第10章 系统级I/O

Unix I/O

- 将所有的I/O设备（例如网络、磁盘和终端）都优雅地映射为文件。
- 打开文件使用一个descriptor标识
- 每个进程创建开始都有三个打开的文件：standard input(descriptor 0 or STDIN_FILENO), standard output(descriptor 1 or STDOUT_FILENO), standard error(descriptor 2 or STDERR_FILENO)
- 文件位置file position可以通过seek操作，初试为0
- 读文件即从文件复制n(n>0)字节到内存,file position响应从k变为k+n,当k大于等于文件字节总数时触发EOF（end-of-file）条件。**文件结尾处并没有EOF符号**
  写操作就是从内存复制n(n>0)字节到文件，从当前file position开始，然后更新file position
- 关闭文件后释放descriptor

Files

- regular file: 包含任意数据。
  - text file 只包含ASCII或Unicode的普通文件
  - binary file 所有其他文件
- directory： 包含一组链接（link）的文件，每个链接将filename映射到一个文件（包括目录）
- socket： 用来与另一个进程进行跨网通信的文件

文件操作：

-
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

//成功返回new file descriptor, 出错返回-1
//flags: O_RDONLY,OWRONLY,O_RDWR,O_CREAT(不存在则新建),O_TRUNC(已存在则截
断),O_APPEND
//访问权限mode & ~umask   umask(some_mode);
//9种权限：S_IR(W or X)USR(GRP or OTH)
          读/写/执行 * 拥有者/所在组成员/其他任何人
int open(char *filename, int flags, mode_t mode);
```

```
#include <unistd.h>
//成功返回0，出错返回-1
int close(int fd);
```

```
#include <unistd.h>
//出现short count的原因：
  //读时遇到EOF，先返回读取值，此后read通过返回0发出EOF信号
  //从终端读文本行，一次读一行
```

```
  //读和写socket，网络延迟
//成功返回读的字节数，若EOF返回0，出错返回-1
ssize_t read(int fd, void *buf, size_t n);
//成功返回写的字节数，出错返回-1
ssize_t write(int fd, const void *buf, size_t n);
```

RIO

- 无缓冲的RIO

```
#include "csapp.h"
//成功返回传送的字节数，若EOF返回0，出错返回-1
ssize_t rio_readn(int fd, void *usrbuf, size_t n);
sszie_t rio_writen(int fd, void *usrbuf, size_t n);
```

- 带缓冲的RIO

```
#include "csapp.h"

//把fd和rp处的一个类型为rio_t的读缓冲区联系起来
void rio_readinitb(rio_t *rp, int fd);

//成功返回读的字节数，EOF返回0，出错返回-1
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
sszie_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
```

Metadata

```
#include <unistd.h>
#include <sys/stat.h>

//检索关于文件的信息
//st_size 文件字节数大小
//st_mode 文件访问许可位
  //S_ISREG(st_mode) 是一个普通文件？
  //S_ISDIR(st_mode) 是一个目录文件？
  //S_ISSOCK(st_mode) 是一个socket文件？
//成功返回0，出错返回-1
int stat(const char *filename, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

Directory Contents

```
#include <sys/types.h>
#include <dirent.h>
```

```
//成功返回directory stream指针，否则为NULL
DIR *opendir(const char *name);

//成功返回下一个directory entry,否则为NULL
struct dirent *readdir(DIR *dirp);

//成功返回0，错误返回-1
int closedir(DIR *dirp);
```

Sharing Files

- Descriptor table: 每个进程都有独立的此表，每个entry使用fd索引，指向File table中的一个entry
- File table：标识打开文件的集合，所有进程共享。每个entry包括pos、refcnt和指向v-node table中一个entry的指针。关闭fd减少对应refcnt，当refcnt=0时删除该entry。
- v-node table：所有进程共享，每个entry包含stat中大部分信息，包括st_mode和st_size。

Redirection

```
#include <unistd.h>
//复制oldfd到newfd并覆盖
//若newfd已经打开，复制前先关闭newfd
//成功返回非负的fd，出错返回-1
int dup2(int oldfd, int newfd);
```

Standard I/O