Chapter 4.
Files and Directories

朱金辉

华南理工大学软件学院

1. Introduction

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

2. stat(), fstat(), lstat()

- #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);
- 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 (permissions) */
  ino_t           st_ino;      /* i-node number (serial number) */
  dev_t           st_dev;      /* device number (file system) */
  dev_t           st_rdev;     /* device number for special files */
  nlink_t         st_nlink;    /* number of links */
  uid_t           st_uid;      /* user ID of owner */
  gid_t           st_gid;      /* group ID of owner */
  off_t           st_size;     /* size in bytes, for regular files */
  struct timespec st_atim;     /* time of last access */
  struct timespec st_mtim;     /* time of last modification */
  struct timespec st_ctim;     /* time of last file status change */
  blksize_t       st_blksize;  /* best I/O block size */
  blkcnt_t        st_blocks;   /* number of disk blocks allocated */
};
```

3. 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**: used for certain types of devices, such as terminal

File Types

- **Block special file**: used for disk devices
- **FIFO**: used for interprocess communication
- **Socket**: used for network communication
- **Symbolic link**: points to another file

Macros to detect file types sys/stat.h

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)

Program 4.3: file types

```
#include        <sys/types.h>
#include        <sys/stat.h>
#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";
#ifdef  S_ISLNK
        else if (S_ISLNK(buf.st_mode))ptr = "symbolic link";
#endif
#ifdef  S_ISSOCK
        else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
#endif
        else        ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }
    exit(0);
}
```

Program 4.3: (output)

```
$ ./a.out /etc/passwd /etc /dev/log /dev/tty \
> /var/lib/oprofile/opd_pipe /dev/sr0 /dev/cdrom

/etc/passwd: regular
/etc: directory
/dev/log: socket
/dev/tty: character special
/var/lib/oprofile/opd_pipe: fifo
/dev/sr0: block special
/dev/cdrom: symbolic link
```

4. 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)

```
➜  ~ ls -l /usr/bin/passwd /etc/shadow
-rw-r----- 1 root shadow  1292 2月   6  2017 /etc/shadow
-rwsr-xr-x 1 root root   54256 5月  17 07:37 /usr/bin/passwd
```

5. 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/dict/words, need “x” for:
  - /
  - /usr
  - /usr/dict
- 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

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.

6. Ownership of New Files/Dirs

- New File or Directory
- UID = Effective UID of process
- GID = Effective GID of process, OR
  GID of directory

7. access()

- #include <unistd.h>
- int access (const char **pathname*,
  int *mode*);
- Returns: 0 if OK, -1 on error
- test accessibility based on the real user and group IDs

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

become superuser

Password:

enter superuser password

chown root a.out

change file's user ID to root

chmod u+s a.out

and turn on set-user-ID bit

ls -l a.out

check owner and SUID bit

-rwsrwxr-x 1 root

15945 Nov 30 12:10 a.out

exit

go back to normal user

$ ./a.out /etc/shadow

access error for /etc/shadow: Permission denied

open for reading OK

8. umask()

- set file mode creation mask
- #include <sys/types.h>
- #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, …}

Program 4.9: umask()

```
#include "apue.h"
#include <fcntl.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          first print the current file mode creation mask
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          see if the file mode creation mask changed
002
```

9. 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*);
- Return: 0 if OK, -1 on error
- Effective UID of process = file owner or root

mode in chmod()

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 indiv

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
```

11. chown(), fchown(), lchown()

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

12. 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 stevens 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

13. File Truncation

- #include <sys/types.h>
- #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
- No standard way of file truncation in UNIX

14. File systems

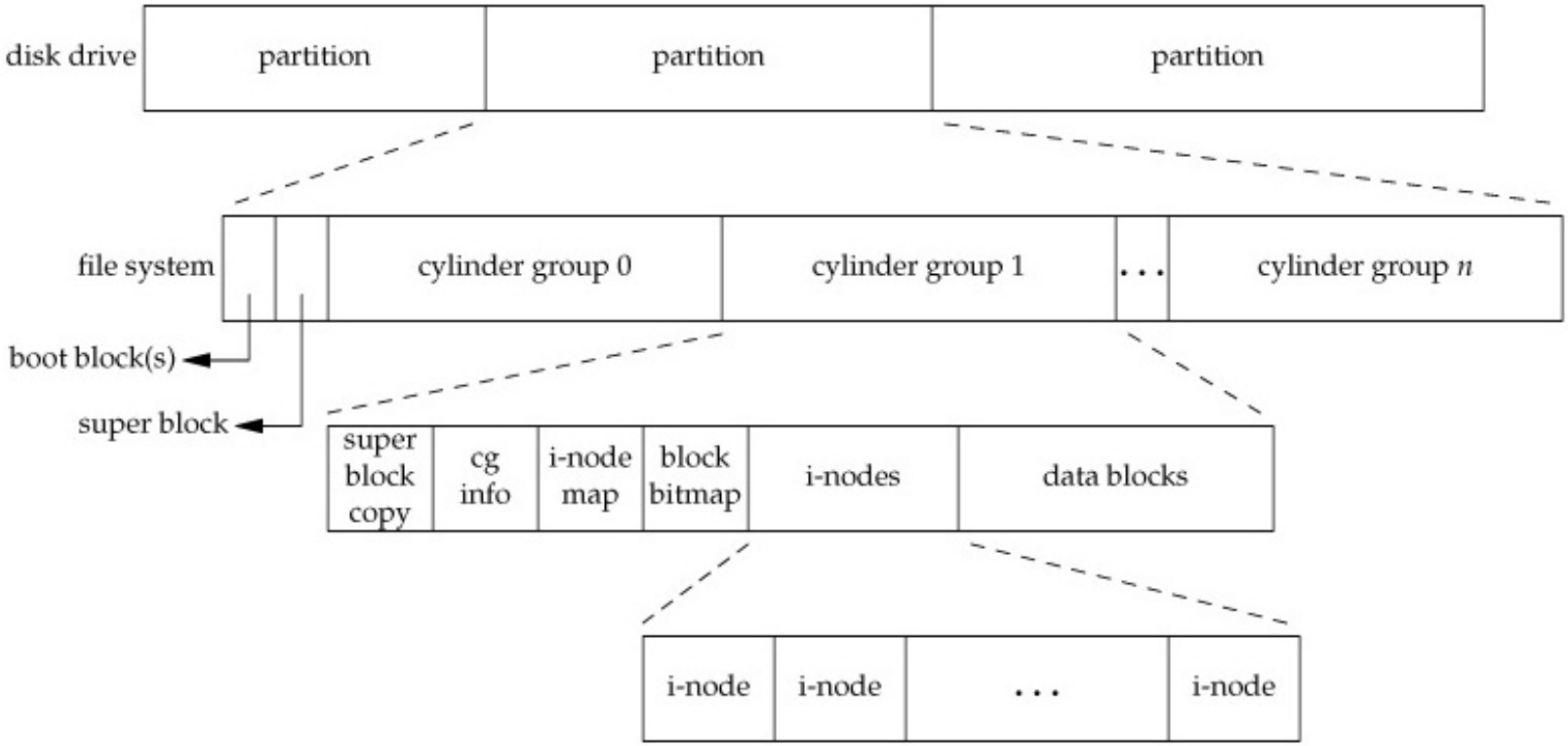


Figure 4.13 Disk drive, partitions, and a file system

File systems

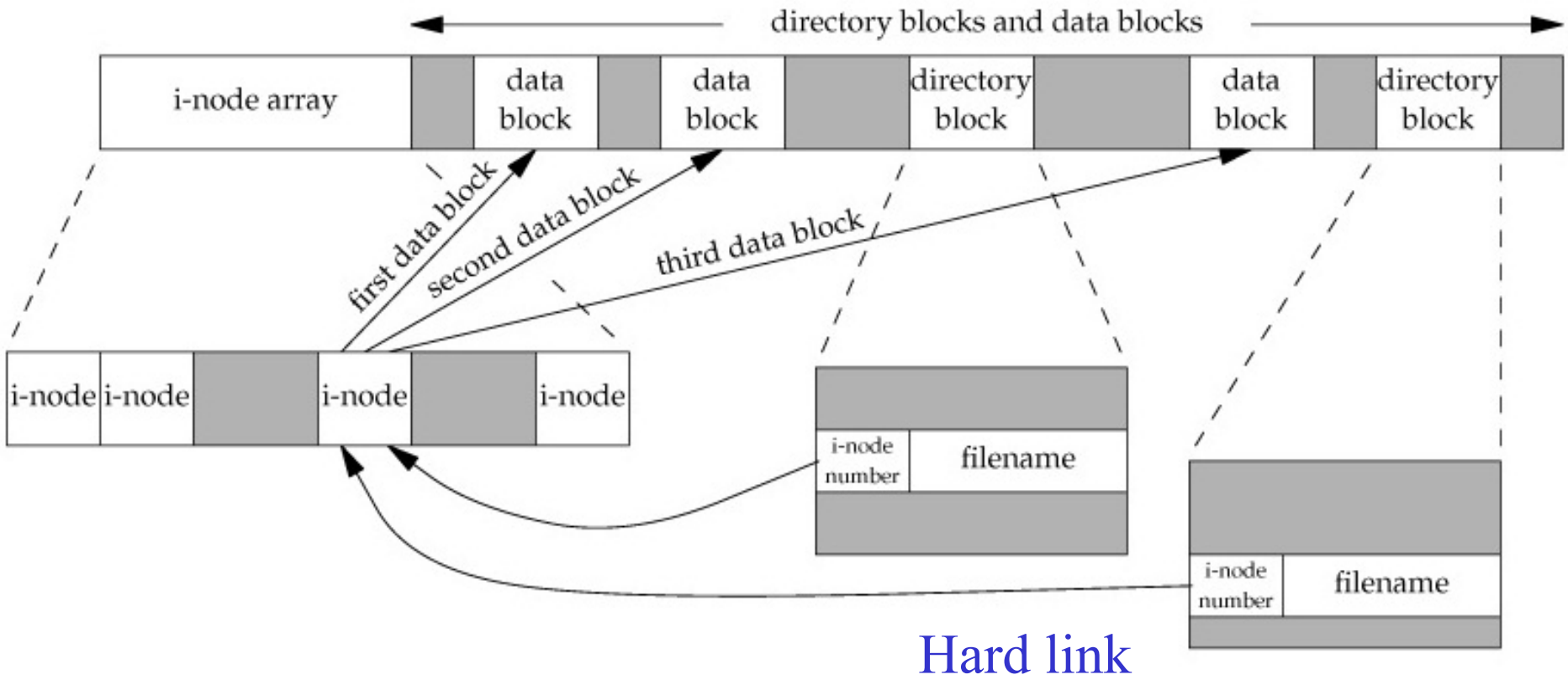


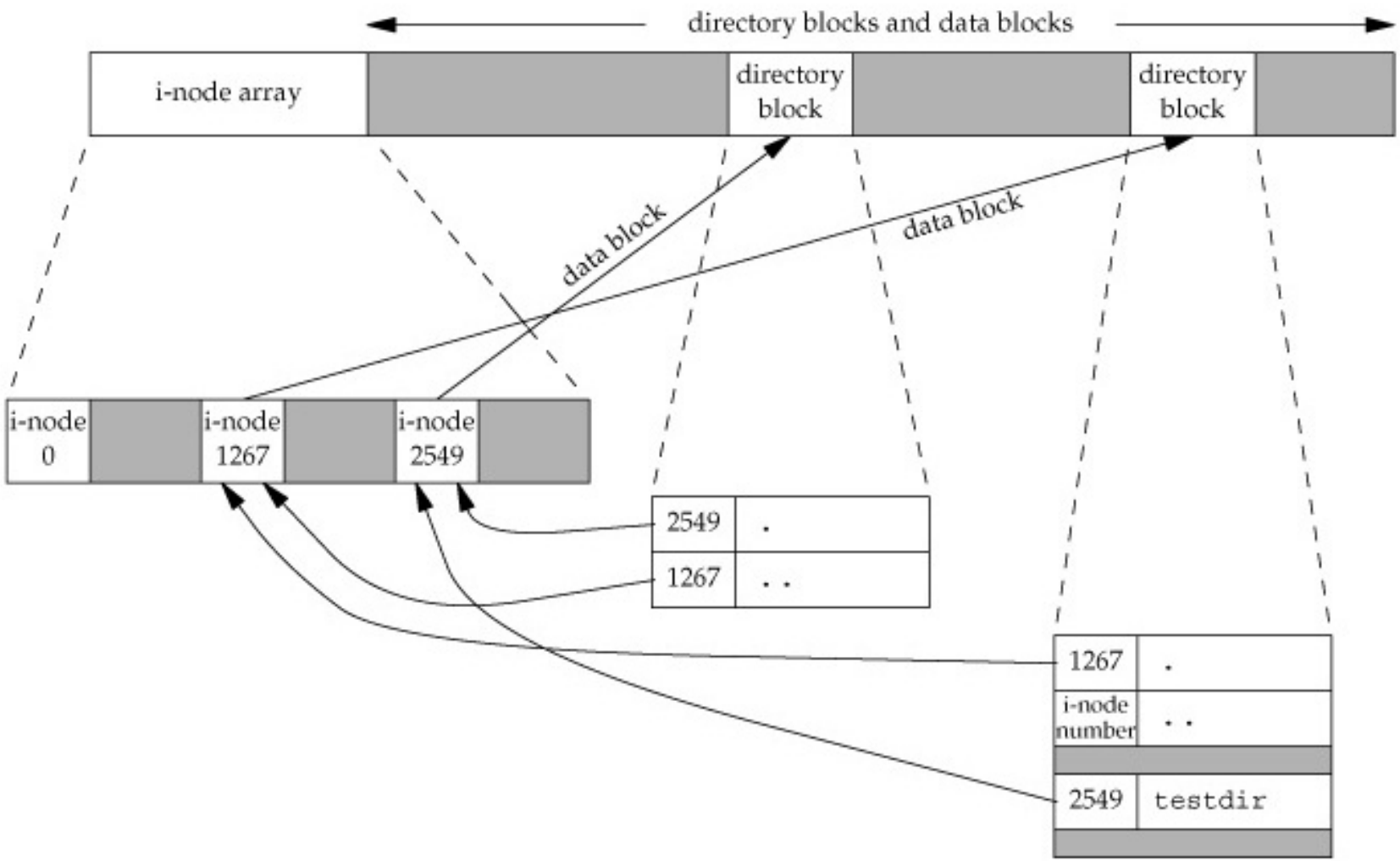
Figure 4.14 Cylinder group's i-nodes and data blocks in more detail

File systems

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

Figure 4.15 Sample cylinder group after creating the directory testdir

File system



15. link()

- make a new name for a file
- creates a new link (also known as a hard link) to an existing file.

- #include <unistd.h>
- int link(const char **existingpath*, const char **newpath*);
- Return: 0 if OK, -1 on error

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()

- deletes a name from the filesystem
- #include <stdio.h>
- int remove(const char **pathname*);
- Returns: 0 if OK, -1 on error
- For files, call unlink()
- For directories, call rmdir()

16. 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 must be empty)
- previous newname is first deleted

17. 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

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

Figure 4.17 Treatment of symbolic links by various functions

18. 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?

```
→ temp ls -l sl.c
lrwxrwxrwx 1 zhu zhu 4 10月  7 14:39 sl.c -> hl.c
```

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

- open() function follows a symbolic link

19. File Times

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

Function	Referenced file or directory			Parent directory of referenced file or directory			Section	Note
	a	m	c	a	m	c		
`chmod`, `fchmod`			•				4.9	
`chown`, `fchown`			•				4.11	
`creat`	•	•	•		•	•	3.4	`O_CREAT` new file
`creat`		•	•				3.4	`O_TRUNC` existing file
`exec`	•						8.10	
`lchown`			•				4.11	
`link`			•		•	•	4.15	parent of second argument
`mkdir`	•	•	•		•	•	4.21	
`mkfifo`	•	•	•		•	•	15.5	
`open`	•	•	•		•	•	3.3	`O_CREAT` new file
`open`		•	•				3.3	`O_TRUNC` existing file
`pipe`	•	•	•				15.2	
`read`	•						3.7	
`remove`			•		•	•	4.15	remove file = `unlink`
`remove`					•	•	4.15	remove directory = `rmdir`
`rename`			•		•	•	4.16	for both arguments
`rmdir`					•	•	4.21	
`truncate`, `ftruncate`		•	•				4.13	
`unlink`			•		•	•	4.15	
`utimes`, `utimensat`, `futimens`	•	•	•				4.20	
`write`		•	•				3.8	

Figure 4.20 Effect of various functions on the access, modification, and changed-status times

20. futimens, utimensat

- change the access time and the modification time of a file
- provide nanosecond granularity
- #include <sys/stat.h>
- int futimens(int fd, const struct timespec times[2]);
- int utimensat(int fd, const char *path, const struct timespec times[2],int flag);

  Both return: 0 if OK, −1 on error

- struct timespec {

```
        time_t tv_sec;      /* seconds */
        long   tv_nsec;     /* nanoseconds */
    };
```

utimes

- change the access time and the modification time of a file
- provide microsecond granularity

```
#include <sys/time.h>
```

- int utimes(const char *pathname, const struct timeval times[2]);

Returns: 0 if OK, −1 on error

```
struct timeval {
        long tv_sec;        /* seconds */
        long tv_usec;       /* microseconds */
    };
```

21. mkdir()

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

rmdir()

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

22. Reading Directories

- Anyone can read a dir with permissions

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

23. chdir()

- change working directory
- #include <unistd.h>
- int chdir(const char **pathname*);
- int fchdir(int *filedes*);
- Return: 0 if OK, -1 on error

Program 4.23

```
#include      "apue.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

Only changes the current working directory of the calling process to the directory specified in path.

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.

24. Device special files

- Every file system is known by its major and minor device numbers, which are encoded in the primitive system data type dev_t.
  - The major number identifies the device driver and sometimes encodes which peripheral board to communicate with;
  - the minor number identifies the specific subdevice.
- The st_dev value for every filename on a system is the device number of the file system containing that filename and its corresponding i-node.
- Only character special files and block special files have an st_rdev value. This value contains the device number for the actual device.

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);
}
```

```
$ ./a.out  /  /home/sar /dev/tty[01]
/: dev = 8/3
/home/sar: dev = 8/4
/dev/tty0: dev = 0/5 (character) rdev = 4/0
/dev/tty1: dev = 0/5 (character) rdev = 4/1
$ mount
/dev/sda3 on / type ext3 (rw,errors=remount-ro,commit=0)
/dev/sda4 on /home type ext3 (rw,commit=0)
$ ls -l /dev/tty[01] /dev/sda[34]
brw-rw---- 1 root 8, 3 2011-07-01 11:08 /dev/sda3
brw-rw---- 1 root 8, 4 2011-07-01 11:08 /dev/sda4
crw--w---- 1 root 4, 0 2011-07-01 11:08 /dev/tty0
crw-------   1 root 4, 1 2011-07-01 11:08 /dev/tty1
```