# Storage, part 1

Michael Brockway

October 25, 2014

# Contents

- File - Concept, structure, attributes, operations, types
- Access methods
- Directory structures and file system organisation
- File-system sounting
- File Sharing
- Protection, permissions

Ref: These notes are based on the module text, Silberschatz et al, chapter 10. You are recommended to read this chapter.

# File Concept

- Contiguous address space
- Types of file:
  - Data
    - numeric
    - character
    - binary
  - Program

# File Structure

- None - sequence of words, bytes
- Simple record structure -
  - lines
  - fixed length
  - varable length
- Complex Structures -
  - Formatted document
  - Can simulate by byte file: insert appropriate control characters
- Who decides -
  - Operating system
  - Program

# File Attributes

- *Name* the only information kept in human-readable form
- *Identifier* unique tag (number) identifies file within file system
- *Type* needed for systems that support different types
- *Location* pointer to file location on storage device
- *Size* current file size
- *Protection* controls who can do reading, writing, executing
- *Time, date, and user* identification data for protection, security, and usage monitoring

Information about files is kept in the directory structure, which is maintained on the storage device (disk or other).

# File Operations

A *File* is an emphabstract data type.

- *Create*
- *Write*
- *Read*
- *Reposition* the read/write point within the file
- *Delete*
- *Truncate*
- *Open*($F_i$) search the directory structure on disk for entry $F_i$, and move the content of entry to memory; create a pointer to this memory location
- *Flush*($F_i$) move the content of entry $F_i$ in memory to directory
- *Close*($F_i$) - flush the file, free the memroy, and delete the pointer

# Open Files

Several pieces of data are needed to manage open files:

- *File pointer*: pointer to last read/write location, per process that has the file open
- *File-open count*: counter of number of times a file is open, to allow removal of data from open-file table when last processes closes it
- *Disk location of the file*: cache of data access information
- *Access rights*: per-process access-mode information

Open file *locking* is provided by some operating systems and file systems: controls processes' access to the file. May be

- *Mandatory*: access is granted/denied depending on locks held and requested
- *Advisory*: processes can find status of locks and decide what to do

# File types; names and extensions

In some operating systems different types of file may be distinguished by an *extension* to the file name.

- *Executible binary file*: ie machine code: `.exe` or `.com` or no extension
- *Object file*: compiler output, not yet linked: `.o`
- *Source file*: just text! `.c`, `.cpp`, `.java`, etc
- *Script file*: just text! `.sh`, `.bat`, `.py`, etc
- *Text file*: just text! `.doc`, `.txt`
- *Library*: for linking: `.lib` - for linking to `.o` files
- *Run-time Library*: `.dll` (Windows), `.so` (Unix)
- *Archived folders*: `.zip`, `.gzip`, `.rar`, ...
- *Multimedia*: `.mp3`, `.mpeg`, `.flac`, `.ogg`, `.mov`, ...

# Access Methods

Sequential access:

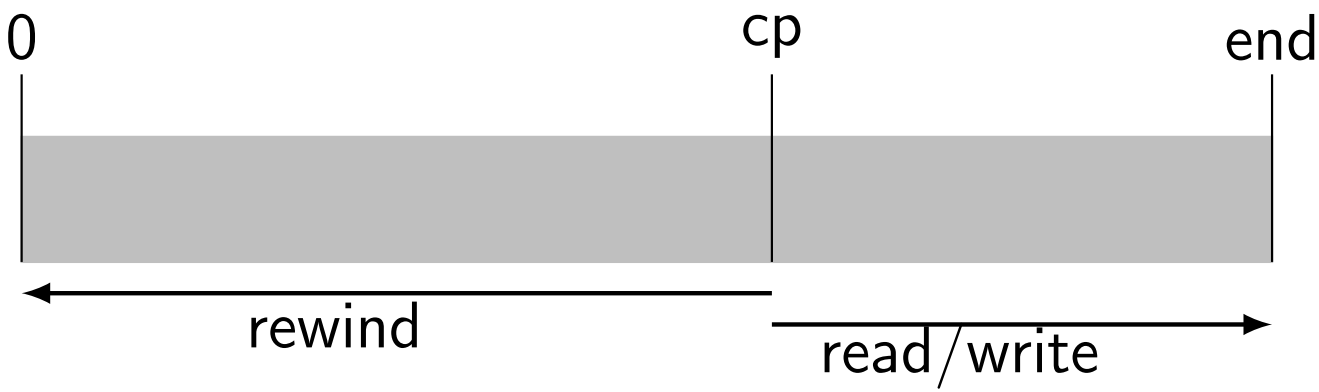
- `read next`
- `write next`
- `reset` - start accessing from the beginning
- `rewrite` - no read after last write

Direct access ($n$ = offset in bytes or blocks of OS-determined size; negative $n$ are counted back from end of file)

- `read` $n$
- `write` $n$
- `seek` $n$ - set current access position `cp` to $n$
- `tell` - report current access position `cp`

# Access Methods ctd

A sequential access file:

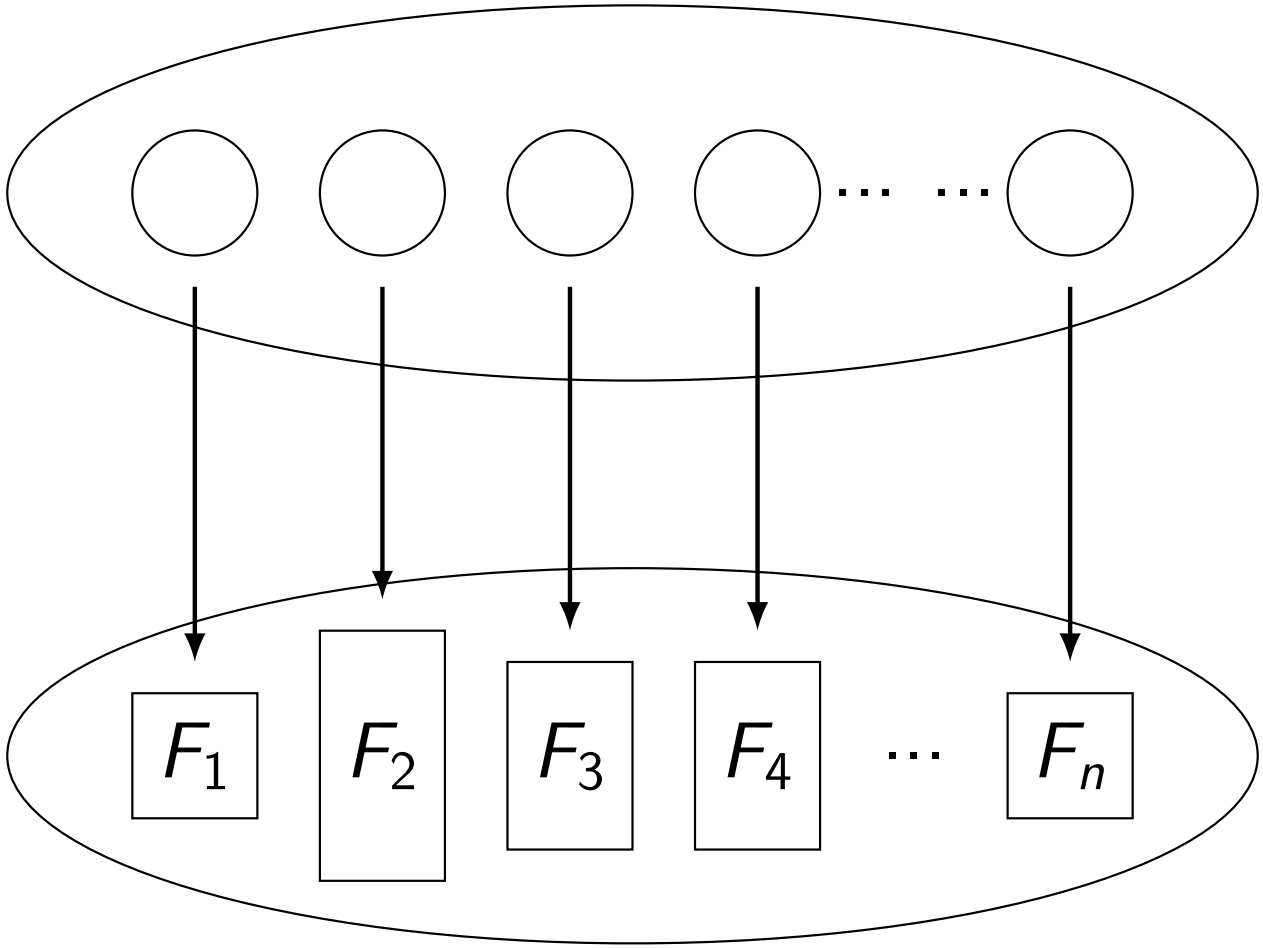



Direct access can simulate sequential access -

- `reset` by `cp` = 0
- `read next` by `read cp; cp += 1;`
- `write next` by `write cp; cp += 1;`

# Directory Structure

A collection of nodes containing information about all files



# Directory Structure

The directory structure consists of emphdirectory entries: each entry contains the a file name and file attributes, and a reference to the physical area of the disk containing the data belonging to the file.

Some entries may not be regular file entries

- Links to files stored elsewhere
- Subdiretories

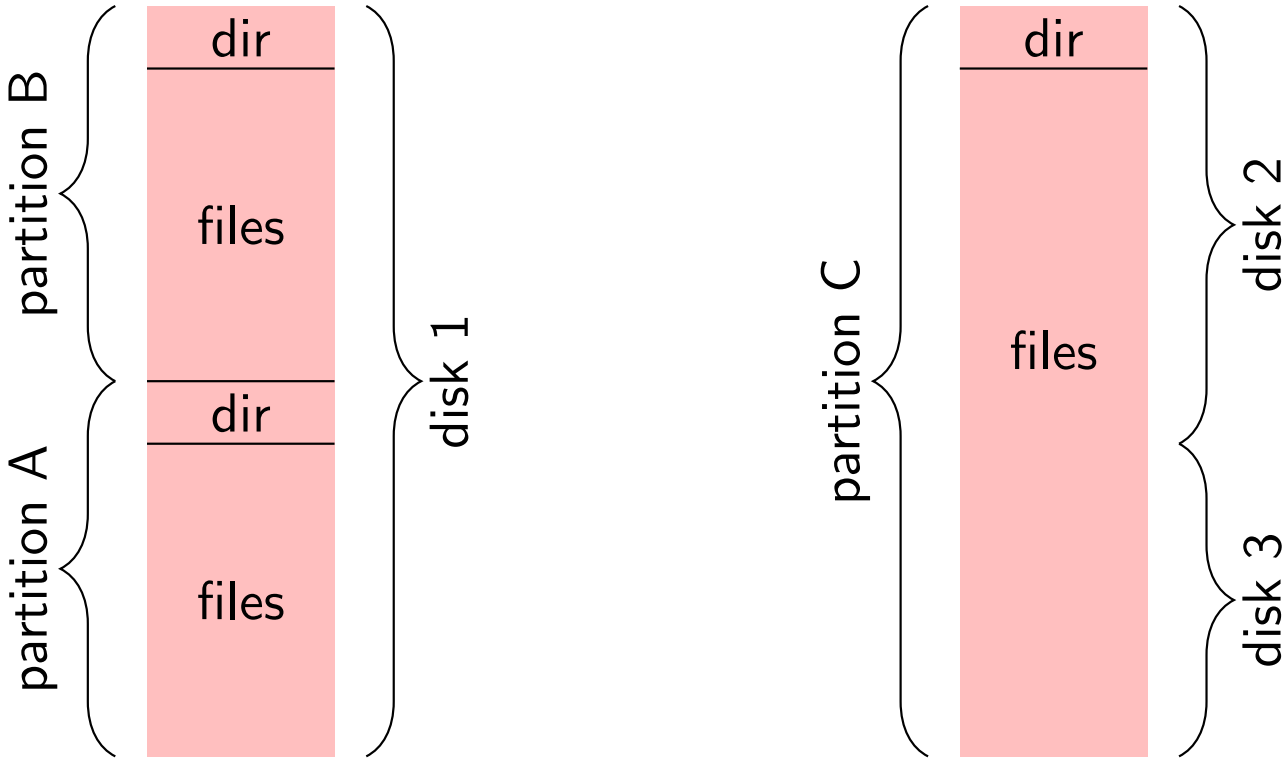
Both the directory structure and the files reside on disk or other permanent storage and are backed up.

# Disk Structure

- A disk or other storage medium can be subdivided into *partitions*.
- Disks or partitions can be *RAID protected* against failure.
- A disk or partition can be used *raw* without a file system, or *formatted* with a file system
- An entity containing file system is called as a *volume*.
- Each volume containing a file system also tracks that file systems information in the *device directory* or the volume *table of contents*.

# File System Organisation

A disk or other physical storage medium (eg a memory stick) can divided into several *partitions*; and a partition can *span* several media:



# Directory Operations and Organisation

Operations on a Directory -

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse (iterate over) the files in the directory

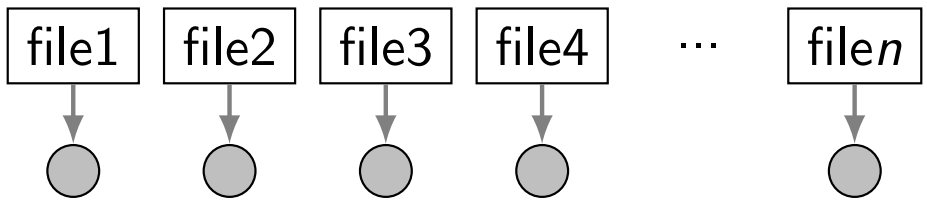
# Directory Operations and Organisation 2

Logical organisation of diectory entries should give

- *Efficiency*: quick location of a file
- *Naming* which is convenient for users
  - Two users can have same name for different files
  - The same file can have several different names
- *Grouping* logical grouping of files by properties, (eg all Java programs, all games, ...)

# Directory Operations and Organisation 3

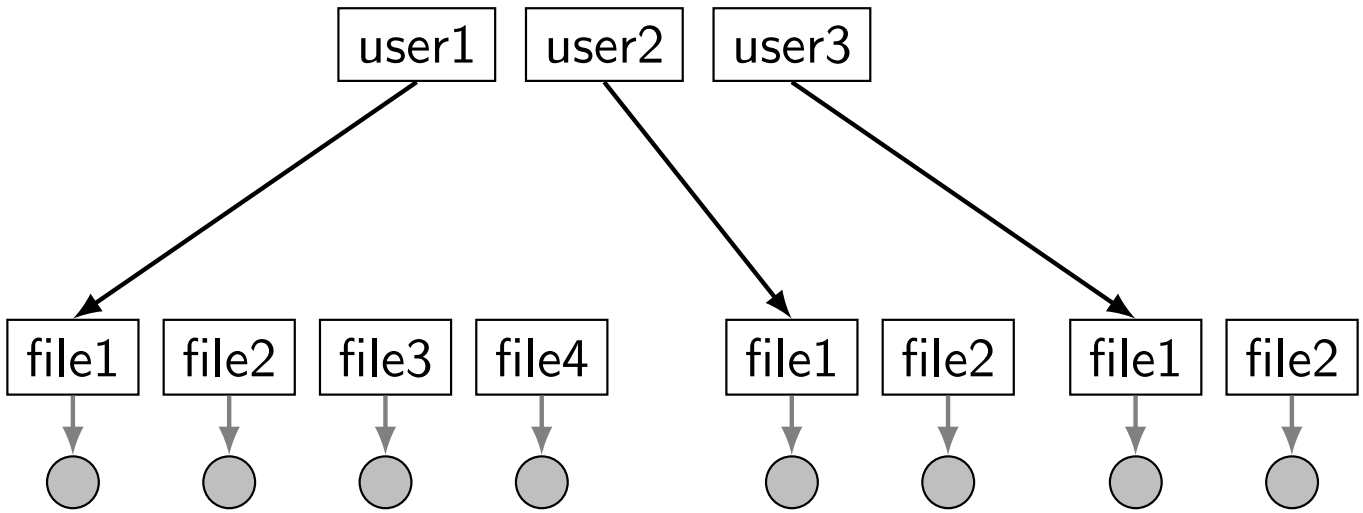
Simple idea: a single list of directory entries




- Naming, grouping not addressed
- Efficiency: not scaleable!

# Directory Operations and Organisation 4

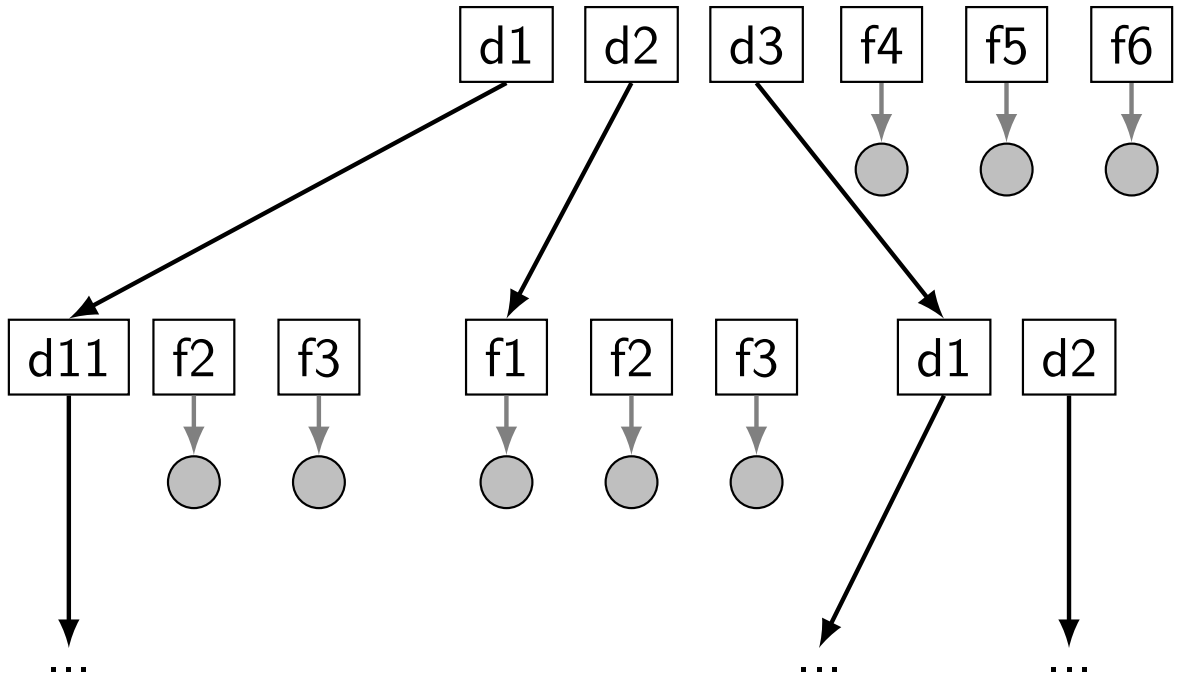
A two-level directory ...




Some progress on efficiency, grouping; still not very flexible, scaleable

# Directory Operations and Organisation 5

Grouping and look-up efficiency are well catered for by a tree-like hierarchy of (sub)directories ...



# Directory Operations and Organisation 6

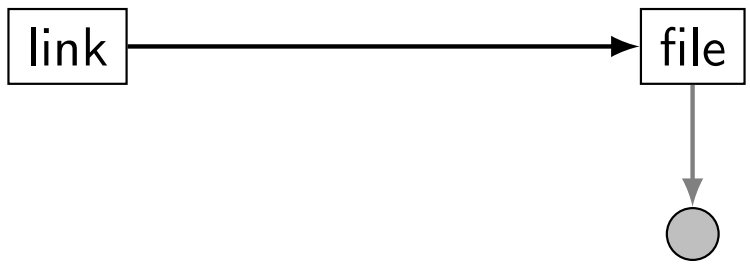
- a *tree* is a *directed graph*
  - *nodes* connected by directed *edges* (arrows)
- ... with no *cycles* and no two edges into the same node
- The nodes of a directory tree are directory entries for
  - regular data files; or for
  - *subdirectories* (subfolders)

Good solution for grouping, look-up/access efficientcy. Also, files in different subdirectories can have the same name.

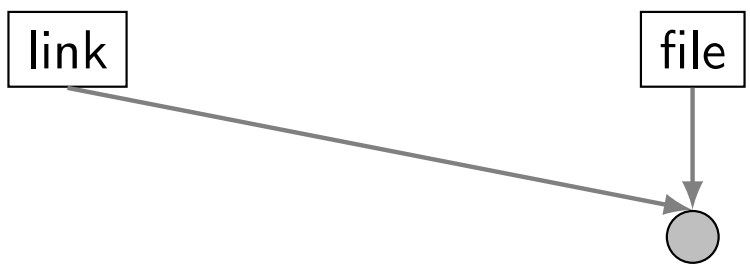
We would also like to have the possiblility of one file having different names (directory entries), possibily in different subdirentories, referring to it.

# Directory Operations and Organisation 7

Soft link - name (directory entry) which is an alias for another directory entry ....

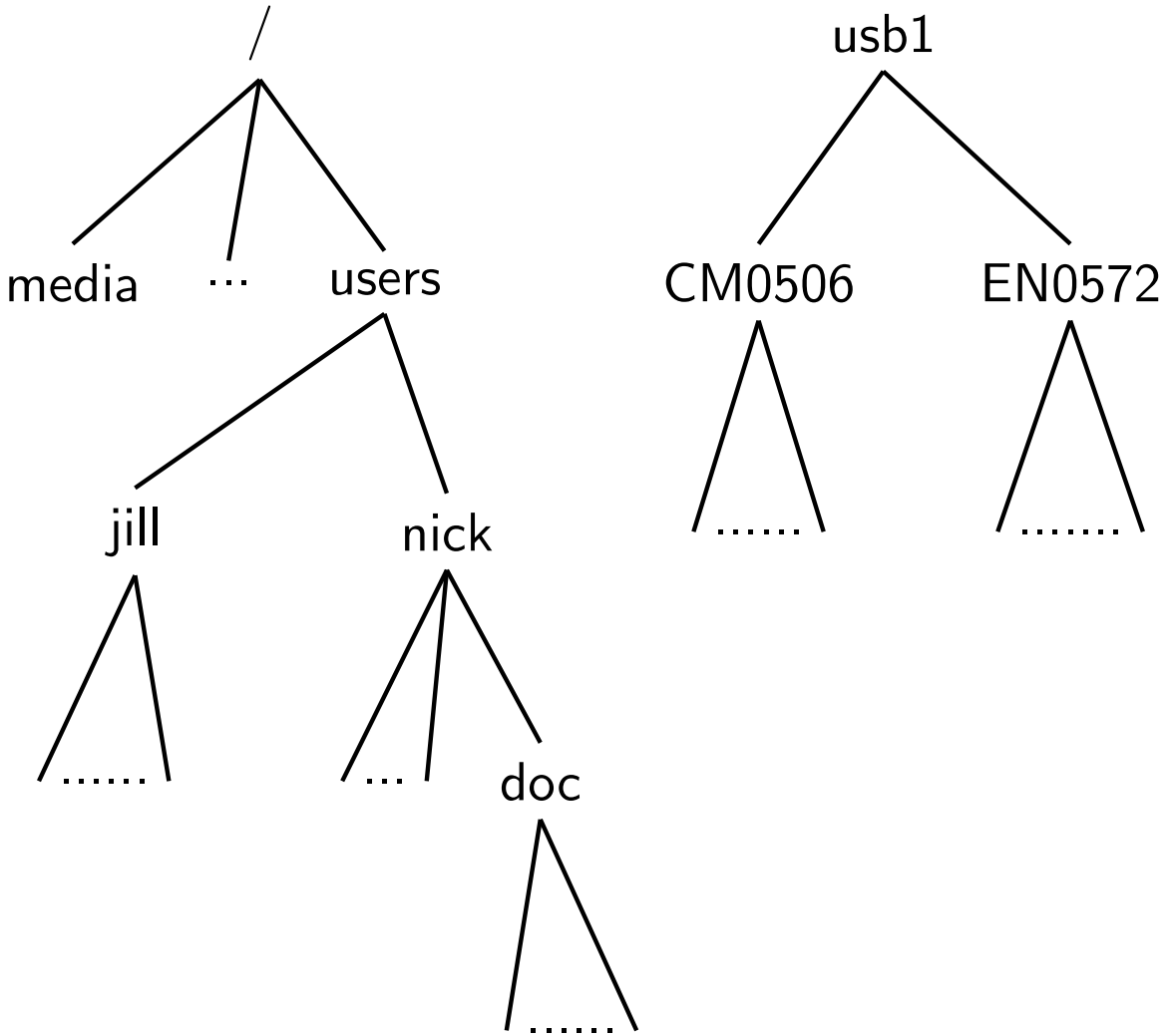



Hard link - two directory entries (probably in different folders) refer tot the same data ....



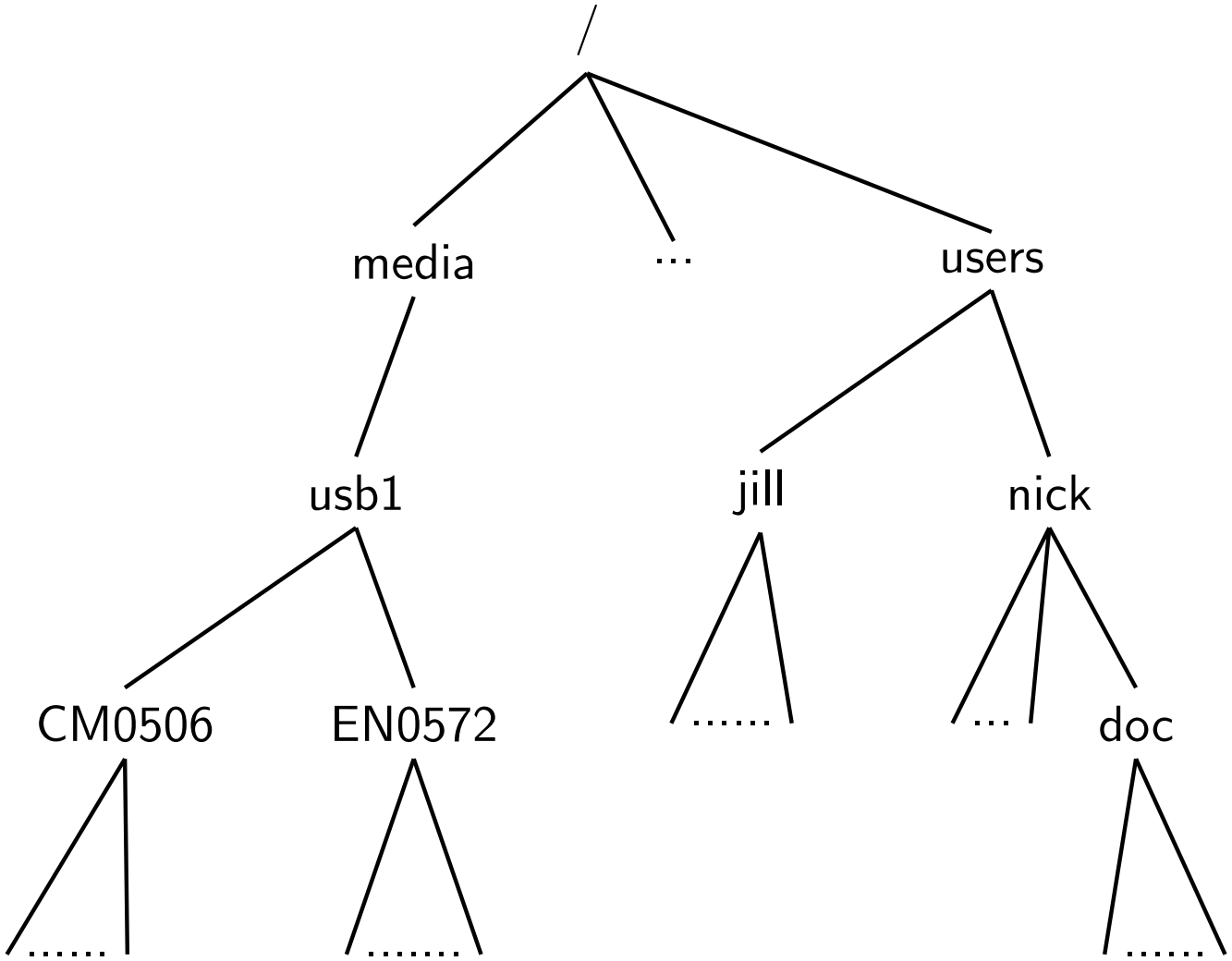
# Mounting a file system ...

Here (on left) is a live file system and (on right) a system on a USB stick, not yet mounted ...



# Mounting a file system ... ctd

Now the USB stick is mounted on the live file system.



# File Sharing

- desirable on multi-user systems
- may be done through a *protection scheme*
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- *User ID*s identify users, allowing permissions and protections to be per-user
- *Group ID*s allow users to be in groups; a group may be given permissions

# File Sharing over a network

- Manual programs like FTP (Filezilla); or automatically, seamlessly using distributed file systems
- WWW is "semi-automatic"
- Client-server model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Security, authenication issues
- NFS is standard UNIX client-server file sharing protocol; CIFS is standard Windows protocol
- Distributed naming services - LDAP, DNS, NIS, Active Directory: implement unified access to information needed for remote computing
- *Consistency semantics* needed to specify how multiple users are to access a shared file simultaneously

# Protection

Concept of file *owner* who specifies

- What can be done
- bywhom

Types of access

- Read
- Execute
- List (directory)
- Write (modify) (create, in a directory)
- Append
- Delete

# UNIX file access specification

- Each file has 9 *access bits*
  - 3 bits specifying the file *owner*'s permissions;
  - 3 bits specifying permissions for users in the owner's *group*;
  - 3 bits specifying everyone else's permissions;
- The three permissions are to
  - *read* the file;
  - *write* the file (includes creating, deleting);
  - *execute* the file (if it is a script or program); Executing a directory lists it.

# UNIX file access specification

Seeing the permissions on a file-

```
$ ls -l a.out
$ -rwxrw-r-x 1 michael michael 7247 Oct 13 08:56 a.out
```

Setting permissions

```
$ chmod g-w a.out
$ ls -l a.out
$ -rwxr--r-x 1 michael michael 7247 Oct 13 08:56 a.out
$ chmod a+x a.out
$ ls -l a.out
$ -rwxr-xr-x 1 michael michael 7247 Oct 13 08:56 a.out
```

# A UNIX Directory listing

```
-rwxr-xr-x 1 michael michael   7247 Oct 13 08:56 a.out
drwxr-xr-x 4 michael michael   4096 Mar  3  2014 Research
-rw-rw-r-- 1 michael michael 773493 Mar  3  2014 Research.html
drwx------ 2 michael michael   4096 Oct 17 10:35 Storage
-rw-r--r-- 1 michael michael 689814 Oct 31  2012 Lecture1.pdf
drwx------ 5 michael michael   4096 Oct 10 10:37 CaseStudy
-rw-rw-r-- 1 michael michael    203 Oct 13 08:55 test.c
drwx------ 2 michael michael   4096 Jul  4 15:12 VisCat
lrwxrwxrwx 1 michael michael 9 Oct 17 10:56 here.c -> ../test.c
```

First character 'd' $\Rightarrow$ a subdirectory; '-' $\Rightarrow$ a regualar file; 'l' $\Rightarrow$ a *link* to a file stored somewhere else.