Files and Directories

Hongik University
Eunsung Jung

Disclaimer: The slides are borrowed from many sources!

File System Overview

- A disk can be divided into logical *partitions*.
- Each partition can contain a *file system*.
- A file system is divided into *multiple cylinder groups in general.*
  - *Different file systems have different structures.*

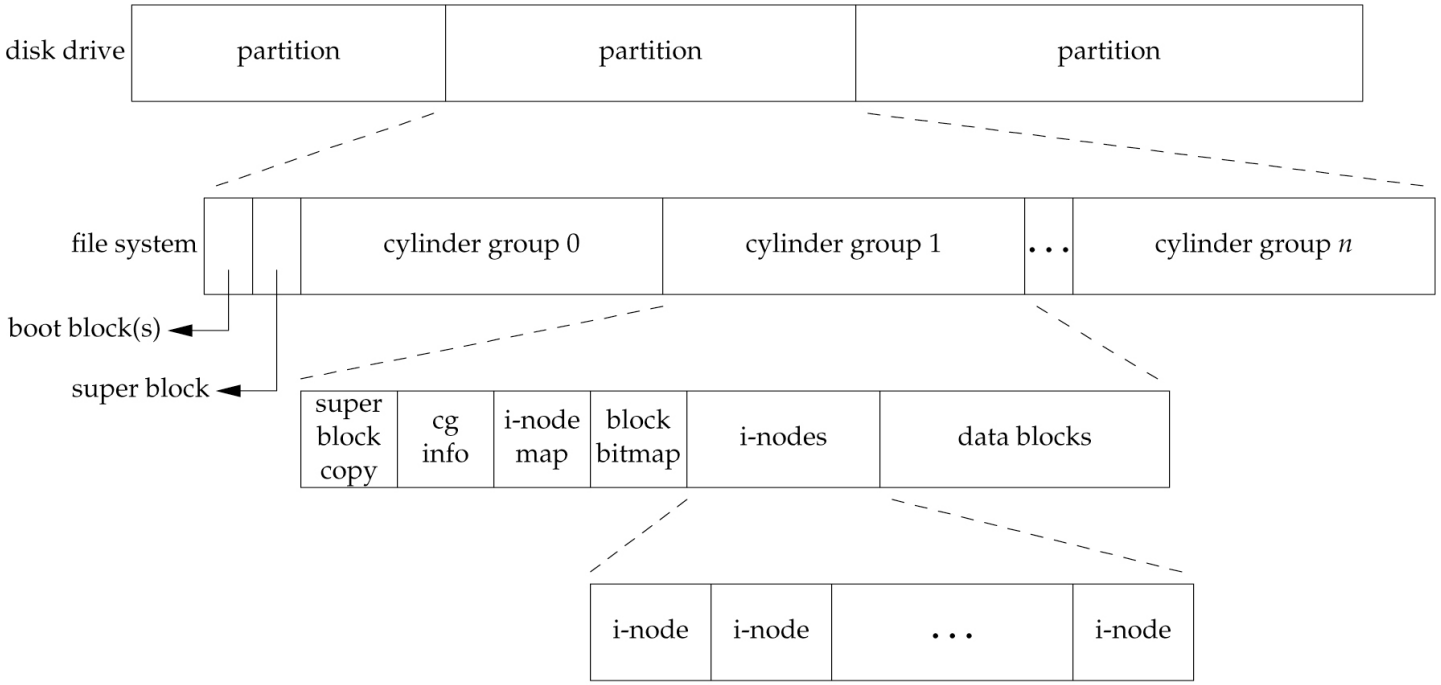


Figure 4.13 Disk drive, partitions, and a file system

Cylinder Group

- Each cylinder group contains *a list of inodes (i-list)* as well as the actual directory- and data blocks
- A directory entry is really just a hard link *mapping a "filename" to an inode*.

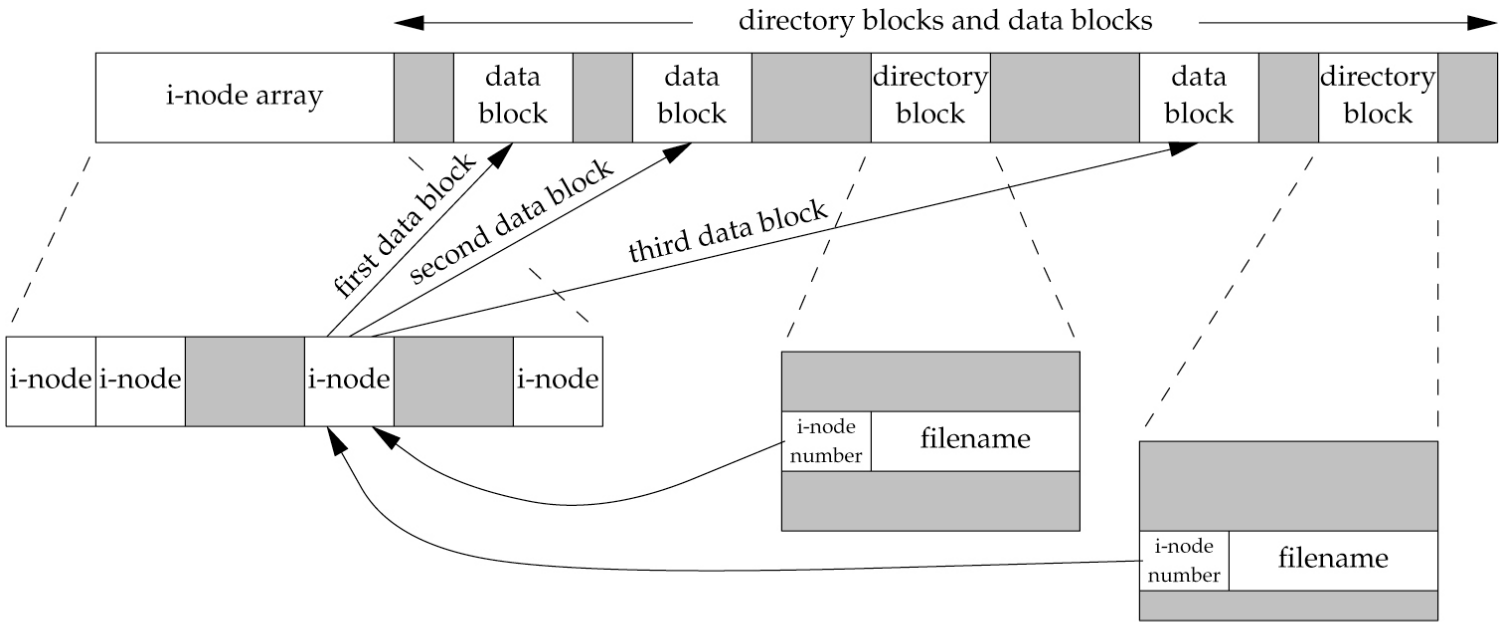


Figure 4.14 Cylinder group's i-nodes and data blocks in more detail

Hard link vs Symbolic link

- Hard link: Multiple directory entries can point to the same i-node. This is tracked by *inode count*, and the i-node can be delete only if the i-node count is 0.
- Symbolic link: The contents of the file stores the path.

```
lrwxrwxrwx  1 root       7 Sep 25 07:14 lib -> usr/lib
```

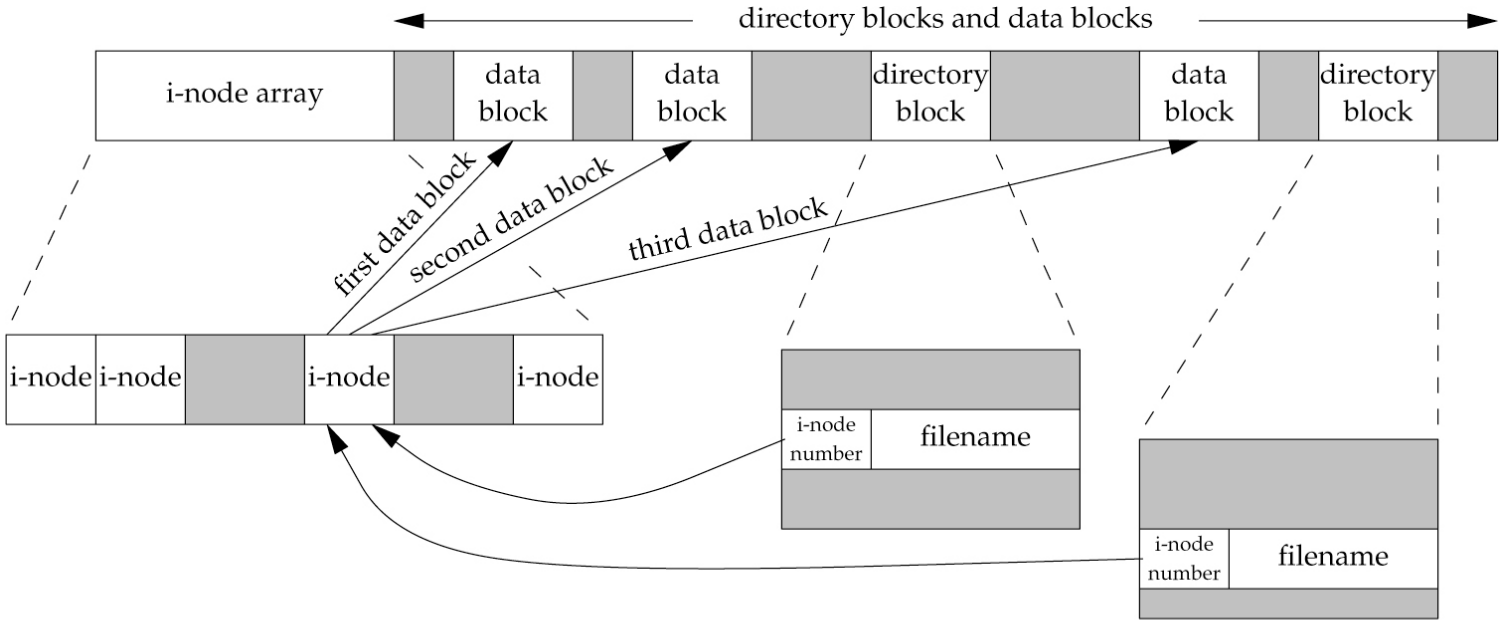


Figure 4.14 Cylinder group's i-nodes and data blocks in more detail

Directories

- Each directory contains at least two entries:
  - . (this directory), .. (the parent directory)
- the link count (st_nlink) of a directory is at least 2: i-node 2549 has . and *directory name* link.

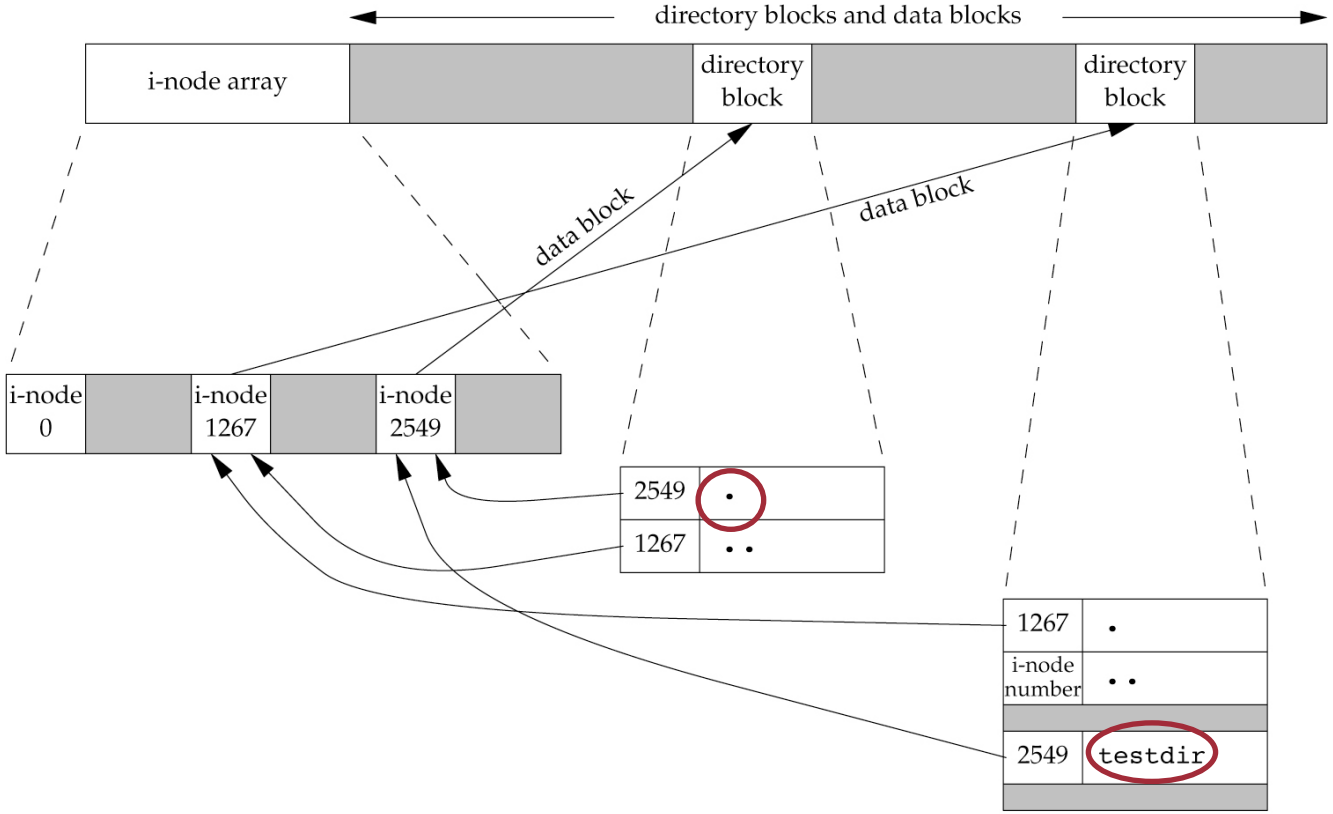


Figure 4.15 Sample cylinder group after creating the directory `testdir`

Inodes

- The inode contains most of the information found in the stat structure.
- Every inode has a link count (st_nlink): it shows how many “things” point to this inode. Only if this link count is 0 (and no process has the file open) are the data blocks freed.
- Inode number in a directory entry must point to an inode on the same file system (no hardlinks across filesystems)
- To move a file within a single filesystem, we can just ”move” the directory entry (actually done by creating a new entry, and deleting the old one).

link(2) and linkat(2)

```
#include <unistd.h>

int link(const char *existingpath, const char *newpath);

int linkat(int efd, const char *existingpath, int nfd, const char *newpath,
           int flag);
```

Both return: 0 if OK, –1 on error

- Creates a link to an existing file (hard link).
- POSIX.1 allows links to cross filesystems, most implementations (SVR4, BSD) don't.
- Only superuser can create links to directories (loops in filesystem are bad)

unlink(2) and unlinkat(2)

```
#include <unistd.h>

int unlink(const char *pathname);

int unlinkat(int fd, const char *pathname, int flag);
```

Both return: 0 if OK, –1 on error

- Removes directory entry and decrements link count of file
- If file link count == 0, free data blocks associated with file (unless processes have the file open)

rename(2) and renameat(2)

```
#include <stdio.h>

int rename(const char *oldname, const char *newname);

int renameat(int oldfd, const char *oldname, int newfd,
             const char *newname);
```

Both return: 0 if OK, –1 on error

- If *oldname* refers to a file:
  - if *newname* exists and it is not a directory, it's removed and *oldname* is renamed *newname.*
  - if *newname* exists and it is a directory, an error results.
  - must have w+x perms for the directories containing *old/newname.*
- If *oldname* refers to a directory:
  - if *newname* exists and is an empty directory (contains only . and ..), it is removed; *oldname* is renamed *newname.*
  - if *newname* exists and is a file, an error results.
  - if *oldname* is a prefix of *newname* an error results.
  - must have w+x perms for the directories containing *old/newname.*

Symbolic Links

```
#include <unistd.h>

int symlink(const char *name1, const char *name2);
```

Returns: 0 if OK, -1 on error

- File whose ”data” is a path to another file.
- Anyone can create symlinks to directories or files

Symbolic Links

- Certain functions dereference the link, others operate on 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

Symbolic Links

- How do we get the contents of a symlink? open(2) and read(2)?
  - readlink()

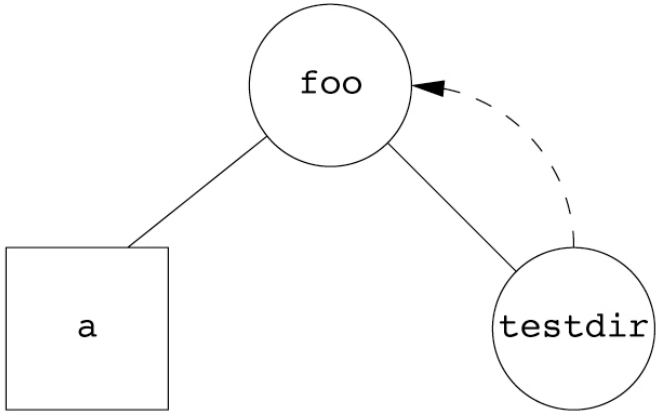
```
#include <unistd.h>

int readlink(const char *path, char *buf, size_t bufsize);

                Returns: number of bytes placed into buffer if OK, -1 on error
```

Symbolic Links: Loop

- A simple program following the link will print the following.
  - To break the loop, use unlink.




```
foo
foo/a
foo/testdir
foo/testdir/a
foo/testdir/testdir
foo/testdir/testdir/a
foo/testdir/testdir/testdir
foo/testdir/testdir/testdir/a
    (many more lines until we encounter an ELOOP error)
```

Figure 4.18 Symbolic link `testdir` that creates a loop

File Times

- Three time values associated with each file

Field	Description	Example	`ls`(1) option
`st_atim`	last-access time of file data	`read`	`-u`
`st_mtim`	last-modification time of file data	`write`	default
`st_ctim`	last-change time of i-node status	`chmod, chown`	`-c`

Figure 4.19 The three time values associated with each file

File Times: Change File Times

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

int utimes(const char *path, const struct timeval times[2]);
int lutimes(const char *path, const struct timeval times[2]);
int futimes(int fd, const struct timeval times[2]);
```

Returns: 0 if OK, -1 on error

- If times is NULL, access time and modification time are set to the current time (must be owner of file or have write permission).
- If times is non-NULL, then times are set according to the timeval struct array. For this, you must be the owner of the file (write permission not enough).

mkdir(2)

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

int mkdir(const char *path, mode_t mode);
```

Returns: 0 if OK, -1 on error

- Creates a new, empty (except for . and .. entries) directory. Access permissions specified by mode and restricted by the umask(2) of the calling process.
- Solaris 10 and Linux 3.2.0 also have the new directory inherit the set-group-ID bit from the parent directory. Files created in the new directory will then inherit the group ID of that directory. With Linux, the file system implementation determines whether this behavior issupported.

rmdir(2)

```
#include <unistd.h>

int rmdir(const char *path);
```

Returns: 0 if OK, -1 on error

- If the link count is 0 (after this call), and no other process has the directory open, directory is removed. Directory must be empty (only . and .. remaining)

Reading Directories

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *filename);
                                    Returns: pointer if OK, NULL on error

struct dirent *readdir(DIR *dp);
                    Returns: pointer if OK, NULL at end of dir or on error

void rewinddir(DIR *dp);
int closedir(DIR *dp);
                                    Returns: 0 if OK, -1 on error
```

- Read by anyone with read permission on the directory
- Format of directory is implementation dependent (always use readdir and friends)
- rewinddir resets an open directory to the beginning so readdir will again return the first entry.

Moving around directories

```
#include <unistd.h>

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

Returns: *buf* if OK, NULL on error

- Get the process's current working directory.

```
#include <unistd.h>

int chdir(const char *path);
int fchdir(int fd);
```

Returns: 0 if OK, -1 on error

- Allows a process to change its current working directory. Note that chdir and fchdir affect only the current process.

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.
  - The minor number identifies the specific subdevice.

```
$ ./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                          which directories are mounted on which devices?
/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
```

Lab#4

- Figure 4.22 Recursively descend a directory hierarchy, counting file types.