Contents

Files and Directories

- Objectives
  - Additional Features of the File System
  - Properties of a File.

- Three major functions:

```
#include <sys/types.h>
#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);
```

Files and Directories

- Differences on stat(), fstat(), lstat():
  - lstat() returns info regarding the symbolic link, instead of the referenced file, if it happens.

```
struct stat {
    mode_t    st_mode; /* type & mode */
    ino_t     st_ino; /* i-node number */
    dev_t     st_dev; /* device no (filesystem) */
    dev_t     st_rdev; /* device no for special file */
    nlink_t   st_nlink; /* # of links */
    uid_t     st_uid;        gid_t    st_gid;
    off_t     st_size; /* sizes in byes */
    time_t    st_atime; /* last access time */
    time_t    st_mtime; /* last modification time */
    time_t    st_ctime; /* time for last status change */
    long      st_blk_size; /* best I/O block size */
    long      st_blocks; /* number of 512-byte blocks allocated */
};
```


File Types

- Regular Files: text, binary, etc.
- Directory Files: Only Kernel can update these files – { (filename, pointer) }.
- Character Special Files, e.g., tty, audio, etc.
- Block Special Files, e.g., disks, etc.
- FIFO – named pipes
- Sockets – not POSIX.1 or SVR4
  - SVR4 uses library of socket functions, instead. 4.3+BSD has a file type of socket.
- Symbolic Links – not POSIX.1 or SVR4

File Types

- Program 4.1 – Page 76
  - lstat() and Figure 4.1
  - <sys/stat.h>

```
#define S_IFMT     0xF000  /* type of file */
#define S_IFDIR    0x4000  /* directory */
#define S_ISDIR(mode)   (((mode)&0xF000)
  == 0x4000)
```

  - st_mode
- Percentage of Files in a Medium-Sized System – Figure 4.2

Access Permissions & UID/GID

- All files have access permissions
  - st_mode mask – owner, group, other

```
#define S_IRWXU 00700      /* read, write, execute: owner */
#define S_IRUSR 00400      /* read permission: owner */
#define S_IWUSR 00200      /* write permission: owner */
#define S_IXUSR 00100      /* execute permission: owner */
#define S_IRWXG 00070      /* read, write, execute: group */
#define S_IRGRP 00040      /* read permission: group */
#define S_IWGRP 00020      /* write permission: group */
#define S_IXGRP 00010      /* execute permission: group */
#define S_IRWXO 00007      /* read, write, execute: other */
#define S_IROTH 00004      /* read permission: other */
#define S_IWOTH 00002      /* write permission: other */
#define S_IXOTH 00001      /* execute permission: other */
```

Access Permissions & UID/GID

- Operations vs Permissions
  - Directory
    - X – pass through the dir (search bit), e.g., /usr/dict/words and PATH env var.
    - R – list of files under the dir.
    - W – update the dir, e.g., delete or create a file.
  - File
    - X – execute a file (which must be a regular file)
    - R – O_RDONLY or O_RDWR
    - W – O_WRONLY, O_RDWR, or O_TRUNC

Access Permissions & UID/GID

- Files
  - st_uid, st_gid in stat
  - S_ISUID, S_ISGID – set st_uid/gid bit in st_mode
    - E.g., Command “passwd” updates /etc/passwd or /etc/shadow
- Process UID/GID – optional with POSIX.1
  - sysconf(_SC_SAVED_IDS)
  - Real User/Group ID (from /etc/passwd)
  - Effective User/Group ID, Supplementary GID’s
    - Check for file access permissions
  - Saved Set-User/Group-ID – saved by exec()

Access Permissions & UID/GID

- File Access Test – each time a process creates/opens/deletes a file
  - If the effective UID == 0 → superuser!
  - If the effective UID == UID of the file
    - Check appropriate access permissions!
  - If the effective GID == GID of the file
    - Check appropriate access permissions!
  - Check appropriate access permissions for others!
- Related Commands: chmod & umask

Ownership of a New File

- Rules:
  - UID of a file = the effective UID of the creating process
  - GID of a file – options under POSIX
    1. GID of the file = the effective GID of the process
    2. GID of the file = the GID of the residing dir
       - 4.3BSD and FIPS 151-1 always do it.
       - SVR4 needs to set the set-group-ID bit of the residing dir (mkdir)!

Function – access

#include <unistd.h>

int access(const char *pathname, int mode);

- Check the real UID/GID!
- R_OK, W_OK, X_OK, F_OK

- Program 4.2 – Page 83
  - access function

Function – umask

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

mode_t umask(mode_t cmask);

- Turn off the file mode
  - cmask = bitwise-OR S_I[RWX]USR, etc (Figure 4.4).
- The mask goes with the process only.
  - Inheritance from the parent!
- Program 4.3 – Page 85
  - umask

Function – chmod & fchmod

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *pathname, mode_t
    mode);
int fchmod(int filedes, mode_t  mode);
```

- fchmod() is not in POSIX.1, but in SVR4/4.3+BSD
- Callers must be a superuser or effective UID = file UID.
- Mode = bitwise-OR S_I[RWX]USR, S_ISVTX (sticky bit), S_IS[UG]ID, etc (Fig 4.6).

Function – chmod & fchmod

- Program 4.4 – Page 87
  - chmod – updates on i-nodes
- set-group-ID
  - If the GID of a newly created file is not equal to the effective GID of the creating process (or one of the supplementary GID's), or the process is not a superuser, clear the set-group-ID bit!
  - Clear up set-user/group-ID bits if a non-superuser process writes to a set-uid/gid file.

Function – chmod & fchmod

- Sticky Bit (S_ISVTX) – saved-text bit
  - Not POSIX.1 – by SVR4 & 4.3+BSD
  - Only superusers can set it!
  - S_ISVTX executable file
    - Used to save a copy of a S_ISVTX executable in the swap area to speed up the execution next time.
  - S_ISVTX directory file, e.g., /tmp
    - Remove/rename its file only if w permission of the dir is set, and the process is belonging to superusers/owner of the file/dir

Function – chown, fchown, lchown

```
#include <sys/types.h>
#include <unistd.h>
int chown(const char *pathname, uid_t  owner,
    gid_t, grp);
int fchown(int filedes, uid_t  owner, gid_t, grp);
int lchown(const char *pathname, uid_t  owner,
    gid_t, grp);
```

- lchown() is unique to SVR4. Under non-SVR4 systems, if the pathname to chown() is a symbolic link, only the ownership of the symbolic link is changed.
- -1 for owner or grp if no change is wanted.

Function – chown, fchown, lchown

- _POSIX_CHOWN_RESTRICTED is in effect (check pathconf())
  - Superuser → the UID of the file can be changed!
  - The GID of the file can be changed if
    - the process owns the file, and
    - Parameter owner = UID of the file & Parameter grp = the process GID or is in supplementary GID's
- set-user/group-ID bits would be cleared if chown is called by non-super users.

File Size

- File Sizes – st_size
  - Regular files – 0~max (off_t)
  - Directory files – multiples of 16/512
  - Symbolic links – pathname length
  - /* a pipe's file size for SVR4 */
- File Holes
  - st_blocks vs st_size (st_blksize)
  - Commands:
    - "ls –l file.hole" == "wc –c file.hole"
    - du –s file.hole → actual size
    - cat file.hole > file.hole.copy

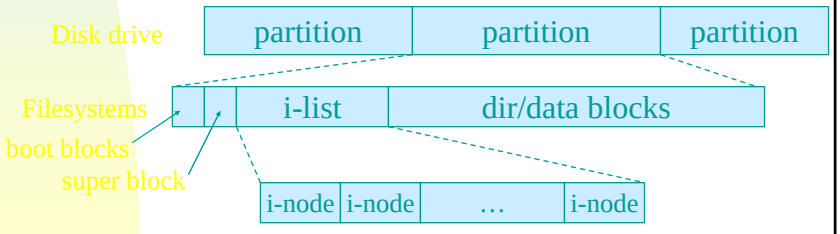
Functions – truncate & ftruncate

```
#include <sys/types.h>
#include <unistd.h>
int truncate(const char *pathname, off_t
    length);
int ftruncate(int filedes, off_t length);
```

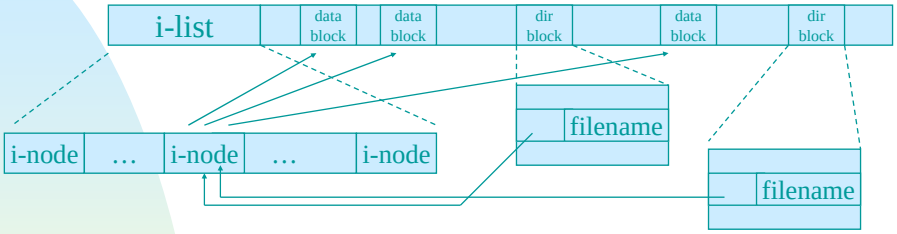
- Not POSIX.1
- SVR4 creates a hole if length > fsize.
  - fcntl with F_FREESP to free any part of a file.
- 4.3+BSD only truncates files.
- Portability?

Filesystems

- A hierarchical arrangement of directories and files – starting in root /
- Several Types, e.g.,SVR4: Unix System V Filesystems (S5), Unified File System (UFS) – Figure 2.6 (Page 39)
- System V:

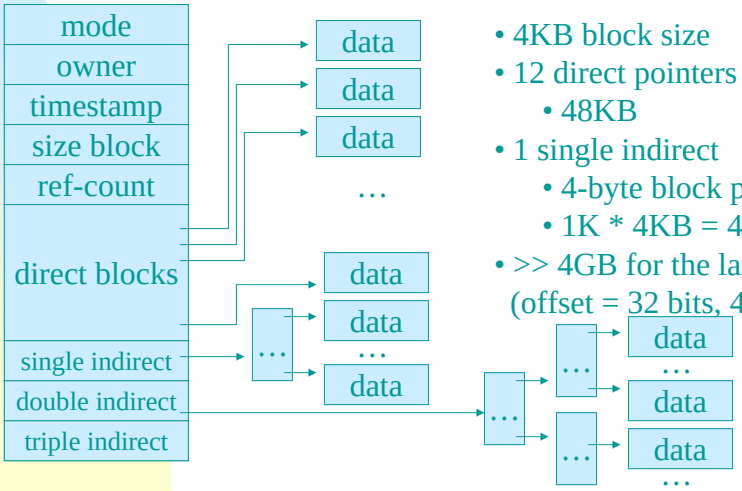


Filesystems



- i-node:
  - Version 7: 64B, 4.3+BSD:128B, S5:64B, UFS:128B
  - File type, access permission, file size, data blocks, etc.
- Link count – hard links
  - st_nlink in stat, LINK_MAX in POSIX.1
  - Unlink/link a file

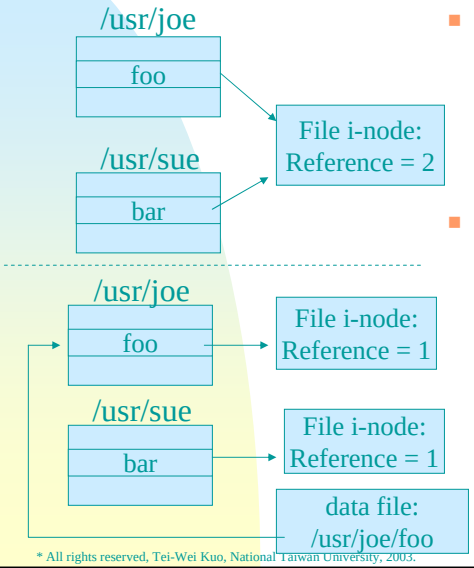
Filesystem – 4.4BSD i-node



- 4KB block size
- 12 direct pointers
  - 48KB
- 1 single indirect
  - 4-byte block ptr
  - 1K * 4KB = 4MB
- >> 4GB for the largest file!
  (offset = 32 bits, 4G=2^{32})

* "Operating system concept", Silberschatz and Galvin, Addison Wesley, pp. 380.

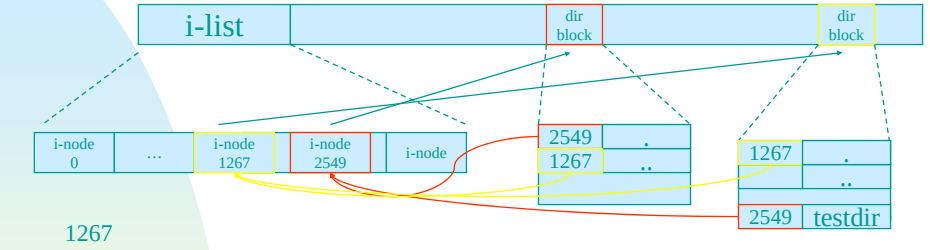
Sharing of Files



- Hard Link
  - Each directory entry creates a hard link of a filename to the i-node that describes the file's contents.
- Symbolic Link (Soft Link)
  - It is implemented as a file that contains a pathname.
  - Filesize = pathname length
  - Example: Shortcut on Windows

* Problem – infinite loop in tracing a path name with symbolic links – 4.3BSD, no 8 passings of soft links
* Dangling pointers

Filesystem



1267

2549 (testdir)

- Example
  - At i-node 1267, mkdir testdir → i-node 2549
  - The link counts of testdir is at least 2
- Command mv only modify dir entries!
- No directory entry points at any i-node residing at other filesystems.

Functions – link, unlink, rename, remove

```
#include <unistd.h>
int link(const char *existingpath, const char
   *newpath);
int unlink(const char *pathname);
```

- Atomic action for link – hard link
  - POSIX.1 allows linking across filesystems
  - Only superusers could create a link to a dir
- Error if newpath exists
- Unlink – WX right at the residing dir
  - Remove the dir entry & delete the file if link count reaches zero and no one still opens the file (Remark: sticky bit & dir rights).

Functions – link, unlink, rename, remove

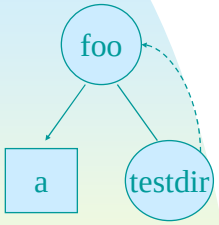
- Program 4.5 – Page 97
  - Open & unlink a file
- Unlink a file
  - Sticky bits set for a residing dir
    - Owner of the file or the dir, or super users
  - If pathname is a symbolic link, unlink references the symbolic link.

Functions – link, unlink, rename, remove

```
#include <stdio.h>
int remove(const char *pathname);
int rename(const char *oldname, const char
   *newname);
```

- remove = rmdir if pathname is a dir. (ANSI C)
- Rename – ANSI C
  - File: both files, newname is removed first, WX permission for both residing directories
  - Directory: both dir, newname must be empty, newname could not contain oldname.

Symbolic Links



- Goal
  - Get around the limitations of hard links: (a) filesystem boundary (b) link to a dir.
  - Initially introduced by 4.2BSD
- Example – ftw
  - ln –s ../foo testdir
- Figure 4.10 – functions follow slinks
  - No functions which take filedes
  - Example: unlink(testdir)

Symbolic Links

```
#include <unistd.h>
int symlink(const char *actualpath, const
    char *sympath);
int readlink(const char *pathname, char
    *buf, int bufsize);
```

- actualpath does not need to exist!
  - They do not need to be in the same file system.
- readlink is an action consisting of open, read, and close – not null terminated.

File Times

- Three Time Fields:

Field	Description	Example	ls-option
st_atime	last-access-time	read	-u
st_mtime	last-modification-time	write	default
st_ctime	last-i-node-change-time	chmod, chown	-c

- Figure 4.13 – Effect of functions on times
- Changing the access permissions, user ID, link count, etc, only affects the i-node!
- ctime is modified automatically! (stat, access)
- Example: reading/writing a file only affects the file, instead of the residing dir (Fig4.13).

File Times

```
struct utimbuf {
 time_t actime;
 time_t modtime;
 }
```

```
#include <sys/types.h>
#include <utime.h>
int utime(const char *pathname, const struct
   utimbuf *times);
```

- time values are in seconds since the Epoch
- times = null → set as the current time
  - Effective UID = file UID or W right to the file
- times != null → set as requested
  - Effective UID = (file UID or superuser) and W right to the file.
- Program 4.6 – Page 105, utime

Functions – mkdir and rmdir

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *pathname, mode_t
   mode);
```

- umask, UID/GID setup (Sec 4.6)
- From 4.2BSD & SVR3 (SVR4 – inheritance of the S_ISGID bit)

```
#include <unistd.h>
int rmdir(const char *pathname);
```

- An empty dir is deleted.
- Link count reaches zero, and no one still opens the dir.

Functions – opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *pathname);
struct dirent *readdir(DIR *dp);
void rewinddir(DIR *dp);
int closedir(DIR *dp);
```

- Only the kernel can write to a dir!!!
- WX for creating/deleting a file!
- Implementation-dependent!

Functions – opendir, readdir, rewinddir, closedir

- dirent struct is very implementation-dependent, e.g.,

```
struct dirent {
   ino_t d_ino;  /* not in POSIX.1 */
   char d_name[NAME_MAX+1];
 } /* fpathconf() */
```

- Program 4.7 – Pages 109-111
  - ftw/nftw – recursively traversing the filesys

Functions – chdir, fchdir, getcwd

#include <unistd.h>

int chdir(const char *pathname);

int fchdir(int filedes);

- fchdir – not POSIX.1
- chdir must be built into shells!
- The kernel only maintains the i-node number and dev ID for the current working directory!
- Per-process attribute – working dir!

- Program 4.8 – Page 113
  - chdir

Functions – chdir, fchdir, getcwd

#include <unistd.h>

char *getcwd(char *buf, size_t size);

- The buffer must be large enough, or an error returns!
- chdir follows symbolic links, and getcwd has not idea of symbolic links!

- Program 4.9 – Page 114
  - getcwd

Special Device Files

- Device Number – dev_t
  - Major and minor numbers: 8 bit each under 4.3+BSD
  - macro definition <sysmacros.h> @ntucsa

```
#define L_BITSMINOR   18  /* # of SVR4 minor device bits */
#define L_MAXMAJ      0x3fff  /* SVR4 max major value */
#define major(x)      (int)((unsigned)((x)>>O_BITSMINOR) &
   O_MAXMAJ)
```

- Program 4.10 – Page 115
  - st_dev vs st_rdev

Functions – sync and fsync

```
#include <unistd.h>
void sync(void);
int fsync(int filedes);
```

- sync() queues all kernel modified block buffers for writing and returns.
- fsync() waits for the I/O of a referred file to complete!
  - fsync vs O_SYNC
- Not POSIX.1, but supported by SVR4 and 4.3+BSD