

The background of the slide is a photograph of a library. At the top, there are two rows of bookshelves. The left row contains several blue books, and the right row contains many brown books. Below the bookshelves is a large grid of dark grey drawers, each with a handle and a small label. The text 'File I/O' and 'File Systems' is overlaid in the center of the grid.

File I/O

File Systems

Topics

- Can we do anything **more than just use data files?**
- How are **file systems organized?**
- What are **hard/soft links?**

The Universality of I/O

Everything is a File

- **UNIX I/O model gives access to many things via files:**

- Actual files!

- ..

- Networks

- Process information

- **/proc File System**

- Shows system and process information using **open()** / **read()** / etc.

- ..

- But they are not "real files" stored on disks.

Example: /proc file system

- | | |
|----------------------|----------------------|
| • /proc/cpuinfo | CPU info |
| • /proc/meminfo | memory info |
| • /proc/PID/status | process info |
| • /proc/PID/fd | file descriptor info |
| • /proc/PID/task/TID | thread info |

E.g., Terminal

- **Universality of file IO: Terminal**
 - 3 standard file descriptors that are always open.
 - These are..
 - **fork()** clones some opened file descriptors; so child processes also has them.

File Descriptor	Purpose	POSIX Name	stdio stream
0	Standard Input	STDIN_FILENO	stdin
1	Standard Output	STDOUT_FILENO	stdout
2	Standard Error	STDERR_FILENO	stderr

Optional: Implicitly Redirect Stdout

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main() {
    close(STDOUT_FILENO);
    open("out.log", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    printf("This goes to out.log\n");
    return 0;
}
```

Behavior: “This goes to out.log” is written to “out.log”

Why?

- close() frees file descriptor 1 (originally assigned to **stdout**)
- open() assigns the lowest-possible (not opened) integer
- printf() prints to file descriptor 1

E.g., Device Files

- Many devices have a "device file" in `/dev/`
 - This is called a node.
- Some are..
 - e.g., a mouse, a disk.
- Some are..
 - `/dev/null` provides a "black hole" of all data written to it.
 - `/dev/zero` provides infinite null characters.
 - `/dev/random` and `/dev/urandom` are pseudorandom number generators.

```
$ od -vAn -N2 -tu2 < /dev/urandom
```

E.g., /sys File System

- File IO in /sys file system
 - /sys..
e.g., various device setups, kernel subsystem info, etc.
- Examples
 - Controlling LEDs
 - Accessing secondary processors
 - Communicating to an accelerometer, etc.
- ioctl syscall
 - Extra syscall for I/O for things
 - ..
 - E.g., Change the speed of a serial port.

Disk Partitions

Disk Partitions

- ..
 - `/proc/partitions` shows the partition info.
 - In Windows, partitions are `C:`, `D:`, etc.
- A partition is typically used as a file system
 - A file system is ..
 - Many different types of file systems.
 - Each partition can have a different file system.
- E.g., BeagleY-AI board has 2 partitions on its micro-SD card:
 - One is `Fat32`, accessible to Windows and storing configuration data.
 - One is `EXT4`, used by Linux to store rest of the root file system.

Disk Partitions (cont)

- User's perspective

- ..

- starts with root directory /.

- Each partition contains a different tree
(More later when talking about mounting)

- Swap Partition

- A partition is also used as a swap space for memory management
e.g., ..

- `/proc/swaps` shows the swap space info.
(Don't always need to have swap space)

I-Nodes

I-Nodes

- A file is associated with an i-node.
 - ..
e.g., file type, permissions, owner, timestamps, etc.
 - An i-node is **identified by a number**.
 - **ls -li** shows i-node numbers (1st column).
- **stat(), lstat(), and fstat()**
 - Functions that work with file metadata mostly from the i-node.
 - Read **man 2 stat** and **man 3 stat** for more details.
 - Under the hood they invoke system calls.

Activity: I-Node

- **Activity:** use `stat()` to display if path is file or directory
 - Use command line argument to get filename (`arg[1]` likely)
 - Read `man inode`, especially about `st_mode`.
- Check out `S_ISREG(...)`, and `S_ISDIR(...)`
 - Print "**Regular file**" if it's a file.
 - Print "**Directory**" if its a directory.
 - Print "**Other**" otherwise.

Hard and Soft Links

Hard Links

- Hard links

..

- A hard link is giving another name to an existing file.

- Hard link limitations

- Cannot hard link a directory

This prevents circular links,

i.e., a child directory that links to the parent directory.

- Hard links should be **within the same file system**, because a hard link is giving another name to an existing file.

Activity: Hard Links

- **[5 min] Activity:**
 - Use `ln` to create a hard link to a file.
 - Read `man ln` to figure out how to create a hard link.
 - Run `ls -li` for both the original file and the hard link.
(They're exactly the same)
 - `ls -li` shows the number of links as well (the third column)
 - `#` links should increase as more hard links are created
- **Modify content of original file**
 - Check contents of the hard link (and vice versa).
 - They should be the same.

How rm works (aside)

- **rm only deletes the hard link.**

–..

(there's a system call used for deleting a file: `unlink()`)

(There's also a more convenient one, `remove()`)

–Only when there's no link left any more, the file gets deleted.

Soft Links (Symbolic Links)

- **Soft links**

..

- Unlike a hard link, ..

- The content of the file is the path to the original file.

- There's a system call **symlink()**.

- **No limitations like hard links**

- Sym links are allowed for directories.

- Sym links do not have to be within the same file system.

Activity: Soft Links

- (5 min) Activity
 - Create a sym link with `ln -s`
 - Run `ls -li`
 - They each have a unique i-node number, meaning they are two different files.
 - The hard link count does not change even if you create a sym link: it's because it's a different file.
 - The sym link will point to nothing if the original gets deleted.
 - This is called **a dangling link**.

Optional:

Bits - setuid, setgid, sticky

Setuid / Setgid bits

- **Program Permission**

- Normally, programs you run will run with your permission.

- **Setuid bit**: if set, the user that runs the program can act as the owner of the program.

- E.g., `passwd` sets a user's password.

It must write to the password file (`/etc/shadow`), which is owned by the root.

- So, use the **setuid bit**:

- When a user runs `passwd`, the program can act as root to modify the password file.

- **Setgid bit**: if set, the user that runs the program can act as if the user belonged to the group of the program.

Sticky Bit

- Sticky bit:
 - Can be set on a **shared directory** for better control.
 - When set, **only able to delete/rename file if:**
 - a) **you own it**
 - b) **you have write permission for it**
(It affects the directory, not the file access permissions)

Sticky Example

- **Situation 1: Regular Directory**

- Create a **shared_photos/** directory that is write-open for others (e.g., **rw-rw-rw-**).
- User **dr-evil** creates a file **selfie.jpg** in it.
- User **boogieman** can delete **selfie.jpg**.

- **Situation 2: Sticky Bit!**

- Set sticky bit on **shared_photos/**
chmod +t shared_photos/
- User **dr-evil** creates a file **selfie.jpg** in it.
- User **boogieman** **cannot** delete **selfie.jpg**.

VFS - Virtual File System

and

Mount/Unmount

VFS (Virtual File System)

- VFS (Virtual File System)

..

- Interface includes: **open**, **read**, **write**, **close**, etc.
VFS in kernel define a function to handle each.

- It's not a file system of real files,

..

- If a file system implements this interface,
it can be used as a Linux file system.

- E.g.**,: **/sys**, **/proc**, **/dev**, ...

Mounting

- Linux presents all file systems as a single tree
 - Starts at root directory /
- In reality, this single file tree
 - ..
- Recall:
 - A partition contains a file tree
 - There can be multiple partitions on a single disk.
 - There can be multiple disks for a single machine.

Mounting and Unmounting

- Mounting

..

- All file systems (from different partitions/disks) are mounted and form a single file tree.

- `mount` command mounts a file tree (a file system) to a specific directory

- This target directory is called a **mount point**

- The `mount` command also shows the current setup. (Shows the same information as `/proc/mounts`).

- The `umount` command unmounts a file system.

Summary

- **Everything is a file**
 - Use file operations to access almost anything.
 - **/proc** for process info
 - **/dev** for devices
 - **/sys** for system info
- **Partitions** split up disks
- **I-Nodes** used for **meta data** about each file/directory.
- **Hard/soft links** allow two entries for one file.
- **Mounting** places one file tree inside another.