CS214-system programming

- Section 03/08 recitation 04

Yunhe Gao
yg397@scarletmail.rutgers.edu

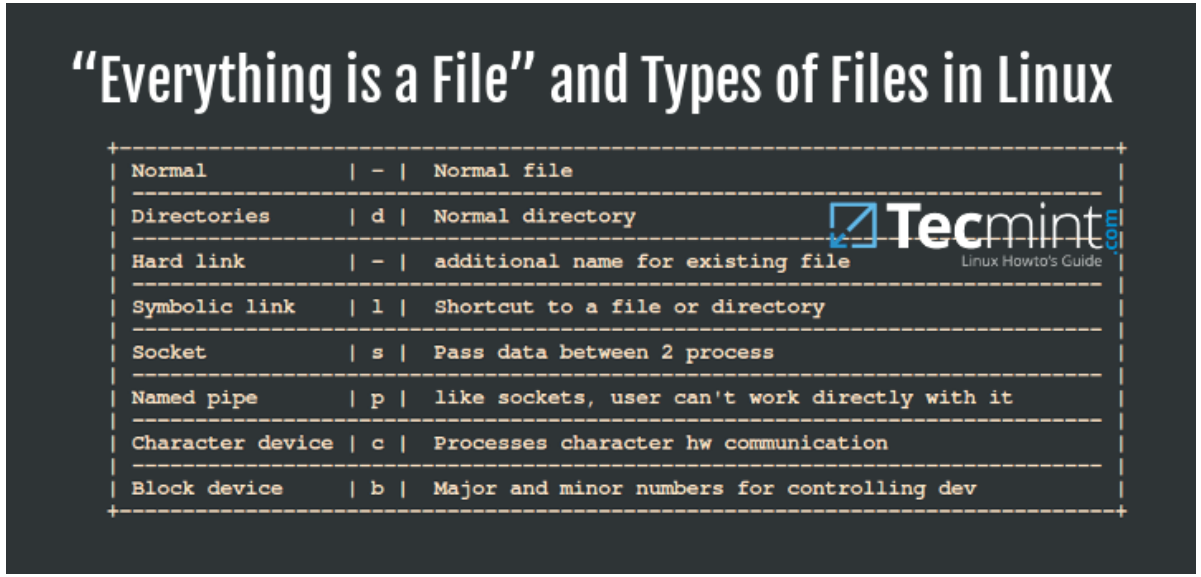
Content

- File operation
- Directory

File types in Linux system

- A file is a sequence of bytes
- All I/O devices (network, disks, ⋯) are modeled as files to generalize all input and output operations as uniform ones

- Regular file: .txt, .out, .o, .c, ⋯
- Directory: a file composing a set of file links
- Another directory can be an element in the set recursively



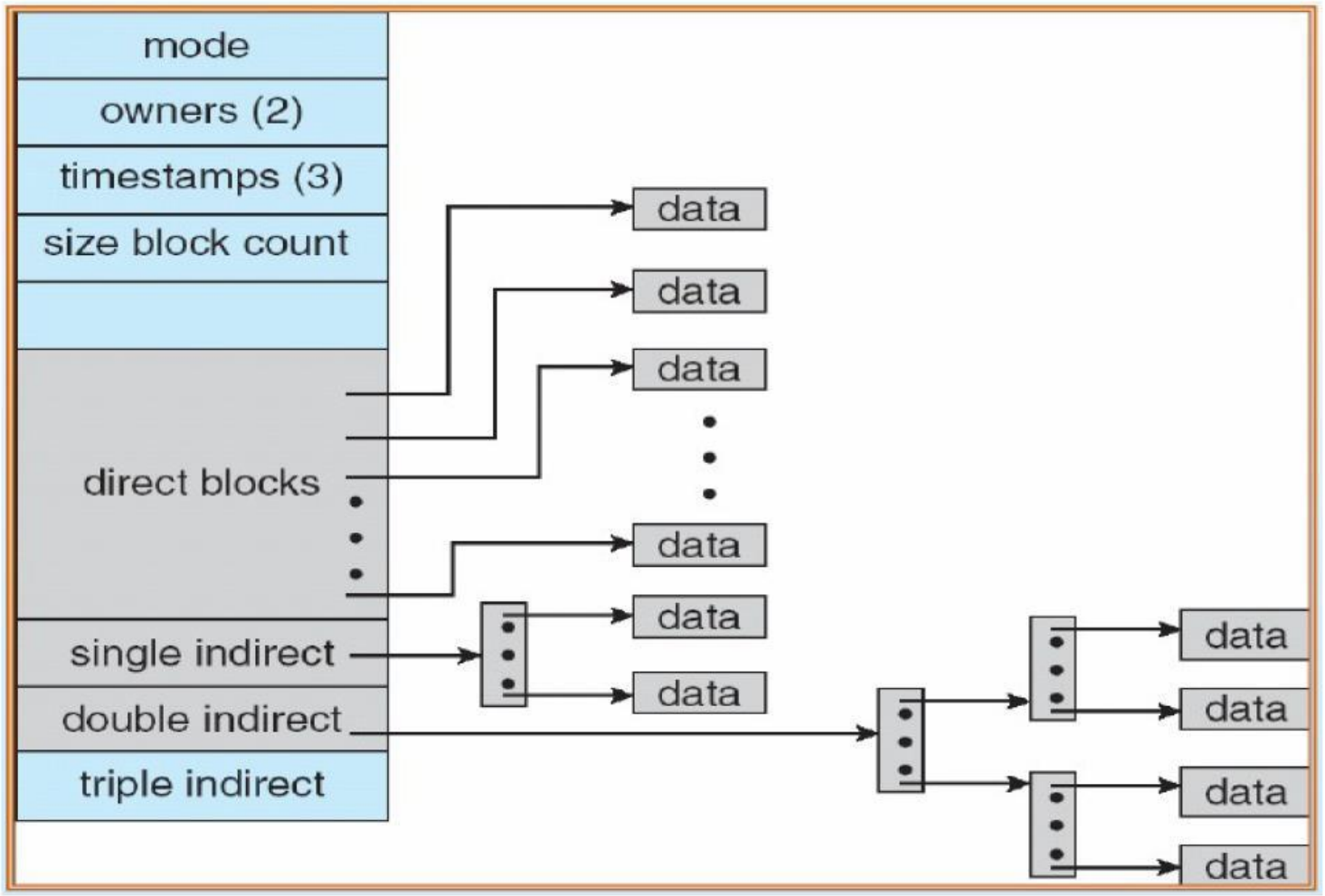
inode

Each file on the disk has an inode associated with it.

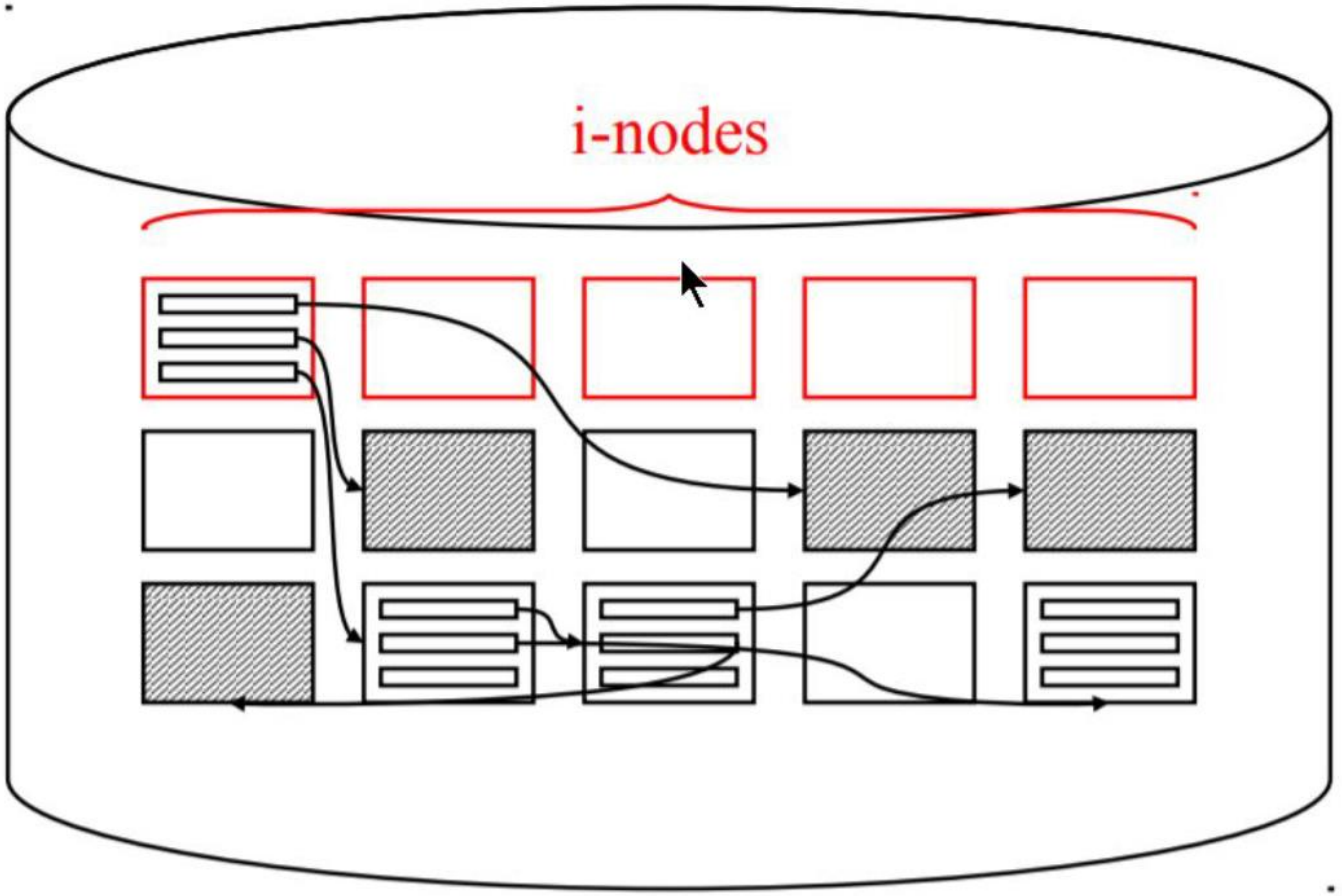
An inode is a data structure on a filesystem that stores meta-information about a file or directory.

Consists of:

- File metadata (mode, owner, info, size, etc.)
- Pointers to data blocks
  - Direct mapped pointers to data blocks
  - Indirect pointers
    - Point to blocks that contain pointers



inode



File operations in Linux

- Three basic types of operations on a file
  - Read (r)
  - Write (w)
  - Execute (x)
- A permission of an operation is associated with the relationship between current user and the file
- Three types of relationships
  - Owner (u)
  - Users in the group where the owner is in (g)
  - Users in other groups (o)

File operations

- File descriptor:
  - Unix / Linux I/O functions: access I/O devices via system calls
  - File descriptor is used for file operations, no stream is needed
  - Open(), close(), read(), write(), …
  - Low-level operations are performed directly using file descriptor

- FILE pointer:
  - Standard I/O functions: access I/O devices in a higher-level
  - A stream (represented with a FILE object) is needed to associate with an opened file
  - Implemented by invoking Linux / Unix I/O functions
  - printf(), scanf(), fopen(), fread(), fscanf(), fgets(), …
  - Streams interface could provide powerful formatted input and output functions

An opened file in Linux

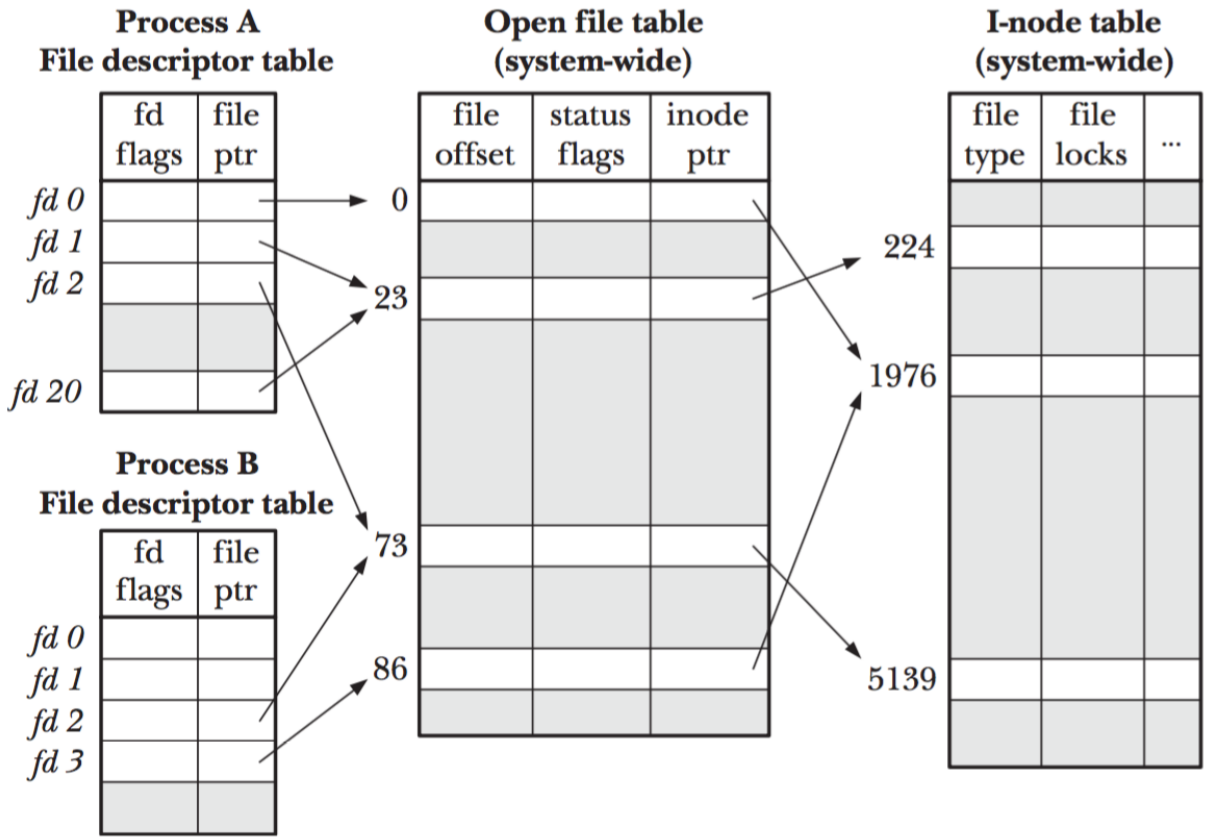


Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

Open

- Open a file: kernel does the work for the program
- After opening, a file descriptor is returned to the program

- `int open(char *path, int flags, mode_t mode);`
- Returns: new file descriptor if OK, -1 otherwise
- Import library: fcntl.h

Close Function and Errno

- Close an opened file represented by a file descriptor
- Prototype:

```
int close(int fd);
```

- Import library: unistd.h
- Returns: 0 on success, -1 on error, ***errno*** is set propriately
- Errno: indicate what goes wrong in the event of an error in the system
- Errno is set by system calls and some library functions

Read Function

- Read: copies ⩽n bytes from the current position of ***fd*** to memory location ***buf***
- Import library: unistd.h
- Prototype:

```
ssize_t read(int fd, void *buf, size_t n);
```

- Return number of bytes read if OK, 0 on EOF, -1 on error

- ssize_t: signed long in essence, which could be a negative number (-1 on error)
- size_t: unsigned long in essence, which should satisfy that is bigger or equal to 0

Write Function

- Write: copies ⩽n bytes from the memory location ***buf*** to the current file position of ***fd***
- Import library: unistd.h
- Prototype of writes

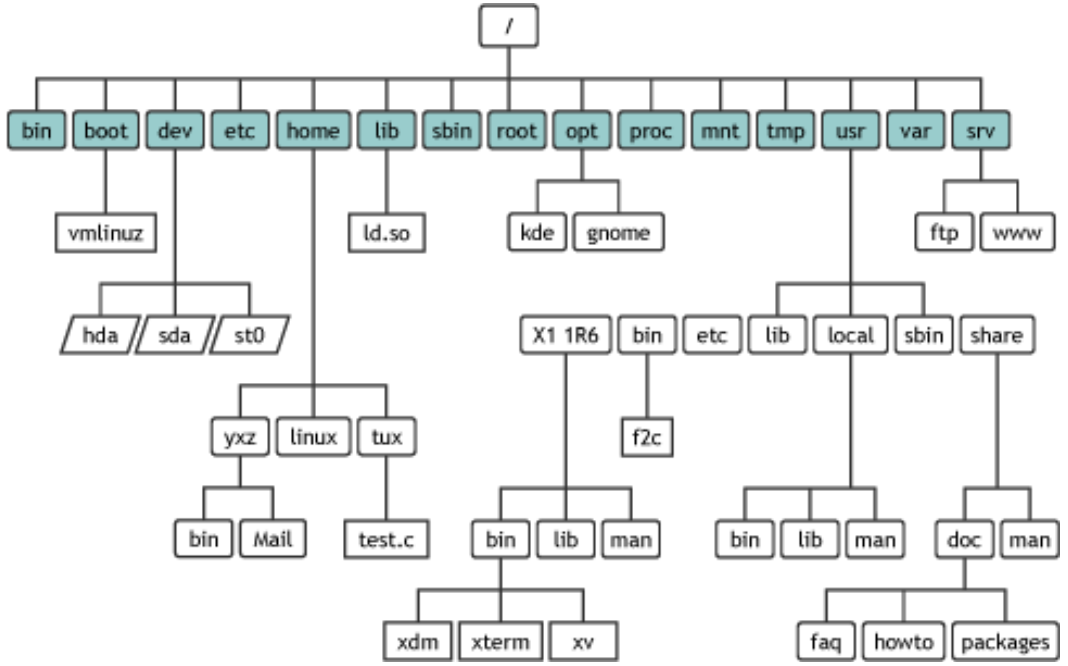
```
ssize_t write(int fd, const void *buf, size_t n);
```

- Returns number of bytes written if OK, -1 on error

Directory

- Directory: special file that contains directory entries
- In Linux, directories are organized in form of a tree and there exists a root directory

- 3 permissions associated with a directory
- Read: get the list of file links
- Write: add/remove a file link
- Execute: enter into the directory (*cd* command), access files inside it and view the metadata of the files (*ls –l* command)



http://researchhubs.com/post/computing/linux-cmd/linux-directory.html

Directory

- It is a special file that contains directory entries
- How to find is a file is a regular file or a directory, and other information?
  - Use `stat`

```
int stat(const char *pathname, struct stat *statbuf);
```

stat

- `int stat(const char *pathname, struct stat *statbuf);`

The stat structure
All of these system calls return a stat structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    nlink_t   st_nlink;       /* Number of hard links */
    uid_t     st_uid;         /* User ID of owner */
    gid_t     st_gid;         /* Group ID of owner */
    dev_t     st_rdev;        /* Device ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
       precision for the following timestamp fields.
       For the details before Linux 2.6, see NOTES. */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

stat

The following mask values are defined for the file type:

S_IFMT	0170000	bit mask for the file type bit field
S_IFSOCK	0140000	socket
S_IFLNK	0120000	symbolic link
S_IFREG	0100000	regular file
S_IFBLK	0060000	block device
S_IFDIR	0040000	directory
S_IFCHR	0020000	character device
S_IFIFO	0010000	FIFO

Thus, to test for a regular file (for example), one could write:

```
stat(pathname, &sb);
if ((sb.st_mode & S_IFMT) == S_IFREG) {
    /* Handle regular file */
}
```

Because tests of the above form are common, additional macros are defined by POSIX to allow the test of the file type in st_mode to be written more concisely:

S_ISREG(m) is it a regular file?

S_ISDIR(m) directory?

S_ISCHR(m) character device?

S_ISBLK(m) block device?

S_ISFIFO(m) FIFO (named pipe)?

S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)

S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

The preceding code snippet could thus be rewritten as:

```
stat(pathname, &sb);
if (S_ISREG(sb.st_mode)) {
    /* Handle regular file */
}
```

Directory Handling

- `DIR *opendir(char *path);`
  - Returns a pointer to an abstract DIR structure
  - Returns NULL and sets errno on failure
- `int closedir(DIR *directory_pointer);`
  - Cloose the directory & deallocate struct
  - Return -1 and sets errno on failure
- `struct dirent *readdir(DIR *directory_pointer);`
  - Obtain the next entry from the directory
  - Return NULL if there are no further entries

Directory Handling

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

       DIR *opendir(const char *name);
       struct dirent *readdir(DIR *dirp);
```

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* not an offset; see NOTES */
    unsigned short d_reclen;    /* length of this record */
    unsigned char  d_type;      /* type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* filename */
};
```

DT_BLK	This is a block device.
DT_CHR	This is a character device.
DT_DIR	This is a directory.
DT_FIFO	This is a named pipe (FIFO).
DT_LNK	This is a symbolic link.
DT_REG	This is a regular file.
DT_SOCK	This is a UNIX domain socket.
DT_UNKNOWN	The file type is unknown.

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <dirent.h>
#include <string.h>

int main()
{
    DIR *dirp = opendir(".");

    struct dirent *dp;

    while ((dp = readdir(dirp))) {
        printf("Got file %llu: %s\n", (unsigned long long) dp->d_ino, dp->d_name);


    closedir(dirp);

    return EXIT_SUCCESS;
}

```

Thanks

- yg397@scarletmail.rutgers.edu