System Security

Files as the True OS Atom

Mohamed Sabt

Univ Rennes, CNRS, IRISA

2023 / 2024

Part I

Files and Directories

Abstractions

- Virtualization of the CPU: Process
- Virtualization of the memory: Address space

- Virtualization of the (persistent) storage: file and directory.

Files 1/2

- From a user's perspective
  - A file is simply a linear array of bytes, each of which you can read or write.
  - Files are persistent across reboots and power failures.

- From OS perspective
  - Map bytes as collection of blocks on storage device.

- Each file has some kind of low-level name, often referred to as its **inode number**.

Files 2/2

- Persistent data on storage is organized as files.

- Files are logical units organized by a file system.
  - The file system maps logical information to bits and bytes on the storage device.

- The file system runs in kernel space
  - Access to files goes through system calls

Unix Design

- Every very persistent resource is accessed through a file
- Consequence of “everything is a file”:
  - User-space processes can operate on files only through syscalls
  - OS can check for each syscall (kernel-space operation), whether the operation is permitted

Files Vs Memory 1/2

- Disk provide persistent storage. Data won’t go away after reboot.
- Disks are much slower than memory:
  - Latency.
  - Throughput.
- Capacity of disks is usually much larger.

Files Vs Memory 2/2

- Every memory location has an address that can be directly accessed.

- In files, everything is relative
  - A location of a file depends on the directory in which it is stored.
  - Open the file before any access and close the file after all accesses are complete.
  - A pointer must be used to store the current read or write position within the file. E.g. To read a byte in a specific file.

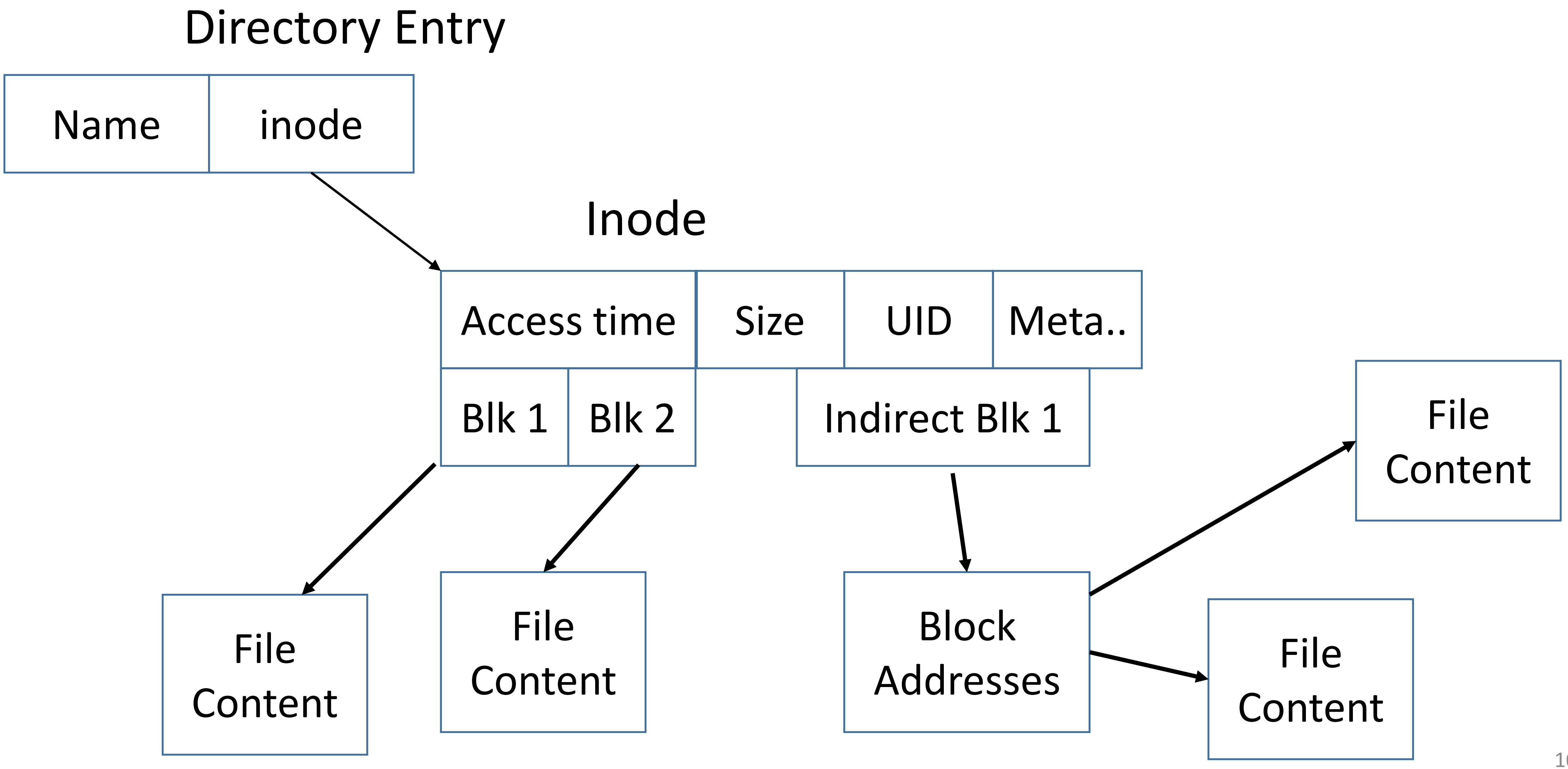
Directory

- Directories are files of type directory (i.e., with metadata type “directory”).
  - Directory contents are quite specific: it contains a list of (user-readable name, low-level name) pairs.

- Each directory has some kind of low-level name, often referred to as its **inode number**.

- By placing directories within other directories, users are able to build an arbitrary **directory tree** (or directory hierarchy), under which all files and directories are stored.

File System Structure



Files Metadata (inodes)

- Each file has an inode containing metadata about the file. An application can retrieve this metadata using ***stat***.
- The following is a list of the information typically found in, or associated with, the file inode:
  - Name. the only information kept in human readable form.
  - Identifier. A number that uniquely identifies the file within the file system. (also called inode number → ls -i).
  - Type. File type.
  - Location. Pointer to location of file on device.
  - Size.
  - Protection. Access control information. Owner, group, permissions, etc.
  - Monitoring. Access time, etc.

Example stat

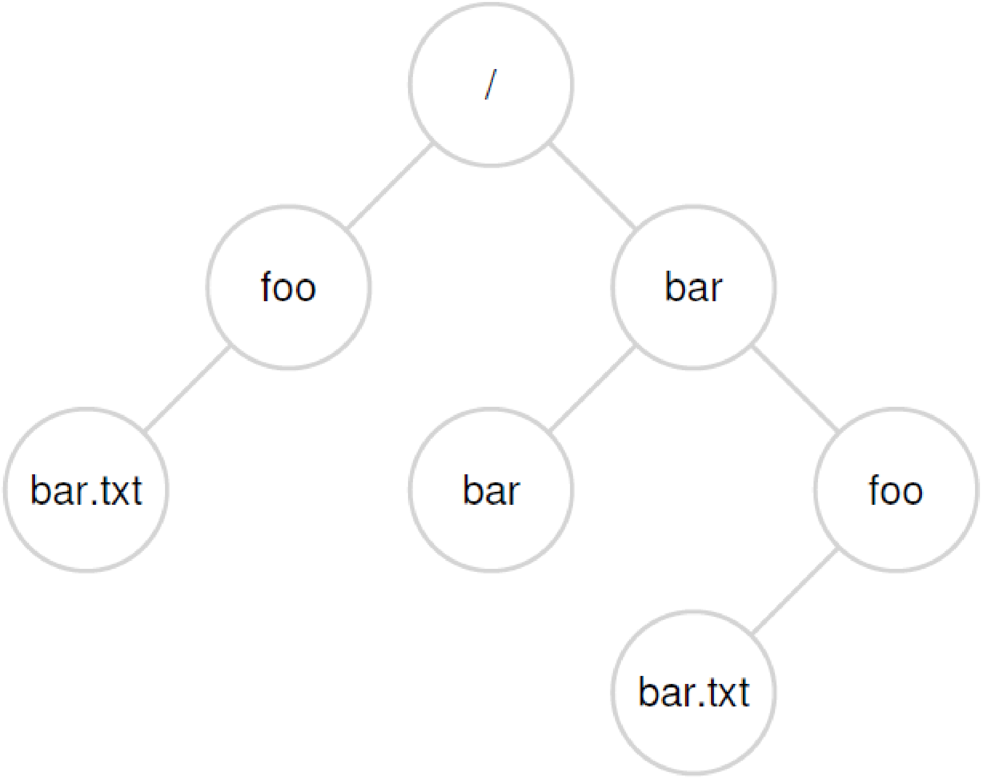
```
sabtmoha@sabtmohav2:~$ stat test.txt
  File: test.txt
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 820h/2080d      Inode: 148911      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/sabtmoha)   Gid: ( 1000/sabtmoha)
Access: 2023-04-24 16:08:53.336639421 +0200
Modify: 2023-04-24 16:08:53.336639421 +0200
Change: 2023-04-24 16:08:53.336639421 +0200
 Birth: 2023-04-24 16:08:53.336639421 +0200
```

Directory Tree

- The directory hierarchy starts at a root directory (in Unix-based systems, the root directory is simply referred to as /) and uses some kind of separator to name subsequent sub-directories until the desired file or directory is named.

- Directories and files can have the same name as long as they are in different locations in the file-system tree.

- Special directories
  - / → root
  - . → current directory
  - .. → parent directory



Unix Directories

- /: The root directory
- /bin: Essential low-level system utilities
  - /user/bin: higher-level system utilities and application programs
  - /sbin: superuser system utilities
- /lib: program libraries (collection of system calls that can be included in programs by a compiler) for low-level system utilities.
  - /usr/lib: program libraries for higher-level user programs
- /tmp: Temporary file storage space (can be used by any user)
- /home: user home directories containing personal file space for each user. Each directory is named after the login of the user.
- /etc: Unix system configuration and information files.
- /dev: hardware devices.

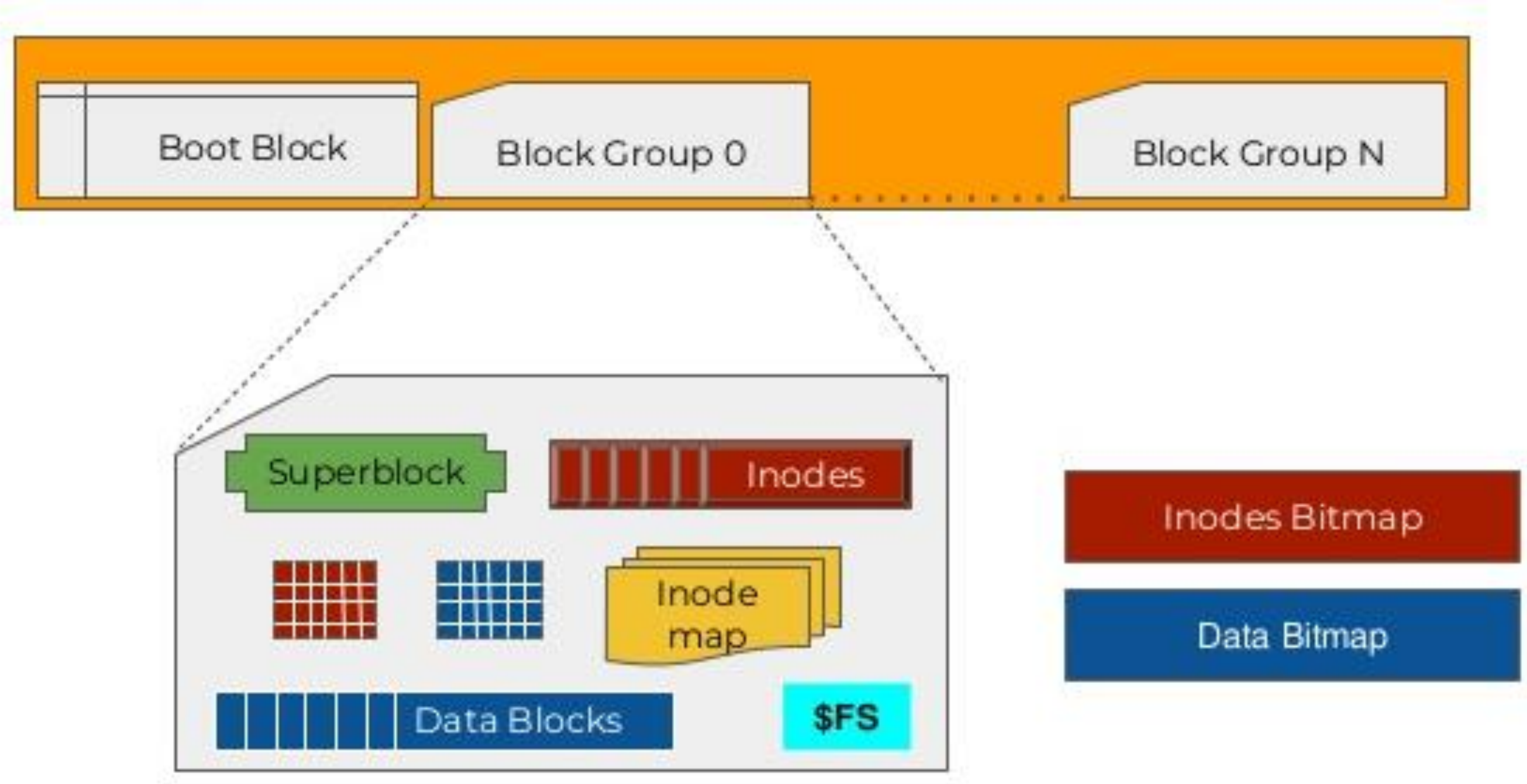
Files Naming

- Names of files consist of two parts separated by a period.
- The first part is an arbitrary name.
- The second part of the file name is usually used to indicate the file type
  - whether it is C code (e.g., .c), or an image (e.g., .jpg), or a music file (e.g., .mp3).
  - However, this is usually just a **convention**
- man basename

Part II

File Constructs

File System Constructs



Blocks

- 💿 Disks are divided into blocks of fixed size.
- 💿 Typically 4 KB blocks.
- 💿 Numbered from 0 to N-1.

Blocks Terminology

- Superblock: this holds data about the system like the version, block size, and the inode number of the root directory.
- Block Bitmap: each bit tells if the corresponding block is free.
- Data Blocks: the files are stored here, but split amongst different blocks.

Files Philosophies

- The 3 views of a file
  - File name (human readable)
  - Inode and device number (operating system)
  - File descriptor (process view)

- Types
  - Typed files: System defines all possible file types (e.g., text document, source file, html file). File type set at creation, file type specifies operations.
  - **Untyped files**: File is a sequence of bytes. System does neither understand nor care about contents. File operations apply to all files

File Descriptor

- One important aspect of open() is what it returns: a file descriptor.

- A file descriptor is just an integer, private per process, and is used in Unix systems to access files;
  - thus, once a file is opened, you use the file descriptor to read or write the file, assuming you have permission to do so.

- You can think of a file descriptor is as a pointer to an object of type file;
  - once you have such an object, you can call other “methods” to access the file, like read() and write().

Special File Descriptors

Integer value	Name
0	Standard input
1	Standard output
2	Standard error

Open File Table

- The OS tracks all the opened files in the system.

- Each entry in this table tracks
  - which underlying file the descriptor refers to,
  - the current offset,
  - other relevant details such as whether the file is readable or writable.
  - Reference count (tracks number of processes that have opened the file)

- A call to open() creates a different entry in the OFT.

Per Process Table

- File descriptors are managed on a per-process basis.
  - To track which files are opened.

- The PCB contains an array of all opened files.

- Each entry of this array points to an entry in the system wide table.

Part III

File System Interface

File Operations

- Files operations include:
  - Open a file.
  - Close a file.
  - Read file content.
  - Write new content into a file.
  - Get/Set file attributes.

Directory Operations

- Directory operations include:
  - Create a file in the directory
  - Delete a file from the directory
  - List a directory's contents
  - Rename a file in the directory
  - Traverse the file system

System Call: Create

- The *creat* system call create a new file in the filesystem.

- fd = creat(pathname, mode), where
  - If pathname of an existing file is passed to *creat*, it will truncate the file (set its size to 0 if permissions allow) , while ignoring mode.

- Fun story: Ken Thompson was asked what he would do differently if he were redesigning Unix, he replied: “I’d spell creat with an e.”

System Call: Open

- Open is the first step to access data in a file.
- fd = open(pathname, flags, mode), where
  - flags indicate the type of open (reading or writing),
  - mode gives the permissions of the file is being created.
  - The returned file descriptor is nothing but the index of the entry in the user file descriptor table.
  - Example: int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);

- Entries in the user file descriptor point to unique entries in the file table, even if the same file is opened twice.

System Call: Close

- A process closes an open file where it no longer wants to access it.
- ret = close(fd)

- When closing a file, the kernel first deals with the entries in the user file descriptor and the file table.
- When a process exists, the kernel examines the active user file descriptors and closes each one.
  - Hence, no process can keep a file open after its termination.

System Call: Read/Write

- number = read/write(fd, buffer, count), where
  - fd is the descriptor returned by open,
  - buffer is the address of the data where the data will be read,
  - count the number of bytes to be read/written.
  - It returns how many bytes where successfully read/written.

System Call: lseak

- off_t lseek(int fd, off_t offset, int whence), where
  - fd is the descriptor returned by open,
  - which positions the file offset to a particular location within the file,
  - Whence: enum to interpret how the offset is computed.

- This syscall explicitly modifies the current offset inside the OFT.

System Call: Dup

- newfd = dup(fd)

- The *dup* system call copies the given file descriptor to the first free slot in the user file descriptor table, and returns the new file descriptor.

- Since it duplicates the entry in the user file descriptor, it increments the reference count in the file table.

System Call: Change Owner/Mode

- Changing the owner or mode (access permissions) of a file are operations on the inode.

- chown(pathname, owner, group)
- chmod(pathname, mode)

System Call: Stat and Fstat

- The system calls *stat* and *fstat* allow processes to query the status of files, returning information such as the file type, file owner, access permissions, number of links, etc.
- stat(pathname, statbuffer)
- fstat(fd, statbuffer), where
  - statbuffer is the address of the data that will get filled in the call.
- The system calls simply write the fields of inodes into the statbuffer.

System Call: Removing Files

- unlink(pathname)
  - Removes a directory entry for a file.

System Call: Link

- The *link* system call links a file to a new name in the directory structure.
  - It creates a new directory entry which points to an existing inode.
- link(source file name, target file name)
- After linking the files, the kernel does not keep track of which file was the original one. Therefore, no name is treated specially.
- **Reference count** within the inode number.
  - This reference count (sometimes called the **link count**) allows the file system to track how many different file names have been linked to this particular inode.
- Even a superuser is not allowed to link directories (why? See TD).

Symbolic (or Soft) Links

- A special kind of files.
- Soft link: a directory entry points to a file that contains a file name, the OS resolves the file name when it is accessed.
- Beware of dangling references!!
  - Soft links don't get updated when the target is moved.

Main File Manipulation Commands

- `touch` Update the access and modification times of FILE to the current time, or create an empty file.
- `mkdir mydir` creates a directory (where you are)
- `rmdir mydir` deletes an empty directory
- `rm -rf` forces to recursively delete a non empty directory.
- `cp file1 file2` copies file1 on file2 (overwriting if it already exists, creating a new one otherwise)
- `rm file1` removes file1
- `mv file1 file2` moves file1 on file2

ls

- `ls` can be used to inquire about various attributes of one or more files or directories.
- You must have read permission to a directory to be able to use the ls command on that directory and files under that directory.
- By default, the list of files within a directory is sorted by filename.
  - You can modify the sort order by using some of the flags.
- You should be aware that the files starting with . (period) will not processed unless you use the –a flag with the ls command.

Cat (also check bat)

- `cat` is used to display a text file or to concatenate multiple files into a single file.

- By default, the cat command generates outputs into the standard output and accepts input from standard input.

- The cat command takes in one or more filenames as its arguments. The files are concatenated in the order they appear in the argument list.

The Ranger Package

- Ranger is a minimal file manager that allows you not only to navigate through the files but also to preview them.

Part IV

Making and Mounting
A File System

Make File System

- To make a file system, most file systems provide a tool, usually referred to as mkfs (pronounced “make fs”).

- The idea is as follows: give the tool, as input,
  - a device (such as a disk partition, e.g., /dev/sda1),
  - and a file system type (e.g., ext3),
  - and it simply writes an empty file system, starting with a root directory, onto that disk partition.

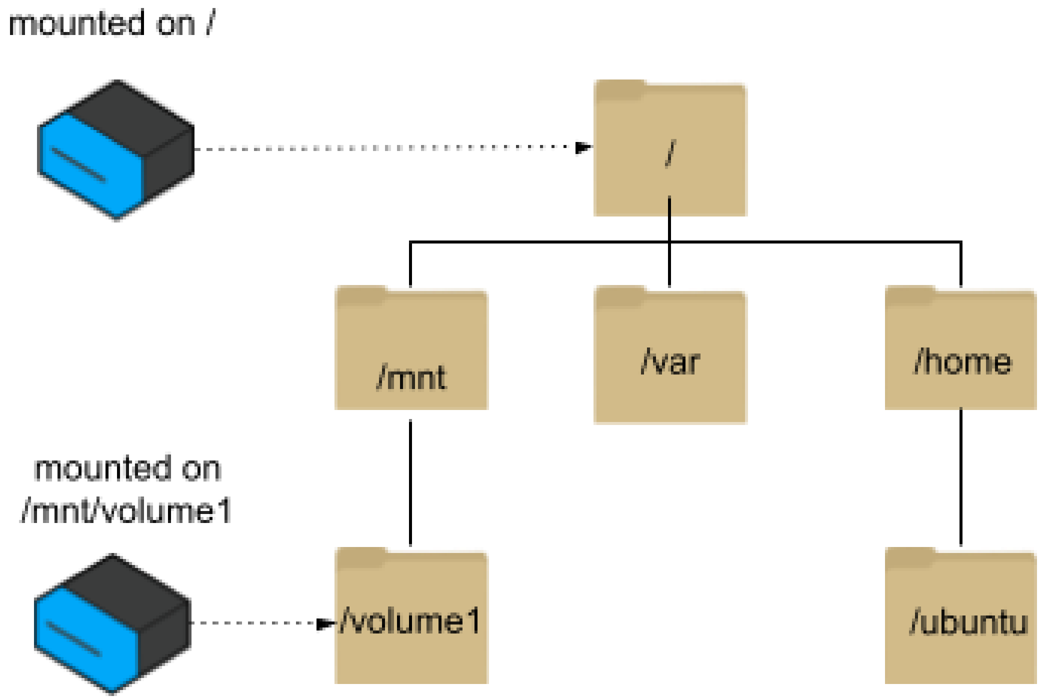
Mounting File System

- However, once such a file system is created, it needs to be made accessible within the uniform file-system tree. This task is achieved via the **mount** program

- What mount does, quite simply is
  - take an existing directory as a target mount point, and
  - essentially paste the new file system onto the directory tree at that point.

- You can permanently configure where a drive attached to a machine should be mounted by editing the file /etc/fstab.

Illustration



/etc/fstab

- The Linux systems filesystem table, aka *fstab*, is a configuration table designed to ease the burden of mounting and unmounting file systems to a machine.
  - It is designed to configure a rule where specific file systems are detected, then automatically mounted every time the system boots.

- Table Structure
  - Device: name or UUID.
  - Mount point. “none” if it is swap.
  - File system type.
  - Options – separated by commas. “defaults” for default options
  - Backup operation – outdated and should not used.
  - File System Check Order – 0 no checks, 1 the root, 2 others.

Example

- Imagine we have an unmounted ext4 file system, stored in device partition /dev/sda1, that has the following contents: a root directory which contains two sub-directories, a and b:
  - Let’s say we wish to mount this file system at the mount point /home/users. We would type something like this: prompt> mount -t ext4 /dev/sda1 /home/users
  - Now, what do we have after prompt> ls /home/users/

- To see what is mounted on your system, and at which points, simply run the mount program.
  - the proc file system (a file system for accessing information about current processes),
  - tmpfs (a file system just for temporary files)