File Systems

- One of the most visible pieces of the OS
- Contributes significantly to usability (or the lack thereof)

1 / 29

Design Choices

2 / 29

Files and File Systems

- What's a file?
- You all know what a file is...
- What's a file system?
- A *file system* is a mapping of *file names* to a subset of a physical medium

2 / 29

Many Types of File Systems

- Unix file systems
- Windows file systems
- CDs
- DVDs
- DECtapes — ancient variant of magnetic tape that permitted seeking and overwriting individual blocks in the middle
- They all had variants

3 / 29

Design Decisions

- File names
- Hierarchical or non-hierarchical
- Types of access control
- Special media characteristics
- Available space
- Media speed
- Read-only or read-write
- Versioning
- Fault recovery
- Record-, byte-, or block-structured
- Many more

4 / 29

File Names

- Case sensitivity
- Extensions — permitted, required, uninterpreted, length
- Character set
- Uniform or non-uniform file-naming scheme

5 / 29

Types of Names

- /this/is/a/unix.file.name
- q:\Windows likes\mOnOcaSE
- >Multics>uses<for>up-a-level
- os.360.looks.hierarchical.but.isnot(really)
- vms-host::device[direct.ory.list]name.type;1

6 / 29

Other Disks

- How are other disk drives named?
- On Unix, they're a subtree of the file system, via the `mount` command
- Multics had a single tree, but directories always lived on permanently-mounted disks
- On Windows and VMS, a drive letter is explicitly given
- On OS/360 and /370, you could use explicit disk identifiers *or* you could use the "system catalog" — a hierarchical structure for file names where the "directories" didn't need to live on the same disks as the files

7 / 29

Extensions

- On Unix and Multics, the kernel knows nothing of filename extensions; they're just characters. But some applications (`make`, the C compiler, etc., care)
- The old DOS filesystem used "8.3" format: 8-character filenames, with optional 3-character extension
- OS/360 confused extensions with directory levels

8 / 29

Media Characteristics

- Today's disks have fixed-size, 512-byte sectors
- Old mainframe disks had variable-size sectors, selected by the application
- CDs, even if writable, aren't block-writable the way disks are
- Blocks on flash disks can only be written a certain number of times before they wear out
- WORM disk blocks can be written and deleted, but not overwritten

9 / 29

Space and Speed

- How big can the file system be? How small must it be?
- Is space really tight? Can you trade space for speed?
- How large can files be? What is a typical size?

10 / 29

Versioning

- Some file systems (VMS, Tenex, TOPS-20) include a version number as part of the file name
- Plan 9 provides file system views at any arbitrary point in time

11 / 29

File Structures

- Old systems: files were sequences of records, intially images of punch cards or printer lines
- Unix: randomly-addressable sequences of bytes
- Page-oriented file systems: sequence of page-sized blocks

12 / 29

How Many Names?

- Can files have more than one name?
- What about directories?
- Are all names equal?
- How do you keep the file system graph acyclic? Or is it just a tree?

13 / 29

Underlying Hardware 14 / 29

Underlying Disk Hardware

- As noted, today's disks use fixed-size blocks; some older disks did not
- IBM disks supported hardware *search keys* — you could seek to a track, then look for that key somewhere on the track
- Goal was good hardware support for databases — offload the CPU, by having the disks do more work

14 / 29

Supporting Such Disks

- Space allocation has to support track granularity
- Underlying I/O primitives have to support that mode of access
- If we mix block-structured files and search key files on the same disk, do we have to support tracks with both kinds of blocks?

15 / 29

Speed and Optimization

- Does the speed of the disk affect the file system design?
- What do we really know about the hardware?
- Old disks were based on cylinders, heads, and sectors; newer disks lie to the OS
- How do keep a file localized? Do we need to?

16 / 29

Interace to Lower Layers

- What is the interface to lower layers?
- Unix assumes "read/write block N"
- Multics has none; it's just the paging system
- OS/360 permitted very general I/O requests

17 / 29

Space Allocation 18 / 29

Space Allocation

- What is our unit of space allocation?
- Does the programmer have to reserve space in advance?
- How much does the OS have to know about the application's data formats?

18 / 29

Units of Space Allocation

- Some systems required space to be allocated in hardware-based units: tracks, cylinders, etc.
- With variable sector sizes, how you packed the data determined how much you could fit onto a track and hence onto a disk
- If you packed more records into a single block, there were fewer blocks and hence fewer interblock gaps on each track

19 / 29

Keeping Track

- The OS has to keep track of all allocated space: which files own which parts of the disk
- What data structure is best?
- Also need to keep track of free space on the disk
- Space allocation is similar to RAM allocation
- Space allocated to a file may or may not need to be contiguous, depending on other design decisions

20 / 29

Metadata

- We need to store a lot of data about the file
- Most obvious: disk blocks
- Other data: file creator, dates created/modified/accessed
- File permissions

21 / 29

Other Metadata

- Read-only flag, independent of other file permissions
- "Hidden"
- System file
- Archive needed (similar to "page modified" flag)
- ASCII/Binary
- Record size

22 / 29

File Types

- Some operating systems consider some files to be special in some way
- On Unix, most I/O devices have filenames
- Directories are sometimes just files
- Some communications channels have filenames

23 / 29

Linux Metadata

```
struct stat {
  dev_t         st_dev;      /* device */
  ino_t         st_ino;      /* inode */
  mode_t        st_mode;     /* protection */
  nlink_t       st_nlink;    /* number of hard links */
  uid_t         st_uid;      /* user ID of owner */
  gid_t         st_gid;      /* group ID of owner */
  dev_t         st_rdev;     /* device type (if inode device) */
  off_t         st_size;     /* total size, in bytes */
  blksize_t     st_blksize;  /* blocksize for filesystem I/O */
  blkcnt_t      st_blocks;   /* number of blocks allocated */
  time_t        st_atime;    /* time of last access */
  time_t        st_mtime;    /* time of last modification */
  time_t        st_ctime;    /* time of last status change */
};
```

24 / 29

Where is Metadata Stored?

- In the file?
- In the directory entry?
- Elsewhere?
- Split?

25 / 29

Crash Recovery 26 / 29

Crash Recovery

- Must ensure that file systems are in a consistent state after a system crash
- Example: don't write out directory entry before the metadata
- Example: File systems are generally trees, not graphs; make sure things always point somewhere sane
- What if the file has blocks but the freelist hasn't been updated?
- Principle: order writes to ensure that the disk is always in a *safe* state

26 / 29

Repairing Damage

- At boot-time, run a consistency checker except after a clean shutdown
- Example: `fsck` (Unix) and `scandisk` (Windows)
- Force things to a safe state; move any referenced blocks to a recovery area

27 / 29

Log-Structured File Systems

- Instead of overwriting data, append the changes to a journaling area
- The file system is thus always consistent, as long as writes are properly ordered.
- Hmm — is that a reasonable assumption?

28 / 29

Modern Disks

- Modern disks do a lot of buffering
- Cache size on new Seagate drives: 2-16M bytes
- The drive will reorder writes to optimize seek times
- If a bad block has been relocated, you can't even predict when seeks will occur; only the drive knows
- What if the power fails when data is buffered?

29 / 29