



西安邮电大学

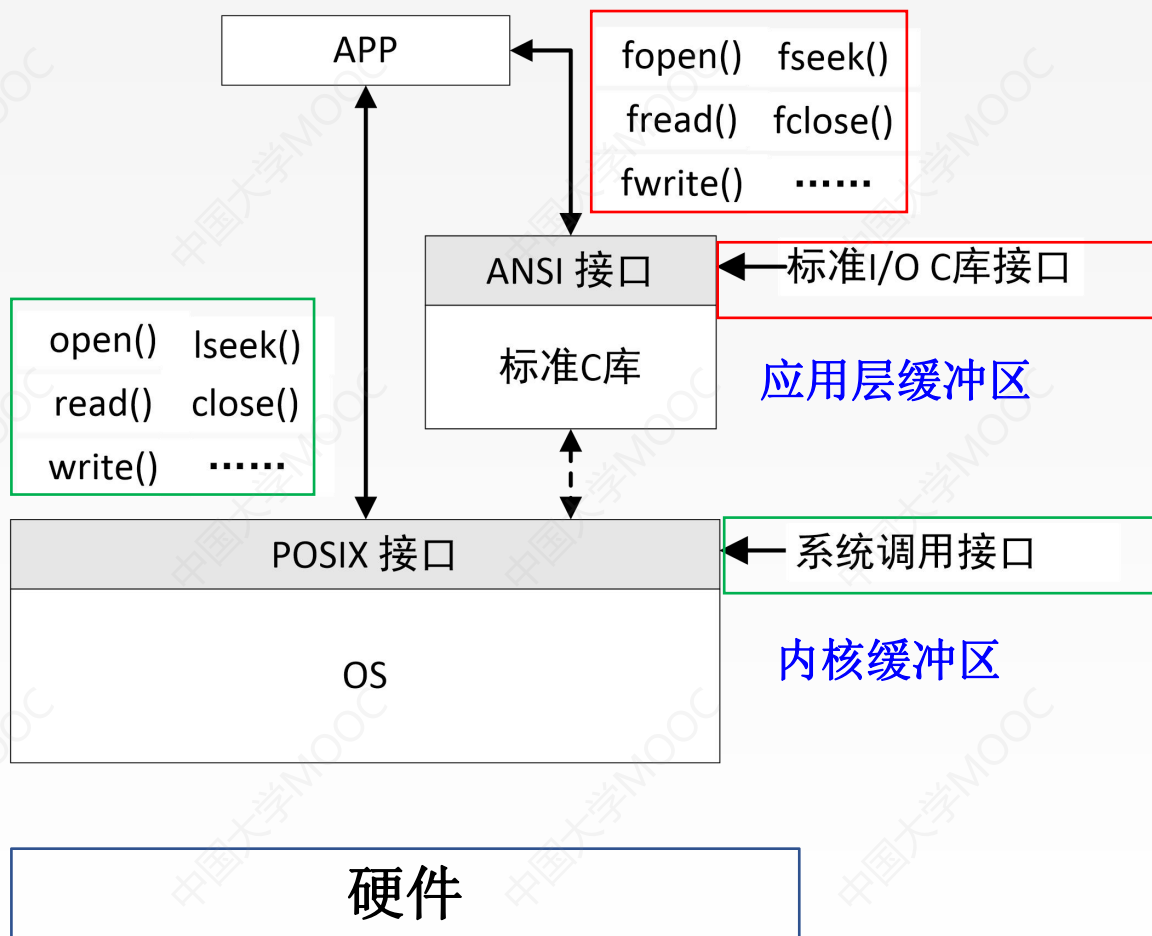
XI'AN UNIVERSITY OF POSTS & TELECOMMUNICATIONS

Linux 编程技
术



第2章 文件、目录管理

——Linux 文件系统的常用接口-上



1.文件的打开、创建和关闭

(1) open

函数名称	open
函数功能	打开或创建文件
函数原型	<code>int open(const char *pathname, int flags);</code> <code>int open(const char *pathname, int flags, mode_t mode);</code>
参数	<code>pathname</code> : 要打开或创建的含路径的文件名 <code>flags</code> : 文件状态标志, 表示打开文件的方式 <code>mode</code> : 如果文件被新建, 指定其权限为mode (八进制表示法)
返回值	大于或等于0的整数: 成功 (即文件描述符) -1: 失败

1.文件的打开、创建和关闭

(1) open

主标志

- O_RDONLY: 只读方式打开文件。
- O_WRONLY: 以只写方式打开文件。
- O_RDWR: 以可读可写方式打开文件。

必须互斥使用


副标志

- | | |
|--------------|-------------|
| •O_CREAT | •O_EXCL |
| •O_TRUNC | •O_APPEND |
| •O_NOCTTY | •O_CLOEXEC |
| •O_DIRECTORY | •O_NOFOLLOW |
| •O_NONBLOCK | |

副标志可同时使用多个，与主标志进行或（|）运算符

1.文件的打开、创建和关闭

(1) open


`fd=open("test3_1.c",O_CREAT|O_EXCL,S_IRUSR|S_IWUSR)==-1`

函数名称	open
函数功能	打开或创建文件
函数原型	<code>int open(const char *pathname, int flags);</code> <code>int open(const char *pathname, int flags, mode_t mode);</code>
参数	pathname: 要打开或创建的含路径的文件名 flags: 文件状态标志, 表示打开文件的方式 mode: 如果文件被新建, 指定其权限为 mode (八进制表示法)
返回值	大于或等于0的整数: 成功 (即文件描述符) -1: 失败

1.文件的打开、创建和关闭

(2) close

函数名称	close
函数功能	关闭一个已打开文件
头文件	#include <unistd.h>
函数原型	int close(int fd);
参数	fd: 即将要关闭的文件描述符
返回值	0: 成功 -1: 失败

例1: open、close举例

```
int main()
{
    int fd;
    if((fd=open("test3_1.c", O_CREAT|O_EXCL, S_IRUSR
|S_IWUSR)) == -1)
    {
        perror("open() error.");
        exit(1);
    }
    else
    {
        printf("create file success\n");
    }
    close(fd);
    return 0;
}
```

新建文件权限为 (mode&~umask)

```
root@ubuntu:3# ls -l test3_1.c
-rw----- 1 root root 0 1月 29 19:22
```

```
root@ubuntu:3# umask -S
u=rwx,g=rx,o=rx
```

2.文件的读、写

(1) read

函数名称	read
函数功能	从指定文件中读取数据
头文件	#include <unistd.h>
函数原型	ssize_t read(int fd, void *buf, size_t count);
参数	fd: 从文件fd读数据 buf: 指向存放读到的数据的缓冲区 count: 从文件fd中读取的字节数
返回值	实际读到的字节数: 成功 (实际读到的字节数小于或等于count) 失败: -1

2. 文件的读、写

(2) write

函数名称	write
函数功能	将数据写入指定的文件
头文件	#include <unistd.h>
函数原型	ssize_t write(int fd, void *buf, size_t count);
参数	fd: 将数据写入文件fd中 buf: 指向要写入fd的数据所在的缓冲区 count: 写入的字节数
返回值	实际写入的字节数: 成功 -1: 失败

- `read`系统调用和`write`系统调用的参数`count`只是一个“愿望值”。
- 读/写操作会使文件当前位置偏移量加上实际读写的字节数，不断地往后移动。

3. 改变文件属性

fcntl

控制命令字

函数名称	fcntl
函数功能	文件控制
头文件	#include <unistd.h>#include <fcntl.h>
函数原型	int fcntl(int fd, int cmd); int fcntl(int fd, int cmd, long arg); int fcntl(int fd, int cmd, struct flock *lock);
参数	fd: 要控制的文件的描述符
	cmd: 控制命令字
	变参: 根据不同的命令字而不同
返回值	根据不同的cmd, 返回值不同: 成功
	-1: 失败

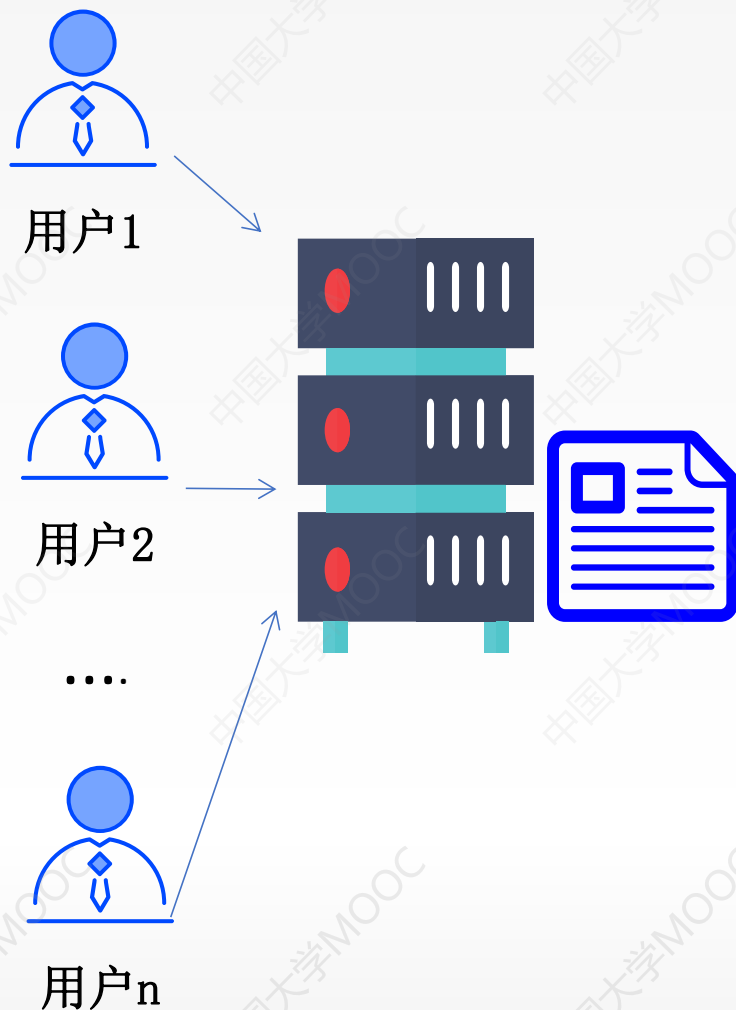
3. 改变文件属性

fcntl

cmd的取值:

- (1) 复制一个已有的描述符 (cmd=F_DUPFD或F_DUPFD_CLOEXEC)
- (2) 获取/设置文件描述符标志 (cmd=F_GETFD或F_SETFD)
- (3) 获取/设置文件状态标志 (cmd=F_GETFL或F_SETFL)
- (4) 获取/设置异步I/O所有权 (cmd=F_GETOWN或F_SETOWN)
- (5) 获取/设置记录锁 (cmd=F_**GETLK**或F_SETLK)

1. 多用户共享文件模型分析



同时允许读文件

`lock.l_type==F_RDLCK`

同时写文件互斥

`lock.l_type==F_WRLCK`

2.fcntl对锁的设置

fcntl(fd, F_SETLK, &lock)

文件
↓
设置锁和释放锁

锁的属性

```
struct flock  
{
```

```
short l_type; <
```

锁类型

```
short l_whence;
```

```
off_t l_start; <
```

锁范围

```
off_t l_len;
```

```
pid_t l_pid; <
```

获得锁
进程

```
};
```

3.共享文件互斥操作实现

```
int main()//fcntl_lock.c
{
    int fd;
    fd=open("hello_lck.c",O_RDWR|O_CREAT ,0666);
    if(fd<0)
    {
        perror("打开出错");
        exit(1);
    }
```

打开文件

```
lock_set(fd,F_WRLCK);
```

设置写锁

```
getchar();
```

```
lock_set(fd,F_UNLCK);
```

释放锁

```
getchar();
```

```
lock_set(fd,F_RDLCK);
```

设置读锁

```
getchar();
```

```
lock_set(fd,F_UNLCK);
```

释放锁

```
close(fd);
```

```
exit(0);
```

```
}
```

3.共享文件互斥操作实现

```
1 void lock_set(int fd, int type)
2 {
3     .....//省略了部分代码
4     while(1)
5     {
6         lock.l_type=type;
7
8         if(fcntl(fd, F_SETLK, &lock)==0)
9         {
10             if(lock.l_type==F_RDLCK)
11                 printf("加上读取锁的是: %d进程\n", getpid());
12             else if(lock.l_type==F_WRLCK)
13                 printf("加上写入锁的是: %d进程\n", getpid());
14             else if(lock.l_type==F_UNLCK)
15                 printf("释放强制性锁的是: %d进程\n", getpid());
16             return ;
17         }
18         .....//省略部分代码
```

设置锁

显示锁由哪个进程所属

3.共享文件互斥操作实现

```
if (fcntl(fd, F_GETLK, &lock) == 0)
{
    if (lock.l_type != F_UNLCK)
    {
        if (lock.l_type == F_RDLCK)
            printf("文件已经加上了读取锁，其进程号是：\n", lock.l_pid);
        else if (lock.l_type == F_WRLCK)
            printf("文件已经加上了写入锁，其进程号是：\n", lock.l_pid);
        getchar();
    }
}
```

获取锁的一些信息

4.运行结果验证

```
root@ubuntu:~# ./fcntl_lock2  
加上写入锁的是: 5844进程  
释放强制性锁的是: 5844进程
```



终端1

```
root@ubuntu:~# ./fcntl_lock2_demo  
锁操作失败  
: Resource temporarily unavailable  
文件已经加上了写入锁, 其进程号是: 5844  
加上写入锁的是: 5845进程  
释放强制性锁的是: 5845进程  
加上读取锁的是: 5845进程
```



终端2

无法获得
写锁

5845号进程获
得写锁

1. 打开、关闭文件

(1) fopen函数

FILE文件流

函数名称	fopen
函数功能	获取指定文件的文件指针
头文件	#include <stdio.h>
函数原型	FILE *fopen(const char *path, const char *mode);
参数	path: 要打开含路径的文件名 mode: 文件打开的方式
返回值	指向FILE的指针: 成功 NULL: 失败

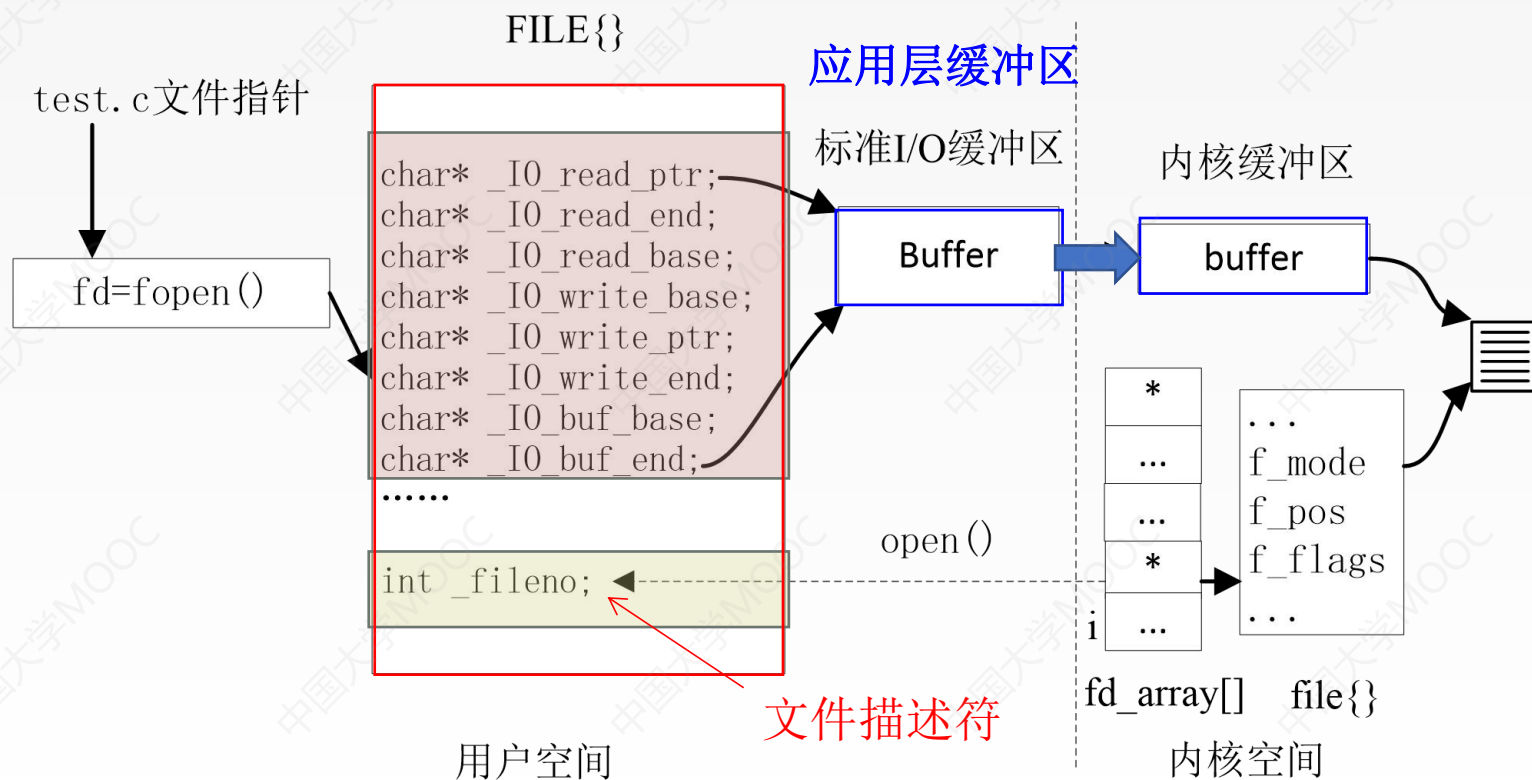
1. 打开、关闭文件

(2) fclose函数

已打开的
FILE文件流

函数名称	fclose
函数功能	关闭指定文件并释放其资源
头文件	#include <stdio.h>
函数原型	int *fclose(FILE *fp);
参数	fp: 即将要关闭的文件
返回值	0: 成功 EOF: 失败

2. 文件流与文件描述符



- 文件操作的两类接口：系统调用+库函数
- 打开（open）文件的含义
- 关闭（close）文件的含义
- 使用fcntl函数实现多用户对共享文件的互斥操作
- 文件流的含义
- 文件流与文件描述符的关系

谢谢大家!

