10.4 文件系统调用

- 10.4.1 文件描述符
- 10.4.2 open和close系统调用
- 10.4.3 read和 write系统调用
- 10.4.4 lseek系统调用
- 10.4.5 访问和显示文件元数据
- 10.4.6 目录

10.4.1 文件描述符

- 特殊文件描述符：进程开始运行时自动打开
  - 0：STDIN_FILENO　标准输入
  - 1：STDOUT_FILENO　标准输出
  - 2：STDERR_FILENO　标准错误输出
- 文件描述符的应用
  - 打开文件，得到文件描述符
  - 通过文件描述符对文件进行读写等操作
  - 通过文件描述符关闭文件

文件描述符

- 用户进程A连续三次打开文件
  - fd1 = open(“/etc/passwd”, O_RDONLY);
  - fd2 = open(“local”, O_RDWR);
  - fd3 = open(“/etc/passwd”, O_WRONLY);
  - 得到三个文件描述符：
    - fd1：3； fd2： 4； fd3：5
- 一个文件可以被某个进程多次打开，每次都分配一个file，并占用fd[ ]的一项，得到一个文件标识号。
  - 不同file中的f_inode都指向同一个inode。

10.4.2 open和close系统调用

```
int  open(const char * pathname,  int  flags);
int  open( const char * pathname, int flags, mode_t mode );
```

文件名　　打开方式　　存取权限

成功:文件描述符，否则:-1

标志	含义	含义
O_RDONLY	只读	用户可读
O_WRONLY	只写	用户可写
O_RDWR	读写	用户可执行
O_APPEND	追加	用户可读、写、执行
O_CREAT	创建	组可读
O_TRUNC	如果文件已经存在，则删除文件的内容	其他人可读
.....	.....	.....

close系统调用

```
int close(int  fd);
```

文件描述符

成功:0，　否则:-1

- 对文件的file中的引用计数减1，如果减为0，则释放该文件描述项。
- 对文件inode中的引用计数减1。
- 释放该文件描述符

10.4.3 read和write系统调用

从fd 所指文件读取 length个字节到 buf

```
int read(int fd,  const void * buf,  size_t  length)
```

把 length个字节从 buf 写到fd 所指文件

```
int write(int fd, const void * buf,  size_t  length)
```

指向缓冲区的指针

缓冲区的大小

成功:实际读写的字节数，否则:-1

出错信息的处理

- 全局变量 errno：上一个函数发生错误的原因
  - 当Linux C API函数发生异常时，会赋一个整数值
  - 通过查看该值推测出错的原因。

error.c

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
int main(void)
{int  fd; extern int errno;
   if((fd = open("no/such/file",0)) < 0) {
        printf("errno=%d \ n",errno);
        char * mesg = strerror(errno);
        printf("Mesg:%s \ n",mesg);
       }
  }
```

stdout -- 标准输出设备
stderr -- 标准错误输出设备
两者默认向屏幕输出。

[5_4] 从键盘输入串，写到文件中。

```
int main(int argc,char **argv)
{ int fd; char buffer[BUFFER_SIZE];
if(argc!=2){
        fprintf(stderr,"Usage:%s outfilename\n\a",argv[0]);
        exit(1);
        }
if((fd=open(argv[1],O_WRONLY|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR))==-1) {
        fprintf(stderr,"Open %s Error:%s\n\a",argv[1],strerror(errno));
        exit(1);
        }
printf("Now,please input string");  printf("(To quit use CTRL+D)\n");
while(1) {
        fgets(buffer,BUFFER_SIZE,stdin);
        if(feof(stdin)) break;
        write(fd,buffer,strlen(buffer));
        }
close(fd);
}
```

[5_6] 简单的文件拷贝。源文件是只有一个字符串构成的文本文件。

```
#define Length 1024
int main( ){
int fdw,fdr,len;  char str[Length];
char sourcename[15],targetname[20];
printf("Please input the name of the source file: ");
gets(sourcename);
printf("Please input the name of the target file: ");
gets(targetname);
fdr=open(sourcename,O_RDONLY);
if(fdr)   len=read(fdr,str, Length);
else{printf("read file error");
      exit(0); }
fdw=open(targetname,O_CREAT|O_RDWR);
write(fdw,str,len);
close(fdr); close(fdw);
}
```

10.4.4 lseek系统调用

将读写指针相对whence移动offset个字节

```
int lseek(int fd, offset_t offset,  int whence);
```

SEEK_SET：相对文件开头
SEEK_CUR：相对文件读写指针的当前位置
SEEK_END：相对文件末尾

成功:文件指针相对于文件头的位置。 否则:-1

```
lseek( fd, 0, SEEK_END);
```

返回文件的长度。

lseek系统调用--文件偏移量

- 文件偏移量（文件读写指针）
  - 标识下一次读或者写文件的位置
  - 一个进程两次打开同一个文件，由于得到两个不同的文件描述符和文件描述项，分别拥有独立的文件偏移量（读写指针）

[5_7]文件的定位操作

```
int main(void)
{       char buf1[]={"abcdefghijk"};     char buf2[]={"1234567890"};
        int fd;             int length;
        if((fd=open("test", NEWFILE,0600))==-1){
                printf("ERROR, open write file error:%s\n", strerror(errno));
                exit(255);}
         length=strlen(buf1);
        if(write(fd,buf1,length)!=length){
                printf("ERROR, write file failed: %s\n", strerror(errno));
                exit(255);}
        if( lseek(fd, 80, SEEK_SET) ==-1){
                printf("E                                \n", strerror(errno));
                exit(255);
        length=strlen(buf2);
        if(write(fd,buf2,length)!=length){
                printf("
                exit(255
         close(fd);
```

$ od - tc test

按地址基数为8进制，以ASCII字符显示文件test内容

```
administrator@ubuntu:~$ od -tc test
0000000   a   b   c   d   e   f   g   h   i   j   k  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000120   1   2   3   4   5   6   7   8   9   0
0000132
```

10.4.5 访问和显示文件元数据

- i节点包含文件的元数据
  - 除了文件名和文件数据外的所有信息
  - 使用stat、fstat、lstat系统调用和ls –l命令访问

文件路径　　文件信息

```
int stat( const char *path, struct stat  *buf);
int fstat( int fd, struct stat *  buf);
```

struct stat

- 能直接看到
  - i节点号、链接数、所有者id、所属组id、文件大小、文件系统磁盘块大小、分配的磁盘块个数
- 经过转换看到
  - 文件类型、权限、时间

```
    ……
};
```

[5_8]从struct stat中获取文件的 time of last access，并用char *ctime（time_t* t）转换成可读形式

```
int main(int args,char *argv[]){
 struct stat   statbuf;  time_t  t;
 if(args < 2){
        printf(“please input a file\n”);
       exit(1); }
 if( stat(argv[1],&statbuf ) == 0){
         t = statbuf.st_atime;
         printf("%s\n",ctime(&t)); }
 else{perror(argv[1]);
       exit(1); }
 return 0;
}
```

perror ( ) :将上一个函数发生错误的原因输出到 stderr。实参所指的字符串会先打印出，后面再加上错误原因字符串。

10.4.6 目录

- 在程序中获得（或改变）进程的当前工作目录
  - getcwd、chdir、fchdir
- 创建、删除目录
  - mkdir、rmdir
- 读取目录文件中的信息
  - opendir、readdir、closedir
  - rewinddir、telldir、seekdir

读目录流操作

目录名

- 基本过程
  - 打开目录　DIR *opendir(const char *path);
    - 返回值：成功DIR指针，否则NULL
  - 操作目录
    读取目录内容　struct dirent　* readdir (DIR * dirp);
    返回目录起始位置
  - 关闭目录　int closedir(DIR *dirp);
    返回值：成功0，否则-1

opendir返回的DIR指针

目录基本操作

- 读目录中的目录项

```
struct dirent  * readdir (DIR * dirp);
```

成功: dirent指针
否则: NULL

opendir返回的DIR指针

```
struct dirent {
    long  d_ino;          //文件对应 inode number
    off_t  d_off;         //该目录项在目录文件中偏移量
   unsigned  short d_reclen;    //文件名长度len of d_name
   char  d_name [NAME_MAX+1];   //文件名file name
 }
```

[5_9]程序有一个参数。如果参数是一个文件名， 输出这个文件的大小和最后修改的时间，如果是一个目录，输出此目录下所有文件(不包括子目录)的大小和修改时间。

```
int main(int argc,char **argv)                                    P102
{ DIR *dirp;   struct dirent *direntp;    int stats;
if(argc!=2){
     printf("Usage:%s filename\n\a",argv[0]);  exit(1); }
if(((stats=get_file_size_time(argv[1]))==0)||(stats==-1))  exit(1);
if( (dirp=opendir(argv[1]))==NULL){
     printf("Open Directory %s Error:%s\n", argv[1], strerror(errno));
     exit(1);   }
while( (direntp=readdir(dirp))!=NULL )
     if (get_file_size_time(direntp->d_name)==-1) break;
closedir(dirp);  exit(1);  }
```

```
static int get_file_size_time(const char *filename){
struct stat statbuf;
if( stat( filename,&statbuf) ==-1 ){
    printf("Get stat on %s Error:%s\n", filename,strerror(errno));
    return(-1);
    }
if(S_ISDIR(statbuf.st_mode))  return(1); /*目录文件*/
if(S_ISREG(statbuf.st_mode))/*普通文件*/
    printf("%s size:%ld bytes\tmodified at %s",
            filename, statbuf.st_size, ctime(&statbuf.st_mtime));
return(0);
}
```

程序的扩充

- 运行时以一个目录名为参数，输出此目录下所有文件的大小和修改时间。（参照**[5_9]**）
- 如果此目录下有名为“**source.txt**”的文本文件，将其内容复制到名为**”target.txt”**的文件中。**(**简单文件复制参照**[5_4])**
- 实现对树形文件结构的广度优先遍历，即按层输出文件信息。