Chapter 4.
Files and Directories

System Programming

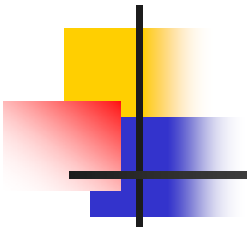
http://www.cs.ccu.edu.tw/~pahsiung/courses/sp

熊博安

國立中正大學資訊工程學系

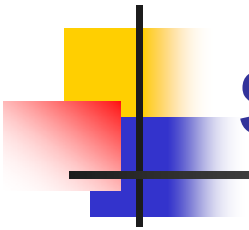
pahsiung@cs.ccu.edu.tw Class: EA-104

(05)2720411 ext. 33119 Office: EA-512



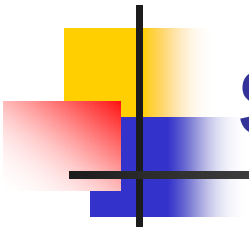
Introduction

- Features of filesystem
- Properties of a file
- stat()
- fstat()
- lstat()
- Symbolic links
- Descending directory hierarchy



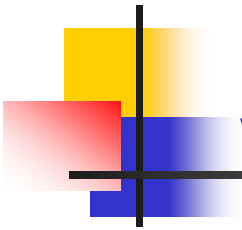
stat(), fstat(), lstat()

- #include <sys/stat.h>
- int stat(const char *pathname,
  struct stat *buf);
- int fstat(int filedes, struct stat *buf);
- int lstat(const char *pathname,
  struct stat *buf);
- Return: 0 if OK, -1 on error



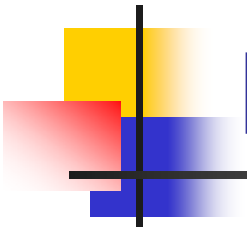
stat(), fstat(), lstat()

- stat() returns a structure of information about a named file
- fstat() returns information about a file that is already open on descriptor *filedes*
- lstat() returns information about symbolic link



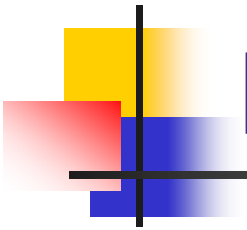
struct stat

```
struct stat {
    mode_t st_mode; /* file type & mode */
    ino_t st_ino;   /* i-node # */
    dev_t st_dev;   /* device # */
    dev_t st_rdev;  /* device # for special file */
    nlink_t st_nlink; /* # of links */
    uid_t st_uid;   /* owner UID (user ID) */
    gid_t st_gid;   /* owner GID (group ID) */
    off_t st_size;  /* size in bytes, for regular files */
    time_t st_atime; /* time of last access */
    time_t st_mtime; /* time of last modification */
    time_t st_ctime; /* time of last file status change */
    long st_blksize; /* best I/O block size */
    long st_blocks;  /* # disk blocks allocated */
};
```



File Types

- **Regular file**: contains data, interpretation left to application program
- **Directory file**: contains names of other files and pointers to information on these files
- **Character special file**: unbuffered I/O, used for certain types of devices, such as terminal



File Types

- **Block special file**: buffered I/O, used for disk devices
- **FIFO**: used for interprocess communication, named pipe (Sec. 15.5)
- **Socket**: used for network communication (Ch. 16)
- **Symbolic link**: points to another file (Sec. 4.16)

<sys/stat.h>

Argument: stat.st_mode

Macros to detect file types

Macro	Type of file
S_ISREG()	Regular file
S_ISDIR()	Directory file
S_ISCHR()	Character special file
S_ISBLK()	Block special file
S_ISFIFO()	Pipe or FIFO
S_ISLNK()	Symbolic link (not in POSIX.1 or SVR4)
S_ISSOCK()	Socket (not in POSIX.1 or SVR4)

Macros to detect IPC type

- Argument: stat
- POSIX.1 allows implementations to represent IPC objects as files, e.g. semaphores, message queues, etc. (**No OS from the book does that now**)

Macro	Type of object
S_TYPEISMQ()	Message queue
S_TYPEISSEM()	Semaphore
S_TYPEISSHM()	Shared memory object

Program 4.3: file types

```
#include     “apue.h"

int
main(int argc, char *argv[])
{
   int               i;
   struct stat       buf;
   char              *ptr;

   for (i = 1; i < argc; i++) {
       printf("%s: ", argv[i]);
       if (lstat(argv[i], &buf) < 0) {
              err_ret("lstat error");
              continue;
       }
```

Program 4.3: file types

```
        if      (S_ISREG(buf.st_mode))    ptr = "regular";
        else if (S_ISDIR(buf.st_mode))    ptr = "directory";
        else if (S_ISCHR(buf.st_mode))    ptr = "character
    special";
        else if (S_ISBLK(buf.st_mode))    ptr = "block
    special";
        else if (S_ISFIFO(buf.st_mode))   ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))    ptr = "symbolic
    link";
        else if (S_ISSOCK(buf.st_mode))   ptr = "socket";
        else                      ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }
    exit(0);
}
```

Program 4.3: (output)

```
$ ./a.out /etc/passwd /etc /dev/initctl /dev/log \
> /dev/tty /dev/scsi/host0/bus0/target0/lun0/cd \
> /dev/cdrom

/etc/passwd: regular
/etc: directory
/dev/initctl: fifo
/dev/log: socket
/dev/tty: character special
/dev/scsi/host0/bus0/target0/lun0/cd: block special
/dev/cdrom: symbolic link
```

Count & % of file types

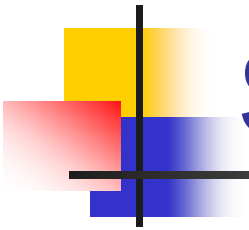
File type	Count	Percentage
Regular file	226,856	88.22%
Directory	23,017	8.95%
Symbolic link	6,442	2.51%
Character special	447	0.17%
Block special	312	0.12%
Socket	69	0.03%
FIFO	1	0.0%

Set-User-ID & Set-Group-ID

Each process has 6 or more IDs:

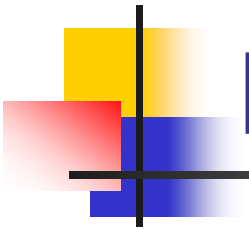
■ Real user ID ■ Real group ID	Who we really are
■ Effective user ID ■ Effective group ID ■ Supplementary group IDs	Used for file access permission checks
■ Saved set-user-ID ■ Saved set-group-ID	Saved by exec functions

Set-User-ID & Set-Group-ID



- A special flag in “st_mode” says:

  “when this file is executed, set the effective user ID of the process to be the owner of the file (st_uid)”
- Example: “passwd” is set-user-id, user can thus write to:
  /etc/passwd or /etc/shadow
  (which are root-writable only)

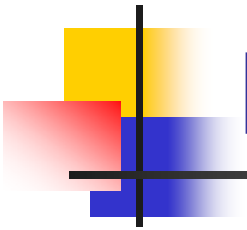


File Access Permission

- 9 permission bits
- E.g.: **`rwxr-xr--`** (read, write, execute)
- Divided into 3 categories
  - User (or owner)
  - Group
  - Other (or world)
- “chmod” command can be used to change permission bits

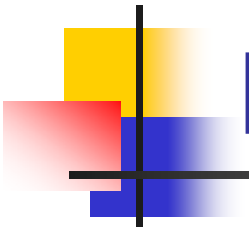
File Access Permissions

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute



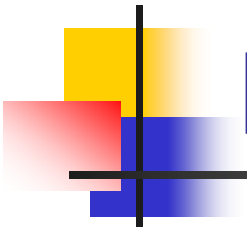
File Access Permissions

- To open a file, we need execute permissions in each directory mentioned in the pathname
- To open /usr/include/stdio.h, need “x” for:
  - /
  - /usr
  - /usr/include
- Read dir → list filenames
- Execute dir → search through



File Access Permissions

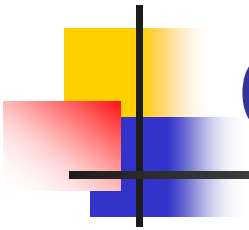
- For O_TRUNC flag, we need write permission for the file
- To create a file, we need write and execute permissions in the directory
- To delete a file, we need write and execute permissions in the directory (no need of read/write permissions for file)



File Access Permissions

Kernel tests:

- If effective UID=0, grant access.
- If effective UID=owner UID:
  - if permission bits set, grant access
  - else deny permission
- If effective GID=owner GID:
  - if permission bits set, grant access
  - else deny permission
- If permission bits set, grant access.



Ownership of New Files/Dirs

- New File or Directory
- UID = Effective UID of process
- GID =
  - Effective GID of process, such as in FreeBSD 5.2.1 and Mac OS X 10.3, OR
  - GID of directory, such as in Linux and Solaris (with set-group-ID bit enabled)

access()

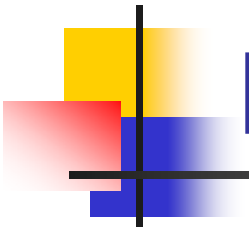
- #include <unistd.h>
- int access (const char *_pathname_, int _mode_);
- Returns: 0 if OK, -1 on error

bitwise OR

mode	**Description**
R_OK	Test for read permission
W_OK	Test for write permission
X_OK	Test for execute permission
F_OK	Test for existence of file

Program 4.8: access()

```
#include     <fcntl.h>
#include     “apue.h"

int main(int argc, char *argv[])
{
   if (argc != 2) err_quit("usage: a.out <pathname>");

   if (access(argv[1], R_OK) < 0)
       err_ret("access error for %s", argv[1]);
   else printf("read access OK\n");

   if (open(argv[1], O_RDONLY) < 0)
       err_ret("open error for %s", argv[1]);
   else printf("open for reading OK\n");

   exit(0);
}
```



Program 4.8: output

$ **ls –l a.out**

-rwxrwxr-x 1 sar 15945 Nov 30 12:10 a.out

$ **./a.out a.out**

read access OK

open for reading OK

$ **ls –l /etc/shadow**

-r-------- 1 root 1315 Jul 17 2002 /etc/shadow

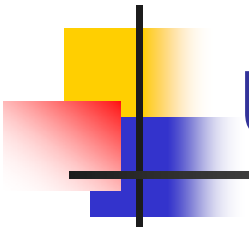
$ **./a.out /etc/shadow**

access error for /etc/shadow: Permission denied

open error for /etc/shadow: Permission denied

Program 4.8: output (contd)

```
$ su
Password:
# chown root a.out
# chmod u+s a.out
# ls –l a.out
-rwsrwxr-x   1        root    15945 Nov 30 12:10          a.out
# exit
$ ./a.out /etc/shadow
access error for /etc/shadow: Permission denied
open for reading OK
```



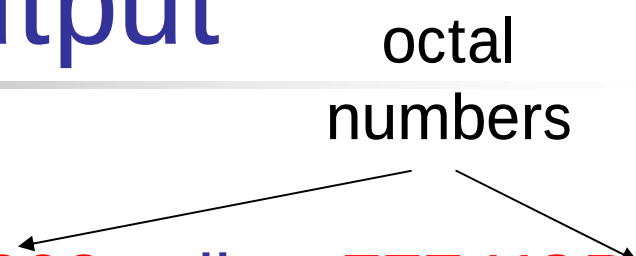
umask()

- #include <sys/stat.h>
- mode_t umask(mode_t *cmask*);
- Returns: previous file mode creation mask
- *cmask* = OR { S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IROTH, …}
- Any bits that are *on* in the file mode creation mask are turned *off* in file's *mode*

Program 4.9: umask()

```
#include<fcntl.h>
#include“apue.h"

#define RWRWRW
    (S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)
int
main(void)
{
    umask(0);
    if (creat("foo", RWRWRW)  <  0 )
        err_sys("creat error for foo");

    umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
    if (creat("bar", RWRWRW)  <  0 )
        err_sys("creat error for bar");
    exit(0);
}
```

Program 4.9: output



- $ **umask**
- 002 (files: 666 XOR 002, dirs: 777 XOR 002)
- $ **./a.out**
- **ls –l foo bar**
- -rw------- 1 sar 0 Dec 7 21:20 bar
- -rw-rw-rw- 1 sar 0 Dec 7 21:20 foo
- $ **umask**
- 002

Umask file access permission bits

Mask bit	Meaning
0400	User-read
0200	User-write
0100	User-execute
0040	Group-read
0020	Group-write
0010	Group-execute
0004	Other-read
0002	Other-write
0001	Other-execute

Single UNIX specification:

Requires symbolic form

$ **umask**

002

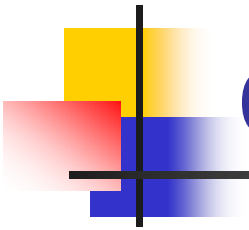
$ **umask –S** /*symbolic form*/

u=rwx, g=rwx, o=rx

$ **umask 027**

$ **umask –S**

u=rwx, g=rx, o=



chmod(), fchmod()

- #include <sys/stat.h>
- int chmod(const char *_pathname_, mode_t _mode_);
- int fchmod(int _filedes_, mode_t _mode_);
- Return: 0 if OK, -1 on error
- Effective UID of process = file owner or root

<sys/stat.h>

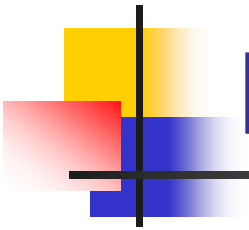
mode in chmod()

Bit-wise OR

mode	**Description**
S_ISUID, S_ISGID, S_ISVTX	Set UID, GID on exec saved-text (sticky bit)
S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR	Read, write, exec by user (owner): all or individual
S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP	Read, write, exec by group: all or individual
S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH	Read, write, exec by other (world): all or individual

Program 4.12: chmod()

```
#include"apue.h"

int
main(void)
{
    struct stat                 statbuf;

    /* turn on set-group-ID and turn off group-execute */

    if (stat("foo", &statbuf) < 0)
        err_sys("stat error for foo");
    if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
        err_sys("chmod error for foo");

    /* set absolute mode to "rw-r--r--" */

    if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
        err_sys("chmod error for bar");

    exit(0);
}
```



Program 4.12: output

$ **ls –l foo bar**

-rw------- 1 sar 0 Dec 7 21:20 bar

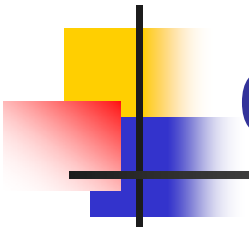
-rw-rw-rw 1 sar 0 Dec 7 21:20 foo

After Program 4.12 execution:

$ **ls –l foo bar**

-rw-r--r-- 1 sar 0 Dec 7 21:20 bar

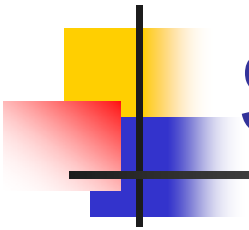
-rw-rwSrw 1 sar 0 Dec 7 21:20 foo



chmod()

chmod() automatically clears 2 bits:

- if we try to set sticky bit (S_ISVTX) without superuser privileges
- setting S_ISGID without privileges
  - To prevent a user from creating a set-group-ID file owned by a group that the user doesn't belong to



Sticky Bit

- Earlier versions of UNIX
  - program's text (machine instructions) was saved in swap space
  - program loaded faster on 2nd execution
- sticky = stuck around (in swap space)
- saved text = sticky
- Newer systems have virtual memory & fast filesystem, no need of sticky bit

Sticky Bit for directories

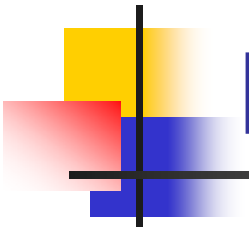
- A file can be removed or renamed only if user has write permissions to dir and
  - owns the file, OR
  - owns the directory, OR
  - is the superuser
- /tmp
- /var/spool/uucppublic

Should not be able to delete or rename others' files

chown(), fchown(), lchown()

- #include <unistd.h>
- int chown(const char *_pathname_,
  uid_t _owner_, gid_t _group_);
- int fchown(int _filedes_,
  uid_t _owner_, gid_t _group_);
- int lchown(const char *_pathname_,
  uid_t _owner_, gid_t _group_);
- Return: 0 if OK, -1 on error

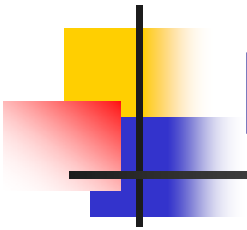
Owner, group = -1 ➔unchanged



File Size

- st_size in stat structure specifies size of file in bytes
- st_size = 0 for an empty regular file (first read() returns EOF)
- st_size = multiple of 16 or 512 for directories
- st_size = #bytes(filename) for links

lrwxrwxrwx 1 root 7 Sep 25 07:14 lib → usr/lib



Holes in a File

$ **ls –l core**

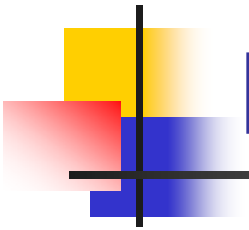
-rw-r--r-- 1 sar 8483248 Nov 18 12:18 core

$ **du –s core**

272 core

(272 512-byte blocks=139,264 bytes)

- Many holes in the file!



Holes in a File

$ **wc –c core**

8483248 core

- Normal I/O operations read through file

$ **cat core > core.copy**

$ **ls –l core***

-rw-r--r-- 1 stevens 8483248 Nov 18 12:18 core

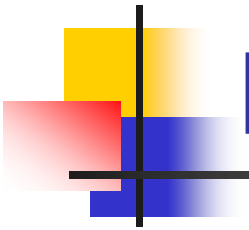
-rw-rw-r-- 1 stevens 8483248 Nov 18 12:27 core.copy

$ **du –s core***

272 core

16592 core.copy

- 16592 x 512 = 8,495,104 bytes



File Truncation

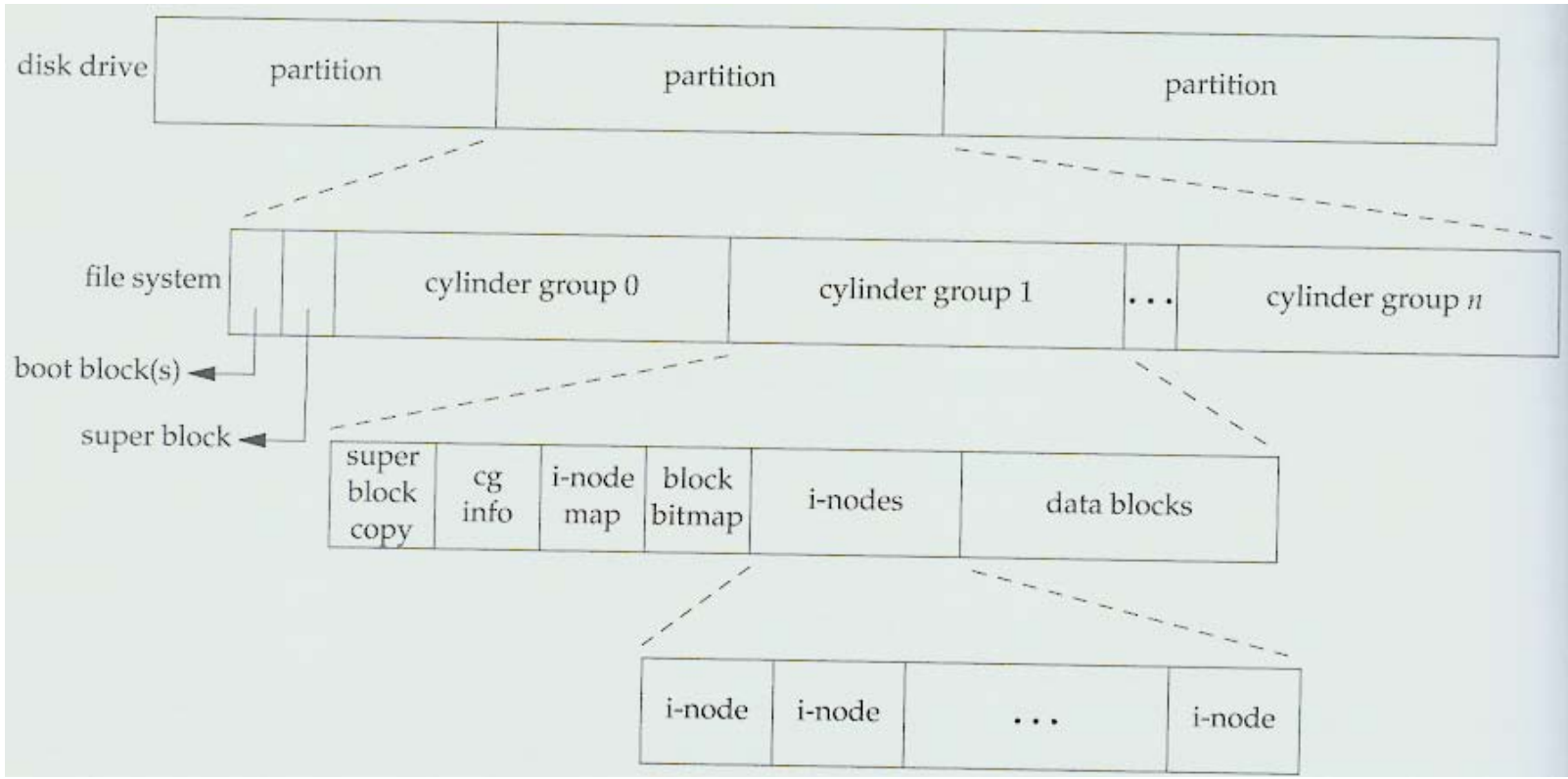
- #include <unistd.h>
- int truncate(const char *_pathname_,
  off_t _length_);
- int ftruncate(int _filedes_, off_t _length_);
- Return: 0 if OK, -1 on error

File Truncation

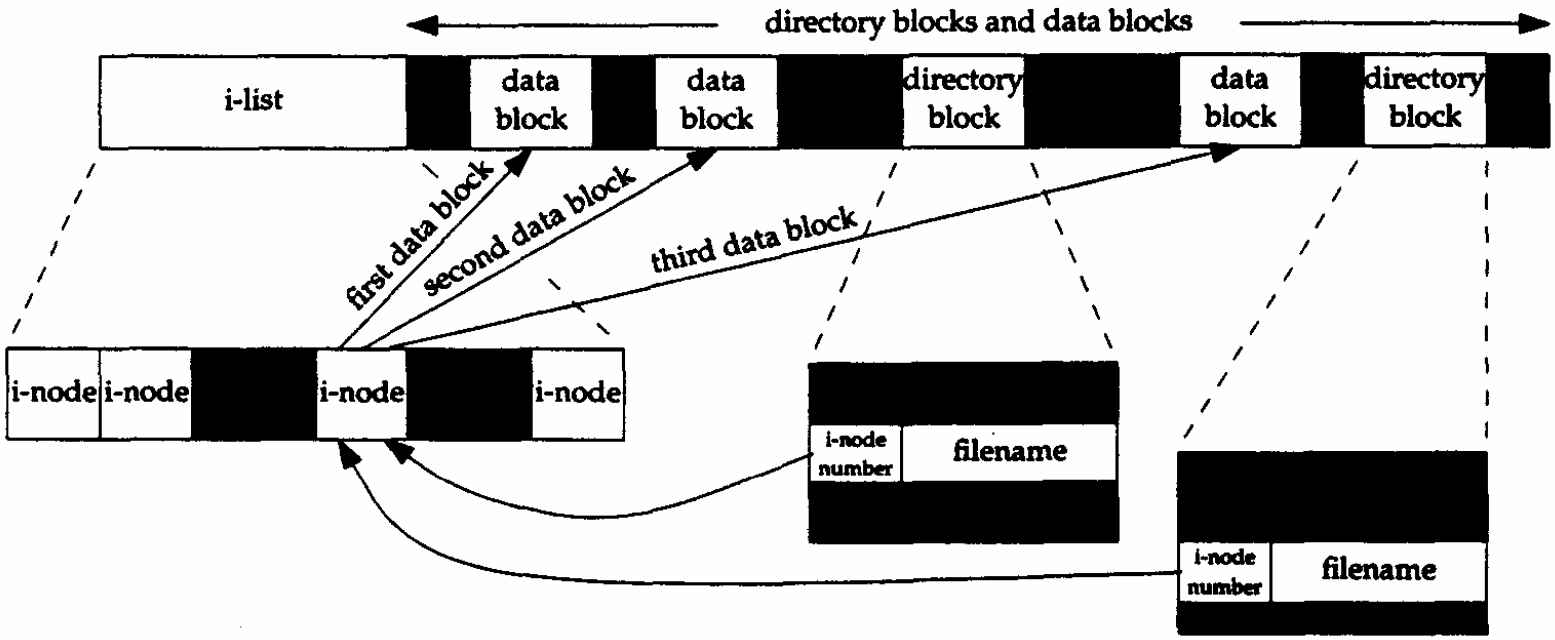
- truncate(*pathname*, *length*)
- Truncates a file to length bytes
  - file size > length ➔ file size:= length
  - file size < length ➔ SVR4 extends file

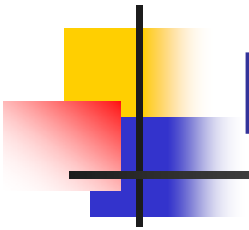
    4.3+BSD does nothing
- Solaris: fcntl(F_FREESP) allows freeing any part of a file, not just a chunk at the end of file

Filesystems (Fig. 4.13)



Filesystems (Fig. 4.14)



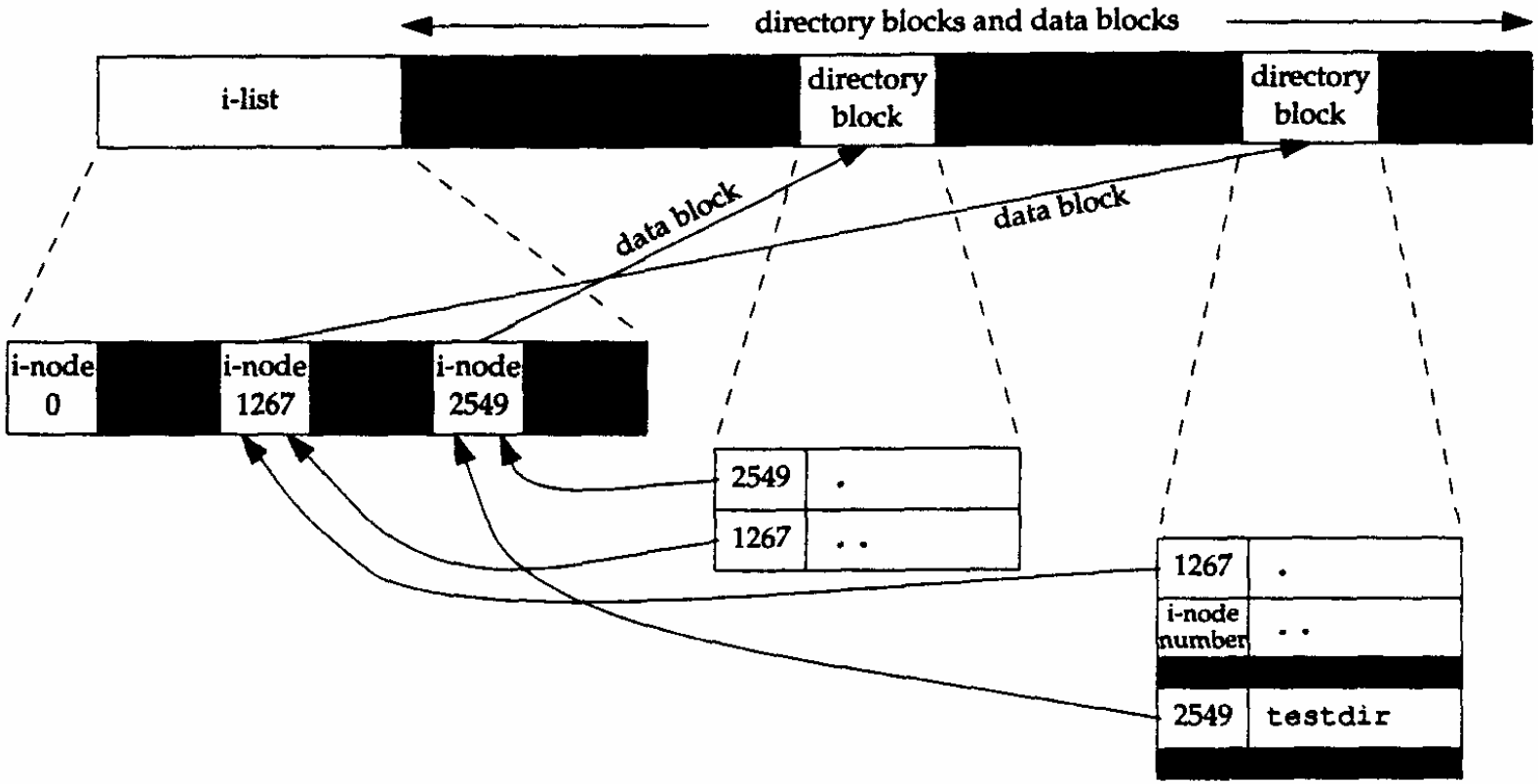


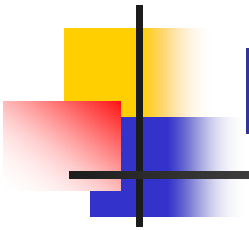
Filesystems

- i-node contains all info about file:
  - file type
  - file's access permission bits
  - file size
  - pointers to data blocks for file
  - …
- Inode # in dir points to an i-node in the same filesystem
- Hence ln cannot link across filesystems

Filesystem (Fig. 4.15)

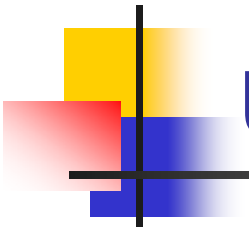
- After creating testdir





link()

- #include <unistd.h>
- int link(const char *_existingpath_, const char *_newpath_);
- Return: 0 if OK, -1 on error
- Creation of a new directory entry and increment of link count must be ATOMIC!!!



unlink()

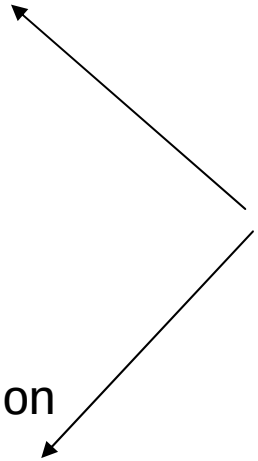
- #include <unistd.h>
- int unlink(const char *_pathname_);
- Returns: 0 if OK, -1 on error
- Removes directory entry
- Decrements link count
- Link count = 0 & open count = 0 ➔ delete file

Program 4.16: unlink()

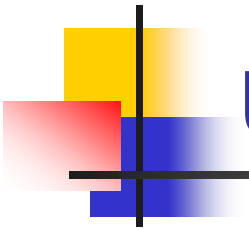
```
#include      <fcntl.h>
#include      "apue.h"

int
main(void)
{
    if (open("tempfile", O_RDWR) < 0)
        err_sys("open error");

    if (unlink("tempfile") < 0)
        err_sys("unlink error");

    printf("file unlinked\n");
    sleep(15);
    printf("done\n");

    exit(0);
}
```

Program 4.16: output

```
$ ls –l tempfile
-rw-r--r-- 1 stevens 9240990 Jul 31 13:42 tempfile
$ df /home
Filesystem kbytes     used    avail capacity  Mounted on
/dev/sd0h 282908 181979 72638     71%   /home
$ a.out &
1364
file unlinked
$ ls –l tempfile
tempfile not found
$ df /home
Filesystem kbytes     used    avail capacity  Mounted on
/dev/sd0h 282908 181979 72638     71%    /home
done
$ df /home
Filesystem kbytes    used    avail capacity  Mounted on
/dev/sd0h 282908 172939 81678    68%    /home
```

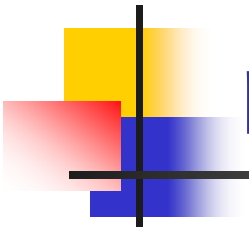


No change in space



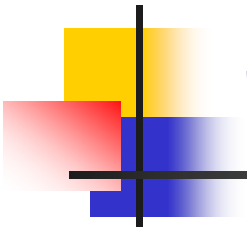
unlink()

- Used for temporary files
- Won’t be left around if program crashes
- open() or creat() temporary file
- unlink() immediately
- File is not deleted, because still open
- File is deleted only when process terminates or closes it



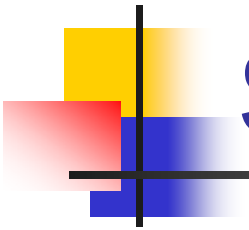
remove()

- #include <stdio.h>
- int remove(const char **pathname*);
- Returns: 0 if OK, -1 on error
- For files: Identical to unlink()
- For directories: Identical to rmdir()



rename()

- #include <stdio.h>
- int rename(const char *_oldname_,
  const char *_newname_);
- Returns: 0 if OK, -1 on error
- If newname exists:
- If both are files, oldname→newname
- If both are dirs, oldname→newname
- (newname dir must be empty)
- previous newname is first deleted



Symbolic Links

- Can link across filesystems
- Anyone can link to a directory
- Used to move a file or an entire directory hierarchy to somewhere else
- If function follows links, argument refers to the actual file
- If function does not follow links, argument refers to the link

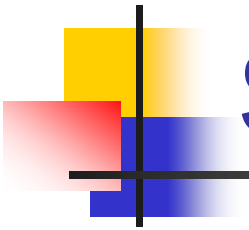
Symbolic Links (Fig. 4.17)

Function	Does not follow symbolic link	Follows symbolic link
access		•
chdir		•
chmod		•
chown	•	•
creat		•
exec		•
lchown	•	
link		•
lstat	•	
mkdir		•
mkfifo		•
mknod		•
open		•
opendir		•
pathconf		•
readlink	•	
remove	•	
rename	•	
stat		•
truncate		•
unlink	•	

Figure 4.10 Treatment of symbolic links by various functions.

Symbolic links create loops

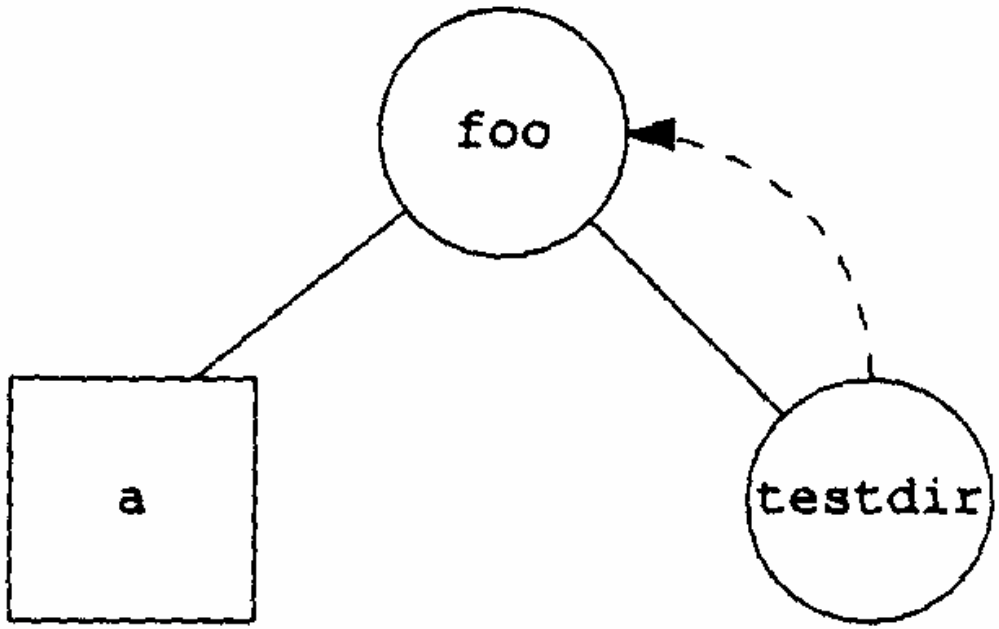
```
$ mkdir foo
$ touch foo/a
$ ln –s ../foo foo/testdir
$ ls –l foo
total 1
-rw-rw-r--    1 stevens 0 Dec 6 06:06 a
lrwxrwxrwx 1 stevens 6 Dec 6 06:06 testdir→../foo
```

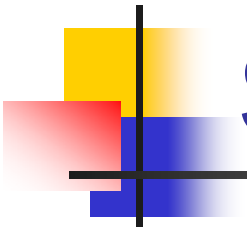


Symbolic Links (loops)

- write a program using ftw (walk fs)
- RESULT:
- foo
- foo/a
- foo/testdir
- foo/testdir/a
- foo/testdir/testdir
- foo/testdir/testdir/a
- foo/testdir/testdir/testdir
- foo/testdir/testdir/testdir/a
- ftw returned -1: Too many levels of symbolic links

Symbolic Link (Fig. 4.18)





Symbolic Link

- open() follows symbolic links

$ **ln –s /no/such/file myfile**

$ **ls myfile**

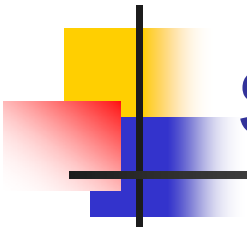
myfile

$ **cat myfile**

cat: myfile: No such file or directory

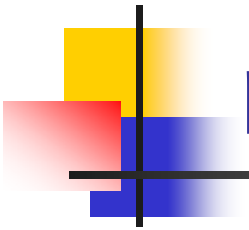
$ **ls –l myfile**

lrwxrwxrwx 1 stevens 13 Dec 6 07:27 myfile→/no/such/file



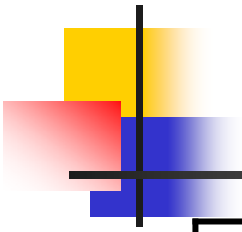
symlink()

- #include <unistd.h>
- int symlink(const char **actualpath*, const char ***sympath*);
- Returns: 0 if OK, -1 on error
- Creates a symbolic link: *sympath* → *actualpath*
- open() cannot open a link, so what if we want to read a link?



readlink()

- #include <unistd.h>
- int readlink(const char **pathname*,
  char **buf*, int *bufsize*);
- Returns: #bytes read if OK, -1 on error
- Combines open, read, & close
- buf: name of link, not null terminated



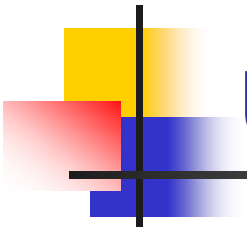
File Times

Field	Description	Example	ls option
st_atime	last-access time of file data	read	-u
st_mtime	last-mod time of file data	write	default
st_ctime	last-change time of i-node status	chmod, chown	-c

3 times of functions (Fig. 4.20)

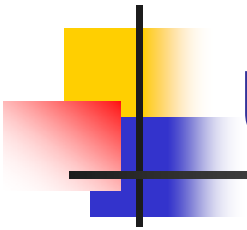
Function	Referenced file (or directory)			Parent directory of referenced file (or directory)			Note
	a	m	c	a	m	c	
chmod, fchmod			•				
chown, fchown			•				
creat	•	•	•		•	•	O_CREAT new file
creat		•	•				O_TRUNC existing file
exec	•						
lchown			•				
link			•		•	•	
mkdir	•	•	•		•	•	
mkfifo	•	•	•		•	•	
open	•	•	•		•	•	O_CREAT new file
open		•	•				O_TRUNC existing file
pipe	•	•	•				
read	•						
remove			•		•	•	remove file = unlink
remove					•	•	remove directory = rmdir
rename			•		•	•	for both arguments
rmdir					•	•	
truncate, ftruncate		•	•				
unlink			•		•	•	
utime	•	•	•				
write		•	•				

Effect of various functions on the access, modification, and changed-status times.



utime()

```
#include <utime.h>
int utime( const char *pathname,
                const struct utimbuf *times);
Returns: 0 if OK, -1 on error
struct utimbuf {
   time_t actime; /* access time */
   time_t modtime; /*modification time */
}
```



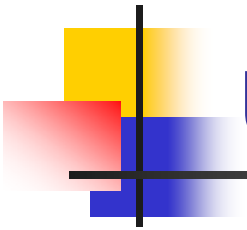
utime()

Depends on *times* pointer

- NULL➔ set a and m times to current
- Not Null ➔ set to the *times* pointer

Access permissions required:

- current: owner of file, write permission
- times: owner of file, superuser



utime()

Used by:

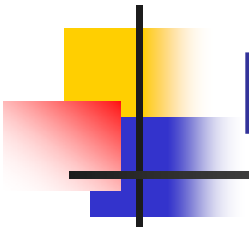
- touch
- tar
- cpio

Program 4.21: utime()

```
#include<fcntl.h>
#include<utime.h>
#include"apue.h"

int
main(int argc, char *argv[])
{
    int                 i, fd;
    struct stat         statbuf;
    struct utimbuf      timebuf;

    for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) {      /* fetch current times */
            err_ret("%s: stat error", argv[i]);
            continue;
        }
```

Program 4.21: utime()

```
            if (fd=open(argv[i], O_RDWR | O_TRUNC) < 0) { /* truncate */
                        err_ret("%s: open error", argv[i]);
                        continue;
            }
            close(fd);
            timebuf.actime  = statbuf.st_atime;
            timebuf.modtime = statbuf.st_mtime;
            if (utime(argv[i], &timebuf) < 0) {          /* reset times */
                        err_ret("%s: utime error", argv[i]);
                        continue;
            }
    }
    exit(0);
}
```



Program 4.21: output

$ **ls –l changemod times**

-rwxrwxr-x 1 sar 24576 Dec 4 16:13 changemod

-rwxrwxr-x 1 sar 24576 Dec 6 09:24 times

$ **ls –lu changemod times**

-rwxrwxr-x 1 stevens 24576 Feb 1 12:44 changemod

-rwxrwxr-x 1 stevens 24576 Feb 1 12:44 times

$ **date**

Sun Feb 3 18:22:33 MST 1991

Program 4.21: output (contd)

$ **./a.out changemod times**

$ **ls –l changemod times**

-rwxrwxr-x 1 stevens 0 Dec 4 16:13 changemod

-rwxrwxr-x 1 stevens 0 Dec 6 09:24 times

$ **ls –lu changemod times**

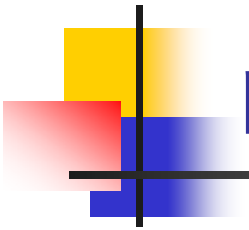
-rwxrwxr-x 1 stevens 0 Feb 1 12:44 changemod

-rwxrwxr-x 1 stevens 0 Feb 1 12:44 times

$ **ls –lc changemod times**

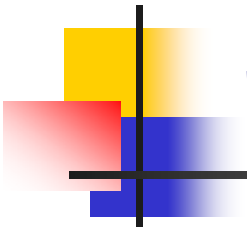
-rwxrwxr-x 1 stevens 0 Feb 3 18:23 changemod

-rwxrwxr-x 1 stevens 0 Feb 3 18:23 times



mkdir()

- #include <sys/stat.h>
- int mkdir(const char *_pathname_, mode_t _mode_);
- Returns: 0 if OK, -1 on error
- Creates a new, empty directory
- . and .. are automatically created
- _mode_ are modified by file mode creation mask of the process



rmdir()

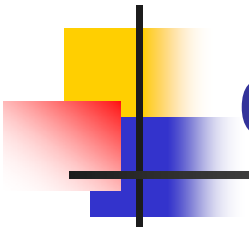
- #include <unistd.h>
- int rmdir(const char *_pathname_);
- Returns: 0 if OK, -1 on error
- Link# = 0 & Open# = 0 ➔ space of dir freed

Reading Directories

- Anyone can read a dir with permissions
- Only kernel can write a dir

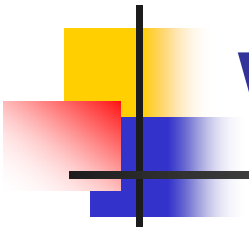
  (diff from create/deleting files in a dir)

```
struct dirent { /* defined in <dirent.h> */
ino_t d_ino; /* i-node # */
char d_name[NAME_MAX + 1];
  /* NULL-terminated filename */
}
```



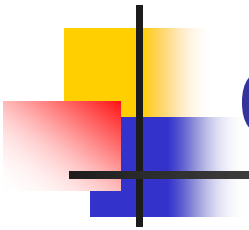
opendir(), rewinddir(), closedir()

- #include <dirent.h>
- DIR *opendir(const char **pathname*);
- struct dirent *readdir(DIR **dp*);
- void rewinddir(DIR **dp*);
- int closedir(DIR **dp*);
- long telldir(DIR **dp*);
- void seekdir(DIR **dp*, long *loc*);



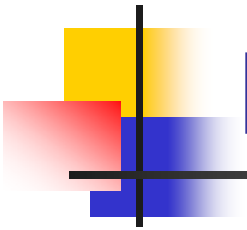
Walking the filesystem

- See Figure 4.22 (pages 121~124)
- Gives the counts and percentages of different file types



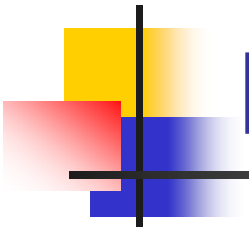
chdir()

- #include <unistd.h>
- int chdir(const char *_pathname_);
- int fchdir(int _filedes_);
- Return: 0 if OK, -1 on error



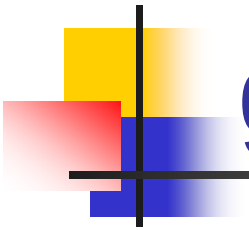
Program 4.23

```
#include      "ourhdr.h"

int
main(void)
{
    if (chdir("/tmp") < 0)
        err_sys("chdir failed");

    printf("chdir to /tmp succeeded\n");
    exit(0);
}
```



Program 4.23: output

- $ **pwd**
- /usr/lib
- $ **mycd**
- chdir to /tmp succeeded
- $ **pwd**
- /usr/lib



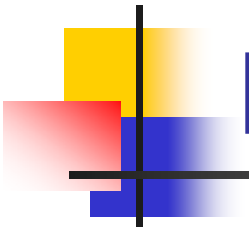
getcwd()

- #include <unistd.h>
- char *getcwd(char *_buf_, size_t _size_);
- Returns: _buf_ if OK, NULL on error
- _buf_ should be large enough to accommodate absolute pathnames plus a terminating null byte, or error.

Program 4.24: getcwd()

```
#include"apue.h"

int
main(void)
{
    char  *ptr;
    int    size;

    if (chdir("/usr/spool/uucppublic") < 0)
        err_sys("chdir failed");

    ptr = path_alloc(&size);          /* our own function */
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd failed");

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

Program 4.24: output



- $ **a.out**
- cwd = /var/spool/uucppublic
- $ **ls –l /usr/spool**
- lrwxrwxrwx 1 root 12 Jan 31 07:57 /usr/spool → ../var/spool

Program 4.25: st_dev, st_rdev

```
#include  "apue.h"

int main(int argc, char *argv[])
{
    int                 i;
    struct stat         buf;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            err_ret("lstat error"); continue; }

        printf("dev = %d/%d", major(buf.st_dev),  minor(buf.st_dev));

        if (S_ISCHR(buf.st_mode) || S_ISBLK(buf.st_mode)) {
            printf(" (%s) rdev = %d/%d",
                (S_ISCHR(buf.st_mode)) ? "character" : "block",
                major(buf.st_rdev), minor(buf.st_rdev)); }
        printf("\n");
    }
    exit(0);
}
```

Program 4.25: output

$ **./a.out / /home/sar /dev/tty[01]**

/: dev = 3/3

/home/sar: dev = 3/4

/dev/tty0: dev = 0/7 (character) rdev=4/0

/dev/tty1: dev=0/7 (character) rdev=4/1

$ **mount**

/dev/hda3 on / type ext2 (rw,noatime)

/dev/hda4 on /home type ext2 (rw,noatime)

$ **ls –lL /dev/tty[01] /dev/hda[34]**

```
brw------- 1 root  3, 3 Jan 31 08:23 /dev/hda3
brw------- 1 root  3, 4 Jan 31 08:23 /dev/hda4
crw------- 1 root  4, 0 Jan 31 08:22 /dev/tty0
crw------- 1 root  4, 1 Jul  9 10:11 /dev/tty1
```