Week 12

File systems

School of Information Technology and Electrical Engineering
The University of Queensland

Outline

File Systems and File I/O
- Files
- Directories
- File Operations
- File Systems

References

File Systems
- See Glass & Ables – pages 572-584, 606-616
- Credits:
  - E. Berger, University of Massachusetts, Amherst
  - Silberschatz et. al, “Operating Systems concepts”
  - Tanenbaum, “Modern Operating Systems”
  - Rochkind, “Advanced UNIX programming”

Recall from Lecture One

Operating Systems provide *abstractions* to
- make computer hardware easier to use
  - for user, programmer, system administrator…
- manage hardware resources

Example abstractions

Abstraction	Resource
Processes	CPU time
Virtual Memory	Memory
Files	Disk space

Files

Computers can store information on storage media (secondary storage)
- magnetic and optical disks, tape

Operating system maps files to physical devices

A **file** is a named collection of related information, can be
- structured (sequence of records)
- unstructured (sequence of bytes)

File Structure

- Files can also contain multiple streams.

Who decides?
- Operating System? – Sometimes
- Application Program - Usually

Most OSs consider files to be linear sequences of bytes
- e.g. UNIX, Windows etc
- Up to each program to know what *format* the file is in

Files can be **programs** or **data**

File Attributes

Each file usually associated with **attributes** or **meta-data**
- Name – human readable
- Type
  - Some systems determine this from name or contents of file
- Location - where is it on the device?
- Size
- Protection – who has what permissions?
- Owner
- Time and date information

Particular attributes stored can vary by OS
- More precisely – can vary by **file system**

File Types

How is type of file determined?

Windows
- Based on filename extension

Macintosh (HFS)
- 4 character type stored as meta-data
  - e.g. MSWD is "Microsoft Word"

UNIX
- Contents of first part of file
  - **magic number**
- See /etc/magic and file(1)

Directories

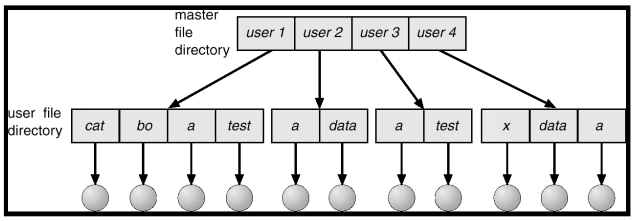
Need a method of retrieving files from disk!
- OS (file system) uses numbers, e.g.
  - index number, or
  - sector number
- People prefer names
- OS provides **directory** to map names to file numbers
  - Example /etc/services on moss has index number 2883627 on its file system.

A **file system** is a collection of files plus a directory structure
- Variety of directory structures possible

Flat File Systems

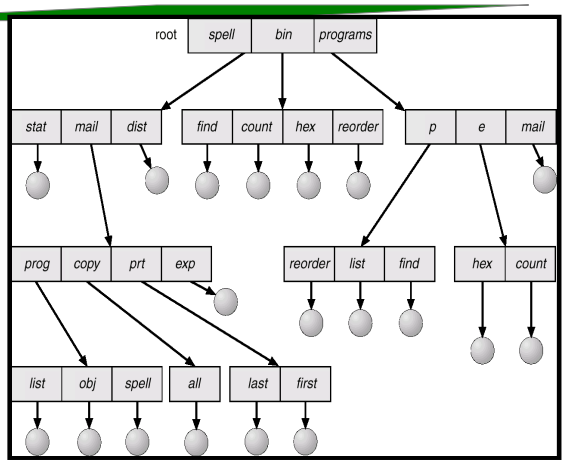
One level directory
- One namespace for entire disk, every name unique
  - Directory contains (name, index) pairs
- Used by Apple, CP/M, DOS 1.0, first MacOS

Two level directories
- Separate directories for each user
- OK for floppies, but not scalable



Hierarchical File Systems

Tree-structured name space
- Used by all modern operating systems



Hierarchical File Systems (cont.)

Top level
- **root directory** (UNIX)
- **drive names** (Windows)
- **volume names** (MacOS – not Mac OS X)

Tree-structured name space
- Directory becomes **special file** on disk
  - Marked by special flag bit
- User programs may read directories, but only system may manipulate directories
- Each directory contains (name, index number) pairs
  - names need not be unique in whole file system
  - names must be unique in that directory

Pathnames

Processes
- have concept of **current** (or **working**) **directory**
- can change their working directory
  - How do you specify a directory name?

Absolute and **relative** pathnames
- Absolute name – will always find the file
  - Name given relative to the root directory
- Relative name
  - Name relative to the current directory

Acyclic Graph Directory Structure

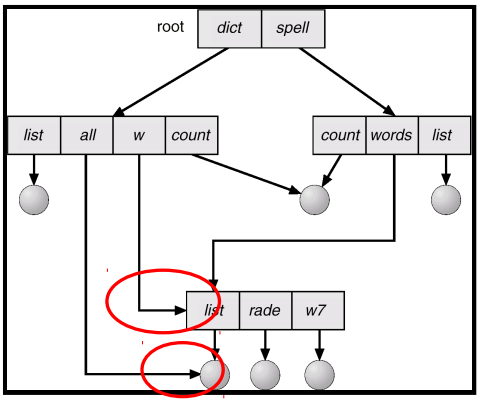
Have shared subdirectories/files

Not a tree

UNIX uses **links**

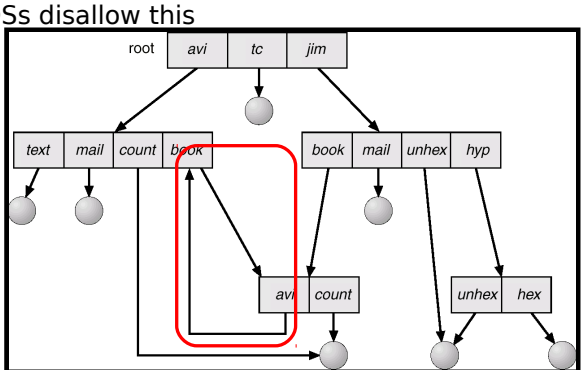
What if delete shared file/ directory?

More later



General Graph Directory

What happens if allow cycles?
- Management becomes more difficult
- Most OSs disallow this



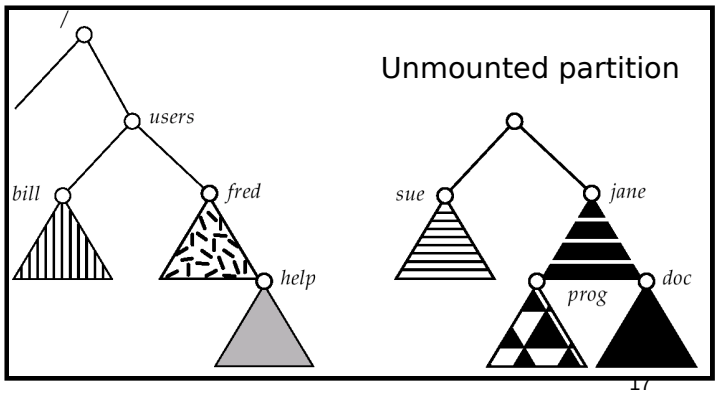
Mounting File Systems

File systems must be **mounted** before use
- Complete directory structure may actually be built out of multiple file systems

Mounting involves associating a file system device (e.g. disk partition) with a mount point
- **Mount point** is the pathname at which the root of the mounted file system appears
- Windows – drive letters
- Unix – more arbitrary mount points

Mount point example

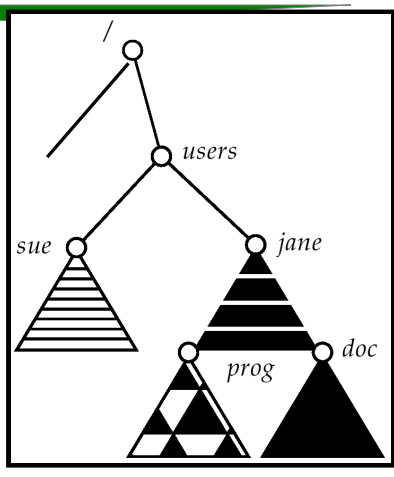
Existing



Mount point example (cont.)

After mounting partition at /users
- /users is mount point

Original /users contents are obscured
- Some OSs prohibit mount over non-empty directory



Break

Links in UNIX

Hard links (UNIX: **ln** command)
- Allows multiple links to single file
- Example: “ln A B”:
  - initially: A -> file number 100
  - after: A, B -> file number 100
    - i.e. multiple names for the same file
- Can’t link across file systems
- OS maintains reference counts
  - Deletes file only after last link to it is deleted

Problem: users could create circular links with directories
- Reference counting can’t reclaim cycles

Solution: can’t hard link directories

Agave example …

Links in UNIX (cont.)

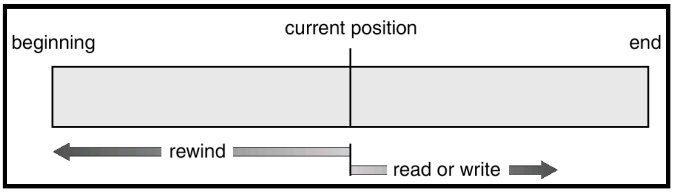
Soft (symbolic) links (UNIX: ln -s)
- Makes symbolic pointer from one file to another
- “ln -s A B”:
  - Initially, A -> file #100
  - After, A -> file #100, B -> A
- Removing B does not affect A
- Removing A: dangling pointer

Problem: Circular links can cause infinite loops
- E.g., list all files in directory & its subdirectories

Shortcuts in Windows are a similar concept

File Access

Sequential
- Bytes read/written in order
- Very common access method



Random
- Jump to any point in file

Directory Operations

Create directory (*mkdir*)

Delete directory (*rmdir*)

Open, read, close directory (*getdents*)

Rename directory (*rename*)

Link (*link*)

Unlink (*unlink*)

Get current directory (*getcwd*)

Change directory (*chdir*)

Protection

OS must allow users to control access to files
- Grant or deny access to file operations depending on protection information

Access lists and groups (Windows - NTFS)
- Access list for each file with user name and access type
- Lists can become large & tedious to maintain

Access control bits (UNIX)
- Three categories of user (user, group, others)
- Three types of access privileges (read, write, execute)
- One bit per operation (111101000 = rwxr-x---)

Examples…

File Sharing

Sharing may occur over a network

Distributed File System

- Remote file system directories are visible locally, e.g. via some mount point
- NFS (Sun's Network File System)
  - Widely used in UNIX world
- CIFS (Common Internet File System)
  - Windows

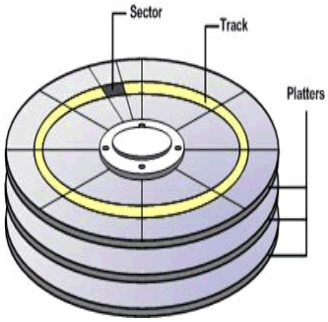
Some systems use different naming convention for network pathnames

- UNC (Uniform Naming Convention) – Windows
  - \\servername\sharename\pathname\...
- DCE DFS
  - /.../servername/pathname/...

Disk Drives

Consist of multiple rotating disks

- Each disk contains concentric tracks
- Each **track** consists of multiple **sectors**
  - Bits laid out serially
- Sector typically 512 bytes (plus preamble, error correction etc)



Disk Drives

Read/write head per surface (usually)

- Heads move together
  - **Seek** to a **cylinder** of tracks

Key fact for our purposes:

- Reads and writes occur in terms of whole sectors
- Minimum unit of transfer

Cost of Disk Operations

In addition to CPU time to start disk operation:

Latency: time to initiate disk transfer

- **Seek time**: time to position head over correct cylinder
- **Rotational time**: time for correct sector to rotate until under disk head

In most cases, disk latency will be some number of milliseconds

Bandwidth: rate of I/O transfer of sectors once initiated

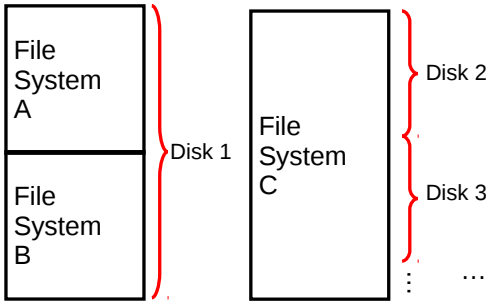
File Systems and Disks

Not necessarily a one-to-one correspondence

File system can span one or more disks

- Does not apply for some file system types

A disk can hold more than one file system



File Organisation on Disk

File system maps file blocks to disk location

- e.g. file 0, block 0 maps to cylinder 0, platter (head) 0, sector 0

File block may not be same size as disk sector

How do we best lay out files on disk?

File Systems

Hundreds of types of file system exist

Common ones:

- Windows
  - NTFS, FAT, FAT32
- Solaris
  - UFS (Unix File System), ZFS
- Linux
  - ext2, ReiserFS, ext3, ext4, btrfs
- BSD
  - FFS (Fast file system)
- Optical disks
  - ISO9660, Joliet extensions, UDF

On-Disk Data Structures

Most systems fit following profile:

- Most files are small
- Most disk space taken up by large files
- I/O operations target both small & large

Per-file cost must be low, but large files must also have good performance

Some possible structures:

- Contiguous allocation
- Linked Files
- Indexed Files

But first... fragmentation

Fragmentation

Internal Fragmentation

- Space allocated but not used
  - E.g. allocations of 4kbyte disk blocks – a 5kbyte file will be allocated 2 blocks (8kbytes) – 3kbytes wasted

External Fragmentation

- Unallocated spaces are
  - too small to be useful, or
  - too spread out

Figure to be drawn in class

Contiguous Allocation

Operating system maintains ordered list of free disk blocks

OS allocates contiguous chunk of free blocks when it creates a file

- Only need to store start location & size in file index

Contiguous Allocation: Pros & Cons

Advantages

- Simple
- Fast – minimises disk seeks

Disadvantages

- May not know size of file at creation time
- Changing file sizes – may not be possible to extend without copying
- Fragmentation – may be hard to find space for a new file

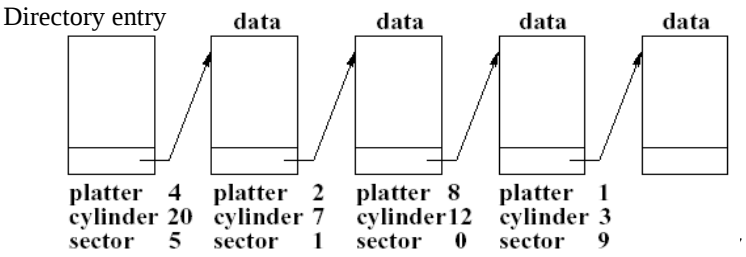
Examples: IBM OS/360, write-only disks, early PCs

Linked Files

Maintain list of all free sectors/blocks

In directory entry, keep pointer to first sector/block

In each sector, keep pointer to next sector



Linked Files: Pros & Cons

Advantages

- No external fragmentation
- Easily handles file size changes
- Good for sequential access

Disadvantages:

- Random access is slow – have to follow the chain of pointers
  - Number of disk seeks required may be large
- Space taken by pointers
  - Can use clusters of disk blocks and have one pointer per cluster (but increases internal fragmentation)

Variation: FAT (File Allocation Table)

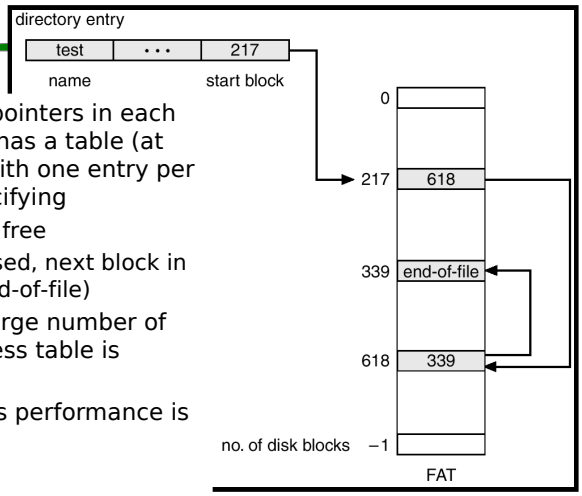
FAT

Doesn't store pointers in each block, instead has a table (at start of disk) with one entry per disk block specifying

- If block is free
- If block used, next block in file (or end-of-file)

Can result in large number of disk seeks unless table is cached

Random access performance is improved



Indexed Files

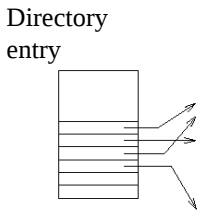
OS keeps array of block pointers for each file

User or OS declares maximum length of file created

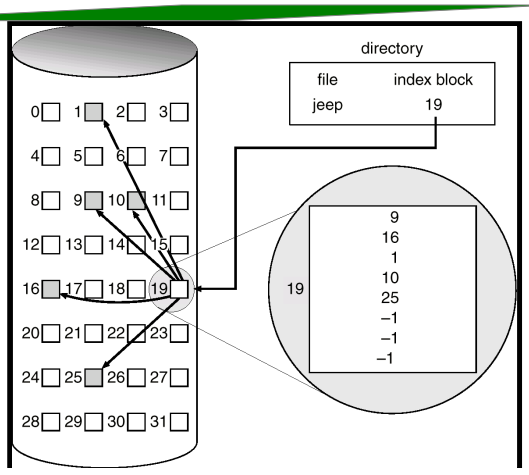
OS allocates array (index block) to hold pointers to all blocks when it creates file

- But allocates blocks themselves only on demand

OS fills pointers as it allocates blocks



Index Allocation Example



Indexed Files: Pros & Cons

Advantages

- Sequential & random access: easy
  - i.e. quick to determine which disk block to read next

Disadvantages

- Sets maximum file size
- Lots of seeks
  - Data blocks may be spread out over disk

Many variations possible

Indexed Files - Variations

Linked Index Blocks
- Index block points to first N blocks in file
- If file grows beyond this, add another index block and link to it from first, etc

Multilevel Index
- First level index block points to a set of second level index blocks (which point to the disk blocks themselves)
- Allocate second level index blocks only as necessary

Combined approach
- This is how most UNIX file systems do it, e.g....

UNIX file system example

Each directory entry contains 14 block pointers

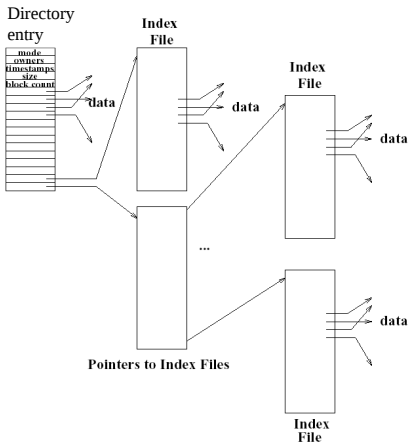
First 12 pointers point to data blocks

13th pointer: one indirection
- Points to block of 1024 pointers to 1024 more data blocks

14th pointer: two indirections
- Points to block of pointers to indirect blocks

Used in BSD UNIX 4.3

Variants used in other UNIXes
- Directory entries are **i-nodes** (index nodes)



Multilevel Indexed Files: Pros & Cons

Advantages
- Simple to implement
- Supports incremental file growth
- Small files?

Disadvantages
- Indirect access: inefficient for random access to very large files
- Lots of seeks (data not contiguous)

Is file size bounded?

Other Issues

Block sizes

Free space management

Journalling

Sparse allocation

Not issues we consider in CSSE2310