Operating Systems

Filesystems

1

What is a file?

- Wikipedia says: The word "file" was used publicly in the context of computer storage as early as February 1950 in an RCA advertisement in Popular Mechanics
    - *"...the results of countless computations can be kept "on file" and taken out again. Such a "file" now exists in a "memory" tube developed at RCA Laboratories. Electronically it retains figures fed into calculating machines, holds them in storage while it memorizes new ones - speeds intelligent solutions through mazes of mathematics."*
  - Also used to refer to information on punched cards
- In general a file is an array of bytes stored on some persistent medium

2

File Attributes

- **Name** – 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 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

3

File Attributes

- **Flags -** indicating status including
  - Hidden
  - Read only
  - System
  - Text/binary
- **Archival information**

4

File Operations

- **Create / Delete**
- **Write / Read**
- **Reposition within file (Seek)**
- **Truncate**
- **Append**
- **Move / Rename**
- **Get/Set Attributes**
- *Open(F_i)* – search the directory structure on disk for entry F_i, and move the content of entry to memory
- *Close (F_i)* – move the content of entry F_i in memory to directory structure on disk

5

Open Files

- An open file has various bits of state
  - 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

6

Open File Locking

- Some file systems support locking part or all of a file
  - This can be implemented like a mutex around a range of bytes
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
    - If the filesystem access is a system call, then the system can enforce the lock by suspending the thread
  - **Advisory** – processes can ignore locks

7

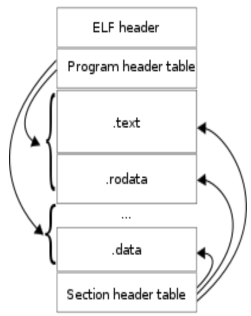
File Structures

- Array of bytes
  - A linear address space
- Array of records
  - Records like in a database or on a punch card
- More complex generic structures
  - Trees
- Specific file types
  - Loadable library
  - Document with formatting
- Any structure can be implemented atop a byte array with appropriate delimiters (control symbols)

8

File Types and Structures

- Different types of files have different internal structure
- The structure could be visible to the OS or only via a library or program
  - The kernel's interface to the filesystem may not be aware of structure, but at some level the OS must understand the structure of executables to create processes from them



9

File Types

- Sometimes the type of a file is communicated via filename convention
- In UNIX/Linux, some files begin with a “magic number” identifying them
  - The “file” command looks this up for you, and not finding that, checks to see if it is text
- The MacOS HFS+ filesystem supports files with a resource fork and a data fork
  - The resource fork contains metadata independent of the data fork

10

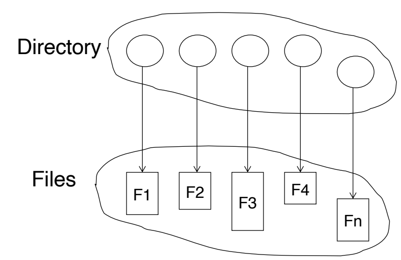
File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - Necessary when the medium was magnetic tape
- Random access
  - bytes/records read in any order
  - essential for database systems
  - simulating random access with sequential access
    - move file marker (seek), then read or …
    - read and then move file marker

11

Directory Structure

- A collection of nodes containing information about all files
- Both the directory structure and the files reside on disk



12

Operations Performed on Directories

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Two types of links

- Link
  - "Hard" link
  - Multiple directory entries point to the same file (inode)
  - Reference count
- Symbolic link
  - Or "soft" link
  - Directory entry that points to the path of the actual file

File System Mounting

- A file system must be mounted before it can be accessed
- Verify and cache filesystem information
- An unmounted file system is mounted at a mount point
- Mounting can be automatic or manual
- Mounting can obscure existing files

Protection

- File owner/creator is able to control what can be done and by whom
  - Enforced by OS
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights
- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **user (owner) access**	7	$\Rightarrow$	1 1 1
			RWX
b) **group access**	6	$\Rightarrow$	1 1 0
			RWX
c) **other (public) access**	1	$\Rightarrow$	0 0 1

- Ask administrator to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

chmod 761 game

Attach a group to a file: `chgrp G game`

17

System Commands You Should Know

- ls
- mv
- rm
- ln
- mknod
- fsck

18

System Calls You Should Know

- read
- write
- open
- close
- rename
- link
- unlink

19

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
- File control block is a structure consisting of information about a file
- Structured in layers
- Device driver controls the physical device

20

File-System Implementation

- Boot control block contains info needed by system to boot OS from that volume
- Volume control block contains volume details
- Directory structure organizes the files
- Per-file File Control Block (FCB) contains many details about the file

21

File Control Block

- Type of the file (special files, etc)
- Link count
- Owner, group and access permissions
- Size of the file
- Date (last modification, creation, etc.)
- Name of the file (maybe)
- Pointers to data blocks

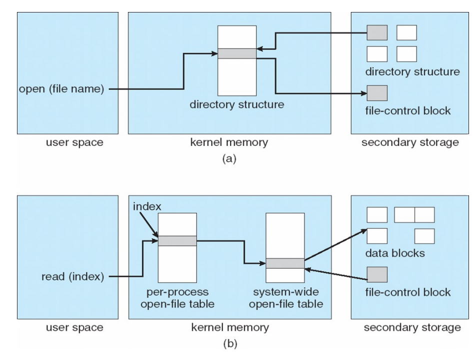
22

Directory Implementation

- **Linear list** of file names with pointer to the metadata or data blocks.
  - simple to program
  - time-consuming to manipulate
- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

23

In-Memory File System Structures



24

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

25

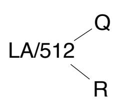
Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow

26

Contiguous Allocation

- Mapping from logical to physical



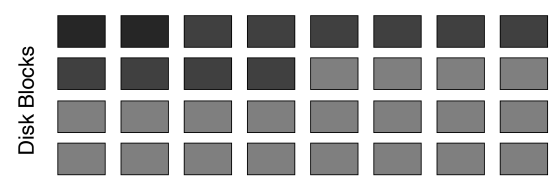
Block to be accessed = starting address / block size
Displacement into block = R

27

Contiguous Allocation

- Like memory, easy to allocate and manage but inflexible
- Good for WORM (write once read many)

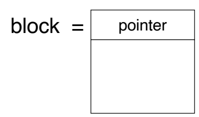
File	Start	Length
Foo	0	2
Bar	2	10



28

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



29

Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

LA/511 Q R

Block to be accessed is the Qth block in the linked list of blocks representing the file.
Displacement into block = R + 1

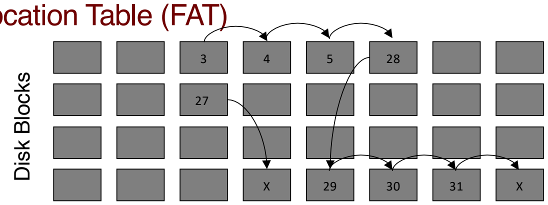
File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

30

Linked Allocation

- Easy to allocate, very flexible
  - Overhead from pointer dereferencing
- Used by MS-DOS File Allocation Table (FAT)

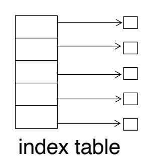
File	Start	Length
Foo	10	2
Bar	2	8
		Optional



31

Indexed Allocation

- Brings all pointers together into the index block.
- Logical view

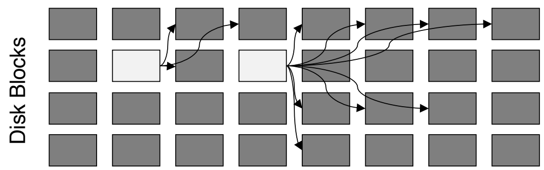


32

Indexed Allocation

- Easy to allocate, flexible
  - Less overhead from pointer dereferencing
- Allocate new index block as needed

File	Index	Length
Foo	9	2
Bar	11	8



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but incurs the overhead of index blocks
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

$$LA/512 \begin{cases} Q \\ R \end{cases}$$

Q = displacement into index table
R = displacement into block

Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).

$$LA / (512 \times 511) \begin{cases} Q_1 \\ R_1 \end{cases}$$

Q_1 = block of index table
R_1 is used as follows:

$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)

- Two-level index (maximum file size is 512^3)

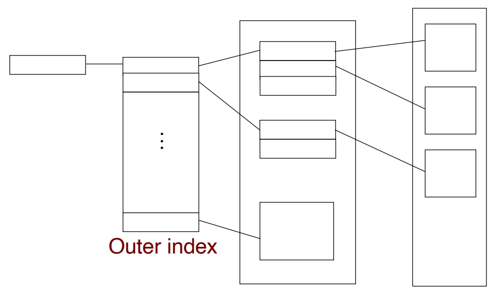
$$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

Q_1 = displacement into outer-index
R_1 is used as follows:

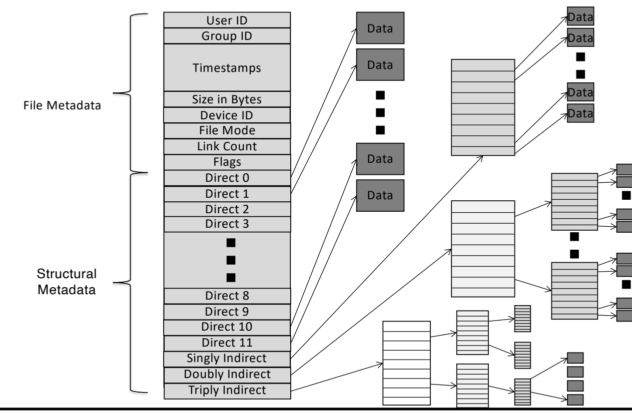
$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

Q_2 = displacement into block of index table
R_2 displacement into block of file:

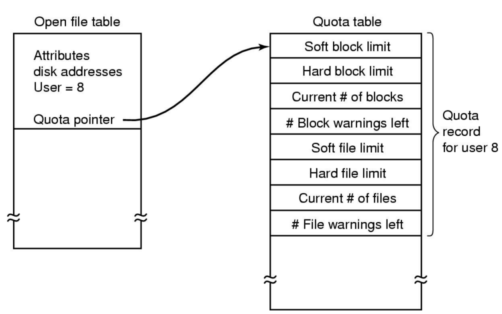
Indexed Allocation – Mapping (Cont.)



The UFS inode (used by ext2/3)

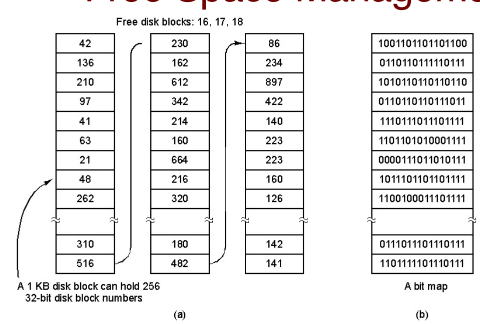


Disk Space Management



Quotas for keeping track of each user's disk use

Free Space Management



(a) Storing the free list on a linked list
(b) A bit map

Free Space Management

- Consider a 32GB disk (2^24 2KB blocks)
  - 24576 blocks to hold a free list
  - 8192 blocks to hold a bitmap
- When the list method is used, only one block of pointers needs to be kept in memory
  - When a file is created, the needed blocks are taken from the in-memory block of pointers
  - When it runs out, a new block is read in; when it fills, it is written to disk

41

Disk Space Management

- With a bitmap, only one block needs to be kept in memory, as well
- With a bitmap, blocks are allocated close together, or contiguously
- Since the structure is a fixed size, it can be completely brought in to memory and paged out if necessary

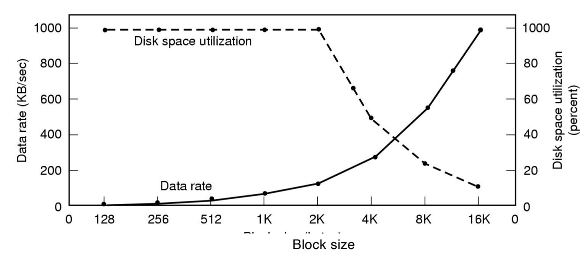
42

Disk Space Management

- Efficiency Concerns
  - How big should the blocks be?
  - Larger blocks perform better
    - More work done for each operation
  - Smaller blocks are more efficient for storage
    - Less internal fragmentation
  - Workload Dependent
    - Most files on Unix systems are small (<2KB)
    - Scientific datasets tend to be quite large

43

Disk Space Management



- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency
- All files 2KB
- Example from Modern Operating Systems, Tanenbaum

44

Disk Performance

```
while ((n = read(IN_FILENO, buf, BUFFSIZE)) > 0)
  if (write(OUT_FILENO, buf, n) != n)
       error("write error");
```

Buffer size and overheads (from Advanced Programming in the UNIX Env.)

BUFFSIZE	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)	Number of loops
1	20.03	117.50	138.73	516,581,760
2	9.69	58.76	68.60	258,290,880
4	4.60	36.47	41.27	129,145,440
8	2.47	15.44	18.38	64,572,720
16	1.07	7.93	9.38	32,286,360
32	0.56	4.51	8.82	16,143,180
64	0.34	2.72	8.66	8,071,590
128	0.34	1.84	8.69	4,035,795
256	0.15	1.30	8.69	2,017,898
512	0.09	0.95	8.63	1,008,949
1,024	0.02	0.78	8.58	504,475
2,048	0.04	0.66	8.68	252,238
4,096	0.03	0.58	8.62	126,119
8,192	0.00	0.54	8.52	63,060
16,384	0.01	0.56	8.69	31,530
32,768	0.00	0.56	8.51	15,765
65,536	0.01	0.56	9.12	7,883
131,072	0.00	0.58	9.08	3,942
262,144	0.00	0.60	8.70	1,971
524,288	0.01	0.58	8.58	986

45

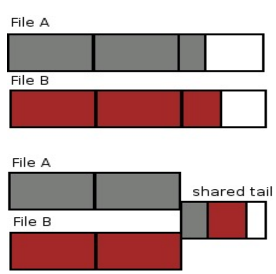
Extent-Based Systems

- Many newer file systems use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous group of disk blocks
  - Extents are allocated for file data
  - A file consists of one or more extents.

46

Disk Space Management

- Making smaller blocks perform better
  - use extents
- Making larger blocks more space efficient
  - tail packing



47

Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - Can improve performance by dedicating section of memory as virtual disk, or RAM disk

48

Buffer Cache (1)

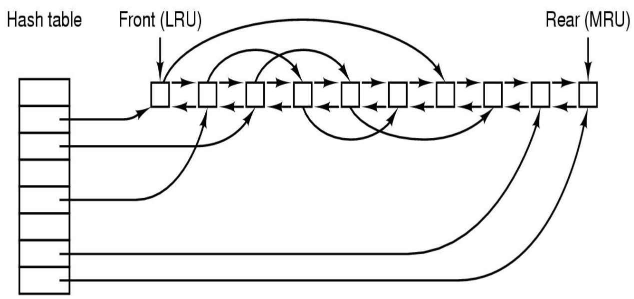
- Disks much slower than memory
- Buffer (or block) cache
  - Cache disk blocks in memory
  - Check cache before reading from/writing to disk
  - The memory for this is limited
    - Use replacement algorithms
- Benefits of Buffer Cache
  - Improves overall system performance
  - Write Merging
  - Proactive caching (read-ahead) can be used
    - E.g., get block N and N+1 in one read

Buffer Cache (2)

- True LRU can be implemented
  - The timescales are much different than for memory
- Buffer Cache Problems
  - Not writing inode blocks back increases the chances of a crash leaving the filesystem in an inconsistent state
- Solutions
  - Write out i-node blocks immediately
  - Put i-node blocks at the front of the LRU list
  - Write out all blocks immediately (used by MS-DOS)
    - known as write-through caching

Buffer Cache (3)

- Buffer cache data structures



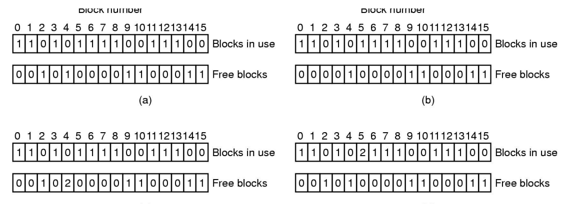
Virtual File Systems

- Virtual File Systems (VFS) provide an "object-oriented" way of implementing file systems
  - In the same way the I/O interface does
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Also: use system programs to back up data from disk to another storage device
- Recover lost file or disk by restoring data from backup

File System Reliability (1)



- File system states
  - (a) consistent
  - (b) missing block
  - (c) duplicate block in free list
  - (d) duplicate data block

File System Reliability (2)

- Keeping the File System Consistent
  - Write disk so it's never in an inconsistent state
  - Divide writing into transaction
    - Like a database transaction: A group of operations that must all succeed, or all fail.
    - Partially completed operations must be undone
  - Write entire transaction to disk journal and then start making changes to disk
  - Write transaction in order and use an atomic “commit”

File System Reliability (3)

- Keeping the Files Consistent
  - Write files to disk regularly
  - Allow user-level transactions
    - concurrency and security issues
  - Checksum each block
  - Write data blocks to disk before you update the relevant inode information

Journaling Filesystems

- Journaling file systems record each update to the file system as a transaction
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
  - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

57

Log Structured Filesystems

- With CPUs faster, memory larger
  - Disk caches can also be larger
  - More read requests can come from cache
  - Thus, most disk accesses will be writes
- A log structured filesystem considers the entire disk as a log
  - Have all writes initially buffered in memory
  - Periodically write these to the end of the disk log
  - When file opened, locate the most recent inode, then find blocks

58

The Storage Stack

System Calls

Buffer Cache

Virtual Filesystem

Filesystem

Block Device

59