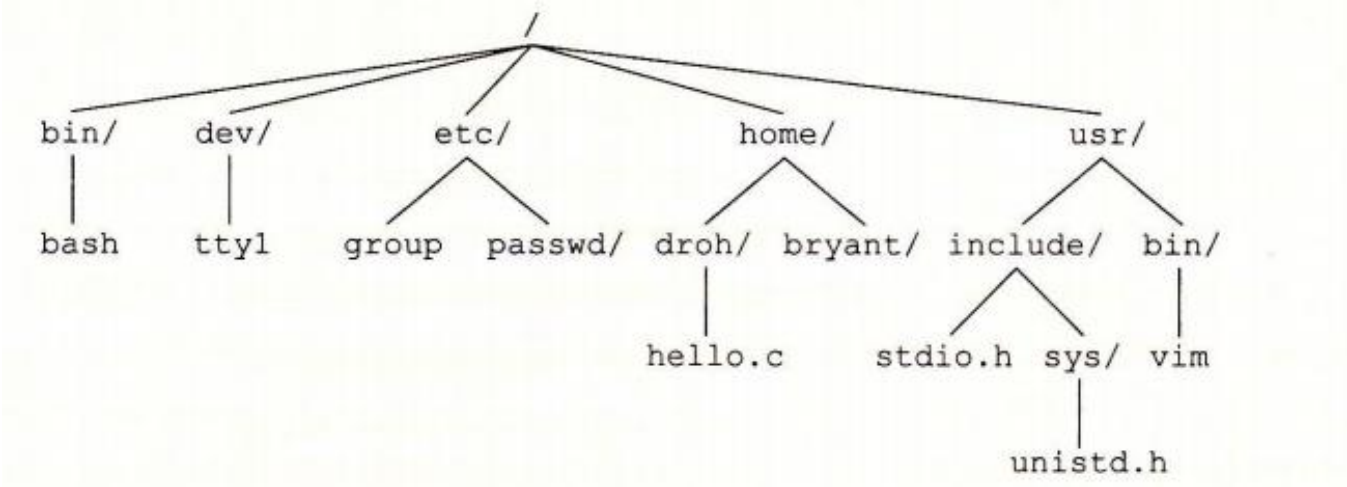
- 万物皆文件（I/O设备，内核，目录……）

- 输入输出统一为一套Unix I/O：
  - 打开open()
  - 读写文件read() write()
  - 改变当前文件位置lseek()
  - 关闭文件close()

# 文件

- 文件类型
  - 普通文件
    - 文本文件
    - 二进制文件
    - EOF
  - 目录文件
    - 一组从文件名到文件的链接
  - 套接字（socket）

- 目录层次结构
  - 绝对路径
  - 相对路径



# 元数据

- 文件的信息 stat() fstat()

```
/* 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 */
};
```

# 内核表示文件

- 描述符表--文件表--v-node表

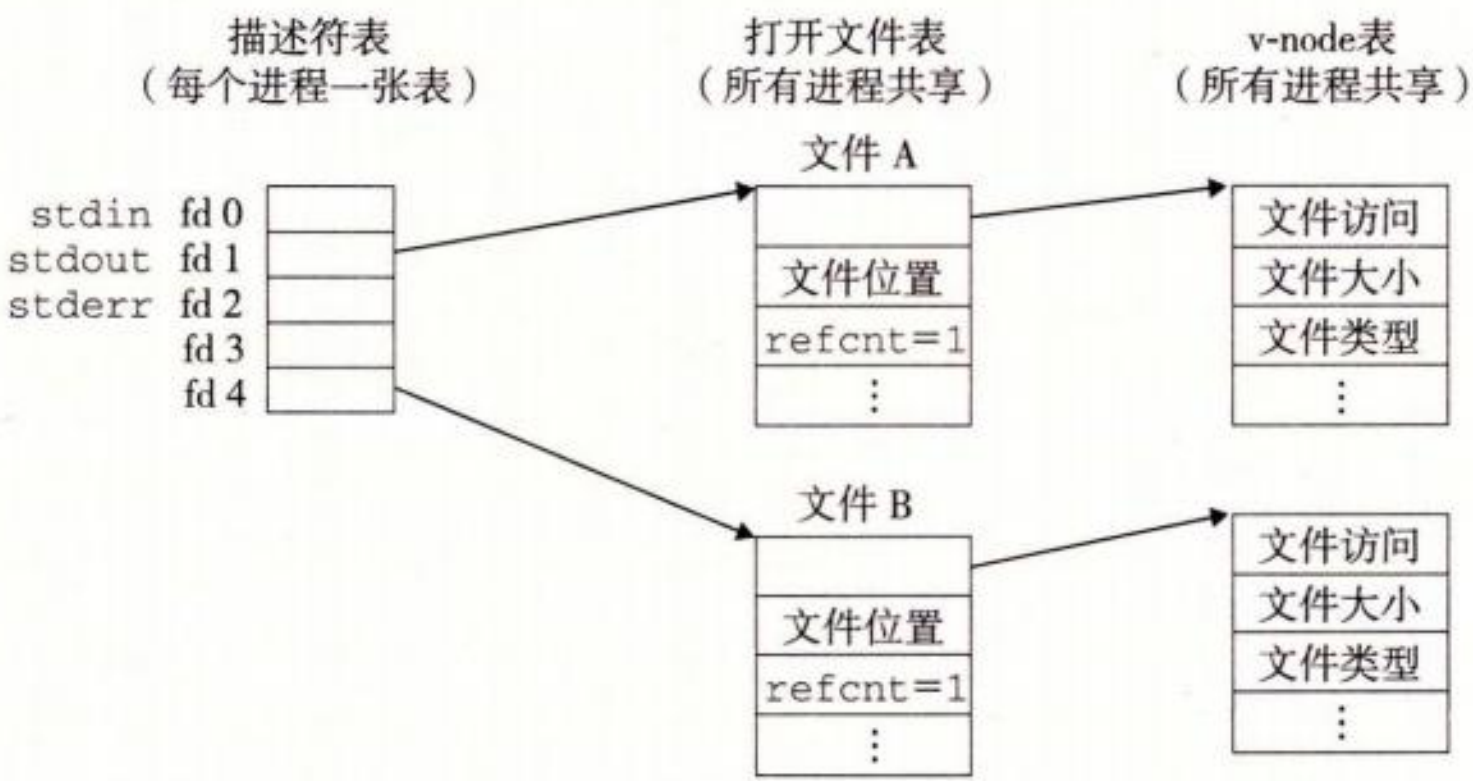



图 10-12　典型的打开文件的内核数据结构。在这个示例中，两个描述符引用不同的文件。没有共享

# 共享文件

- 一个进程打开用同一个文件两次

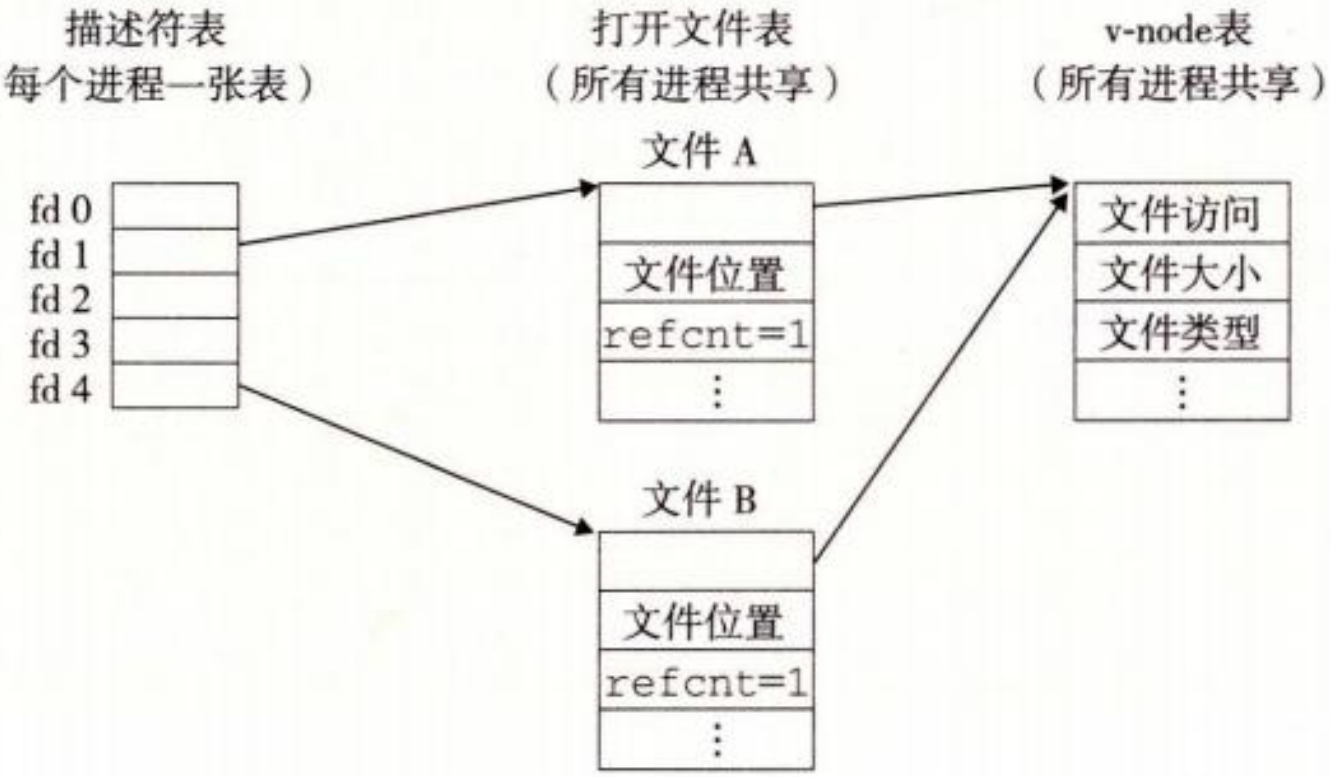
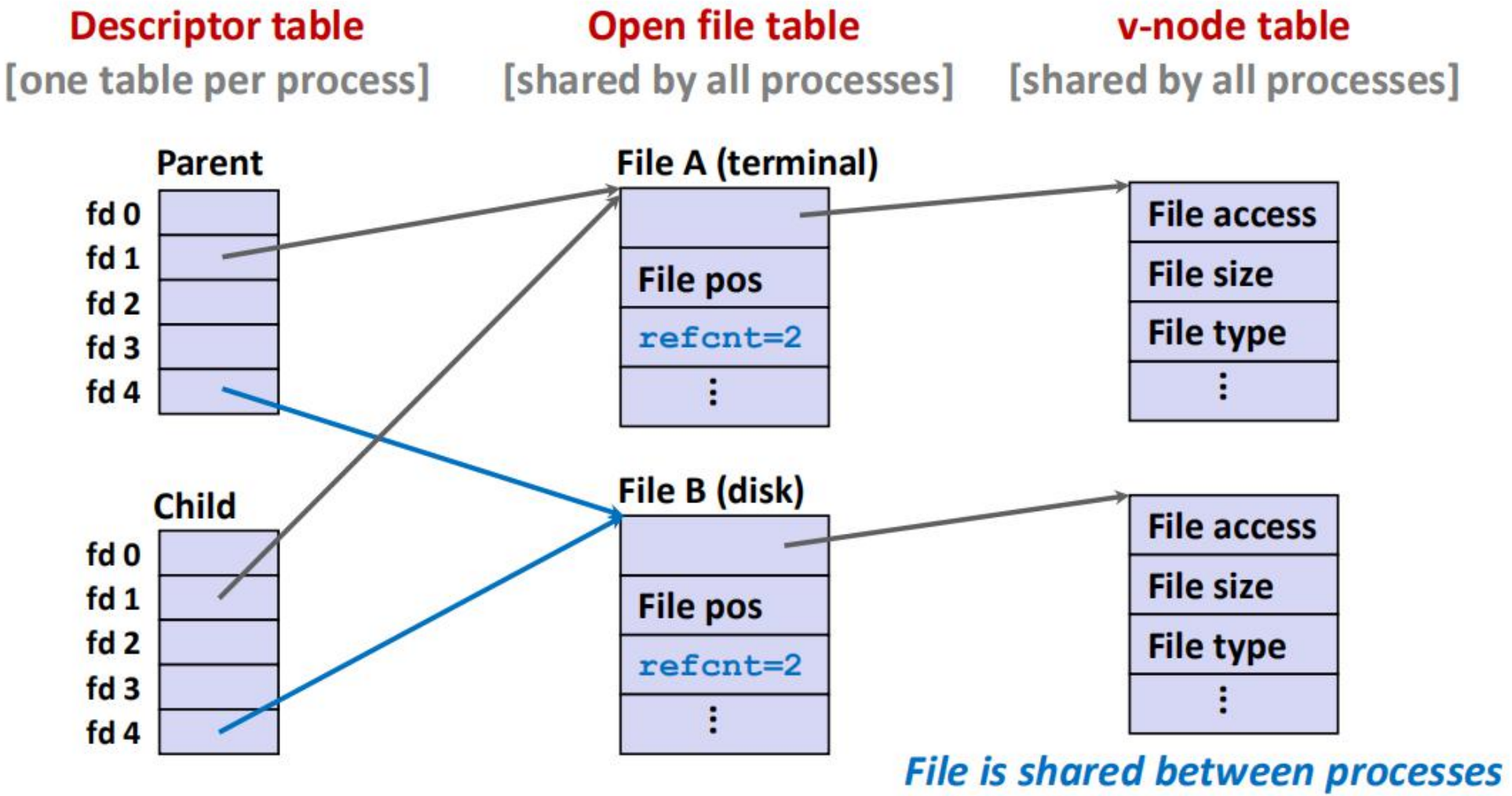



图 10-13　文件共享。这个例子展示了两个描述符通过两个打开文件表表项共享同一个磁盘文件

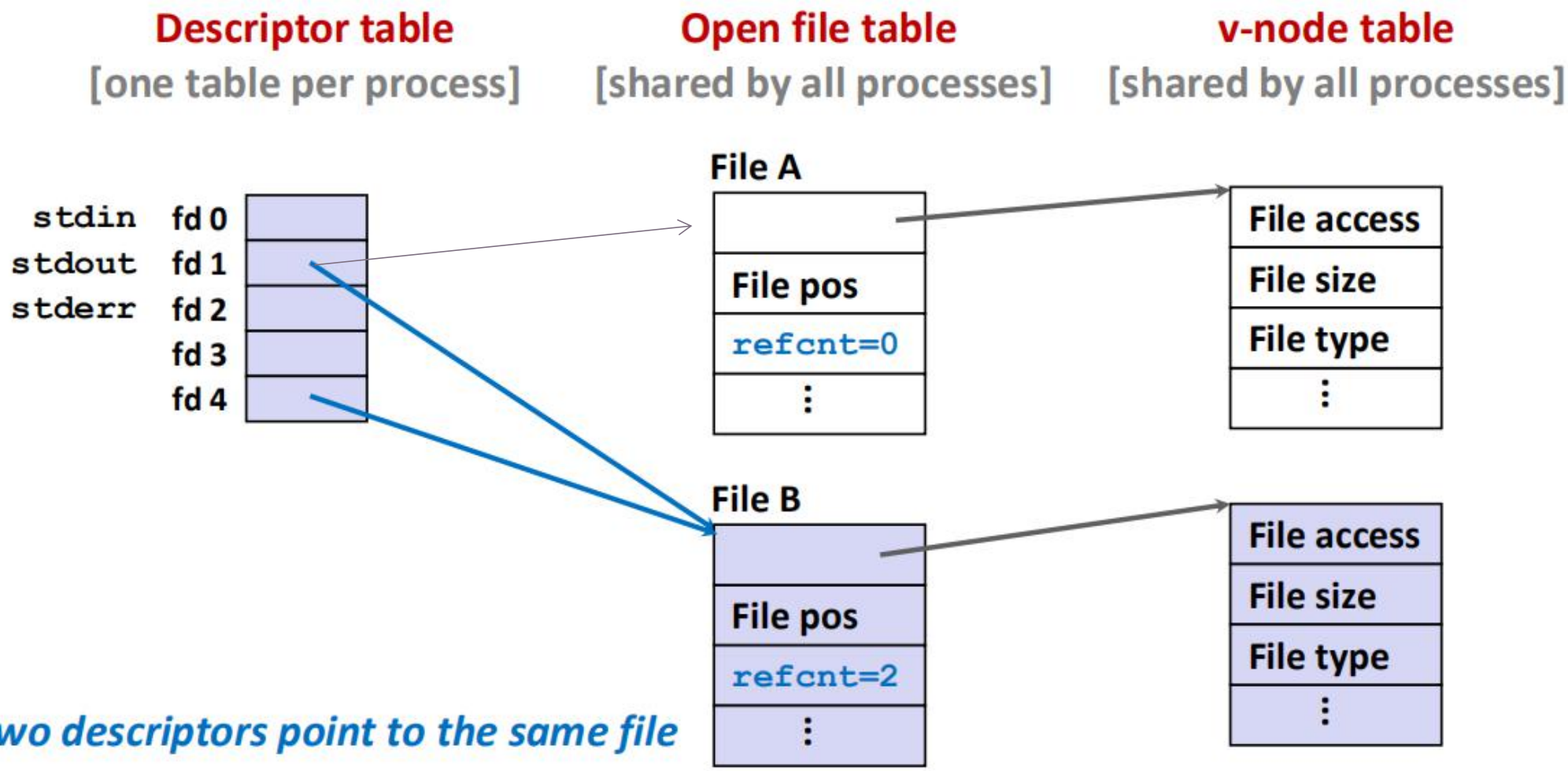
# 共享文件

- Fork()


Descriptor table
[one table per process]
Open file table
[shared by all processes]
v-node table
[shared by all processes]
Parent
fd 0
fd 1
fd 2
fd 3
fd 4
Child
fd 0
fd 1
fd 2
fd 3
fd 4
File A (terminal)
File pos
refcnt=2
File B (disk)
File pos
refcnt=2
File access
File size
File type
File access
File size
File type
File is shared between processes

# I/O重定向

- calling the dup2(oldfd, newfd) function　eg.dup2(4,1)



# Standard I/O

- streams
  - Opening and closing files (**fopen** and **fclose**)
  - Reading and writing bytes (**fread** and **fwrite**)
  - Reading and writing text lines (**fgets** and **fputs**)
  - Formatted reading and writing (**fscanf** and **fprintf**)
- 进程开始就打开的三个

```
#include <stdio.h>
extern FILE *stdin;  /* standard input  (descriptor 0) */
extern FILE *stdout; /* standard output (descriptor 1) */
extern FILE *stderr; /* standard error  (descriptor 2) */
```

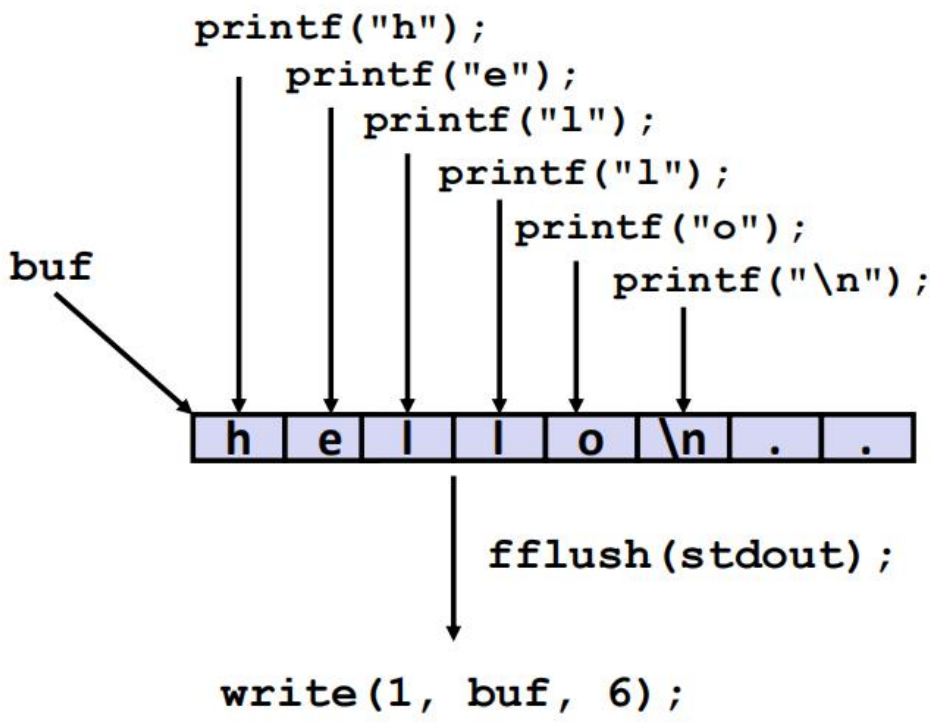
# 问题

- read,write等系统调用代价大
- 每次读写的数据量小，但次数多

- 使用缓冲区(buffer)

# buffered I/O

- Use Unix read to grab block of bytes, user input functions take one byte at a time and refill when empty
- 利用局部性提高性能
- 磁盘<->buffer<->user




Question:

```
for(int i=0;i<2;i++){
    Fork();
    printf("-");
}
```

# 问题

- 不足值（Short Counts）

  read write传送的字节比程序要求的少

```
/* Read at most max_count bytes from file into buffer.
   Return number bytes read, or error value */
ssize_t read(int fd, void *buffer, size_t max_count);
```

```
/* Write at most max_count bytes from buffer to file.
   Return number bytes written, or error value */
ssize_t write(int fd, void *buffer, size_t max_count);
```

- 健壮读写 Robust I/O
- 允许有不足值
- 处理器程序和信号

# RIO

- Unbuffered input and output of binary data
  - rio_readn（returns short count only if it encounters EOF）
  - rio_writen

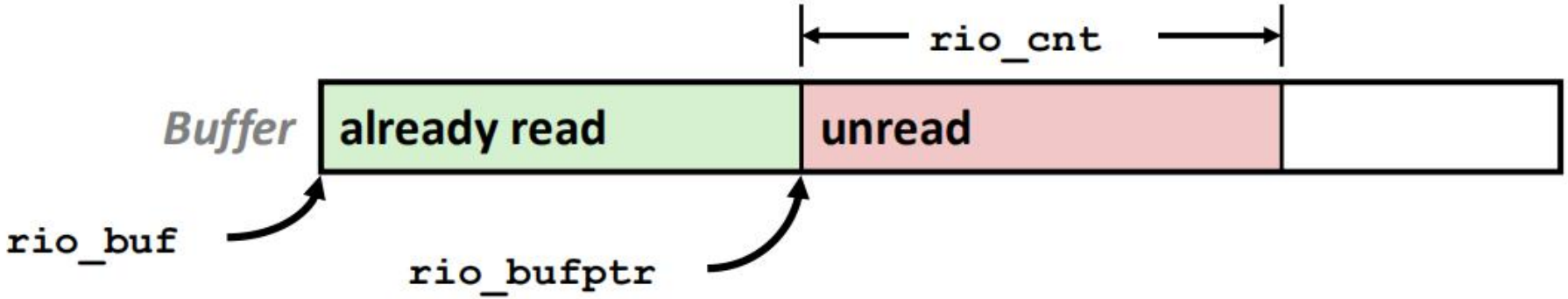
```
ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

- Buffered input of text lines and binary data
  - rio_readlineb
  - rio_readnb（不能和readn一起用）

```
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
```

# RIO buffer

- buffer of file




```
typedef struct {
    int rio_fd;                /* descriptor for this internal buf */
    int rio_cnt;               /* unread bytes in internal buf */
    char *rio_bufptr;          /* next unread byte in internal buf */
    char rio_buf[RIO_BUFSIZE]; /* internal buffer */
} rio_t;
```

# IO选择

- 网络socket一定用RIO
- 磁盘和终端文件用STDIO
- 信号处理器用raw Unix IO

谢谢~