第四章 文件和目录

```
#define __S16_TYPE          short int
#define __U16_TYPE          unsigned short int
#define __S32_TYPE          int
#define __U32_TYPE          unsigned int
#define __SLONGWORD_TYPE    long int
#define __ULONGWORD_TYPE    unsigned long int
/usr/include/bits/types.h [FORMAT=unix:utf-8]
```

```
localhost include # pwd
/usr/include
localhost include # grep -r '__MODE_T_TYPE' *
bits/typesizes.h:#define __MODE_T_TYPE          __U32_TYPE
bits/types.h:__STD_TYPE __MODE_T_TYPE __mode_t; /* Type of file attribute bitmasks.  */
```

```
struct stat
  {
    __dev_t st_dev;     /* Device.  */
#if __WORDSIZE == 32
    unsigned short int __pad1;
#endif
#if __WORDSIZE == 64 || !defined __USE_FILE_OFFSET64
    __ino_t st_ino;     /* File serial number.  */
#else
    __ino_t __st_ino;           /* 32bit file serial number.    */
#endif
#if __WORDSIZE == 32
    __mode_t st_mode;           /* File mode.  */
    nlink_t st_nlink;           /* Link count.  */
/usr/include/bits/stat.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001]
```

```
/* Test macros for file types.  */

#define __S_ISTYPE(mode, mask)  (((mode) & __S_IFMT) == (mask))

#define S_ISDIR(mode)    __S_ISTYPE((mode), __S_IFDIR)
#define S_ISCHR(mode)    __S_ISTYPE((mode), __S_IFCHR)
#define S_ISBLK(mode)    __S_ISTYPE((mode), __S_IFBLK)
#define S_ISREG(mode)    __S_ISTYPE((mode), __S_IFREG)
#ifdef __S_IFIFO
# define S_ISFIFO(mode)  __S_ISTYPE((mode), __S_IFIFO)
#endif
#ifdef __S_IFLNK
# define S_ISLNK(mode)   __S_ISTYPE((mode), __S_IFLNK)
/usr/include/sys/stat.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=026]
```

https://www.kernel.org/doc/Documentation/devices.txt

Offset	Size	Name	Description
0x0	__le16	i_mode	File mode. Any of: 0x1 S_IXOTH (Others may execute) 0x2 S_IWOTH (Others may write) 0x4 S_IROTH (Others may read) 0x8 S_IXGRP (Group members may execute) 0x10 S_IWGRP (Group members may write) 0x20 S_IRGRP (Group members may read) 0x40 S_IXUSR (Owner may execute) 0x80 S_IWUSR (Owner may write) 0x100 S_IRUSR (Owner may read) 0x200 S_ISVTX (Sticky bit) 0x400 S_ISGID (Set GID) 0x800 S_ISUID (Set UID) These are mutually-exclusive file types: 0x1000 S_IFIFO (FIFO) 0x2000 S_IFCHR (Character device) 0x4000 S_IFDIR (Directory) 0x6000 S_IFBLK (Block device) 0x8000 S_IFREG (Regular file) 0xA000 S_IFLNK (Symbolic link) 0xC000 S_IFSOCK (Socket)

```
localhost ~ #        普通文件
localhost ~ # ls -l /etc/fstab ; file /etc/fstab
-rw-r--r-- 1 root root 973  1月 27 2013 /etc/fstab
/etc/fstab: ASCII text
localhost ~ # ls -ld /etc ; file /etc 普通目录
drwxr-xr-x 53 root root 4096 10月 16 12:41 /etc
/etc: directory
localhost ~ # ls -l /bin/bash ; file /bin/bash
-rwxr-xr-x 1 root root 737808  1月 10 2013 /bin/bash
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/L
inux 2.6.9, stripped  可执行文件
localhost ~ # ls -l /bin/sh ; file /bin/sh 链接文件
lrwxrwxrwx 1 root root 4  1月 10 2013 /bin/sh -> bash
/bin/sh: symbolic link to `bash'
localhost ~ # ls -l /dev/tty ; file /dev/tty 字符设备文件
crw-rw-rw- 1 root tty 5, 0 10月 16 12:06 /dev/tty
/dev/tty: character special
localhost ~ # ls -l /dev/sda ; file /dev/sda 块设备文件
brw-rw---- 1 root disk 8, 0 10月 16 12:07 /dev/sda
/dev/sda: block special
```

```
localhost ~ # ls -l /var/run/mysqld/mysqld.sock ;\
>             file /var/run/mysqld/mysqld.sock
srwxrwxrwx 1 mysql mysql 0 10月 21 17:28 /var/run/mysqld/mysqld.sock
/var/run/mysqld/mysqld.sock: socket    socket文件
```

```
localhost ~ # tty
/dev/pts/1
localhost ~ # mkfifo test.pipe
localhost ~ # ls -l test.pipe ; file test.pipe
prw-r--r-- 1 root root 0 10月 22 16:46 test.pipe
test.pipe: fifo (named pipe) 管道文件
```

```
localhost ~ # tty
/dev/pts/2 <-------- 新开的终端
localhost ~ # tail -f test.pipe
```

```
localhost ~ # echo 'Hello James!' > test.pipe
localhost ~ # tty
/dev/pts/1
```

```
localhost ~ # tty
/dev/pts/2
localhost ~ # tail -f test.pipe
Hello James!
```

http://en.wikipedia.org/wiki/Linux_Security_Modules

一个任务的安全上下文

组成该上下文的部分可分解成两个类别：

1、一个客观任务的上下文，当一些其它的任务试图去影响它时，这些部分会被用到。

2、一个主观任务的上下文，当一个任务作用于其它对象上时(如一个文件、进程、key或其它任何)，这些详细信息会被用到。

注意，结构体中的成员，有些属于两个类别 —— 以LSM安全指针为例。

任务有两个安全指针。task->real_cred指到客观上下文，它定义了该任务的实际细节。每当任务被采用时，该上下文的客观部分会被使用。

task->cred指向主观上下文，它定义了任务如何作用到其它对象上

的细节。
它可能被重写，使之暂时指向其它对象的上下文。但通常指向相同的上下文，如task->real_cred。
进程凭据

```
/* process credentials */
    const struct cred __rcu *real_cred; /* objective and real subjective task
                    * credentials (COW) */
    const struct cred __rcu *cred;  /* effective (overridable) subjective task
                    * credentials (COW) */
```

进程凭据的结构体

```
struct cred {
    atomic_t    usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
    atomic_t    subscribers;    /* number of processes subscribed */
    void        *put_addr;
    unsigned    magic;
#define CRED_MAGIC  0x43736564
#define CRED_MAGIC_DEAD 0x44656144
#endif
    kuid_t      uid;        /* real UID of the task */
    kgid_t      gid;        /* real GID of the task */
    kuid_t      suid;       /* saved UID of the task */
    kgid_t      sgid;       /* saved GID of the task */
    kuid_t      euid;       /* effective UID of the task */
    kgid_t      egid;       /* effective GID of the task */
    kuid_t      fsuid;      /* UID for VFS ops */
    kgid_t      fsgid;      /* GID for VFS ops */
    unsigned    securebits; /* SUID-less security management */
/usr/src/linux-3.7.10-gentoo/include/linux/cred.h [FORMAT=unix:utf-8]
```

http://zh.wikipedia.org/wiki/UID
真实uid为运行该进程的用户id；
有效uid用在进行权限监测时的用户id，其值通常为真实uid，如果一个可执行文件设置了suid，则该程序在执行时，其有效用户id为该文件的所属主id；
保存uid，意在该进程可有的特殊权限。
因为seteuid(*uid*)，其中的*uid*只能是有效uid或保存uid，其它值无法被正常赋予。

以参数为文件路径，依次读入并打印

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define BUFFER_SIZE 4096
int main(int argc, const char *argv[]) {
    char buf[BUFFER_SIZE];
    int idx, fd1;    // idx for index;
    printf("argc %d\n", argc);
    for (idx = 1; idx < argc; idx++) {
        fd1 = open((char *) argv[idx], O_RDONLY);
        bzero(buf, BUFFER_SIZE);
        read(fd1, buf, BUFFER_SIZE);
        close(fd1);
        printf("%s", buf);
    }
    sleep(86400);    // 将任务保活一天时间;
    return 0;
}
```

/opt/cpp/reader.cpp [FORMAT=unix:utf-8] [TYPE=CPP]

```
mail cpp # g++ reader.cpp -o reader
mail cpp # chown vpn.vpn reader
mail cpp # chmod 4511 reader
mail cpp # ls -l reader
-r-s--x--x 1 vpn vpn 8040 10月 27 04:42 reader
```

```
mail cpp # su - vpn
vpn@mail ~ $ echo 'Hello, I am vpn!' > vpn_file
vpn@mail ~ $ chmod 0400 vpn_file ; ls -l vpn_file
-r-------- 1 vpn vpn 17 10月 27 04:37 vpn_file
vpn@mail ~ $ logout
mail cpp # su - jim
jim@mail ~ $ echo 'Hello, I am jim!' > jim_file
jim@mail ~ $ chmod 0400 jim_file ; ls -l jim_file
-r-------- 1 jim jim 17 10月 27 04:38 jim_file
```

```
jim@mail ~ $ tty
/dev/pts/0
jim@mail ~ $ /opt/cpp/reader /home/vpn/vpn_file /home/jim/jim_file
argc 3
Hello, I am vpn!
```

jim_file无权被读取，因为当前reader进程有效uid为vpn用户的uid，所以无法读取。

```
mail ~ # su - vpn
vpn@mail ~ $ tty
/dev/pts/2
vpn@mail ~ $ ps o user,pid,args | grep reader
vpn        2347 /opt/cpp/reader /home/vpn/vpn_file /home/jim/jim_file
vpn        2585 grep --colour=auto reader
vpn@mail ~ $ kill 2347
```

非特权用户只能结束掉属于自己的进程。

```
jim@mail ~ $ tty
/dev/pts/0
jim@mail ~ $ /opt
argc 3
Hello, I am vpn!
終止
jim@mail ~ $
```

发现上面的程序只能读取特权用户的文件(不能读取执行程序真实用户的文件)，如何才能使程序对特权用户和自己的文件都能读取呢?请看下面修改后的程序。

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define BUFFER_SIZE 4096
int main(int argc, const char *argv[]) {
    char buf[BUFFER_SIZE];
    int idx, fd1, euid;     // idx for index;
    euid = geteuid();
    printf("argc %d\n", argc);
    for (idx = 1; idx < argc; idx++) {
        if (idx == 2) {
            seteuid(getuid());
        }
        fd1 = open((char *) argv[idx], O_RDONLY);
        bzero(buf, BUFFER_SIZE);
        read(fd1, buf, BUFFER_SIZE);
        close(fd1);
        printf("%s", buf);
    }
    seteuid(euid);
    sleep(86400);    // 将任务保活一天时间;
    return 0;
}
```

红框中为新增部分
仅为演示，所以代码写的比较死

当前进程euid为该文件所属主的uid

该文件当前所属主为vpn

该进程的真实uid为执行该进程的用户id，执行该进程的用户是jim

当idx等于2时，执行该框中代码的euid为jim的uid

```
mail cpp # g++ reader.cpp -o reader
mail cpp # chown vpn.vpn reader
mail cpp # chmod 4511 reader
mail cpp # ls -l reader
-r-s--x--x 1 vpn vpn 8198 10月 27 05:18 reader
```

```
mail cpp # su - jim
jim@mail ~ $ /opt/cpp/reader /home/vpn/vpn_file /home/jim/jim_file
argc 3
Hello, I am vpn!
Hello, I am jim!
```

和预期的一样，正常读出

```
vpn@mail ~ $ tty
/dev/pts/2
vpn@mail ~ $ ps o user,pid,args | grep reader
vpn       5027 /opt/cpp/reader /home/vpn/vpn_file /home/jim/jim_file
vpn       5322 grep --colour=auto reader
```

进程当前euid仍是特权用户的id

文件系统之目录项部分

```
struct dentry {      文件系统目录项结构体
    /* RCU lookup touched fields */
    unsigned int d_flags;        /* protected by d_lock */
    seqcount_t d_seq;        /* per dentry seqlock */
    struct hlist_bl_node d_hash;     /* lookup hash list */
    struct dentry *d_parent;     /* parent directory */
    struct qstr d_name;
    struct inode *d_inode;       /* Where the name belongs to - NULL is
                     * negative */
    unsigned char d_iname[DNAME_INLINE_LEN];     /* small names */

    /* Ref lookup also touches following */
    unsigned int d_count;        /* protected by d_lock */
    spinlock_t d_lock;       /* per dentry lock */
    const struct dentry_operations *d_op;
    struct super_block *d_sb;    /* The root of the dentry tree */
    unsigned long d_time;        /* used by d_revalidate */
    void *d_fsdata;          /* fs-specific data */

    struct list_head d_lru;      /* LRU list */
    /*
     * d_child and d_rcu can share memory
     */
    union {
        struct list_head d_child;    /* child of parent list */
        struct rcu_head d_rcu;
    } d_u;
    struct list_head d_subdirs; /* our children */
    struct hlist_node d_alias;  /* inode alias list */
};
/usr/src/linux-3.6.11-gentoo/include/linux/dcache.h [FORMAT=unix:utf-8]
```

```
struct list_head {
    struct list_head *next, *prev;
};
</linux-3.6.11-gentoo/include/linux/types.h
```

从代码看，它是个双向列表，如果把列表的头尾相接，就组成了双向循环列表（空双向列表除外）。

http://www.ibm.com/developerworks/cn/linux/kernel/l-chain/

http://blog.csdn.net/sealyao/article/details/4626875

以下权限仅对普通用户起作用，root用户不受下面限制

新创建文件或文件夹的所属主(组)为进程的有效用户(组)id

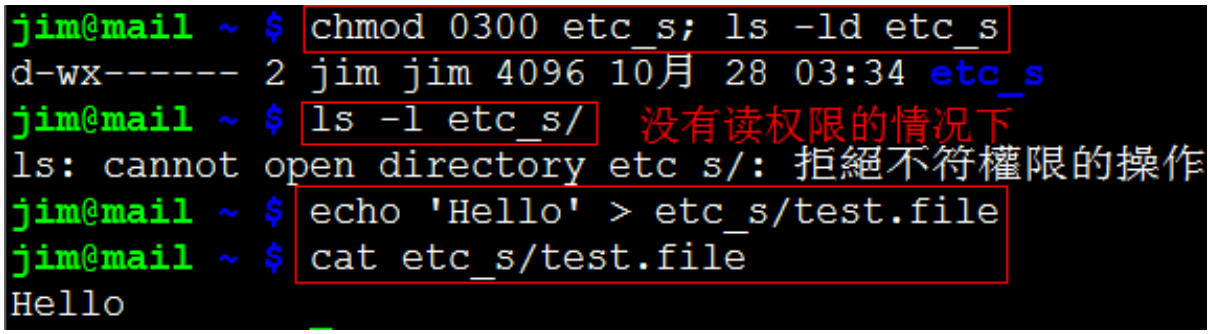
进程设置了sgid的情况下，

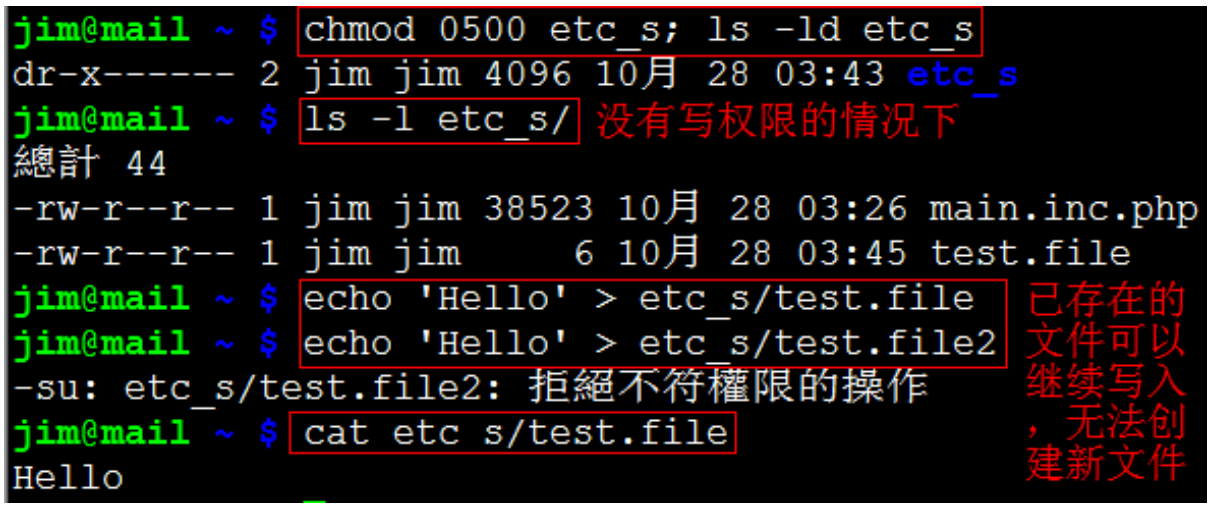
```
jim@mail ~ $ chmod 0700 etc_s; ls -ld etc_s
drwx------ 2 jim jim 4096 10月 28 03:34 etc_s
jim@mail ~ $ ls -l etc_s/
總計 40                                  用于测试的目录状态
-rw-r--r-- 1 jim jim 38523 10月 28 03:26 main.inc.php
jim@mail ~ $
```

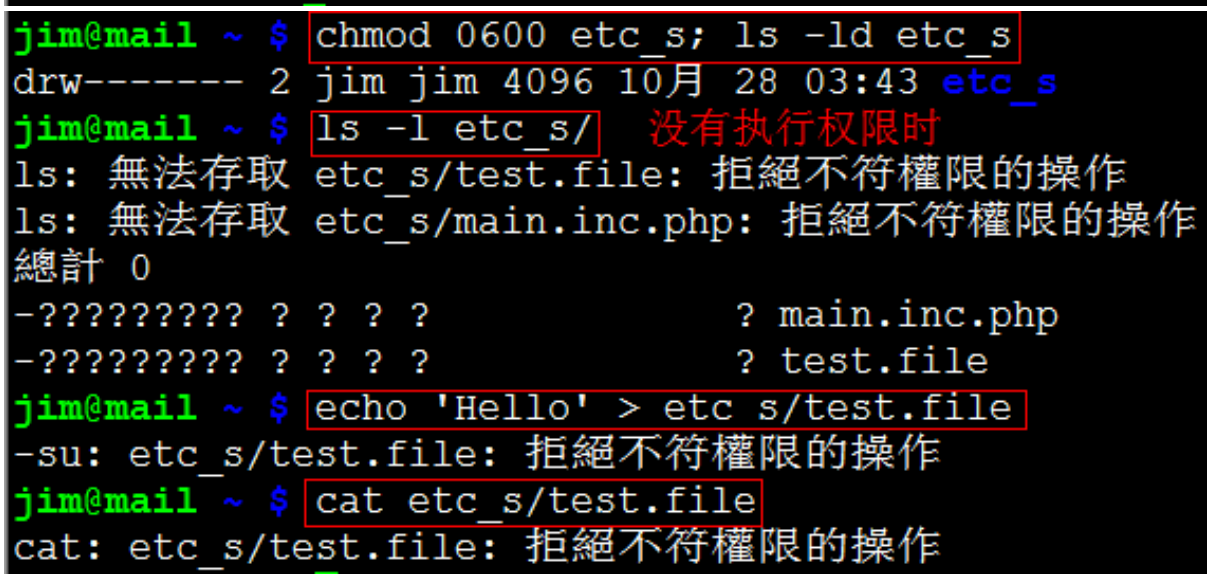
```
jim@mail ~ $ chmod 0400 etc_s; ls -ld etc_s
dr-------- 2 jim jim 4096 10月 28 03:34 etc_s
jim@mail ~ $ ls -l etc_s/    当目录只有读权限时
ls: 無法存取 etc_s/main.inc.php: 拒絕不符權限的操作
總計 0
-????????? ? ? ? ?                ? main.inc.php
jim@mail ~ $ echo 'Hello' > etc_s/test.file
-su: etc_s/test.file: 拒絕不符權限的操作
jim@mail ~ $ cat etc_s/main.inc.php
cat: etc_s/main.inc.php: 拒絕不符權限的操作
```

```
jim@mail ~ $ chmod 0200 etc_s; ls -ld etc_s
d-w------- 2 jim jim 4096 10月 28 03:34 etc_s
jim@mail ~ $ ls -l etc_s/仅有写权限时，发现什么都干不了
ls: cannot open directory etc_s/: 拒絕不符權限的操作
jim@mail ~ $ echo 'Hello' > etc_s/test.file
-su: etc_s/test.file: 拒絕不符權限的操作
jim@mail ~ $ cat etc_s/main.inc.php
cat: etc_s/main.inc.php: 拒絕不符權限的操作
```

```
jim@mail ~ $ chmod 0100 etc_s; ls -ld etc_s
d--x------ 2 jim jim 4096 10月 28 03:34 etc_s
jim@mail ~ $ ls -l etc_s/   仅有执行权限时
ls: cannot open directory etc_s/: 拒絕不符權限的操作
jim@mail ~ $ echo 'Hello' > etc_s/test.file
-su: etc_s/test.file: 拒絕不符權限的操作
jim@mail ~ $ cat etc_s/main.inc.php
<?php    发现可以读目录中文件的数据
```







http://users.sosdg.org/~qiyong/mxr/source/sys/sys/stat.h#L214

```
#define __S_IFMT    0170000 /* These bits determine file type.  */
                                    文件类型的定义
/* File types.  */
#define __S_IFDIR   0040000 /* Directory.  */
#define __S_IFCHR   0020000 /* Character device.  */
#define __S_IFBLK   0060000 /* Block device.  */
#define __S_IFREG   0100000 /* Regular file.  */
#define __S_IFIFO   0010000 /* FIFO.  */
#define __S_IFLNK   0120000 /* Symbolic link.  */
#define __S_IFSOCK  0140000 /* Socket.  */

/* POSIX.1b objects.  Note that these macros always evaluate to zero.  But
   they do it by enforcing the correct use of the macros.  */
#define __S_TYPEISMQ(buf)   ((buf)->st_mode - (buf)->st_mode)
#define __S_TYPEISSEM(buf)  ((buf)->st_mode - (buf)->st_mode)
#define __S_TYPEISSHM(buf)  ((buf)->st_mode - (buf)->st_mode)

/* Protection bits.  */                         基础权限的定义

#define __S_ISUID   04000   /* Set user ID on execution.  */
#define __S_ISGID   02000   /* Set group ID on execution.  */
#define __S_ISVTX   01000   /* Save swapped text after use (sticky).  */
#define __S_IREAD   0400    /* Read by owner.  */
#define __S_IWRITE  0200    /* Write by owner.  */
#define __S_IEXEC   0100    /* Execute by owner.  */
/usr/include/bits/stat.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=192
```

```
#define S_IRUSR __S_IREAD   /* Read by owner.  */
#define S_IWUSR __S_IWRITE  /* Write by owner.  */
#define S_IXUSR __S_IEXEC   /* Execute by owner.  */
/* Read, write, and execute by owner.  */
#define S_IRWXU (__S_IREAD|__S_IWRITE|__S_IEXEC)

#if defined __USE_MISC && defined __USE_BSD
# define S_IREAD    S_IRUSR
# define S_IWRITE   S_IWUSR
# define S_IEXEC    S_IXUSR
#endif

#define S_IRGRP (S_IRUSR >> 3)  /* Read by group.  */
#define S_IWGRP (S_IWUSR >> 3)  /* Write by group.  */
#define S_IXGRP (S_IXUSR >> 3)  /* Execute by group.  */
/* Read, write, and execute by group.  */
#define S_IRWXG (S_IRWXU >> 3)

#define S_IROTH (S_IRGRP >> 3)  /* Read by others.  */
#define S_IWOTH (S_IWGRP >> 3)  /* Write by others.  */
#define S_IXOTH (S_IXGRP >> 3)  /* Execute by others.  */
/* Read, write, and execute by others.  */
#define S_IRWXO (S_IRWXG >> 3)
/usr/include/sys/stat.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=185
```

引用定义

集合读、写、执行权限

通过位操作产生新定义

```
MKDIR(2)                          Linux Programmer's Manual                          MKDIR(2)



NAME
       mkdir - create a directory

SYNOPSIS
       #include <sys/stat.h>
       #include <sys/types.h>

       int mkdir(const char *pathname, mode_t mode);

DESCRIPTION
       mkdir() attempts to create a directory named pathname.

       The  argument  mode  specifies  the permissions to use.  It is modified by the process's umask in the
       usual way: the permissions of the created directory are (mode & ~umask & 0777).  Other mode  bits  of
       the created directory depend on the operating system.  For Linux, see below.
```

```
See NOTES below for details of glibc extensions for mode.
```

man fopen

```
Any created files will have mode S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH (0666), as
modified by the process's umask value (see umask(2)).
```

系统调用函数映射

```
0   common  read            sys_read
1   common  write           sys_write
2   common  open            sys_open
3   common  close           sys_close
4   common  stat            sys_newstat
5   common  fstat           sys_newfstat
6   common  lstat           sys_newlstat
7   common  poll            sys_poll
8   common  lseek           sys_lseek
9   common  mmap            sys_mmap
10  common  mprotect        sys_mprotect
11  common  munmap          sys_munmap
12  common  brk         sys_brk
13  64  rt_sigaction        sys_rt_sigaction
14  common  rt_sigprocmask      sys_rt_sigprocmask
15  64  rt_sigreturn        stub_rt_sigreturn
16  64  ioctl           sys_ioctl
/usr/src/linux-3.6.11-gentoo/arch/x86/syscalls/syscall_64.tbl
```

http://blog.csdn.net/xdsoft365/article/details/5911596

```
#else
#define SYSCALL_DEFINEx(x, sname, ...)                \
    __SYSCALL_DEFINEx(x, sname, __VA_ARGS__)
#endif

#ifdef CONFIG_HAVE_SYSCALL_WRAPPERS

#define SYSCALL_DEFINE(name) static inline long SYSC_##name

#define __SYSCALL_DEFINEx(x, name, ...)                    \
    asmlinkage long sys##name(__SC_DECL##x(__VA_ARGS__));        \
    static inline long SYSC##name(__SC_DECL##x(__VA_ARGS__));    \
    asmlinkage long SyS##name(__SC_LONG##x(__VA_ARGS__))         \
    {                                   \
        __SC_TEST##x(__VA_ARGS__);                \
        return (long) SYSC##name(__SC_CAST##x(__VA_ARGS__));     \
    }                                   \
    SYSCALL_ALIAS(sys##name, SyS##name);                 \
    static inline long SYSC##name(__SC_DECL##x(__VA_ARGS__))

#else /* CONFIG_HAVE_SYSCALL_WRAPPERS */

#define SYSCALL_DEFINE(name) asmlinkage long sys_##name
#define __SYSCALL_DEFINEx(x, name, ...)                    \
    asmlinkage long sys##name(__SC_DECL##x(__VA_ARGS__))
/usr/src/linux-3.6.11-gentoo/include/linux/syscalls.h [FORMAT=unix:
```

```
#define SYSCALL_DEFINE1(name, ...) SYSCALL_DEFINEx(1, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE2(name, ...) SYSCALL_DEFINEx(2, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE3(name, ...) SYSCALL_DEFINEx(3, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE4(name, ...) SYSCALL_DEFINEx(4, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE5(name, ...) SYSCALL_DEFINEx(5, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE6(name, ...) SYSCALL_DEFINEx(6, _##name, __VA_ARGS__)
/usr/src/linux-3.6.11-gentoo/include/linux/syscalls.h [FORMAT=unix:utf-8] [
```

```
由上得知，该宏展开后为 sys_open(const char __user * filename, int flags, umode_t mode)
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
    long ret;                                gcc中，如果变量不赋初值，则该变量为随机值

    if (force_o_largefile())
        flags |= O_LARGEFILE;

    ret = do_sys_open(AT_FDCWD, filename, flags, mode);
    /* avoid REGPARM breakage on x86: */
    asmlinkage_protect(3, ret, filename, flags, mode);
    return ret;
}
/usr/src/linux-3.6.11-gentoo/fs/open.c [FORMAT=unix:utf-8] [TYPE=C] [COL=012] [ROW=961
```

```
static inline int build_open_flags(int flags, umode_t mode, struct open_flags *op)
{
    int lookup_flags = 0;
    int acc_mode;

    if (flags & O_CREAT)
        op->mode = (mode & S_IALLUGO) | S_IFREG;
    else
        op->mode = 0;
/usr/src/linux-3.6.11-gentoo/fs/open.c [FORMAT=unix:utf-8] [TYPE=C] [COL=019] [ROW=840
```

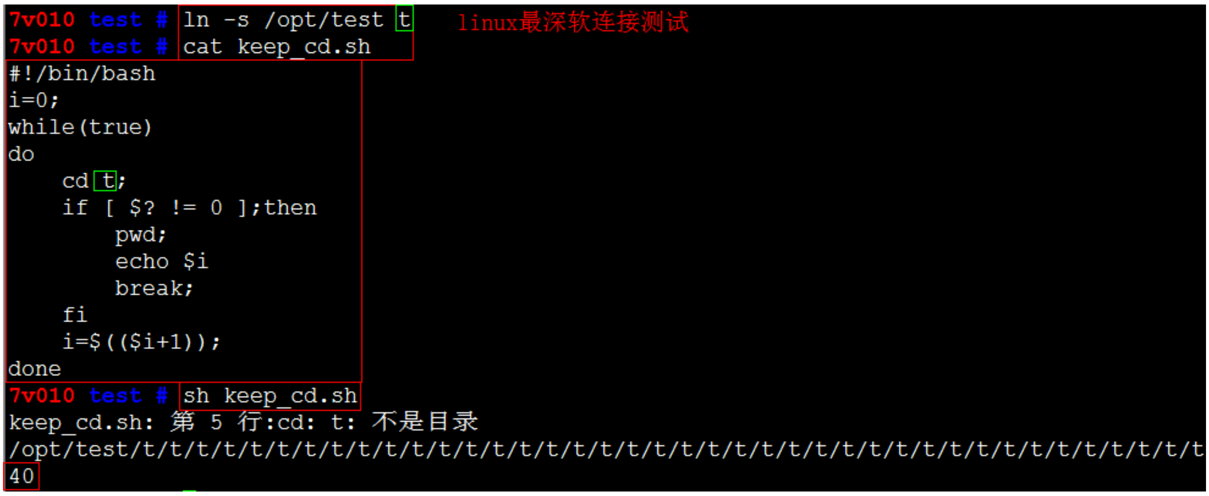
```
#include <stdio.h>       // For fopen, fclose;
#include <unistd.h>      // For close;
#include <fcntl.h>       // For open;
int main(int argc, const char *argv[]) {
    int fd1;
    fd1 = open((char *) argv[1], O_CREAT | O_WRONLY);
    close(fd1);

    FILE *fp2 = fopen((char *) argv[2], "a");
    fclose(fp2);
    return 0;
}    测试POSIX、ISO创建文件的默认权限
```

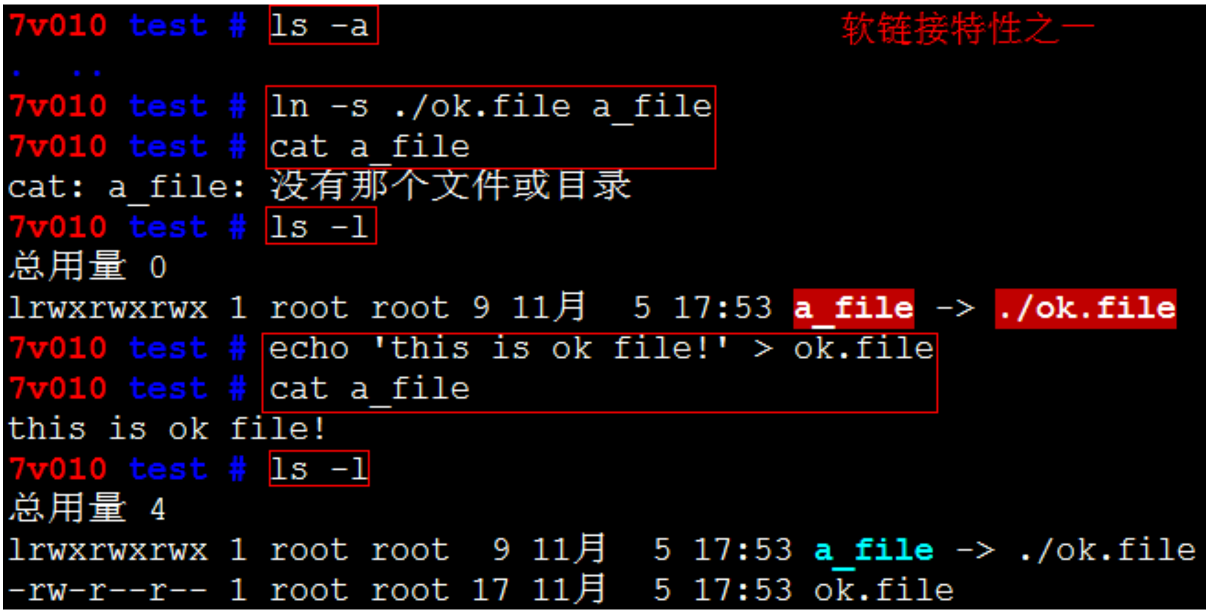
```
mail cpp # g++ test_posix\&iso_mode.cpp -o test_posix\&iso_mode
mail cpp # umask -S
u=rwx,g=rx,o=rx
mail cpp # ./test_posix\&iso_mode file1 file2
mail cpp # touch file3
mail cpp # ls -l file*
-rws--s--T 1 root root 0 11月  3 04:42 file1 权限为随机值
-rw-r--r-- 1 root root 0 11月  3 04:42 file2
-rw-r--r-- 1 root root 0 11月  3 04:42 file3
```

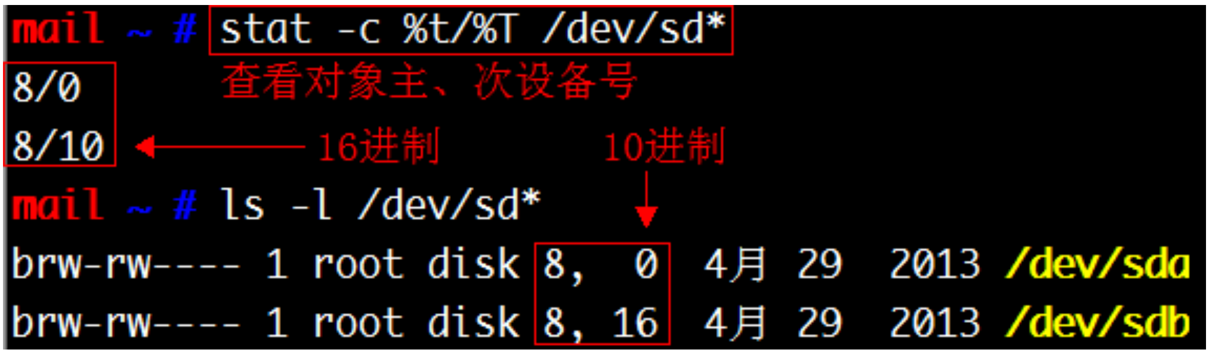
关于文件系统，有如下几个概念需清楚
扇区、块、inode、超级块、分区、MBR
dumpe2fs /dev/vg01/lv_root
https://ext4.wiki.kernel.org/index.php/Main_Page
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout
http://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/

每个分区都维护着本区的inode，硬链接指向的是目标文件的inode，而不同的分区会有着相同的inode号。所以硬链接不能跨分

区。软连接记录的是一个路径，而路径相对于一个系统来说是唯一的，它并不关心目标节点存放在哪个分区上。
http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard
硬链接的参考点是分区，软连接的参考点是FHS；







习题：

4-1、lstat与stat的唯一区别为，是否去追随链接目标。lstat不追随，stat反之。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, const char *argv[]) {
    int32_t i;
    struct stat buf;
    for(i = 1; i < argc; i++) {
        if (stat(argv[i], &buf) < 0) {
            printf("Stat error!");
            continue;
        }
        printf("%s", argv[i]);
        if (S_ISREG(buf.st_mode)) {
            printf(" %s\n", "is regular!");
        } else if (S_ISLNK(buf.st_mode)) {
            printf(" %s\n", "is symbolic link!");
        }
    }
    return 0;
}
/opt/cpps/4-1.cpp [FORMAT=unix:utf-8] [TYPE=CPP]
```

```
client cpps # ./4-1 4-1.cpp l_4-1.cpp
4-1.cpp is regular!
l_4-1.cpp is regular!   <-- 表明stat函数会追随链接到目标文件
client cpps # ls -l *4-1.cpp
-rw-r--r-- 1 root root 515 11月  9 11:26 4-1.cpp
lrwxrwxrwx 1 root root   7 11月  9 11:27 l_4-1.cpp -> 4-1.cpp
```

4-2、参看该书77页内核对文件访问权限的测试规则。

```
jim@client ~ $ umask 777; umask          普通用户jim的一系列操作
0777
jim@client ~ $ echo 'Hello!' > a_file; \
> ls -l a_file; \
> cat a_file; \
> rm -f a_file; \
> ls -a
---------- 1 jim jim 7 11月  9 11:40 a_file
cat: a_file: 权限不够
.  ..  .bash_logout  .bash_profile  .bashrc  .ssh  .vimrc
```

```
client ~ # umask 777; umask          超级用户的一系列操作
0777
client ~ # echo 'Hello!' > a_file; \
> ls -l a_file; \
> cat a_file; \
> rm -f a_file; \
> ls -a
---------- 1 root root 7 11月  9 12:54 a_file
Hello!          <-------与普通用户的区别
.  ..  .bash_history  .gitconfig  .ssh  .vim  .vimrc
```

4-3、上图已证，root用户除外；

4-4、只有时间会变，文件以新创建的形式产生。

```
man creat;open的选项表示目标不存在时则创建，并且以只写模式，文件截短至0打开。
creat() is equivalent to open() with flags equal to O_CREATIO_WRONLYIO_TRUNC.
```

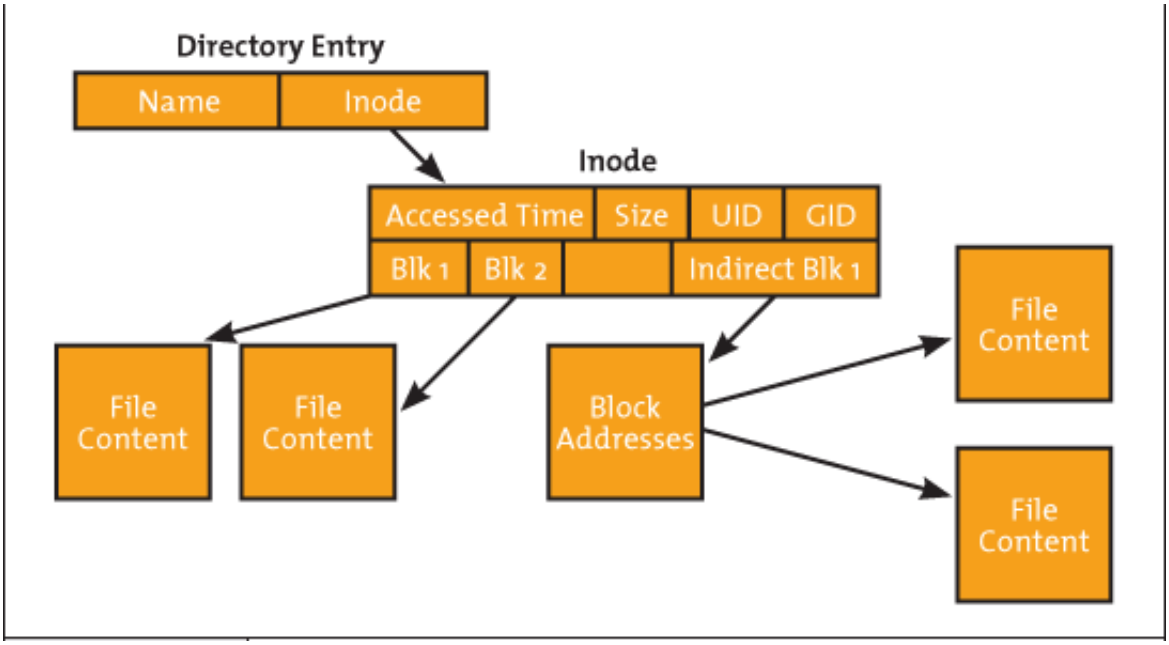
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define RWRWRW (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)
int main(int argc, const char *argv[]) {
    umask(0);
    if (creat(argv[1], RWRWRW) < 0) {
        printf("%s %s %s", "Create", argv[1] ,"error!");
    }
    umask(S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
    if (creat(argv[2], RWRWRW) < 0) {
        printf("%s %s %s", "Create", argv[2] ,"error!");
    }
    return 0;
}
```

/opt/cpps/4-4.cpp [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=016

```
client cpps # ./4-4 foo bar; ls -l foo bar; \
>             sleep 60; \
>             ./4-4 foo bar; ls -l foo bar
-rw------- 1 root root 0 11月  9 13:36 bar
-rw-rw-rw- 1 root root 0 11月  9 13:36 foo
-rw------- 1 root root 0 11月  9 13:37 bar
-rw-rw-rw- 1 root root 0 11月  9 13:37 foo
```

4-5、下图给出了详细的数据格式，按照该格式，可以组建任意想要的文件组成。但是在链接文件和目录大小为零的情况下，对系统而言，是没意义的，也是不可用的，站在系统的角度去看，应该是不可以为0。通过系统提供的命令，是无法创建大小为0的目录和链接文件;

http://users.nccs.gov/~fwang2/linux/lk_debugfs.html
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Linear_.28Classic.29_Directories



Offset	Size	Name	Description
0x0	__le32	inode	Number of the inode that this directory entry points to.
0x4	__le16	rec_len	Length of this directory entry.
0x6	__u8	name_len	Length of the file name.
0x7	__u8	file_type	File type code, one of: 0x0 Unknown. 0x1 Regular file. 0x2 Directory. 0x3 Character device file. 0x4 Block device file. 0x5 FIFO. 0x6 Socket. 0x7 Symbolic link.
0x8	char	name[EXT4_NAME_LEN]	File name.

```
mail opt # mkdir -p dir_test/' '; \
>          touch dir_test/FFFFFFFFFF; \
>          ls -laF dir_test
總計 12                    生成测试目录
drwxr-xr-x  2 root root 4096 11月 10 05:34  /
drwxr-xr-x  3 root root 4096 11月 10 05:34 ./
drwxr-xr-x 13 root root 4096 11月 10 05:34 ../
-rw-r--r--  1 root root    0 11月 10 05:34 FFFFFFFFFF
```

```
mail opt # debugfs                    使用该工具配合演示
debugfs 1.42 (29-Nov-2011)
debugfs:  open /dev/vg01/lv_root
debugfs:  stat /opt/dir_test
Inode: 574808   Type: directory    Mode:  0755   Flags: 0x80000
Generation: 214802061    Version: 0x00000000:00000003
User:     0   Group:     0   Size: 4096
File ACL: 0    Directory ACL: 0
Links: 3   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x527eaa76:e74f83b4 -- Sun Nov 10 05:34:46 2013
 atime: 0x527eaa76:e74f83b4 -- Sun Nov 10 05:34:46 2013
 mtime: 0x527eaa76:e74f83b4 -- Sun Nov 10 05:34:46 2013
crtime: 0x527eaa76:e74f83b4 -- Sun Nov 10 05:34:46 2013
Size of extra inode fields: 28
EXTENTS:
(0):2103972  <-- 该对象的实际内容，在文件系统blocks中的位置
lines 1-13/13 (END)
mail opt # dd if=/dev/vg01/lv_root of=./dir_block.raw bs=4096 skip=2103972 count=1
1+0 records in
1+0 records out
4096 bytes (4.1 kB) copied, 0.000262827 s, 15.6 MB/s   该分区文件系统block的大小
mail opt # hexdump -C dir_block.raw                    宽字符串结束符
00000000  58 c5 08 00 0c 00 01 02  2e 00 00 00 ec 00 06 00  |X...............|
00000010  0c 00 02 02 2e 2e 00 00  59 c5 08 00 0c 00 01 02  |........Y.......|
00000020  20 00 00 00 5a c5 08 00  dc 0f 0a 01 46 46 46 46  | ...Z.......FFFF|
00000030  46 46 46 46 46 46 00 00  00 00 00 00 00 00 00 00  |FFFFFF..........|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*                 文件名长度  文件类型  文件名
00001000                目录项长度
mail opt # echo $((16#08c558)); echo $((16#0c))
574808   <-- inode
12
mail opt # stat -c %i dir_test{/,/.}
574808
574808
mail opt #
```

4-6、如下；

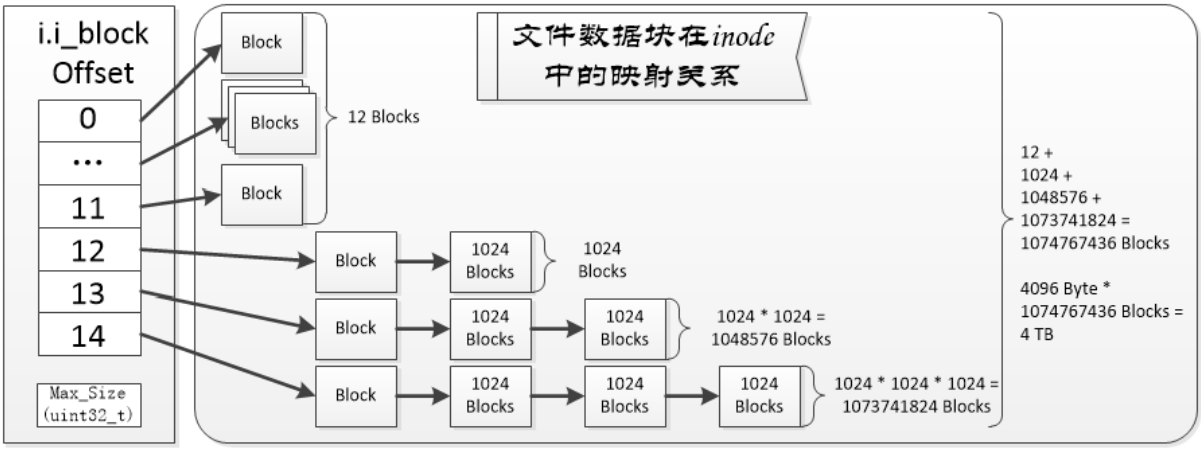
File System Maximums

	32-bit mode			
Item	**1KiB**	**2KiB**	**4KiB**	**64KiB**
Blocks	2^32	2^32	2^32	2^32
Inodes	2^32	2^32	2^32	2^32
File System Size	4TiB	8TiB	16TiB	256PiB
Blocks Per Block Group	8,192	16,384	32,768	524,288
Inodes Per Block Group	8,192	16,384	32,768	524,288
Block Group Size	8MiB	32MiB	128MiB	32GiB
Blocks Per File, Extents	2^32	2^32	2^32	2^32
Blocks Per File, Block Maps	16,843,020	134,480,396	1,074,791,436	4,398,314,962,956
File Size, Extents	4TiB	8TiB	16TiB	256TiB
File Size, Block Maps	16GiB	256GiB	4TiB	256PiB

0x28	60 bytes	i_block[EXT4_N_BLOCKS=15]	Block map or extent tree. See the section "The Contents of inode.i_block".

i.i_block Offset	**Where It Points**
0 to 11	Direct map to file blocks 0 to 11.
12	Indirect block: (file blocks 12 to ($block_size / 4) + 11, or 12 to 1035 if 4KiB blocks) **Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Direct map to ($block_size / 4) blocks (1024 if 4KiB blocks)
13	Double-indirect block: (file blocks $block_size/4 + 12 to ($block_size / 4) ^ 2 + ($block_size / 4) + 11, or 1036 to 1049611 if 4KiB blocks) **Double Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Map to ($block_size / 4) indirect blocks (1024 if 4KiB blocks) **Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Direct map to ($block_size / 4) blocks (1024 if 4KiB blocks)
14	Triple-indirect block: (file blocks ($block_size / 4) ^ 2 + ($block_size / 4) + 12 to ($block_size / 4) ^ 3 + ($block_size / 4) ^ 2 + ($block_size / 4) + 12, or 1049612 to 1074791436 if 4KiB blocks) **Triple Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Map to ($block_size / 4) double indirect blocks (1024 if 4KiB blocks) **Double Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Map to ($block_size / 4) indirect blocks (1024 if 4KiB blocks) **Indirect Block Offset** / **Where It Points** 0 to ($block_size / 4) / Direct map to ($block_size / 4) blocks (1024 if 4KiB blocks)

60/15=4Byte、8Bit = 1Byte、8Bit * 4 = 32Bit、max(uint_32_t) = 4294967296、1Block = 4096Byte、1Block * 4294967296 = 17592186044416Byte = 16TB




```
mail cpp # echo 'BBBBBBBBBB' > strB.file
mail cpp # echo 'AAAAAAAAAA' > hole.file
mail cpp # dd if=./strB.file of=./hole.file bs=4096 seek=5 count=1;
0+1 records in
0+1 records out
11 bytes (11 B) copied, 0.000255756 s, 43.0 kB/s
mail cpp # debugfs -R "stat /opt/cpp/hole.file" /dev/vg01/lv_root
```

```
Inode: 1001607   Type: regular    Mode:  0644   Flags: 0x80000
Generation: 214903385    Version: 0x00000000:00000001
User:     0   Group:     0   Size: 20491
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 16  单位为 512Byte
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x5287c304:97562560 -- Sun Nov 17 03:09:56 2013
 atime: 0x5287c2ed:97560170 -- Sun Nov 17 03:09:33 2013
 mtime: 0x5287c304:97562560 -- Sun Nov 17 03:09:56 2013
crtime: 0x5287c2ed:97560170 -- Sun Nov 17 03:09:33 2013
Size of extra inode fields: 28
EXTENTS:
(0):5041026, (5):5023744
```

```c
#include <stdio.h>          // For printf;
#include <stdlib.h>         // For exit;
#include <unistd.h>         // For read, lseek, write, close;
#include <fcntl.h>          // For open, creat;
#include <string.h>         // For bzero;
#include <sys/stat.h>       // For stat;
#include <sys/ioctl.h>      // For ioctl;
#include <linux/fs.h>       // For FIBMAP;
int main(int argc, const char *argv[]) {
    struct stat sb;
    if (stat(argv[1], &sb) == -1) {
        printf("stat error!\n");
        exit(EXIT_FAILURE);
    }
    int i, read_count, buf_size, fd1, fd2, blk_num, blocks;
    buf_size = sb.st_blksize;
    blocks = (sb.st_size + (sb.st_blksize - 1)) / sb.st_blksize;
    char buf[buf_size];获取被分配数据块的实际编号，如果所
    fd1 = open(argv[1], O_RDONLY);映射的数据块没被分配，则
    fd2 = creat(argv[2], S_IRUSR | S_IWUSR);blk_num的值为0。
    for (i = 0; i < blocks; i++) {
        blk_num = i;
        bzero(buf, buf_size);
        ioctl(fd1, FIBMAP, &blk_num);
        printf("%d\n", blk_num);
        read_count = read(fd1, buf, buf_size);
        if (0 == blk_num) {
            lseek(fd2, buf_size, SEEK_CUR);
        } else {
            write(fd2, buf, read_count);
        }
    }
    close(fd1);
    close(fd2);
    return 0;
}
/opt/cpp/4-6.cpp [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=001/36(2%)]
```

```
mail cpp # g++ 4-6.cpp -o 4-6
mail cpp # ./4-6 hole.file hole.file2
5041026
0
0
0
0
5023744
mail cpp # md5sum hole.file*
95038907059f38875cdf0196fd411361  hole.file
95038907059f38875cdf0196fd411361  hole.file2
mail cpp # debugfs -R "stat /opt/cpp/hole.file2" /dev/vg01/lv_root
Inode: 1001606   Type: regular    Mode:  0600   Flags: 0x80000
Generation: 214903529    Version: 0x00000000:00000001
User:     0   Group:     0   Size: 20491
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 16
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x5287c5da:169b9684 -- Sun Nov 17 03:22:02 2013
 atime: 0x5287c5da:169b9684 -- Sun Nov 17 03:22:02 2013
 mtime: 0x5287c5da:169b9684 -- Sun Nov 17 03:22:02 2013
crtime: 0x5287c5da:169b9684 -- Sun Nov 17 03:22:02 2013
Size of extra inode fields: 28
EXTENTS:
(0):5042244, (5):5023747
```

4-7、该题容易产生误导，考察的是shell进程的umaks，即新创建的文件由当前shell进程生成；

4-8、du必须的一个参数是文件名，unlink使要查看的文件的目录项不存在了，所以通过变通的方法df来验证；

4-9、因为unlink更改了该文件inode的链接数，由于改动了该文件的inode信息，所以会改变状态；

4-10、101页，有个dopath函数在自己的opendir与closedir之间，有条件的自我调用，当条件满足的情况下，最深能读到ulimit -n层目录；

4-11、使用chdir切到当前目录，可以省去后续读叶文件而需要的，层层目录穿过的过程；

4-12、以下是对系统coreutils手册中chroot的翻译，关于用它的集大成者，可参考Linux-VServer, FreeVPS 和 OpenVZ，以及LFS；

info coreutils 'chroot invocation'

23.1 `chroot': 在一个不同的根目录中运行一个命令

==========

`chroot' 在一个指定的根目录中运行命令。在很多系统中，只有超级用户可以运行它。(1) 摘要：

chroot OPTION NEWROOT [COMMAND [ARGS]...]

chroot OPTION

按理来说，文件名的查找起始于目录结构的根，即‘/’。`chroot' 改变根到目录NEWROOT(目标目录必须已存在)，并和可选的ARGS运行COMMAND。如果COMMAND没被指定，默认值为环境变量 `SHELL' 或`/bin/sh'(如果`SHELL'没被设置)，选项为 `-i'。

COMMAND 不能是一个特殊的内置工具(*注意 特殊的内置工具::)。

http://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_14

http://pubs.opengroup.org/onlinepubs/009696899/idx/sbi.html

- break - exit from for, while, or until loop (Special Built-in Utility)
- colon - null utility (Special Built-in Utility)
- continue - continue for, while or until loop (Special Built-in Utility)
- dot - execute commands in current environment (Special Built-in Utility)
- eval - construct command by concatenating arguments (Special Built-in Utility)
- exec - execute commands and open, close, or copy file descriptors (Special Built-in Utility)
- exit - cause the shell to exit (Special Built-in Utility)
- export - set export attribute for variables (Special Built-in Utility)
- readonly - set read-only attribute for variables (Special Built-in Utility)
- return - return from a function (Special Built-in Utility)
- set - set or unset options and positional parameters (Special Built-in Utility)
- shift - shift positional parameters (Special Built-in Utility)
- times - write process times (Special Built-in Utility)
- trap - trap signals (Special Built-in Utility)
- unset - unset values and attributes of variables and functions (Special Built-in Utility)

可用选项如下。同样需注意，命令选项::。选项必须在操作数的前面。

`--userspec=USER[:GROUP]'

默认，COMMAND使用相同的凭据作为调用进程运行。使用该选项以一个不同的USER和/或不同的主GROUP来运行。

`--groups=GROUPS'
使用该选项指定新进程中使用的候补组GROUPS。列表中的每一个(组名或组ID)必须用逗号分开。
这是一些帮助避免使用chroot常见问题的技巧。从一个简单实例开始，创建一个引用静态链接库的COMMAND。如果你已经使用了一个动态链接的可执行程序，那么你必须安排它的共享库到你新root目录的正确位置。
例，如果你创建一个静态链接的`ls' 可执行程序，并置它到`/tmp/empty'中，你可以以root身份运行该命令:
$ chroot /tmp/empty /ls -Rl /
接着你将看到类似下面的输出:
/:
total 1023
-rwxr-xr-x 1 0 0 1041745 Aug 16 11:17 ls
如果你想使用动态链接库的执行程序，谈谈`bash'，首先运行`ldd bash' 来看看它需要哪些共享对象。接着，除了复制实际的二进制程序，也复制所需的文件到你意欲作为新root目录的适当位置。最后，如果可执行程序需要任何其它文件(如，数据、状态、设备文件)，同样复制它们到适当位置。
退出状态:

125 if `chroot' itself fails
126 if COMMAND is found but cannot be invoked
127 if COMMAND cannot be found
the exit status of COMMAND otherwise

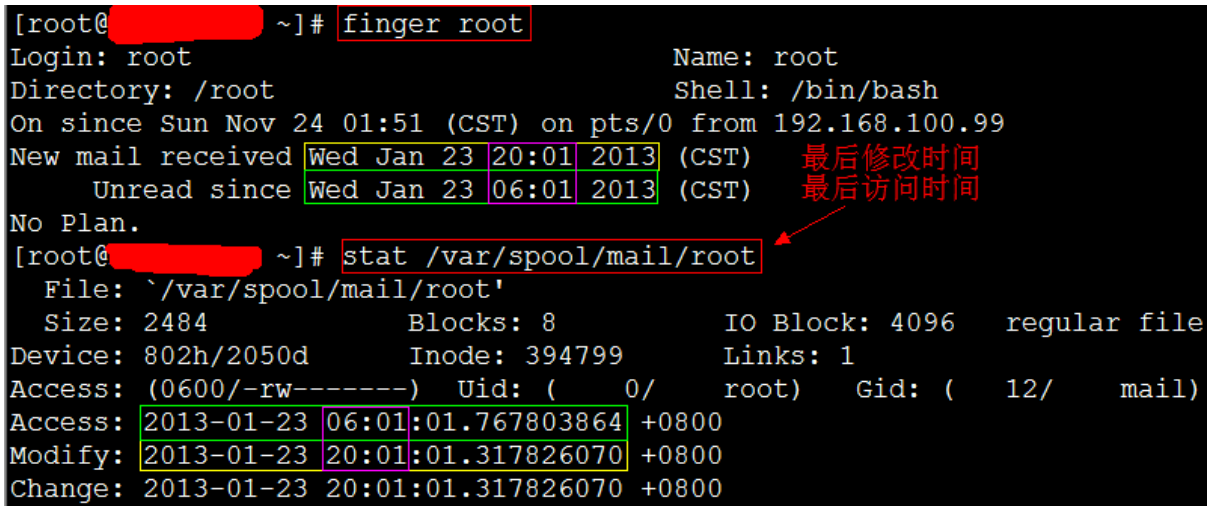
---------- 脚注 ----------
(1)然而，一些系统(如，FreeBSD)可以通过配置，允许某些普通用户去使用`chroot'系统调用，因而运行此程序。同样，在Cygwin上，任何人都可以运行`chroot'命令，因为相关方法是非特权的，由于在 MS-Windows缺乏支持。
4-13、这是在考验读者的编程能力吗?

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>
#include <utime.h>
int main(int argc, const char *argv[]) {
    struct stat sb;
    struct utimbuf tb;
    stat(argv[1], &sb);
    tb.actime = sb.st_atime;
    tb.modtime = sb.st_mtime;
    if (0 == strcmp("acc", argv[2])) {
        tb.actime = atol(argv[3]);
    }
    if (0 == strcmp("mod", argv[2])) {
        tb.modtime = atol(argv[3]);
    }
    if (0 > utime(argv[1], &tb)) {
        printf("utime error!\n");
    }
    return 0;
}
```

/opt/cpp/4-13.cpp [FORMAT=unix:utf-8] [TY

```
mail cpp # g++ 4-13.cpp -o 4-13
mail cpp # stat strB.file
  File: ‘ strB.file’
  Size: 11              Blocks: 8          IO Block: 4096   普通檔案
Device: fd00h/64768d    Inode: 1001605     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2013-11-17 03:09:28.304748487 +0800   原始访问时间
Modify: 2013-11-17 03:09:28.304748487 +0800
Change: 2013-11-17 03:09:28.304748487 +0800
 Birth: -
mail cpp # ./4-13 ./strB.file acc 0           修改访问时间
mail cpp # stat strB.file
  File: ‘ strB.file’
  Size: 11              Blocks: 8          IO Block: 4096   普通檔案
Device: fd00h/64768d    Inode: 1001605     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800   修改后的访问时间
Modify: 2013-11-17 03:09:28.000000000 +0800
Change: 2013-11-24 01:41:04.794651053 +0800
 Birth: -
mail cpp # ./4-13 ./strB.file mod 100         修改改动时间
mail cpp # stat strB.file
  File: ‘ strB.file’
  Size: 11              Blocks: 8          IO Block: 4096   普通檔案
Device: fd00h/64768d    Inode: 1001605     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 08:00:00.000000000 +0800
Modify: 1970-01-01 08:01:40.000000000 +0800   修改后的改动时间
Change: 2013-11-24 01:41:14.844652058 +0800
 Birth: -
```

4-14、需了解一下系统的邮件系统部分；



4-15、参考信息info cpio, info tar；

cpio与tar均记录了文件的最后修改时间；

默认cpio恢复出来的最后修改时间为当前时间，tar解压出来的最后时间为原始时间，cpio使用-m参数，可以使恢复出来的最后修改时间为原始修改时间；

最后访问时间，默认均为当前时间，cpio使用-m参数后，最后访问时间与最后修改时间相同；

关于本题的最后‘为什么？’，应该问的是对表4-11中creat和utime的理解；

```
struct header_old_cpio {  man 5 cpio
        unsigned short   c_magic;
        unsigned short   c_dev;
        unsigned short   c_ino;
        unsigned short   c_mode;
        unsigned short   c_uid;
        unsigned short   c_gid;
        unsigned short   c_nlink;
        unsigned short   c_rdev;
        unsigned short   c_mtime[2];
        unsigned short   c_namesize;
        unsigned short   c_filesize[2];
};
```

```
struct header_old_tar {  man 5 tar
        char name[100];
        char mode[8];
        char uid[8];
        char gid[8];
        char size[12];
        char mtime[12];
        char checksum[8];
        char linkflag[1];
        char linkname[100];
        char pad[255];
};
```

cpio部分源码

```
void
set_file_times (int fd,                                    设置文件的时间
        const char *name, unsigned long atime, unsigned long mtime)
/opt/cpp/cpio-2.11/src/util.c [FORMAT=unix:utf-8] [TYPE=C] [COL=005]
```

```
  if (retain_time_flag)如果使用了-m参数，恢复出来的文件，其访问时间
    set_file_times (-1, file_hdr->c_name, file_hdr->c_mtime,
            file_hdr->c_mtime);  也变为了原始修改时间
/opt/cpp/cpio-2.11/src/copyin.c [FORMAT=unix:utf-8] [TYPE=C]
```

4-16、有对2.5.5的温习；可以通过tar和cpio对目录进行归档；

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#define BUF_SIZE 1024 * 4
int main(int argc, const char *argv[]) {
    int i = 0;
    char buf[BUF_SIZE];
    while(true) {
        if(0 > mkdir("./d", S_IRWXU)) {
            printf("mkdir error!\n");
            break;
        }
        if(0 > chdir("./d")) {
            printf("chdir error!\n");
            break;
        }
        if (0 < argc) {
            if (0 == strcmp("getcwd", argv[1])) {
                if (NULL == getcwd(buf, BUF_SIZE)) {
                    printf("getcwd error!\n");
```

```
                    break;
                }
                printf("cur_dir %s\n", buf);
            } else if (0 == strcmp("count", argv[1])) {
                if (i == atoi(argv[2])) {
                    break;
                }
            }
        }
        i++;
        printf("dir_dep %d\n", i);
    }
    return 0;
}
/opt/cpps/4-17.cpp [FORMAT=unix:utf-8] [TYPE=CPP] [COL=
```

```
localhost cpps # g++ 4-17.cpp -o 4-17
```

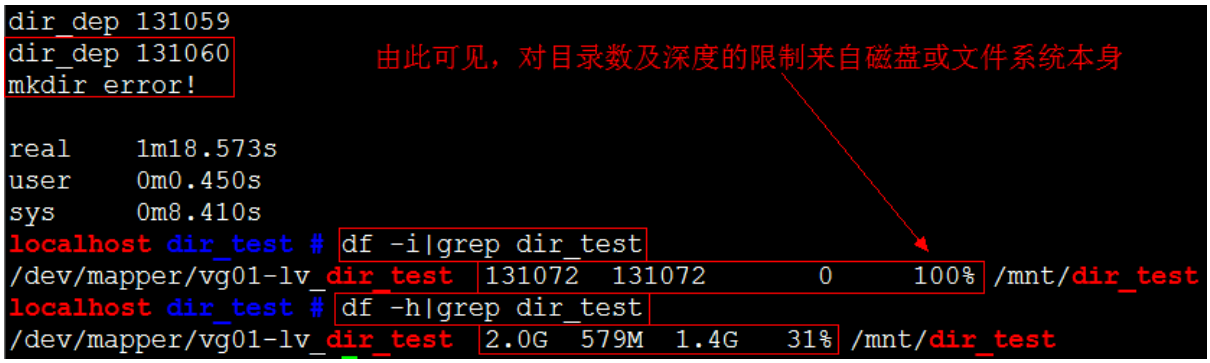
```
localhost cpps # lvcreate -L 2G -n lv_dir_test vg01
  Logical volume "lv_dir_test" created
localhost cpps # mkdir /mnt/dir_test
localhost cpps # mkfs.ext4 /dev/vg01/lv_dir_test
```

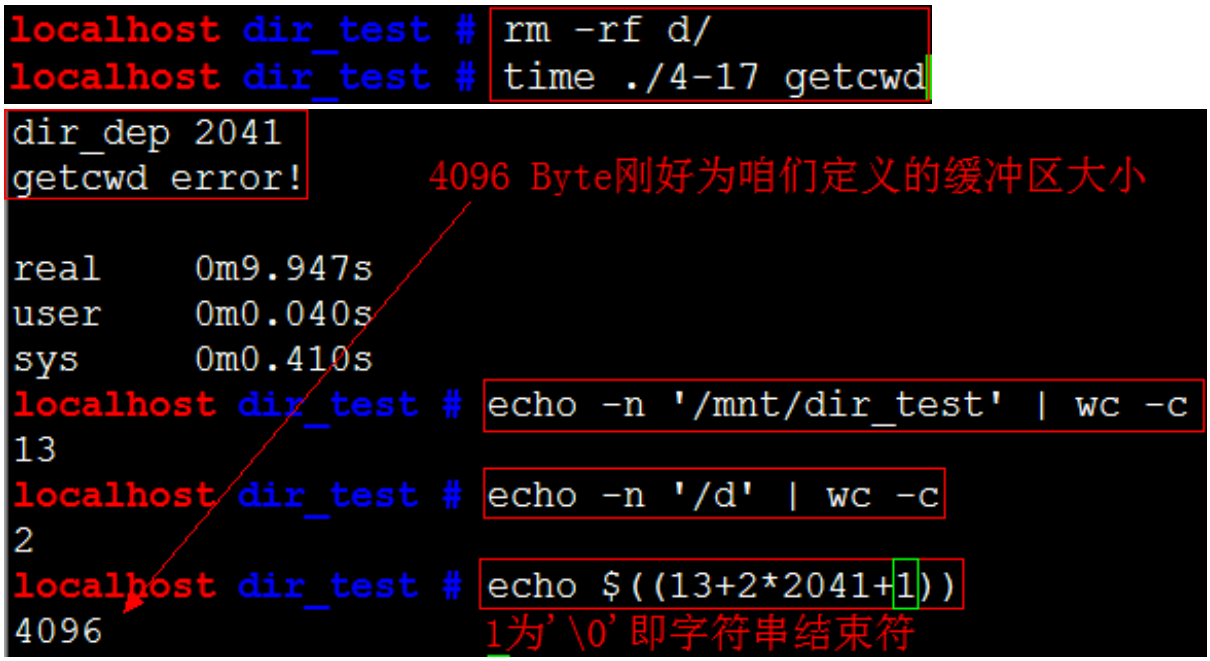
```
localhost cpps # mount /dev/vg01/lv_dir_test /mnt/dir_test/
localhost cpps # mv 4-17 /mnt/dir_test/
localhost cpps # cd /mnt/dir_test/
```

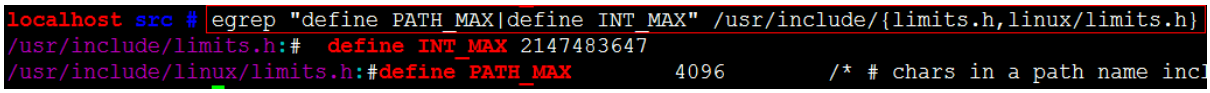
```
localhost dir_test # mv /opt/cpps/4-17 .
```

```
localhost dir_test # df -i|grep dir_test
/dev/mapper/vg01-lv_dir_test  131072      12  131060       1% /mnt/dir_test
localhost dir_test # time ./4-17
```

```
dir_dep 131059
dir_dep 131060
mkdir error!

real    1m18.573s
user    0m0.450s
sys     0m8.410s
localhost dir_test # df -i|grep dir_test
/dev/mapper/vg01-lv_dir_test  131072  131072       0     100% /mnt/dir_test
```


dir_dep 131059
dir_dep 131060
mkdir error!
由此可见，对目录数及深度的限制来自磁盘或文件系统本身
real 1m18.573s
user 0m0.450s
sys 0m8.410s
localhost dir_test # df -i|grep dir_test
/dev/mapper/vg01-lv_dir_test 131072 131072 0 100% /mnt/dir_test
localhost dir_test # df -h|grep dir_test
/dev/mapper/vg01-lv_dir_test 2.0G 579M 1.4G 31% /mnt/dir_test


localhost dir_test # rm -rf d/
localhost dir_test # time ./4-17 getcwd
dir_dep 2041
getcwd error!
4096 Byte刚好为咱们定义的缓冲区大小
real 0m9.947s
user 0m0.040s
sys 0m0.410s
localhost dir_test # echo -n '/mnt/dir_test' | wc -c
13
localhost dir_test # echo -n '/d' | wc -c
2
localhost dir_test # echo $((13+2*2041+1))
4096
1为'\0'即字符串结束符


localhost src # egrep "define PATH_MAX|define INT_MAX" /usr/include/{limits.h,linux/limits.h}
/usr/include/limits.h:# define INT_MAX 2147483647
/usr/include/linux/limits.h:#define PATH_MAX 4096 /* # chars in a path name incl

pwd的部分源码

```c
static void
file_name_free (struct file_name *p)
{
  free (p->buf);
  free (p);
}

static struct file_name *
file_name_init (void)
{
  struct file_name *p = xmalloc (sizeof *p);

  /* Start with a buffer larger than PATH_MAX, but beware of systems
     on which PATH_MAX is very large -- e.g., INT_MAX.  */
  p->n_alloc = MIN (2 * PATH_MAX, 32 * 1024);
                    取其两个数中的较小者
  p->buf = xmalloc (p->n_alloc);开辟缓冲区空间
  p->start = p->buf + (p->n_alloc - 1);
  p->start[0] = '\0';
  return p;
}
/opt/test/coreutils-8.20/src/pwd.c [FORMAT=unix:utf-8] [TYPE=C] [COL=
```

```c
static void
file_name_prepend (struct file_name *p, char const *s, size_t s_len)
{                                    s，欲前塞的文件名
  size_t n_free = p->start - p->buf;  s_len，s的长度
  if (n_free < 1 + s_len)  1是欲存放路径中文件名间的分隔符'/'的大小
    {
      size_t half = p->n_alloc + 1 + s_len;
      /* Use xnmalloc+free rather than xnrealloc, since with the latter
         we'd end up copying the data twice: once via realloc, then again
         to align it with the end of the new buffer.  With xnmalloc, we
         copy it only once.  */
      char *q = xnmalloc (2, half); 开辟两倍half大小的空间
      size_t n_used = p->n_alloc - n_free;
      p->start = q + 2 * half - n_used;
      memcpy (p->start, p->buf + n_free, n_used);
      free (p->buf);
      p->buf = q;
      p->n_alloc = 2 * half;
    }

  p->start -= 1 + s_len;
  p->start[0] = '/';
  memcpy (p->start + 1, s, s_len);
}
```

4-17、该题综合考察了对3.10、3.16、4.15等章节的理解，尤其需注意89页倒数第二段，关于进程打开文件对unlink的影响；

```
localhost cpps # ls -l /dev/fd/1
lrwx------ 1 root root 64 12月  2 11:23 /dev/fd/1 -> /dev/pts/1
localhost cpps # echo $$    返回当前进程的ID号
2218
localhost cpps # ls -l /proc/2218/fd/1  查看当前进程打开文件描述符为1的状态
lrwx------ 1 root root 64 12月  2 11:19 /proc/2218/fd/1 -> /dev/pts/1
localhost cpps # lsof /dev/pts/1      查看该文件被哪些进程所占用
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash     2218 root    0u   CHR  136,1      0t0    4 /dev/pts/1  该文件由本shell所产生
bash     2218 root    1u   CHR  136,1      0t0    4 /dev/pts/1
bash     2218 root    2u   CHR  136,1      0t0    4 /dev/pts/1
bash     2218 root  255u   CHR  136,1      0t0    4 /dev/pts/1
lsof     2232 root    0u   CHR  136,1      0t0    4 /dev/pts/1
lsof     2232 root    1u   CHR  136,1      0t0    4 /dev/pts/1
lsof     2232 root    2u   CHR  136,1      0t0    4 /dev/pts/1
```

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
int main(int argc, const char *argv[]) {
    int fd;
    if (argc < 1) {
        printf("Need an arguments!");
        return 1;
    }
    if (unlink(argv[1]) < 0) {
        printf("Unlink error!\n");
    }
    if ((fd = creat(argv[1], S_IRUSR | S_IWUSR)) < 0) {
        printf("Creat error!");
        return 1;
    }
    if (write(fd, "Hello Jim!\n", sizeof("Hello Jim!\n")) < 0) {
        printf("Write error!");
        return 1;
    }
    return 0;
}
/opt/cpps/4-18.cpp [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=
```

```
localhost cpps # g++ 4-18.cpp -o 4-18
localhost cpps # ./4-18 /dev/fd/1
Unlink error!
Hello Jim!
```