CS214-system programming

- Section 03/08 recitation 03

Yunhe Gao
yg397@scarletmail.rutgers.edu

Content

- Makefile
- File operation

Makefile

A **Makefile** is a raw text file that specify rules on how to build a program.

- A convenient way to call gcc to compile and link large projects
- Can enable modular compilation
  - You have several objects linked together into an executable file. You make a change to one file, only one object will be regenerated rather than recompling the entire project.
- Can be named "Makefile" or "makefile"

Makefile

The format for and item in Makefile:

```
<target> : <files>
<TAB><command>
```

The second line must have a TAB character.

```
main: foo.c bar.c
     gcc –o output foo.c bar.c
```

To invoke in terminal:
`make`

Makefile

Makefiles support macros.

```
CC=gcc
CFLAGS=-Wall
OUTPUTS=output

main: main.c otherfile.c
	$(CC) $(CFLAGS) –o output main.c otherfile.c
clean:
	rm $(OUTPUTS)
```

```
make   \\ it will compile using the command under “main”

make clean    \\ it will remove all output files
```

Makefile

- Process
  1. Recursively update all dependencies of the target.
  2. Check whether the target exists
  3. If so, check whether the target is older than its dependencies
  4. If target does not exist or is older than its dependencies, run the command

Makefile

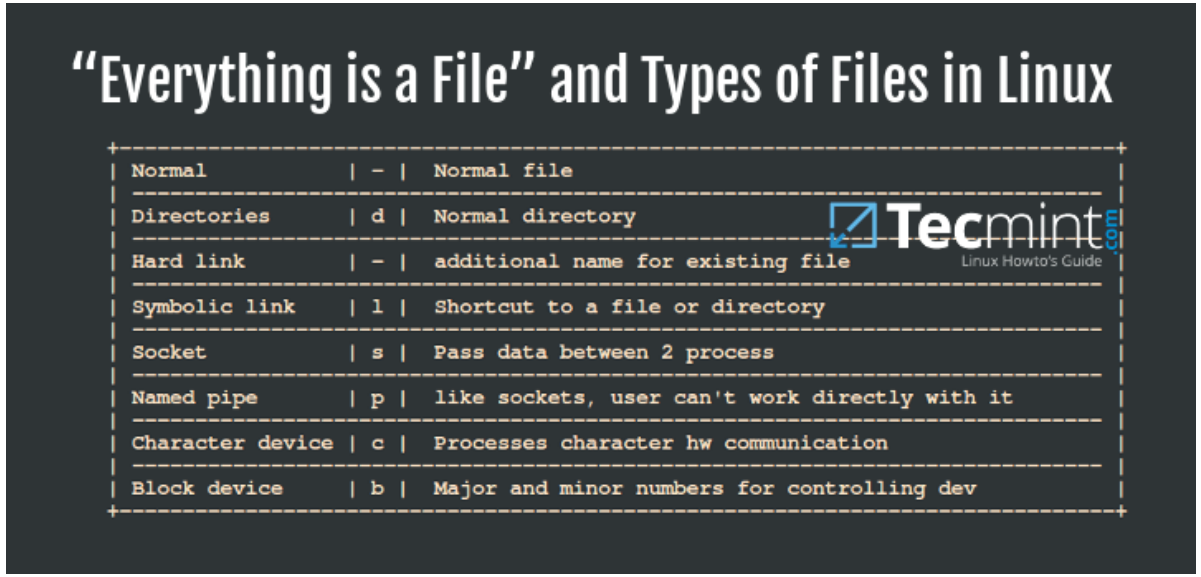
```
CC=gcc
CFLAGS=-Wall
OUTPUTS=output *.o

main: foo.o bar.o
	$(CC) $(CFLAGS) foo.o bar.o –o output
foo.o: foo.c
	$(CC) $(CFLAGS) –c foo.c
bar.o: bar.c bar.h
	$(CC) $(CFLAGS) –bar.c
clean:
	rm $(OUTPUTS)
```

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



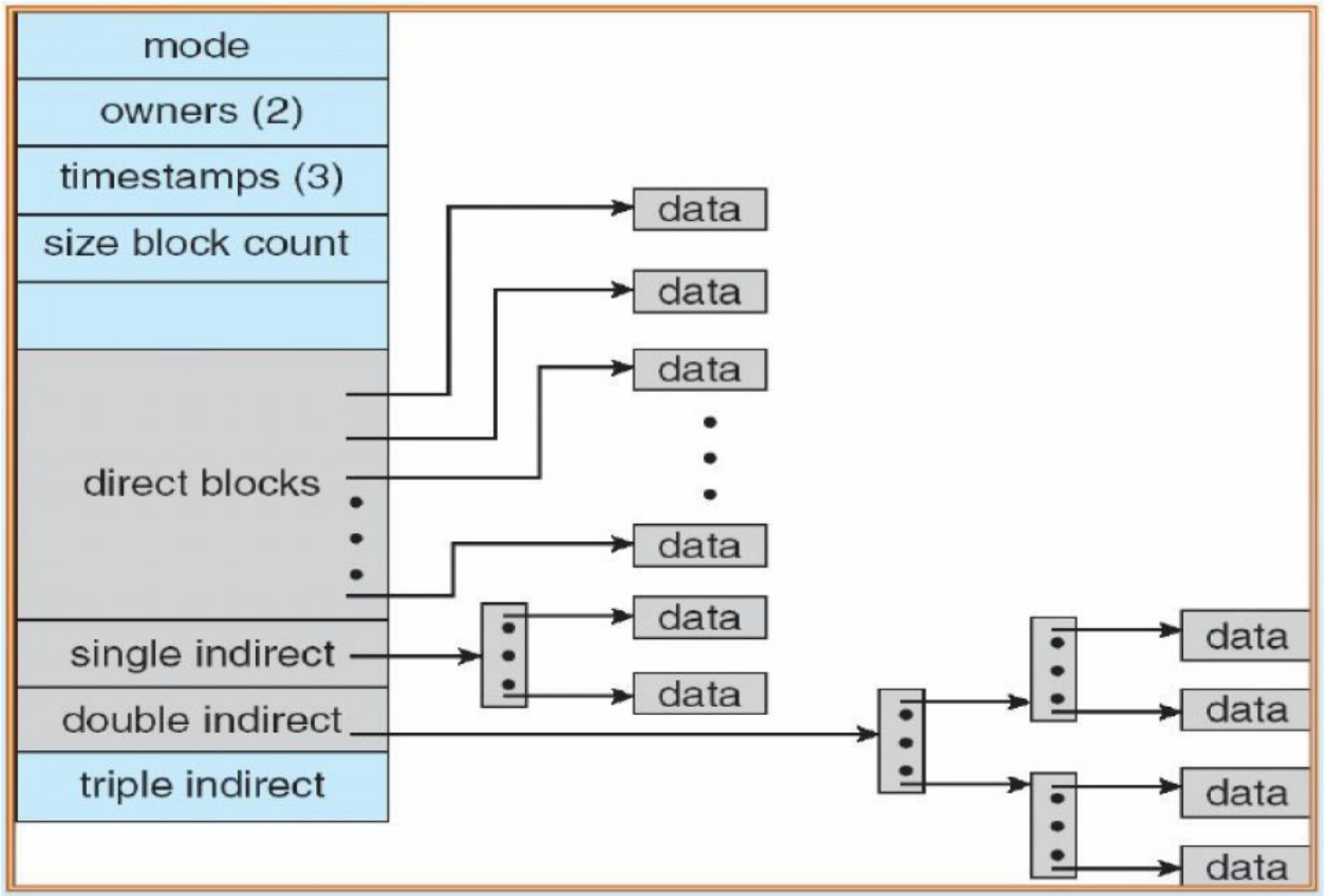
inode

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

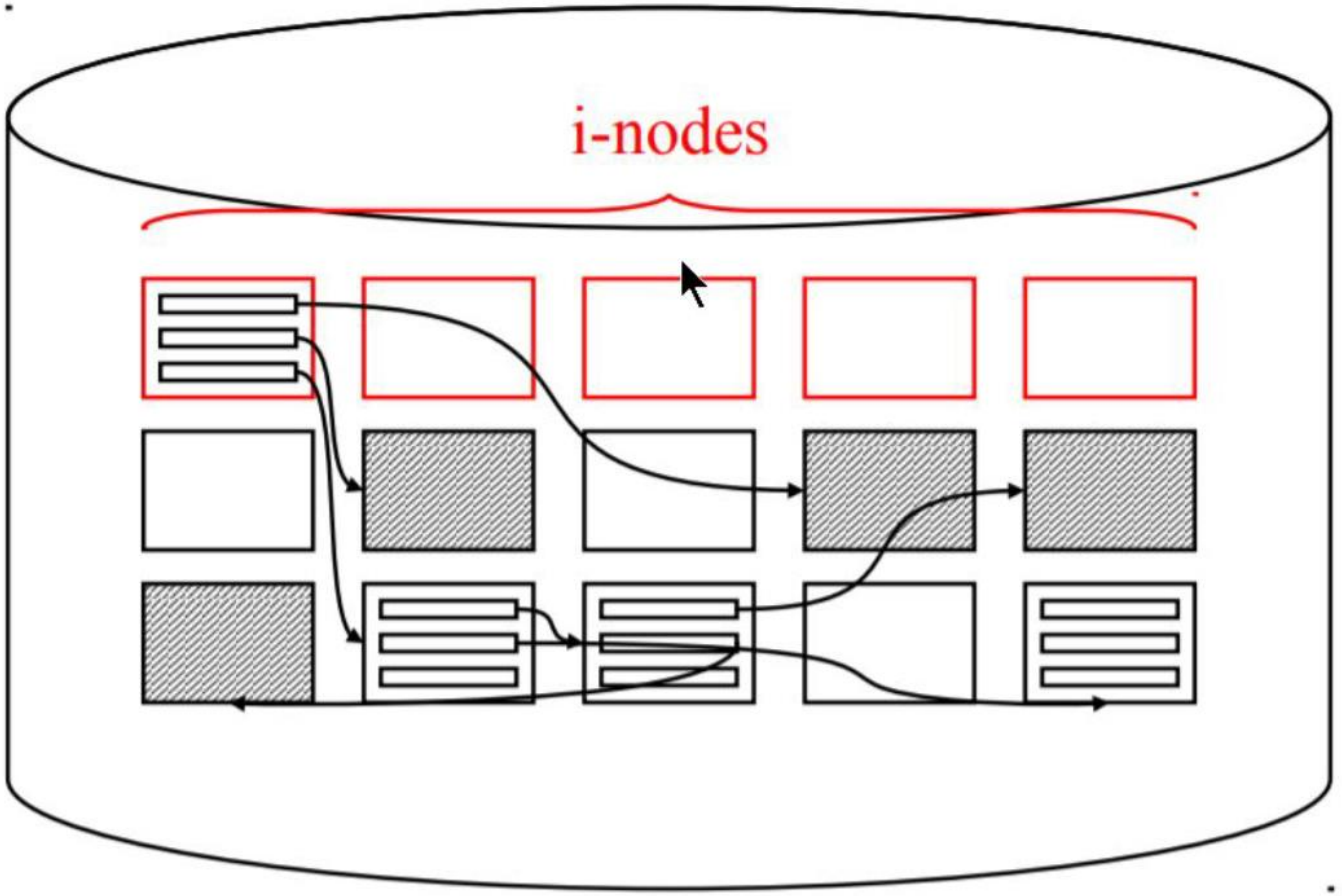
An inode is a data structure on a filesystem that stores meta-information about a file or directory.

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



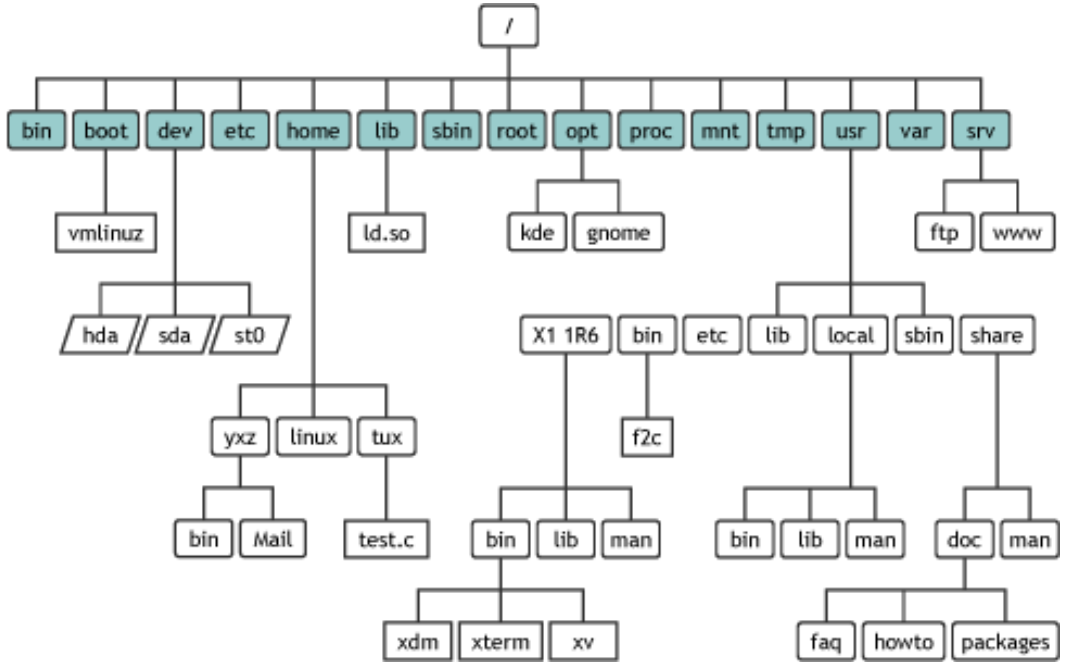
inode



Directory

- Directory: file composing a set of file links
- 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

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

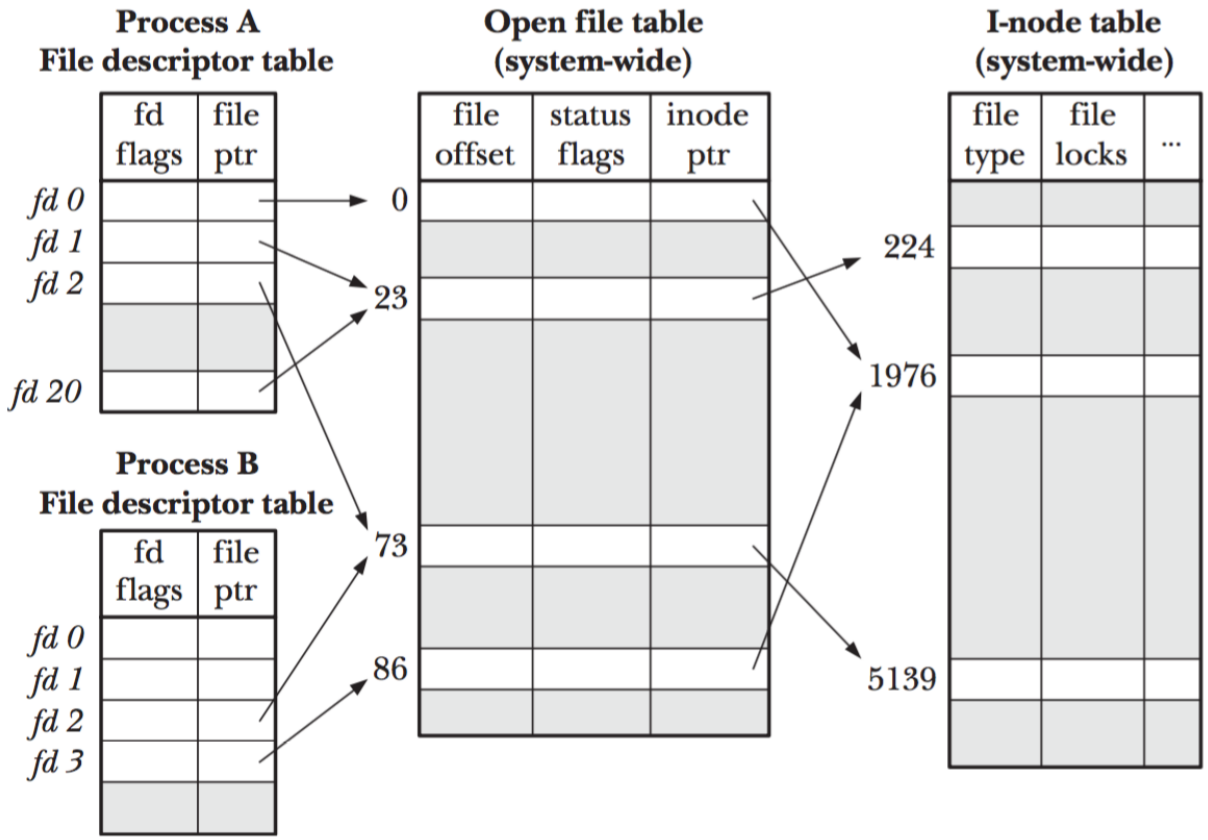


Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

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

Open: flags

- Prototype:

```
int open(char *path, int flags, mode_t mode);
```

- Flags: indicate the way a program accesses the file
- O_RDONLY, O_WRONLY, O_RDWR

- Additional instructions for operating files
- O_CREAT: if the file doesn't exist, then created an empty version
- O_TRUNC: if the file already exists, then truncate it
- O_APPEND: set the starting position to the end of the file before each write operation

- Combining usage allowed
- `fd = open("new_file", O_WRONLY|O_CREAT|O_TRUNC, 0600);`

Open: modes

- Prototype:
  - `int open(char *path, int flags, mode_t mode);`
- Modes: access permission bits

- S_IRUSR: User(owner) can read this file
- S_IWUSR: User(owner) can write this file
- S_IXUSR: User(owner) can execute this file
- S_IRGRP: Members of the owner's group can read this file
- S_IWGRP: Members of the owner's group can write this file
- S_XGRP: Members of the owner's group can execute this file
- S_IROTH: Others (anyone) can read this file
- S_IWOTH: Others (anyone) can write this file
- S_IXOTH: Others (anyone) can execute this file

Or use number representation:

- Read: 4
- Write: 2
- Execute: 1

Examples:
read/write/execute: 4+2+1 = 7
read/write: 4+2 = 6
read/execute: 4+1 = 5

S_IWUSR|S_IRGRP
240

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

Thanks

- yg397@scarletmail.rutgers.edu