System-Level I/O

周新凯

2024/11/20

Unix I/O

在Linux当中，所有的 IO 设备也都被模型化为文件。输入输出都被当作简单的读写文件，这使得我们所有的输入和输出都可以简单一致的方式去访问：

- 打开文件：通过 open 函数打开一个文件，**内核**会记录有关这个打开文件的**所有信息**，并向用户层会返回一个文件描述符，**用户层**要操作文件只需要对**文件描述符**操作即可。
- 改变当前文件的位置：对于每个打开的文件，内核会记录文件所在的位置 k，初始为 0。应用程序可以通过 seek 操作，显式地改变这个值。
- 读写文件：读文件就是把文件中从 k 开始到 k+size 的文件内容复制到内存，写文件就是把文件中从 k 开始到 k + size 的文件内容用内存中的某些值替换。
- 关闭文件：完成了访问之后，我们应当使用 close 函数去通知内核关闭这个文件，释放系统资源。

Unix I/O

注意:

- Linux shell 在创建进程的时候有默认的三个打开的文件：标准输入（stdin），标准输出（stdout），标准错误（stderr），他们的描述符分别为0（STDIN_FILENO , 以此类推)，1，2。
- 应用程序能检测到EOF（end-of-file）条件，文件结尾处并**没有**明确的“EOF符号”。

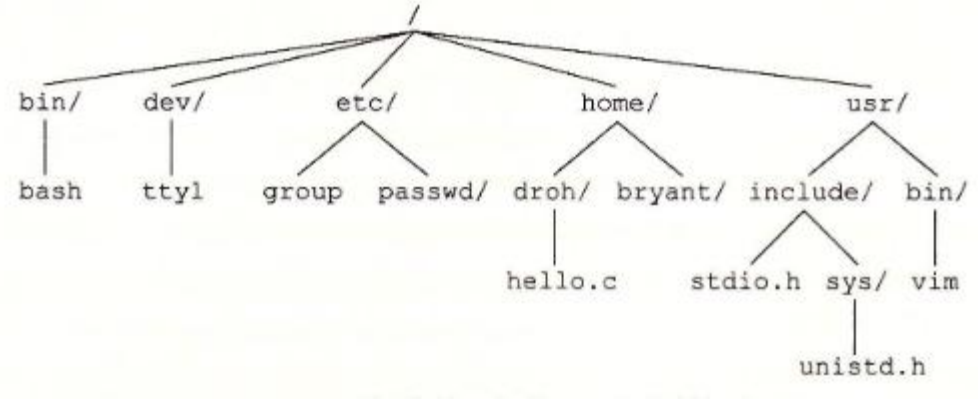
文件

- 普通文件（regular file）：包含任意数据。应用程序常常要区分文本文件和二进制文件，而对内核而言，文本文件和二进制文件**没有区别**。

  (文本文件：只含有ASCII或Unicode字符；二进制文件：其它所有文件)

文件

- 目录（directory）：是包含一组链接的文件，其中每个链接都将一个文件名映射到一个文件，这个文件可能是另一个目录。每个目录至少包含**两**个条目：. 是到该目录自身的链接；以及 .. 是到目录层次结构中父目录的链接。



文件

- 套接字（socket）：是用来与另一个进程进行跨网络通信的文件。

- 其它文件类型，包括命名通道（named pipe），符号链接（symbolic link），字符和块设备（character and block device)**不在讨论范畴。**

对文件的操作

- int open(char *filename, int flags, mode_t mode)：打开或创建

  open 函数将 filename 转换为一个文件描述符，并且返回描述符数字。

  返回的描述符总是在进程中**当前没有打开的最小描述符**。(见课本练习题10.1)

  flags 参数指明了进程打算如何访问这个文件，也可以是一个或者更多位掩码的或，为写提供给一些额外的指示。

  mode 参数是我们创建文件时的权限， Linux 的文件权限由 9 位二进制数字组成，因此它也有定义九个宏分别表示这些权限。

- int close(int fd)：关闭

  关闭一个**已关闭的**描述符会出错。

对文件的操作

- ssize_t read(int fd, void *buf, size_t n)：读/输入
- ssize_t write(int fd, const void *buf, size_t n)：写/输出

  read 函数从描述符为 fd 的当前文件位置复制**最多** n 个字节到内存位置 buf 。
  write 函数从内存位置 buf 复制**最多**n 个字节到描述符 fd 的当前文件位置。
  在某些情况下，read 和 write 传送的字节比应用程序要求的要少——不足值（short count）
  不足值**并不表示有错误。**可能原因有：EOF,从终端读文本行,读和写网络套接字。
  注：读磁盘文件（除非EOF）**不会**遇到不足值，写也不会。

RIO包

RIO（Robust I/O）就是一个 I/O 包，它会自动处理上文中提到的不足值。RIO 提供了两类不同的函数：

- **无缓冲的输入输出函数。**这些函数直接在内存和文件之间传输数据，没有应用级的缓冲。它们对将**二进制数据**读写到网络和从网络读写**二进制数据**尤其有用。
- **带缓冲的输入函数。**这些函数允许我们高效地从文件中读取**文本行**和**二进制数据**，这些文件的内容缓存在应用级缓冲区内，类似于为 printf 这样的标准 I/O 函数提供的缓冲区。

其中后者是**线程安全**的。(见12.7.1)

RIO包

- ssize_t rio_readn(int fd, void *usrbuf, size_t n)：无缓冲输入
- ssize_t rio_writen(int fd, void *usrbuf, size_t n)：无缓冲输出

  对于同一个描述符，可以任意的交错调用rio_readn和rio_wiriten。

  rio_writnen 函数**不会**返回不足值。

RIO包

- void rio_readinitb(rio_t *rp, int fd);
- 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_t 是一个 读缓冲区。

  rio_readinitb函数将描述符 fd 和地址 rp 处的一个类型为 rio_t 的读缓冲区联系起来。

  rio_readlineb函数处理文本行，最多读取maxlen - 1个字节，余下的一个字符留给结尾的NULL 字符。超过maxlen - 1字节的文本行被截断，并用一个NULL字符结束。

  rio_readnb函数处理二进制数据，从文件 rp 最多读 n 个字节到内存位置 usrbuf。

  对同一描述符，对 rio_readlineb 和 rio_readnb 的调用**可以任意交叉进行。**

  然而对于这些**带缓冲**的函数的调用不能和**无缓冲**的 rio_readn 函数交叉使用。

读取文件原数据

- int stat(const char *filename, struct stat *buf);
- int fstat(int fd, struct stat *buf);

  应用程序能够通过调用 stat 和 fstat 函数，检索到关于文件的信息（有时也称为文件的元数据（metadata））。

  stat 结构体定义在 sys/stat.h 头文件中。需要的成员有 st_mode 和 st_size 。

  st_size 成员包含了文件的字节数大小。st_mode 成员则编码了文件访问许可位和文件类型。

  Linux 在 sys/stat.h 中定义了宏谓词来确定 st_mode 成员的文件类型。

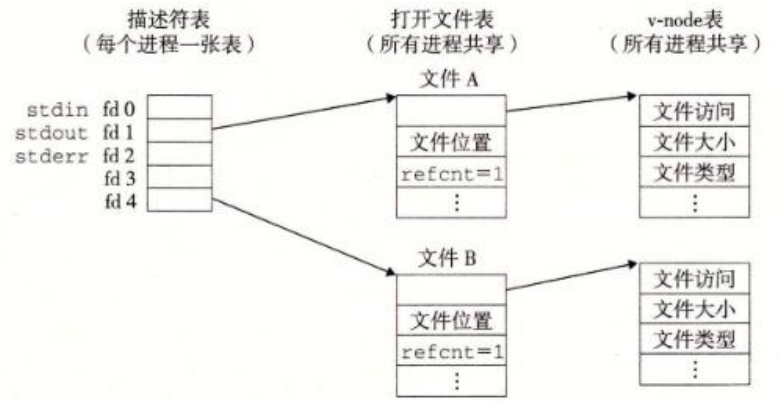
读取目录内容

- DIR *opendir(const char *name)：打开
  返回指向目录流的指针
- struct dirent *readdir(DIR *dirp)：读
  返回指向下一个目录的指针
- int closedir(DIR *dirp)：关闭

共享文件

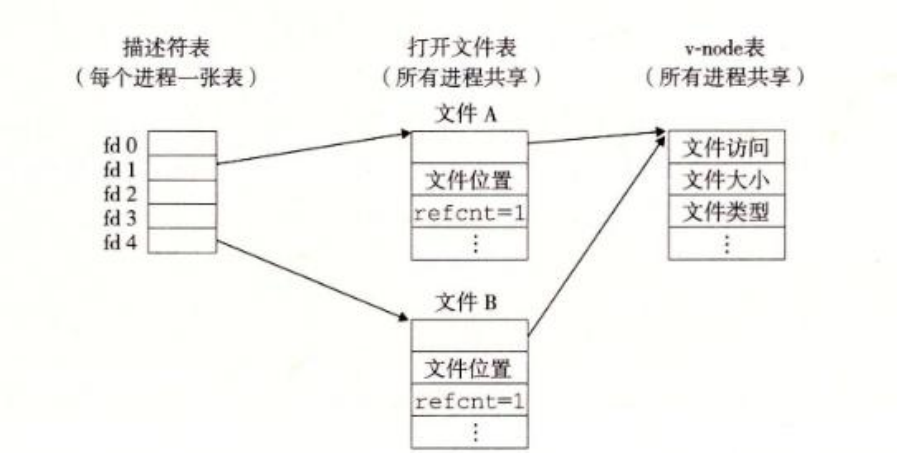
内核用三个相关的数据结构来表示它打开的文件：

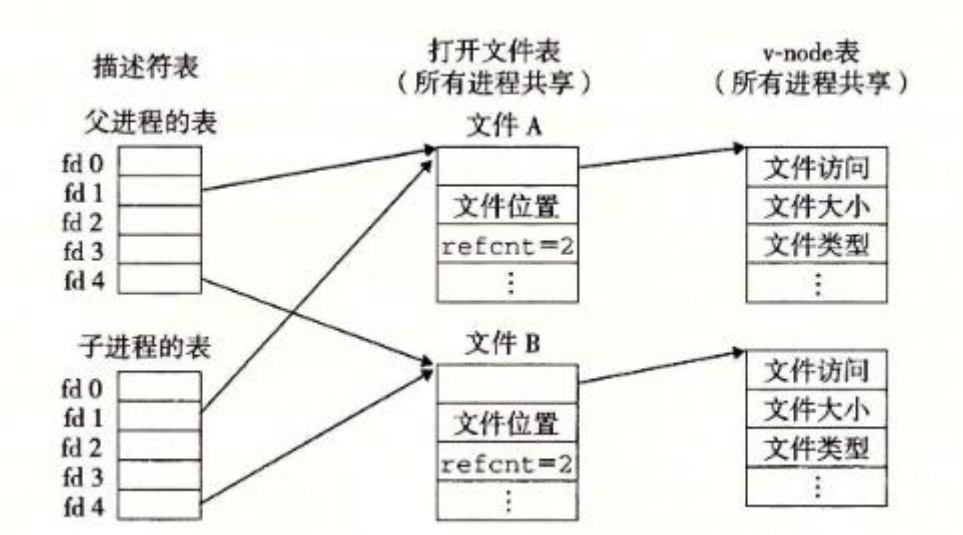
- 描述符表：进程之间**独立**，每个打开的文件描述符表项指向文件表中的一个表项。
- 文件表：所有进程**共享**。每个表项的组成包括了文件位置、引用计数、以及指向v-node 表项对应的指针。
- v-node 表：所有进程**共享**，里面的一个表项包含了 stat 结构信息以及其它一些额外的字段。



共享文件

- 多个描述符可以通过不同的文件表项来引用同一个文件（左图）
- 父子进程共享文件（右图）





I/O重定向

- int dup2(int oldfd, int newfd)

  dup2 函数复制描述符表表项 oldfd 到描述符表表项 newfd，覆盖描述符表表项 newfd 以前的内容。如果 newfd 已经打开了，dup2 会在复制 oldfd 之前关闭 newfd。

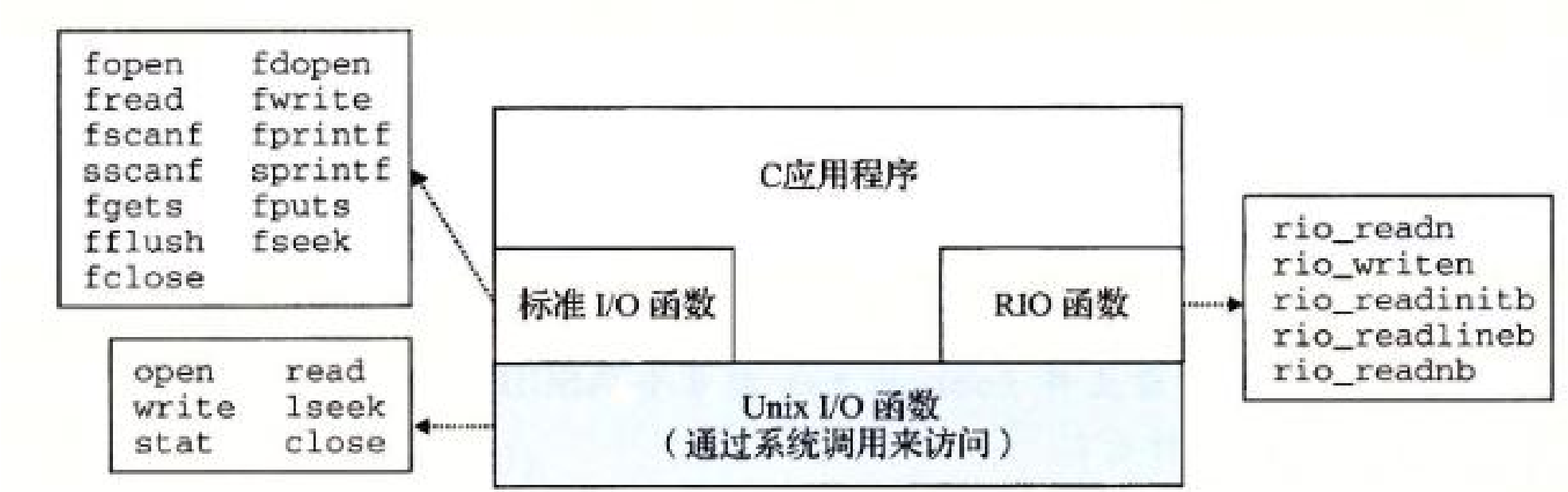
标准I/O

标准I/O库提供了打开和关闭文件的函数（fopen 和 fclose），读和写字节的函数（fread 和 fwrite），读和写字符串的函数（fgets 和 fputs），以及复杂的格式化的 I/O 函数（scanf 和 printf）。

标准 I/O 库将一个打开的文件模型化为一个流。类型为 FILE 的流是对文件描述符和流缓冲区的抽象。流缓冲区的目的和 RIO 读缓冲区的一样,就是使开销较高的 Linux I/O 系统调用的数量尽可能得小。

我们该使用哪些I/O函数

下图是 Unix I/O、标准 I/O、和 RIO 之间的关系：



我们该使用哪些I/O函数

- **只要有可能就使用标准 I/O。**对磁盘和终端设备 I/O 来说，标准 I/O 函数是首选方法。除了 stat 读取文件基本信息以外，使用 stdio 封装的函数。
- **不要使用 scanf 或 rio_readlineb 来读二进制文件。**像 scanf 或 rio_read-lineb 这样的函数是专门设计来读取文本文件的。二进制文件可能会散布很多的 0xa 字节，而 scanf 遇到会将它们识别为终止符，因此会出现错误。
- **对网络套接字的 I/O 使用 RIO 函数。**

谢谢！