Basics of Operating Systems (2)

Dr. Qin Zhou

Coventry University
email: q.zhou@coventry.ac.uk

Files and directories

- Files are stored on disks, CDs etc.
  - these devices store data in fixed-size blocks
- Different devices use different block sizes
- Different devices have different capabilities
  - layout of data blocks may effect efficiency
- Operating system need to hide these device-dependent aspects

Files and directories

- The filesystem lets you refer to files by name
  - – the filesystem defines a *namespace* (a set of names, in the same way that an address space is a set of addresses)
- Where the files are located becomes irrelevant
  - – the filesystem can organise data layout to optimise access to the data

Files and directories

- Filesystems store information in *volumes*
  - usually a single device, e.g. a disk or CD
  - a single devices can be *partitioned* into multiple volumes
  - a single volume might span multiple devices
- Each volume is treated as a numbered sequence of blocks
  - filesystem needs to keep track of file names and corresponding block numbers

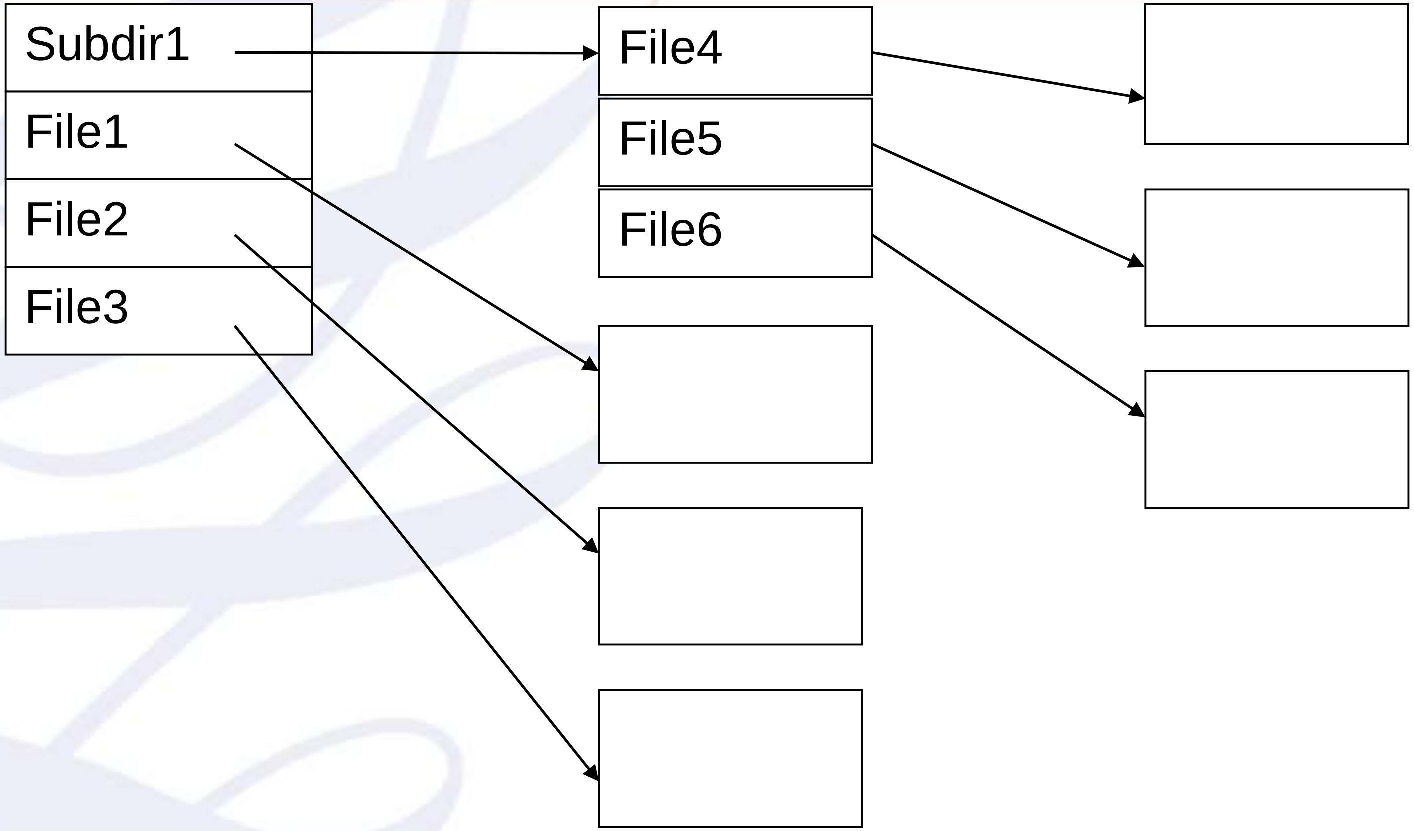
Files and directories

- Each volume is treated as a numbered sequence of blocks
  - filesystem needs to keep track of file names and corresponding block numbers
- A list of filenames and locations is called a *directory*
  - this is data that can be stored as just another file
  - there must be a *root directory* at a known place on each volume

Files and directories

- By treating directories as (special) files, there can be multiple directories on a volume
  - directories can contain entries for files and other (sub-) directories
  - this leads to a tree-structured *hierarchical filesystem*

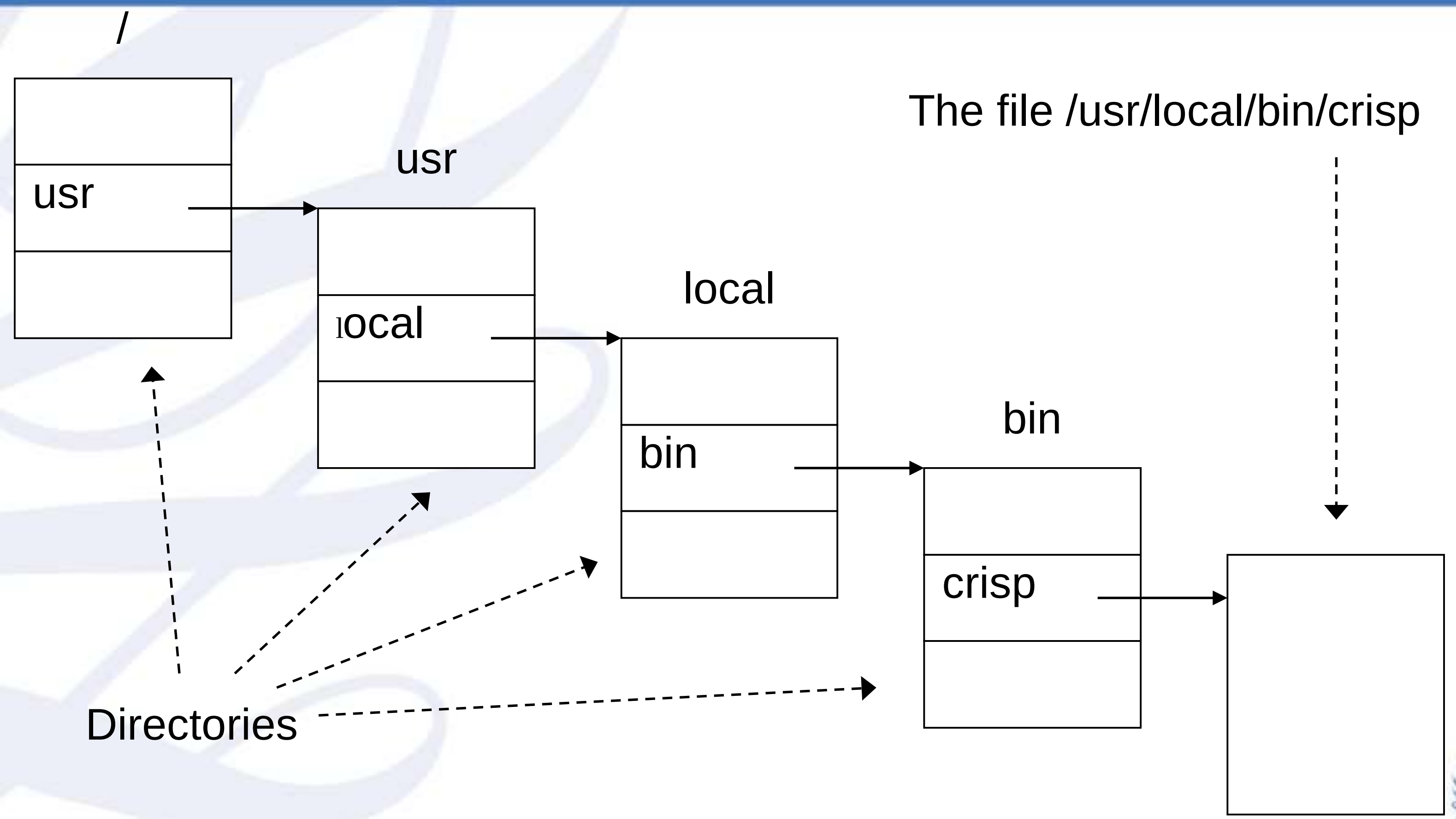
Files and directories



Pathnames

- To identify a file uniquely, we need to use a *pathname*
  - this specifies the path from the root directory through any subdirectories to the file
- Unix naming conventions:
  - the root directory is called ‘/’
  - ‘/’ is also used to separate individual directory and file names in a pathname

Pathnames


/
usr
usr
local
local
bin
bin
crisp
The file /usr/local/bin/crisp
Directories

Pathnames

- Most filesystems keep track of a *current directory*
  - names which do not begin with ‘/’ (the root directory) are taken to be relative to the current directory
- Special names:
  - ‘**.**’ is the current directory
  - ‘**..**’ is the *parent directory* which contains the current directory

Pathnames

- If the current directory is ‘/usr/local/bin’:
  - the file ‘/usr/local/bin/crisp’ can be referred to as ‘crisp’ or as ‘**.**/crisp’
  - the directory ‘/usr/local/bin’ can be referred to as ‘**.**’
  - the directory ‘/usr/local’ can be referred to as ‘**..**’
  - the file ‘/usr/local/man/crisp.1’ can be referred to as ‘**..**/man/crisp.1’

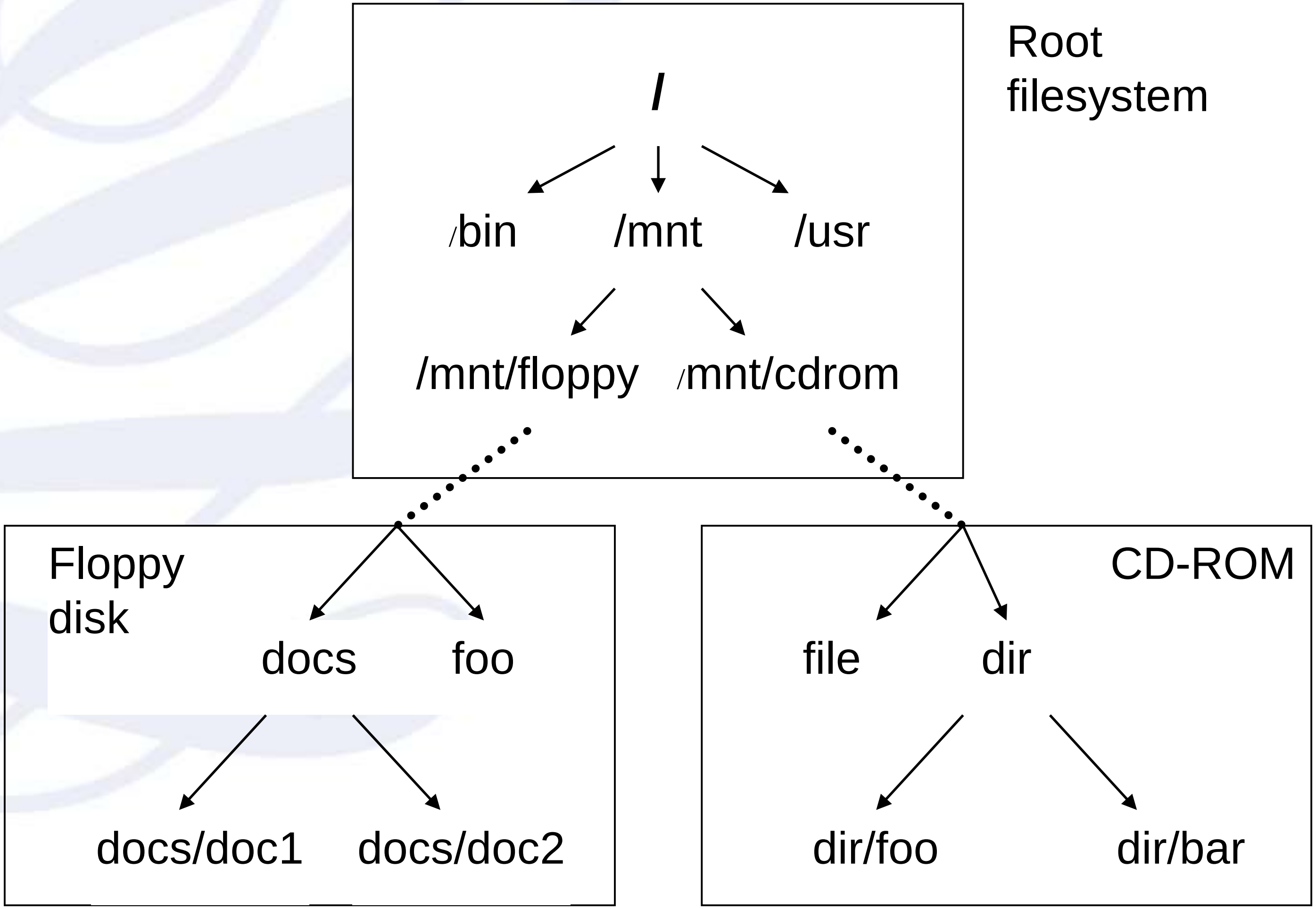
Windows pathnames

- Same as Unix, but:
  - – ‘\’ used instead of ‘/’
  - – names must be preceded by a volume identifier, e.g. ‘C:’ for the primary hard disk
- ‘Drive letters’ like C: limits system to 26 volumes
  - – disks from remote machines can be *mapped* to drive letters
  - – UNC names: refer to files on remote machine using \\machine as a prefix instead of e.g. C:

Multiple volumes in Unix

- Unix uses a unified filesystem
- There is a *root filesystem* (initially a single volume)
  - additional volumes are *mounted* as subdirectories within the root filesystem
  - the mounted volume’s directory structure replaces the directory it is mounted onto

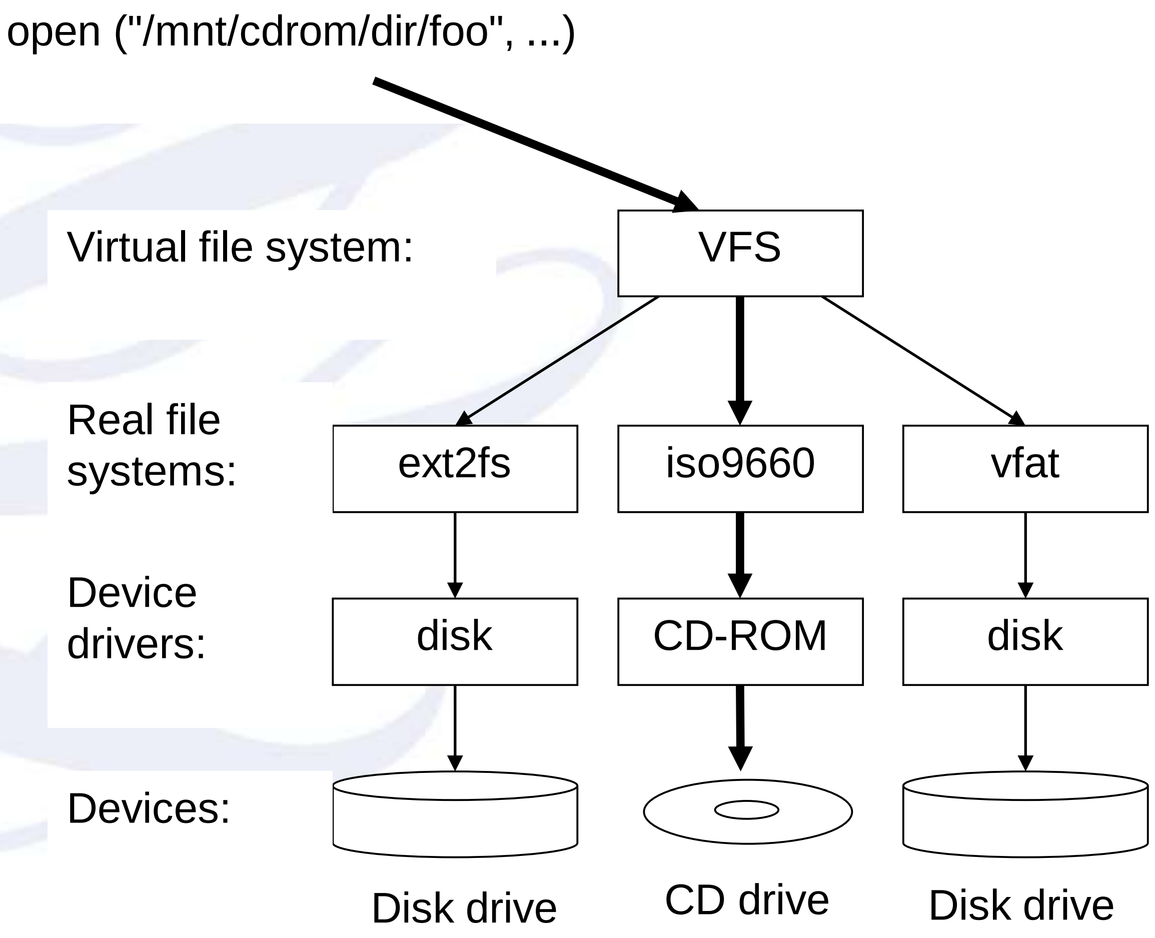
Multiple volumes in Unix



Multiple volumes in Unix

- Different volumes will be organised differently
  - the *virtual filesystem* (VFS) switches between different filesystems as *mount points* are encountered when tracing through a pathname
- Volumes must be explicitly unmounted before removal

The virtual filesystem



Types of files

- VMS (ancestor of Windows NT) provided different file types:
  - sequential: could process from start to end
  - direct access: fixed-size records, could process in any sequence by record number
  - indexed: similar, but records identified by a symbolic key
- This makes it hard to write a program to process any file

Types of file

- Still need to distinguish different types of file according to content
  - e.g. text files vs. executable binary files
- Both Unix and Windows use suffixes (extensions) to identify file contents:
  - e.g. file.txt or file.exe
  - Windows associates different actions with different extensions
  - Unix extensions are just a naming convention

Types of file

- Executable programs are identified:
  - by their extension in Windows (.com, .exe, .bat or .cmd)
  - by a file attribute in Unix
- Unix shell scripts can be made executable
  - first line (‘shebang’ line) specifies pathname of program to use to process file

```
!#/bin/sh
```

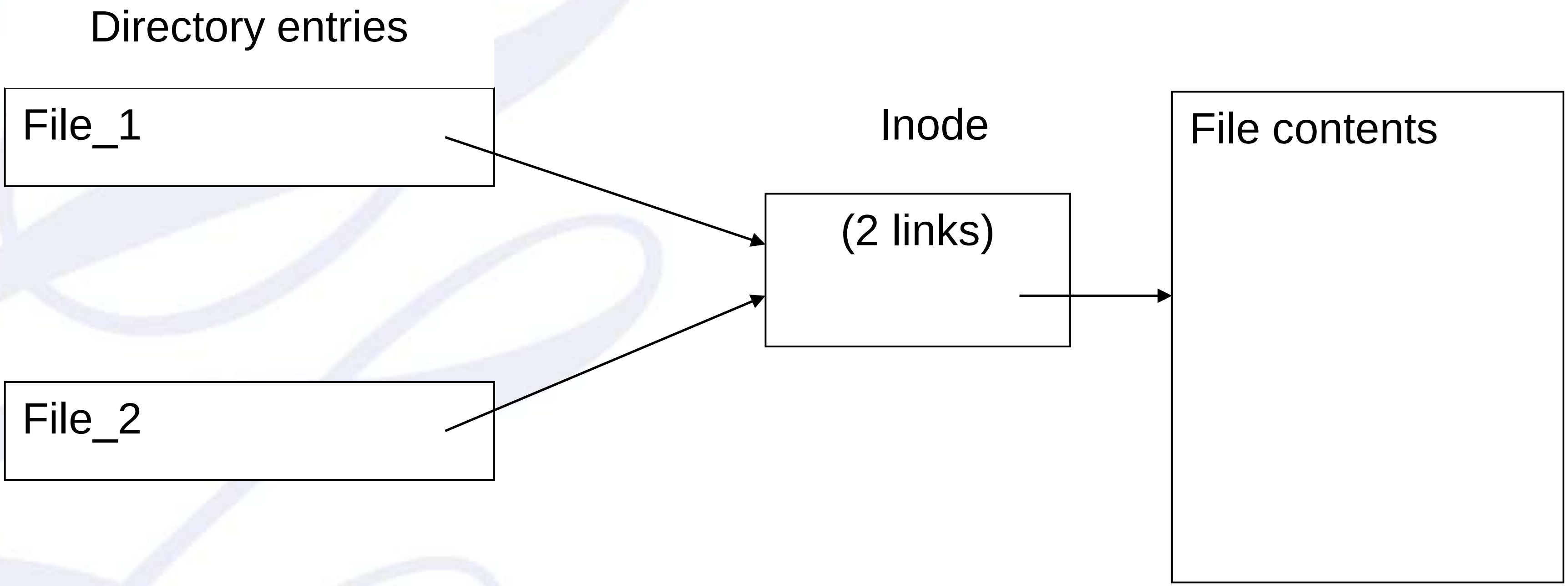
Filenames

- Limited character set
  - – characters like ‘*’ are interpreted specially and should not be used
  - – names including spaces must be quoted (“...”)
- Limited length
  - – MS-DOS: 8 characters + 3 for extension
  - – early Unix systems: 14 characters
  - – now: 255 characters

Links and shortcuts

- It is sometimes useful to keep the same file in several places
  - keep copies? but then they need to be re-copied after any changes...
- Unix uses *inodes* to hold all information about file layout on disk etc.
  - directory entries refer to inodes
  - multiple directories can contain *links* to the same inode

Links and shortcuts



Directory entries

File_1

File_2

Inode

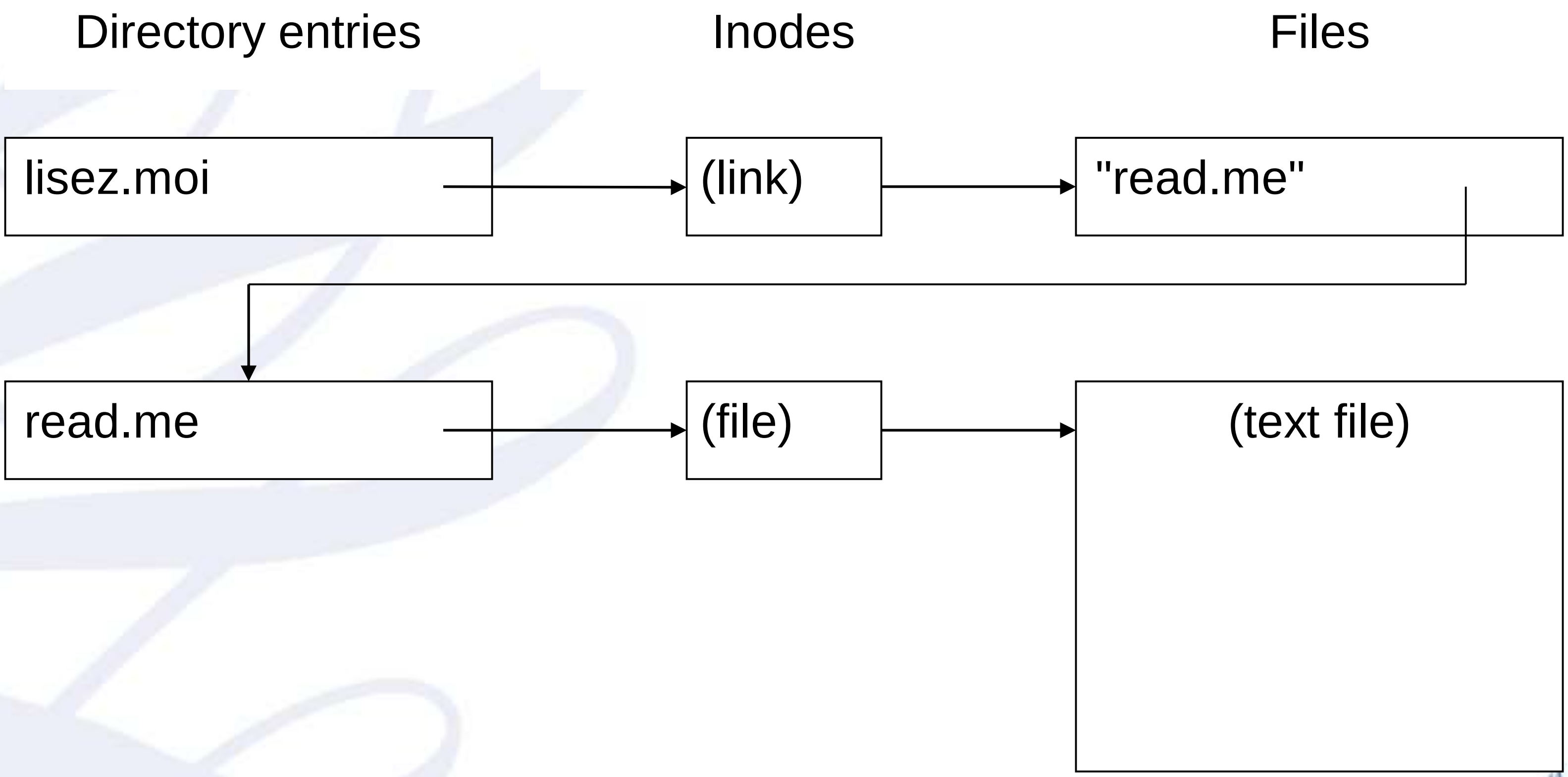
(2 links)

File contents

Links and shortcuts

- Links can only refer to files on the same volume
  - – solution: *soft links* whici are files containing the pathname of the file being linked to
  - – file attribute used to mark file as a soft link
- Windows provides *shortcuts*
  - – contains more information than just the name
  - – not exactly the same: shell accesses real file when icon activated, but otherwise a shortcut is a normal file

Links and shortcuts



Directory entries

Inodes

Files

lisez.moi

(link)

"read.me"

read.me

(file)

(text file)

File attributes

- Apart from name, directory entry (or inode) should provide some or all of:
  - timestamps (creation, last update, last access)
  - ownership (individual and group)
  - file size (bytes)
  - file type (e.g. file or directory)
  - access permissions
- Other attributes possible (version number, password, ...)

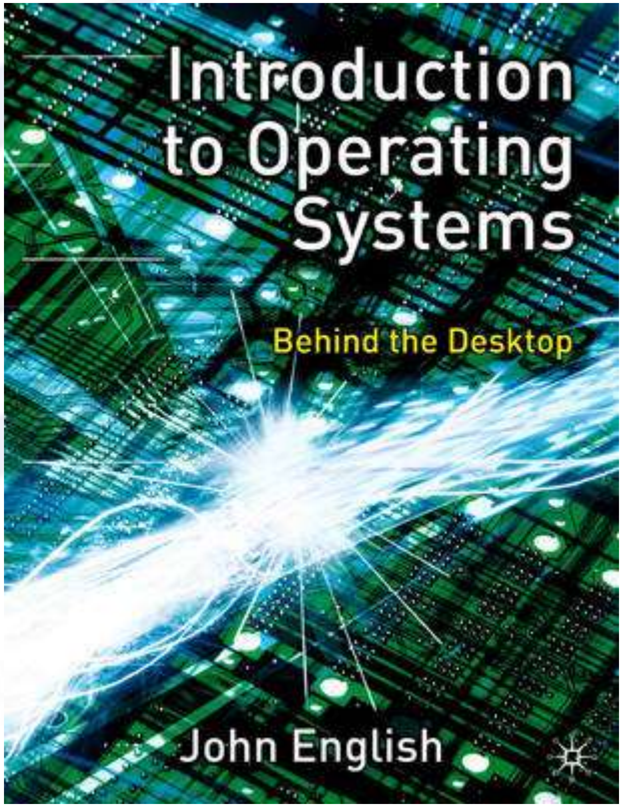
Access permissions

- Permissions include read, write, execute, delete, append, ...
- Unix: each user belongs to one or more groups
  - separate permissions for file owner, owning group and others (read, write, execute)
- Windows: each file has an access control list
  - separate permissions for individuals or groups

Example file systems

- FAT (MS-DOS, Windows)
  - other systems can also use this as a ‘lowest common denominator’
  - used for floppy disks
- Ext 2/3/4 (Linux filesystem’)
  - based on original BSD Unix filesystem
- NTFS (Windows NT, 2000, XP, Vista, 7/8)

Reference


Introduction to Operating Systems
Behind the Desktop
John English