File Systems Management and Examples

Today

- Efficiency, performance, recovery
- Examples

Next

- Mass storage and I/O

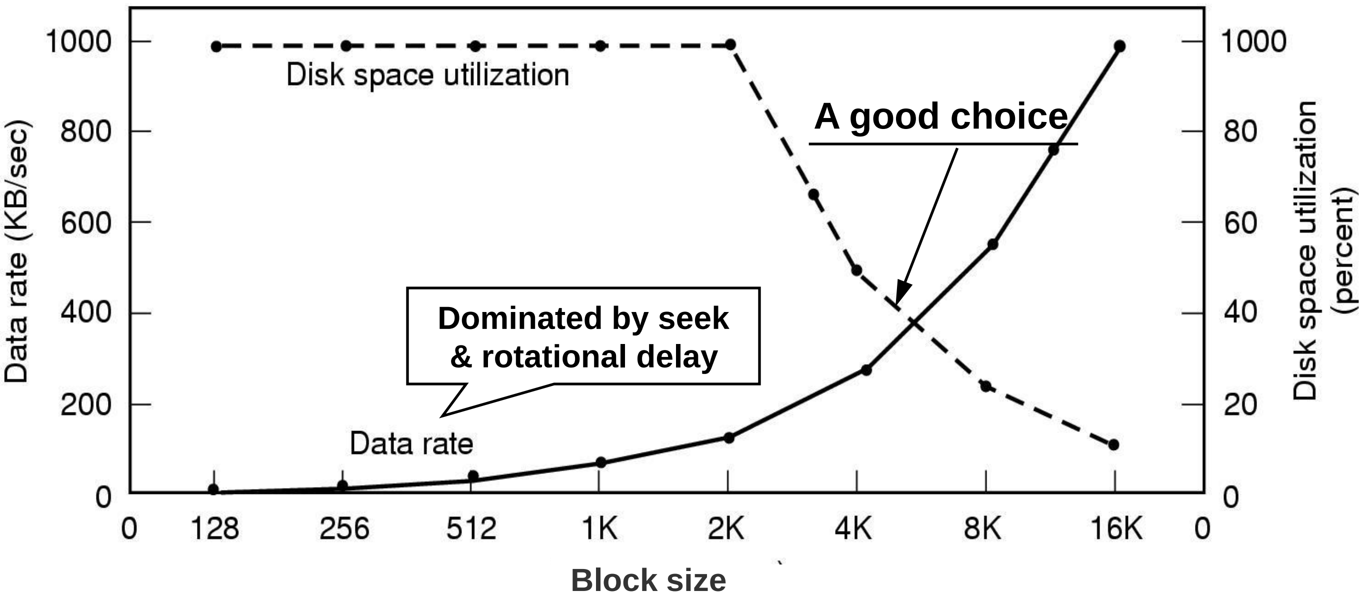
Disk space management

- Once decided to store a file as sequence of blocks
  - What's the size of the block?
    - Good candidates: Sector, track, cylinder, page
    - Pros and cons of large/small blocks
    - Decide base on median file size (instead of mean)
    - Clearly performance and space utilization are in conflict

Dark line (left) gives data rate of a disk

Dotted line (right) gives disk space efficiency

Assume all files 2KB



Disk space management

- Keeping track of free blocks
  - Storing the free list on a linked list
    - Use a free block for the linked list (holding as many free disk block numbers as possible)
  - A bit map (only one bit per block)
  - *When would the linked list require fewer blocks than the bitmap?*
    - Only if the disk is nearly full
- And if you tend to run out of free space, control usage
  - Quotas for user's disk use
  - Open file entry includes pointer to owner's quota rec.
  - Soft limit may be exceeded (warning)
  - Hard limit may not (log in blocked)

File system reliability

- Need for backups
  - Bad things happen & while HW is cheap, data is not
- Backup - needs to be done efficiently & conveniently
  - Not all needs to be included – /bin?
  - Not need to backup what has not changed – incremental
    - Shorter backup time, longer recovery time
  - Still, large amounts of data – compress?
  - Backing up active file systems
  - Security
- Strategies for backup
  - Physical dump – from block 0, one at a time
    - Simple and fast
    - You cannot skip directories, make incremental backups, restore individual files

File system reliability

- Logical dumps
  - Keep a bitmap indexed by i-node number
  - Bits are set for
    - Modified files
    - Every directories
  - Unmarked directories w/o modified files in or under them
  - Dump directories and files marked
- Some more details
  - Free list is not dump, reconstructed
  - Files linked from multiple places restored just one
  - Unix files may have holes (core files are a good example)
  - Special files, named pipes, etc. are not dumped

File system reliability

- File system consistency
- fsck/scandisk ideas
  - Two kind of consistency checks: blocks & files
  - Blocks:
    - Build two tables – a counter per block and one pass; every block should be in exactly one list
  - Similar check for directories – link counters kept in i-nodes

Consistent state

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

Missing block

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

Solution – add it to the free list

Twice in free list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1	Free blocks

Solution – rebuild the free list

Part of more than one file

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

Solution – duplicate data block

File system performance …

- Caching – to reduce disk access
  - Too many blocks in a cache – Hash (device & disk address) to find block in it
  - Cache management ~ page replacement
    - But cache references are rare so you can keep a LRU linked list
  - The catch? Plain LRU is undesirable
    - Essential blocks should be written out right away
    - If blocks would not be needed again, no point on caching
  - Unix sync and MS-DOS write-through cache
    - Why the difference? At the beginning …
    - UNIX – Hard disk were the norm
    - MS-DOS – Started out with floppy disks

File system performance

- Block read ahead
  - Clearly useless for non-sequentially read files
  - Maybe the file system can keep the track of access pattern for an open file
- Reducing disk arm motion
  - Put blocks likely to be accessed in sequence close to each other
  - I-nodes placed at the start of the disk
  - Disk divided into cylinder groups - each with its own blocks & i-nodes (McKusick et al.' FFS)
    - To allocate according to cylinder groups you need enough free space → save 10% of the disk just for that!

Log-structured file systems

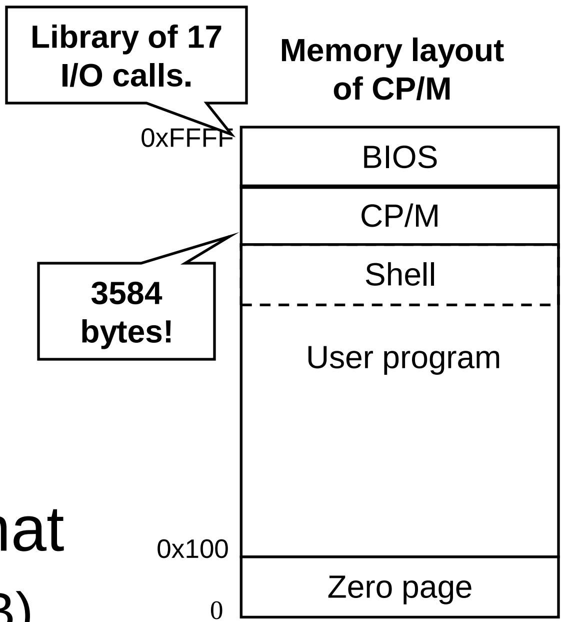
- CPUs getting faster, memories larger, disks bigger
  - But disk seek time lags behind
  - Since disk caches can also be larger → increasing number of read requests can come from cache
  - *Thus, most disk accesses are writes*
- To make matter worse
  - Writes are normally done in very small chunks
    - To create a new file, writes for: directory i-node, directory block, file i-node, and file’s blocks
  - Low disk efficiency, most of the time gone on seek and rotational delay
- LFS strategy - structure entire disk as a log to achieve the full bandwidth of the disk

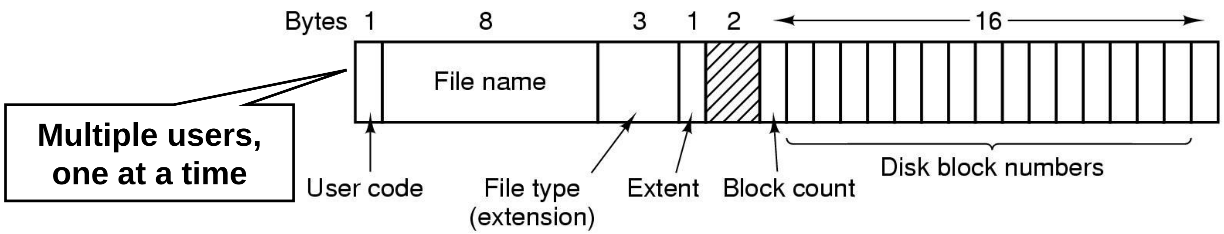
LFS – Disk as a log

- All writes initially buffered in memory
- Periodically write buffer to end of disk log
  - Each segment may contain, all mixed together, i-nodes, directory blocks, and data blocks
  - Each segment has a summary at the start
- When file opened, locate i-node, then find blocks
  - But i-nodes are not at a fixed position but scattered all around
  - Keep an i-node map in disk, index by i-node, and cache it
- To deal with finite disks: cleaner thread
  - Compact segments starting at the front, first reading the summary, creating a new segment, marking the old one free

The CP/M file system

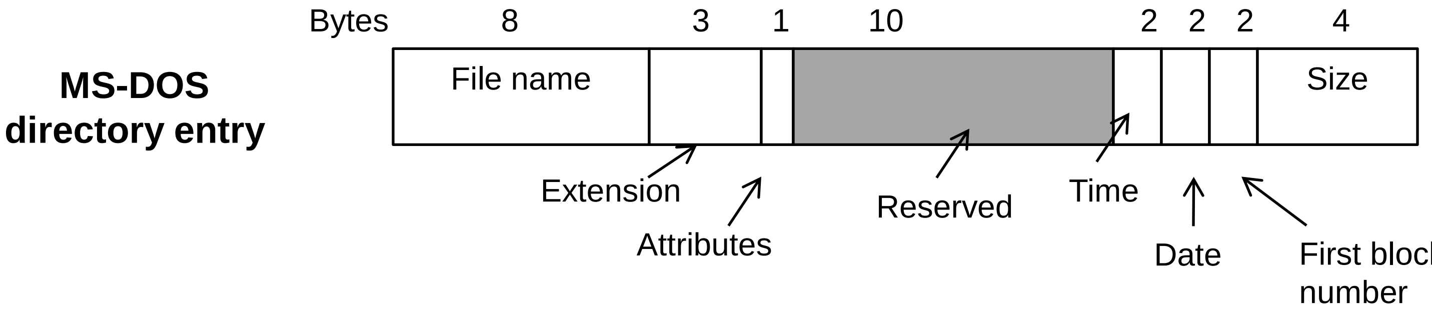
- Control Program for Microcomputers
- Run on Intel 8080 and Zilog Z80
  - 64KB main memory
  - 720KB floppy as secondary storage
- Separation bet/ BIOS and CP/M for portability
- Multiple users (but one at a time)
- The CP/M (one) directory entry format
  - Each block – 1KB (but sectors are 128B)
  - Beyond 16KB – Extent
  - (soft-state) Bitmap for free space





The MS-DOS file system

- Based on CP/M; used by the iPod
- Biggest improvement: hierarchical file systems (v2.0)
  - Directories stored as files – no bound on hierarchy
  - No links – so basic tree
- Directory entries are fixed-size, 32 bytes
- Names shorter than 8+3 characters are left justified and padded



The MS-DOS file system

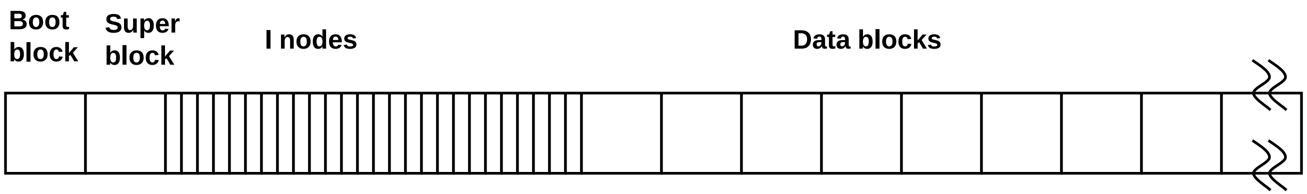
- Attributes include:
  - Read-only – to avoid accidental damage
  - Hidden – to prevent it from appearing in a listing
  - Archived – used by, for instance, a backup system
  - System – Hidden and cannot be deleted using ‘del’
- Time – 5b for seconds, 6b for minutes, 5b for hours
  - Accurate only to +/-2 sec (2B – 65,536 sec of 86,400 sec/day)
- Date – 7b for year (i.e. 128 years) starting at 1980 (5b for day, 4b for month)
- Size is a 32bit number (so, theoretical up to 4GB files)

The MS-DOS file system

- Another difference with CP/M – FAT
  - First version FAT-12: 12bit disk addresses & 512B blocks
  - Max. partition 2^{12}x 512 ~ 2MB
  - FAT with 4096 entries of 2 bytes each – 8KB
- Later versions: FAT-16 and FAT-32 (actually only the low-order 28-bits are used)
- Disk block sizes can be set to multiple of 512B
- FAT-16:
  - 128KB of memory for the FAT table
  - Largest partition – 2GB ~ with block size 32KB
  - Largest disk - 8GB (MS-DOS limit of 4 partitions per disk)
- *How do you keep track of free blocks?*

The UNIX V7 file system

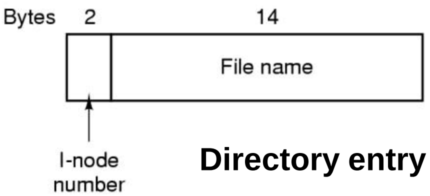
- Unix V7 on a PDP-11
- Tree structured as a DAG
- File names up to 14 chars (anything but “/” and NUL)
- Disk layout in classical UNIX systems



- Each i-node – 64 bytes long
- I-node’s attributes
  – file size, three times (creation, last access, last modif.), owner, group, protection info, # of dir entries pointing to it
- Following the i-nodes – data blocks in no particular order

The UNIX V7 file system

- A directory – an unsorted collection of 16-bytes entries

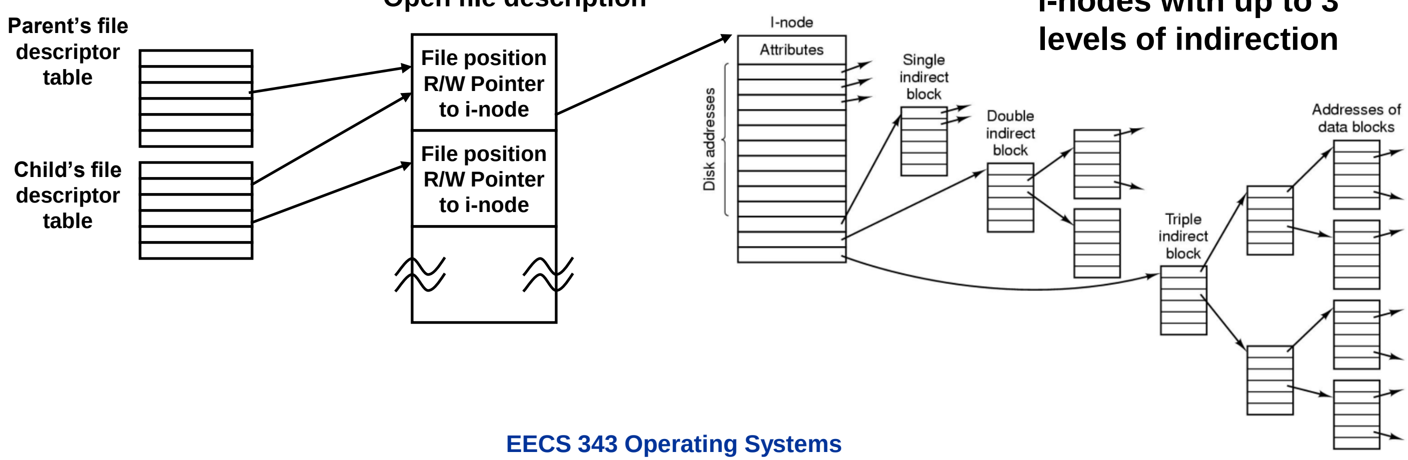


Directory entry

- File descriptor table, open file descriptor table and i-node table – starting from file descriptor, get the i-node
  - Pointer to i-node in the FD table? No, where to put the current pointer? Multiple processes each w/ their own
  - File descriptor table? No, parent and children cannot share it
  - New table – the open file description

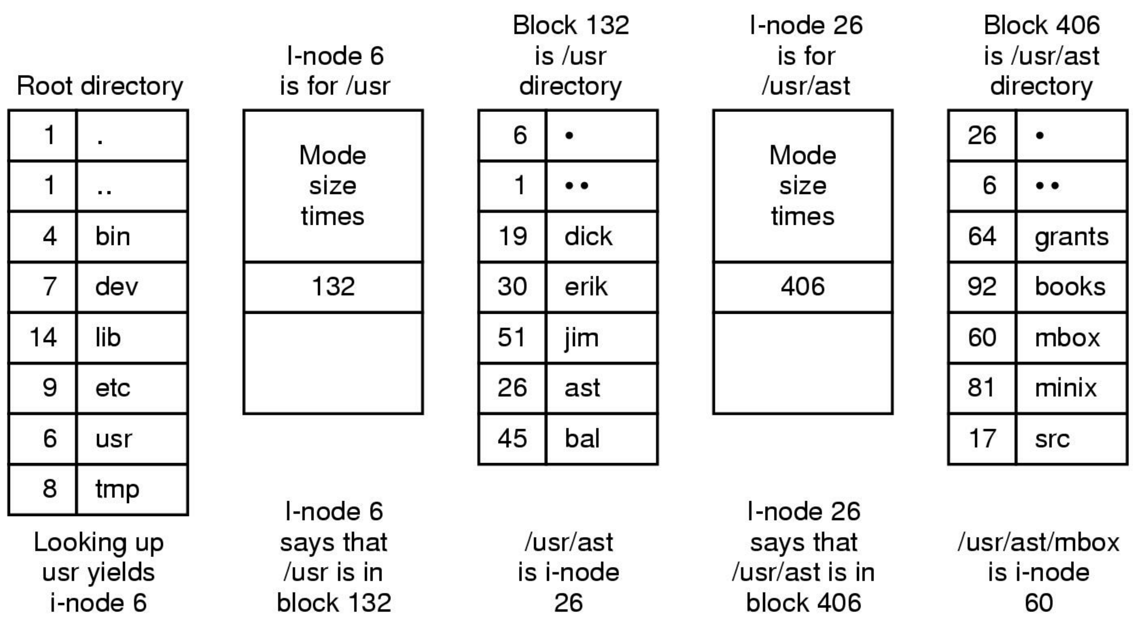
Open file description

i-nodes with up to 3 levels of indirection



The UNIX V7 file system

- Steps in looking up /usr/ast/mbox
  - Locate root directory – i-node in a well-known place
  - Read root directory
  - Look for i-node for /usr
  - Read /usr and look for ast
  - …



Next Time

- Mass storage and I/O