operating systems

UNIT - 4

STORAGE MANAGEMENT

feedback/corrections: vibha@pesu.pes.edu

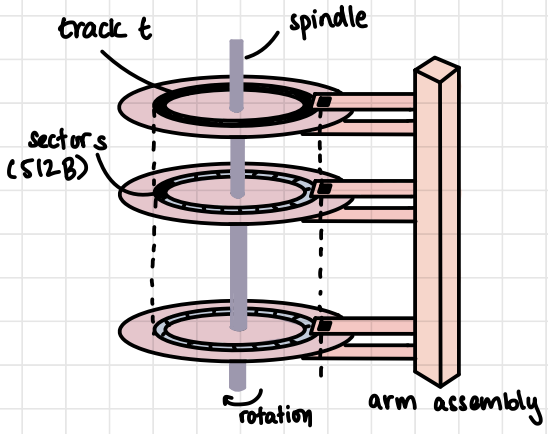
VIBHA MASTI

MASS STORAGE STRUCTURES

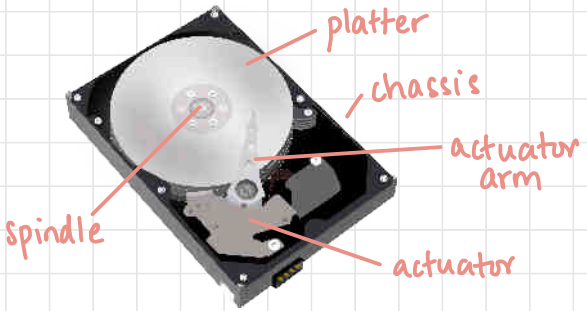
- secondary storage in modern computers
- Hard Disks (magnetic disks), SSDs, magnetic tape

1. Magnetic Disk

- consists of multiple flat cylindrical disks (platters) capable of rotating about a central axis called the spindle
- The surface of the platters is made of a magnetic material
- The disks are divided into concentric circular regions known as tracks, and the set of all tracks at a given distance from the spindle forms a cylinder (under arm)
- Around each disk is a read/write head that attaches to the main arm assembly. The read/write head can move linearly to access different tracks of the disk (innermost: track 0)
- The spindle rotates the disks so that the r/w head can access all the sectors of a track (speed: 60-250 rps)



- Diameter of platter: ranges from 1.8" to 3.5"
- Storage capacity typically ~30 GB to 4 TB per drive



Performance

- Transfer rate of HD: rate at which data transferred from secondary storage to primary storage ($t_{transfer}$: time taken to transfer one unit of data to RAM)
- $t_{transfer}$ depends on disk positioning time, which is the sum of seek time (time taken to move arm to the right cylinder) and rotational latency (time taken to rotate the disk such that the correct sector is under the r/w head)

$$t_{transfer} = \frac{\text{data to transfer}}{\text{transfer rate}}$$

$$t_{transfer} = \alpha (t_{seek} + t_{rot})$$

- Average access time = avg seek time + avg rotational latency

$$t_{avg\ rot\ latency} = \frac{\text{time period of rotation}}{2}$$

Head Crash

- There is a small cushion of air maintained between the read/write head and the disk at all times to protect the data
- A head crash occurs when the read/write head comes in contact with the rotating disk
- This typically results in a loss of data that cannot be recovered

I/O Bus

- Disk drives are attached to the computers (externally or internally) via I/O buses or interfaces
- Buses: EIDE, ATA, Serial ATA (SATA), Fibre Channel, SAS, Firewire, Universal Serial Bus (USB)
- Computer has host controller to communicate with HD's disk controller via I/O bus

2. Solid State Drive

- Nonvolatile memory for secondary permanent storage
- Storage technology: DRAM with battery, flash memory
- More reliable than HDs as there are no moving parts and consume less power
- More expensive and shorter life spans, lower capacity

- SSDs faster than HDs and bus interfaces are too slow for access (directly connected to PCI in many cases)
- SSDs are sometimes used as a tier of cache between HDs and primary storage

3. Magnetic Tape

- Old technology for secondary storage
- Permanent (lasts much longer), slow to retrieve data from (~1000x slower random access than HD)
- Storage of infrequently accessed data, backups
- Tape kept in a spool and wound past read/write head
- Accessing correct spot on tape can take minutes (access time) but read/write once spot is found is as fast as some disks (~140 MB/s)
- Tapes can have a capacity of 200 GB – 1.5 TB, typically
- Categorised by width: 4mm, 8mm, 19mm, ¼ inch, ½ inch
- Technologies: LTO-3, LTO-4, LTO-5, SDLT, T10000

Computer specialist John Smith arranges and examines cannisters of magnetic tape used in the processing of medical data at the National Library of Medicine.

(1960s)

Disk Scheduling

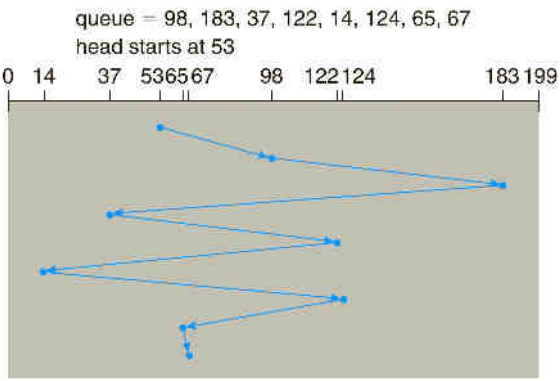
- Fast access time and large disk bandwidth; minimise seek time (time taken to move arm)
- Disk bandwidth = $\frac{\text{total no. of bytes transferred}}{\text{time between first req and transfer completion}}$
- Scheduling of I/O request servicing allows for faster transfer and improved bandwidth
- Whenever process needs to perform disk I/O operations, it issues a system call to the OS with specific information in the request
  - Operation type: input or output
  - Disk address for data transfer (logical)
  - Memory address for data transfer (logical)
  - No. of sectors to transfer
- If disk drive and disk controller available when request made, serviced immediately
- If not, request added to the end of pending requests queue, maintained by OS

DISK SCHEDULING ALGORITHMS

- To minimise access time (seek time and rotational latency) by scheduling requests made

1. FCFS (First Come First Serve)

- First request in queue serviced first (FIFO)
- Not efficient if addresses far apart

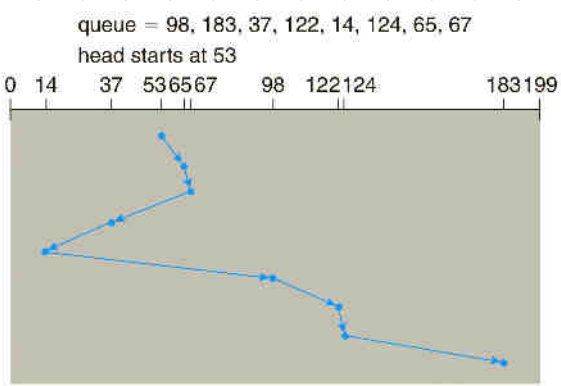


- Head movement: (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

  = 640 head movements

2. SSTF (Shortest Seek Time First)

- Request with shortest seek time serviced first
- Can lead to starvation of requests with large seek distances

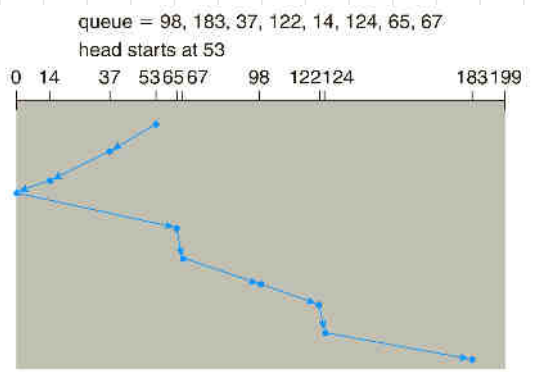


- Head movement: $(65-53)+(67-65)+(67-37)+(37-14)+(98-14)+(122-98)+(124-122)+(183-124)$

  $= 236$ head movements

3. SCAN Scheduling

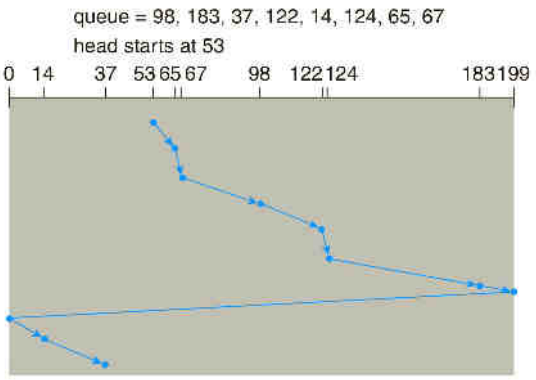
- Disk starts at one end of the disk and moves towards the other end, servicing requests until it reaches other end of the disk
- After reaching the other end, it starts scanning back to the first end, continuing to service requests
- Also called elevator algorithm
- Eg: head starts at 53, moving towards 0



- Head movements: $(53-0)+(183-0) = 236$
- Requests that appear right after r/w head crosses must wait for r/w head to reverse directions and come back

4. C-SCAN Scheduling

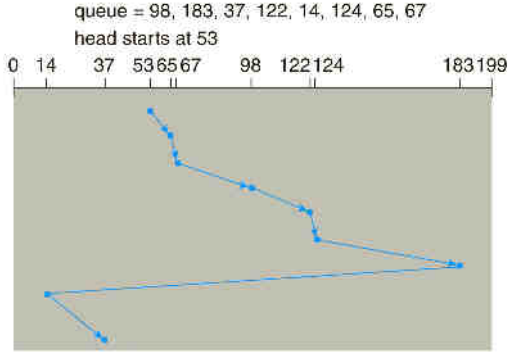
- More uniform wait time than SCAN
- Head moves from one end to another and then goes back to the first end without scanning in the opposite direction



- Cylinder treated as circular list

5. LOOK and C-LOOK Scheduling

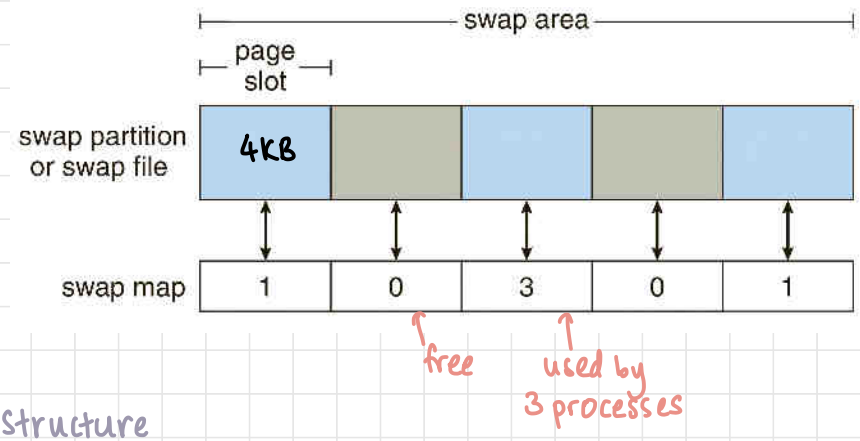
- LOOK is similar to SCAN but it goes only to the last request and not the end of the disk
- C-LOOK is similar to C-SCAN
- Eg: C-LOOK



Swap Space Management

- Virtual memory uses disk space as an extension of main memory when memory availability is critically low
- Swap space is present in disk as a part of the normal file system or as a separate disk partition (Linux)
- Swap space management – task of OS
- Swapping takes time and disk access is slower; need to optimise
- 4.3 BSD allocates swap space when process starts
  - holds text segment (program) and data segment
- Solaris 2 (Sun Microsystems) allocates swap space only when dirty page evicted from memory
- Faster to reread page from file system than to swap in and then swap out (Solaris)
- Swap space only used for anonymous memory (not backed by files — stack, heap)
- Kernel uses swap maps — array of integer counters (0 — available, positive integer — number of mappings to swapped page)

Data Structures for Swapping on Linux Systems



RAID Structure

- Redundant Array of (Independent) Inexpensive Disks — RAID — techniques
- Redundant data stored on multiple disks ; more reliable
- Higher data transfer rates

Redundancy vs Reliability

- A single disk less likely to fail than any disk out of N disks
- Mean time to failure of disk — approx. lifetime of single disk
- Array of disks of different data decreases mean time to failure (lifetime)

$$MTTF_N = (MTTF_1) / N$$

- Balance between redundancy and reliability

Mirroring

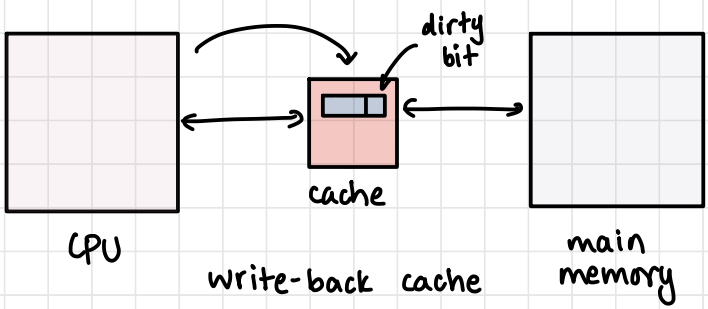
- Duplicate (mirror) data of a disk onto another disk
- Most expensive, most redundant
- One logical disk: 2 physical disks (mirrored volume)
- Data loss very rare

- Mean time to failure depends on mean time to failure of individual disks and mean time to repair

$$MTTF = \frac{(MTTF_1)^2}{2 \times MTTR}$$

assuming disk failures are independent

- Solid state non-volatite RAM (writeback cache) common to both disks used to protect from data loss during power failures (NVRAM)



- Rate of handling disk read requests: double
- Transfer rate same

Data Striping

- Split bits of each byte accross different disk; bit-level striping

- Eg: one byte split into 8 disks (bit i stored in i^{th} disk)

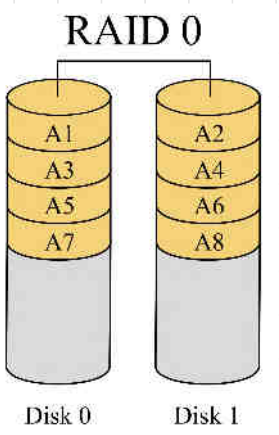
- Access rate: 8x normal rate
- Block-level striping: n disks, i^{th} block of a file goes to block $(i \bmod n) + 1$ —— block level most common
- Sector-level and byte-level striping also possible

Levels of RAID

- Combination of reliability of mirroring and data transfer rates of striping
- Low-cost redundancy and parity bits (error correction) —— combination to trade off cost and reliability

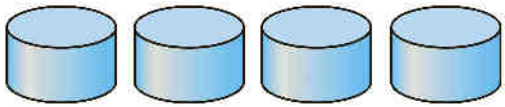
(a) RAID 0

- Non-redundant striping (block level)
- To store two disks' worth of data, two disks used



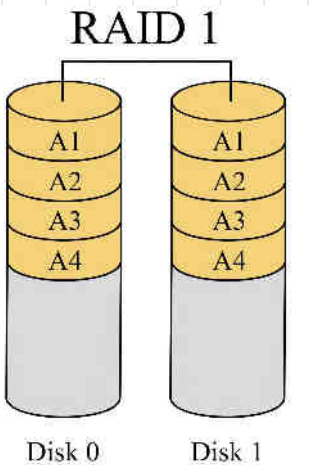
source: wikipedia

- For four disks:



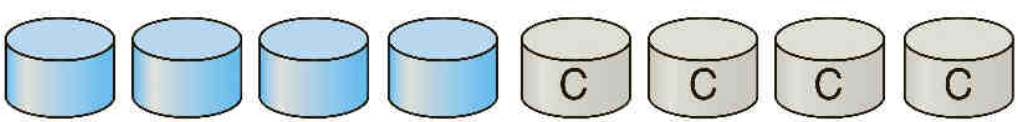
(b) RAID 1

- Complete mirroring (full redundancy, expensive)



source: Wikipedia

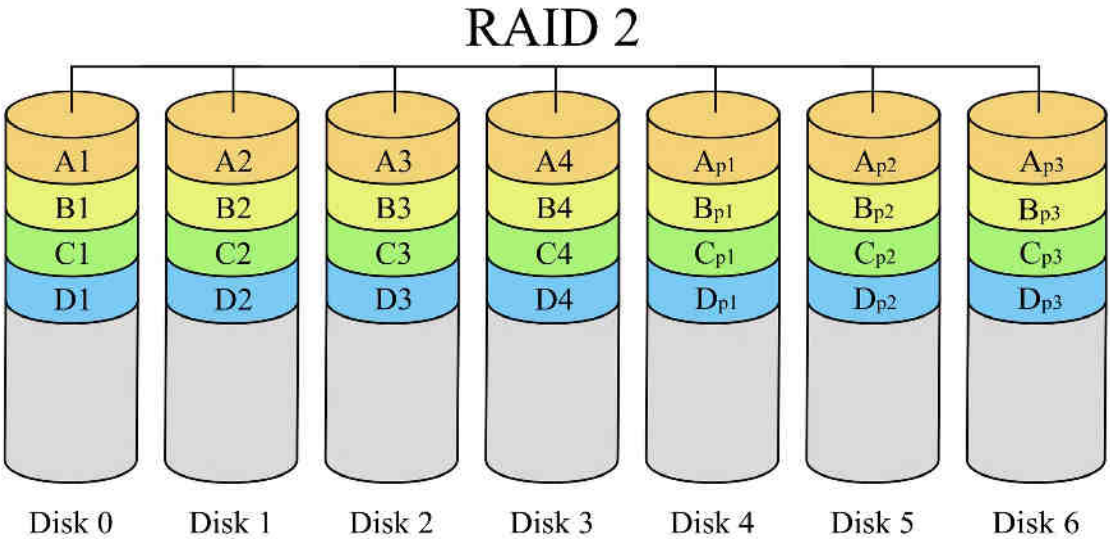
- Four disks



(c) RAID 2

- Memory style error-correcting codes (ECC) and bit striping (not block)

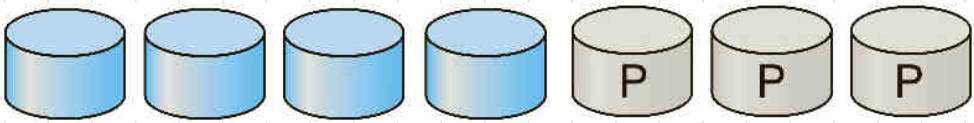
- Each byte in memory has parity bit associated with it: number of set bits in byte (bit=1) is even (parity=0) or odd (parity=1)
- Uses Hamming code for error detection (linear error correcting code) — can detect upto 2-bit errors and correct 1-bit errors
- [7,4] Hamming code encodes 4 data bits into 7 bits by adding 3 parity bits



source: wikipedia

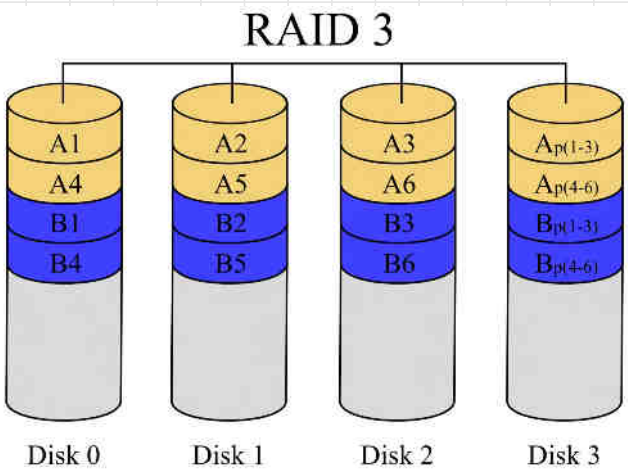
- For four disks' data, 3 disks overhead (Hamming(7,4) code — 3 parity bits for 4 data bits)

- Only original level of raid not currently used



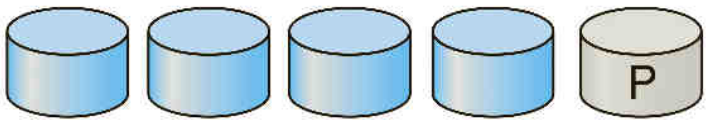
(d) RAID 3

- Bit interleaved parity organisation
- Disk controllers can detect if sector has been read correctly
- Single parity bit for error correction and detection
- Byte-level striping, single dedicated parity disk
- High transfer rates for apps that make long sequential read and write requests



source: wikipedia

- Bad performance for short, random access
- Rarely used in practice

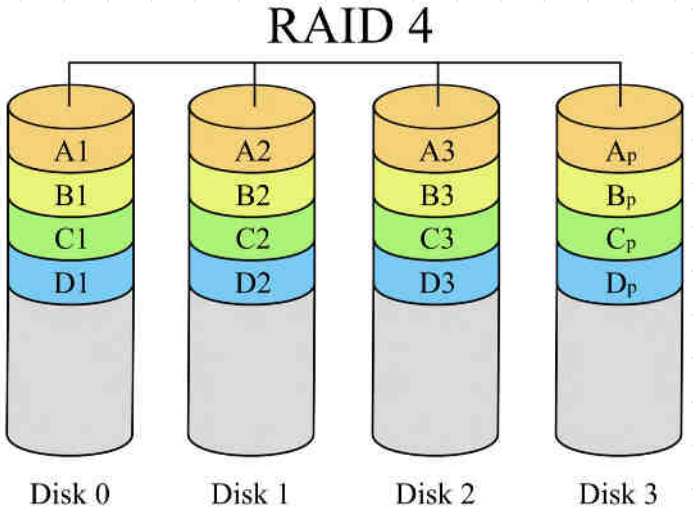


- Faster read time than RAID 1 (due to striping)
- Only single I/O request at a time as each byte split accross multiple disks and every disk must participate in single I/O request
- Overhead — computing parity bit implies slow write times
- Solution — dedicated hardware for parity computation, offloaded from CPU and cache to buffer disk blocks (NVRAM)

(c) RAID 4

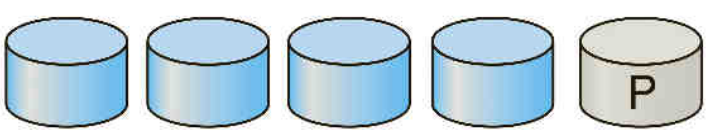
- Block-interleaved parity organisation
- Block level striping (RAID 0)
- One disk for parity bit for corresponding block; disk failure — missing bit can be reconstructed from other bits and parity bits
- Good performance for random reads, bad performance of random writes

- Each block access requires only one disk access, so some level of parallelism can be achieved (eg: A1 & B2)



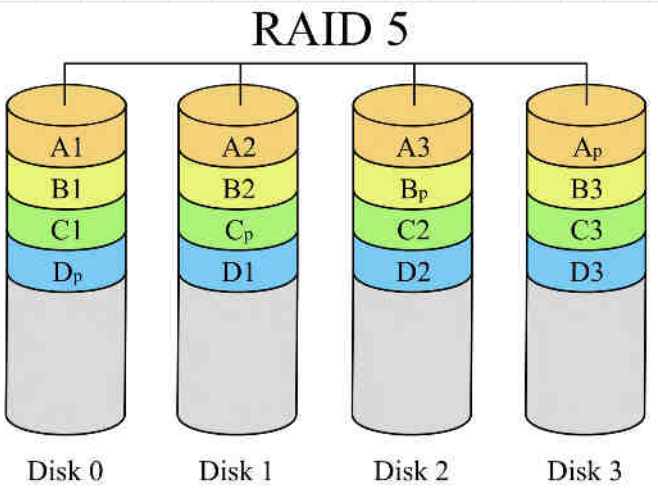
source: wikipedia

- Read-modify-write cycle; single write requires 4 disk accesses —— 2 reads (block & parity) and 2 writes (block & parity)
- Block must be read, modified and written back
- RAID 4 is scalable; allows new disks to be added (if they are initialised with 0's, the parity does not change



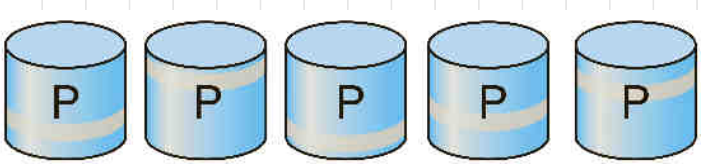
(f) RAID 5

- Block - interleaved distributed parity
- Instead of data in N disks and parity in 1 disk, data and parity distributed evenly across all disks



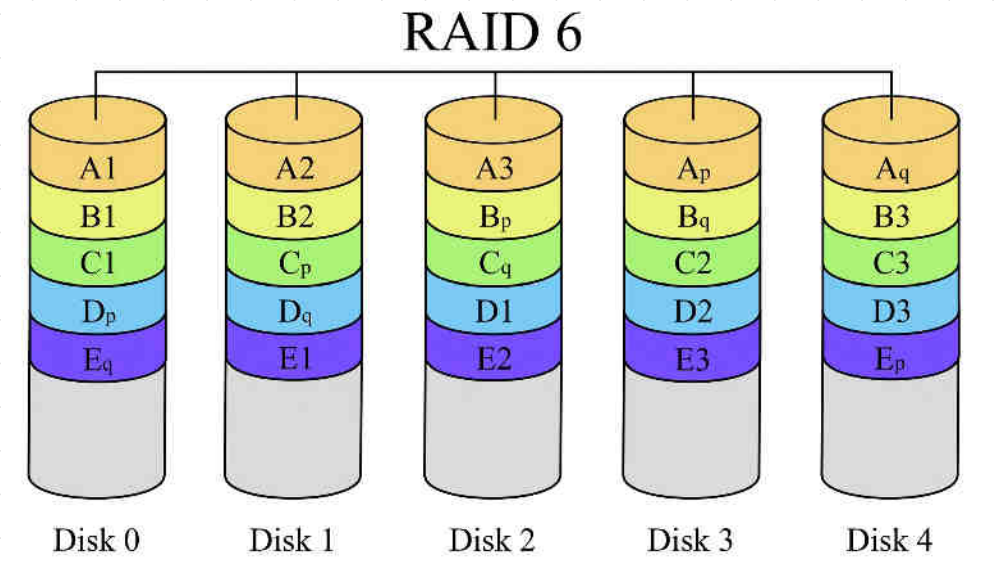
source: wikipedia

- For an array of n disks, the parity of the m^{th} block is stored in disk $(m \bmod n) + 1$
- A single disk cannot store a block and its parity as disk failure would result in loss of parity as well as block
- RAID 5 most commonly used parity system



(g) RAID 6

- P + Q redundancy scheme
- Additional redundant information to RAID 5 to guard against multiple disk failure
- Error checking codes (Reed-Solomon codes) used instead of parity in some layouts
- 2 bits of redundant data for every 4 bits of data; can tolerate two disk failures

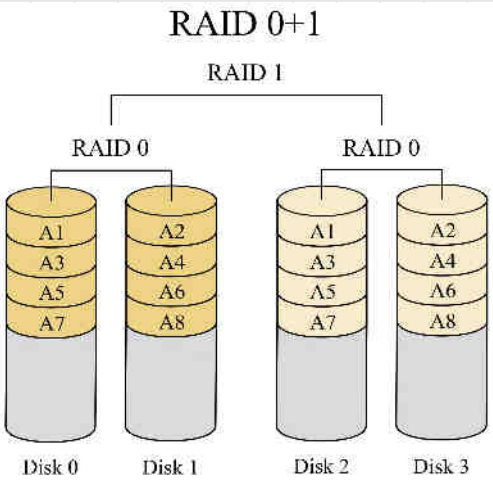


source: wikipedia

- Unlike RAID 2, 3, 4, 5, can tolerate upto 2 disk failures

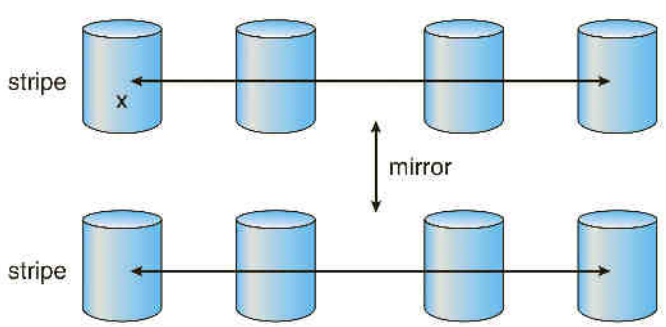
(h) RAID 0+1

- Combination of RAID 0 and RAID 1 — parallelism allowed
- Performance of RAID 0, reliability of RAID 1 (double the number of disks needed — like RAID 1)
- Better performance than RAID 5



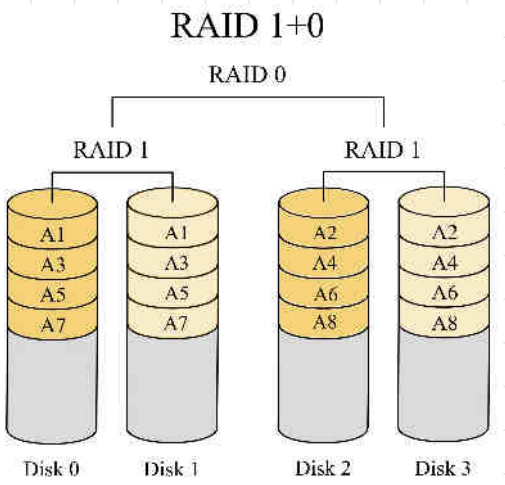
source: wikipedia

- RAID 0 striping + RAID 1 mirroring

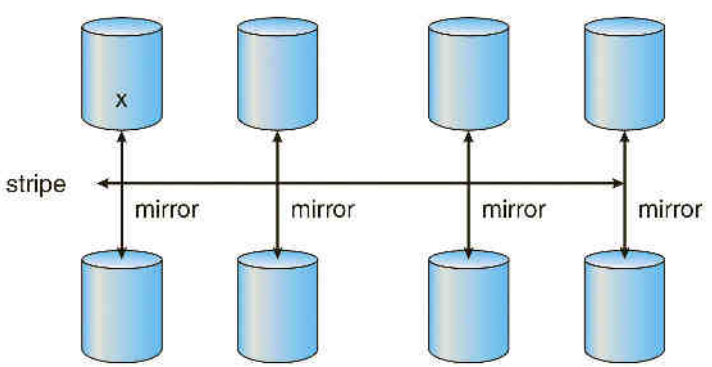


(i) RAID 1+0

- Disks mirrored in pairs and then striped
- Advantage over RAID 0+1 : if single disk fails, strip accessible



Source: Wikipedia

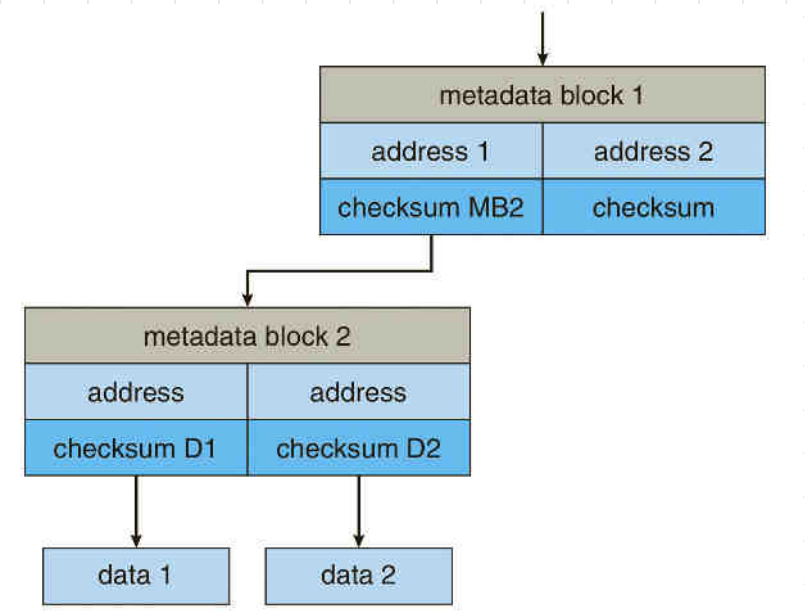


Useful Features

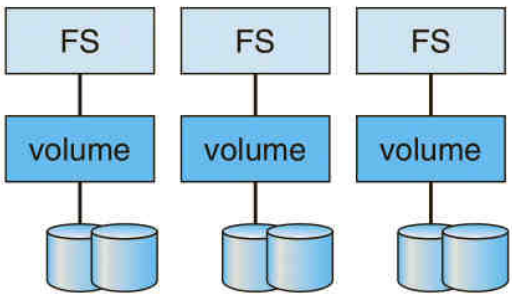
- **Snapshot:** view of file system before last update
- **Replication:** automatic duplication of writes for redundancy (synchronous and asynchronous replication)
- **Hot spare disk:** used as replacement in case of disk failure without waiting for disk to be replaced by human

Extensions

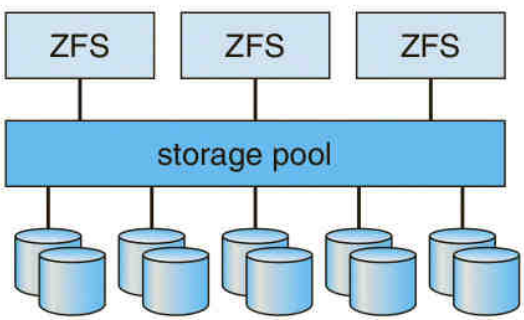
- **ZFS:** adds checksums of all data (data integrity) — Solaris
- Disk allocation in pools
- No volumes/partitions and volume management
- ZFS aggregates the devices into a storage pool



- LVM (Logical Volume Manage)



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.

FILE SYSTEM

File

- Named collection of related data stored on secondary storage
- Attributes: name, unique tag number / identifier, type, location (pointer), size, protection (rwx), time & date, user identification
  ↓ disk, track, sector

Operations

- Create
- Read
- Write
- Seek
- Delete
- Truncate
- Open — search & move contents to memory
- Close — move from memory to directory structure on disk

Open Files

- OS maintains open file table (cache) allowing for direct indexing
- File open count maintained for each file : no. of processes that have opened the file
  - open() increases count
  - close() decreases count

Locks

- Shared lock — reader lock ; several processes can acquire concurrently
- Exclusive lock — writer lock
- Mandatory lock — OS prevents other process from obtaining exclusive lock if a process already has acquired it
- Advisory lock — upto programmer to obtain and release locks

- From textbook:

For example, assume a process acquires an exclusive lock on the file system.log. If we attempt to open system.log from another process—for example, a text editor—the operating system will prevent access until the exclusive lock is released. This occurs even if the text editor is not written explicitly to acquire the lock. Alternatively, if the lock is advisory, then the operating system will not prevent the text editor from acquiring access to system.log. Rather, the text editor must be written so that it manually acquires the lock before accessing the file. In other words, if the locking scheme is mandatory, the operating system ensures locking integrity. For advisory locking, it is up to software developers to ensure that locks are appropriately acquired and released. As a general rule, Windows operating systems adopt mandatory locking, and UNIX systems employ advisory locks.

File Extensions

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Structure

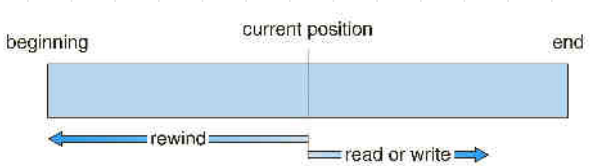
- Certain files must conform to a structure understood by the OS
- Executable files follow structure so OS knows where the first instruction is located and where in memory to load the file
- Some OSes have set of system-supported file structures
- Disadvantage of supporting multiple file structures: size of OS becomes too large

Internal File Structure

- UNIX defines all files to be a stream of bytes, each addressible by its offset from the beginning of the file

(1) Sequential Access File

- read_next() reads next portion of file and advances file pointer
- write_next() appends to the file and updates file pointer with the new eof
- Based on tape model of file; works well on sequential and random access devices



(2) Direct Access File

- random access
- databases
- Based on disk model of file; when query entered, block containing the record is directly fetched
- read(n) to read block number n directly
- write(n) to write into block number n directly
- Can use read_next() and write_next() with another function position_file(n) instead of read(n) and write(n)
- n = relative block number (from beginning of file) so that OS decides where to allocate memory for the file (allocation problem)

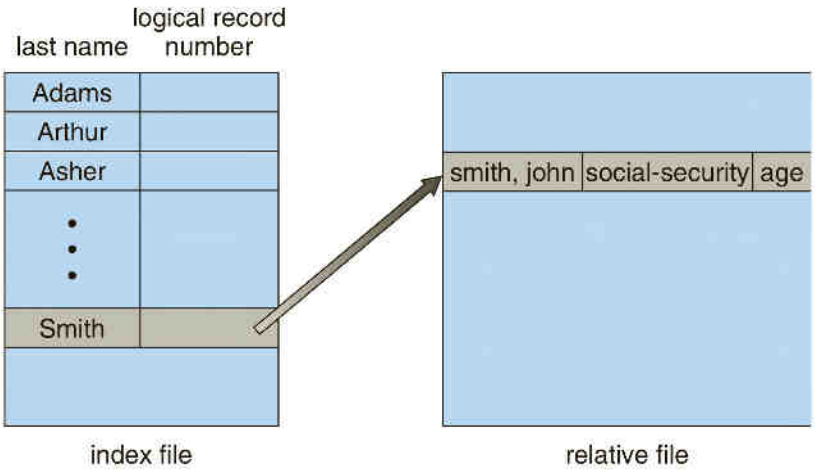
Simulation of Sequential Access on Direct Access

- Keep track of current position cp

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

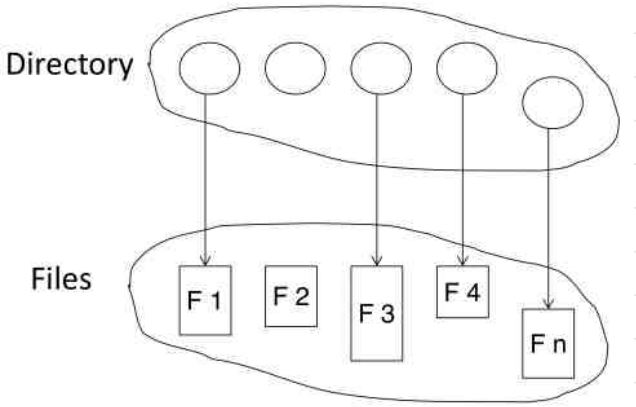
(3) Indexing

- Can be built on top of direct access
- creation of index in memory for file for fast determination of location of data
- If index file too large, index file for index file
- IBM's indexed sequential access method (ISAM)
  - small master index points to disk blocks
  - disk block read and secondary index located
  - secondary index used to find file block
- VMS OS — index & relative files

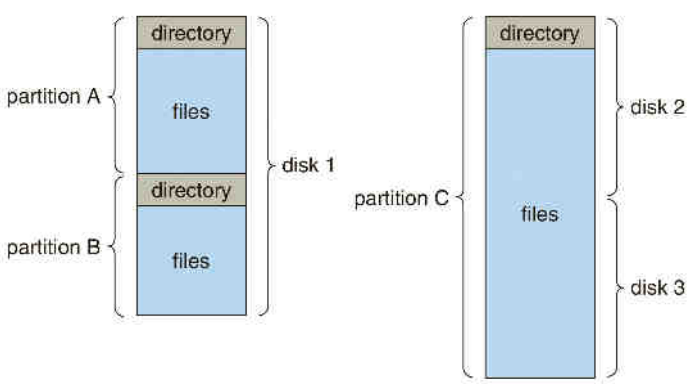


DIRECTORY STRUCTURE

- Collection of nodes containing information about files



- Disk can be entirely used for single file system, or it can be partitioned into volumes each of a particular file system
- Each volume with file system contains information about the files in the system in device directory or directory
- Directory stores information (name, size, location, type etc.) of all files on that volume



Solaris File Systems

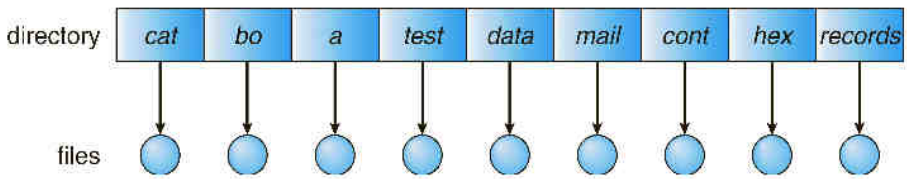
/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

DIRECTORY OPERATIONS

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

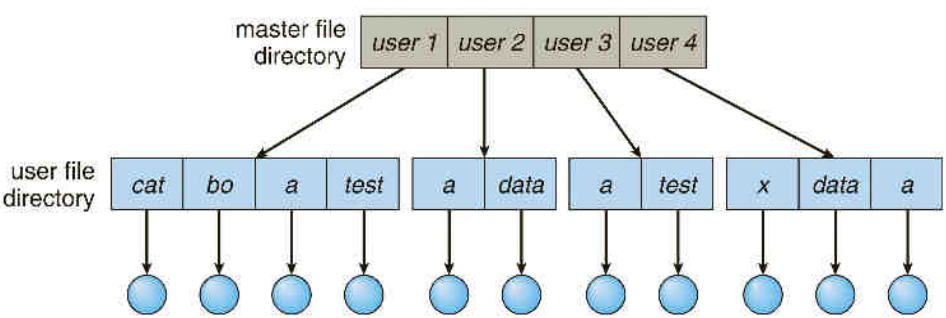
Single-Level DIRECTORY

- All files in single directory
- All unique names



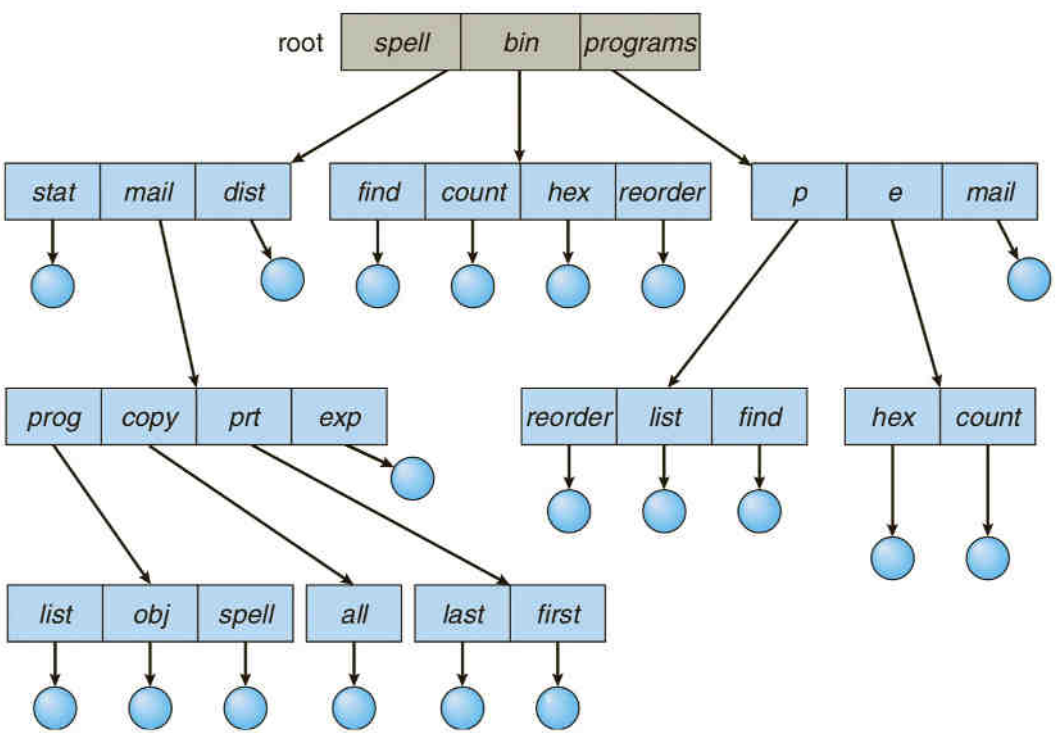
Two-Level DIRECTORY

- Separate directory for each user (UFD - user file directory)
- Users isolated
- User name & file name define path name
- Search path: sequence of directories searched when a file is named (Linux, MacOS: $PATH env variable — echo $PATH)



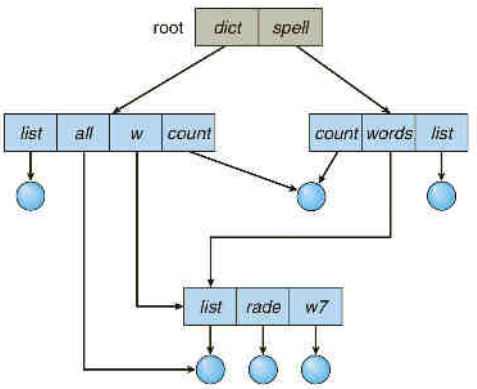
Tree-structured DIRECTORY

- Tree of arbitrary height
- Directories and subdirectories
- Each file has unique path
- One bit in each directory entry defines it as a file (0) or a subdirectory (1)
- Each process has current working directory
- When reference to file made, current working directory searched, then search path
- Change directory: change_directory()
- Absolute or relative path name



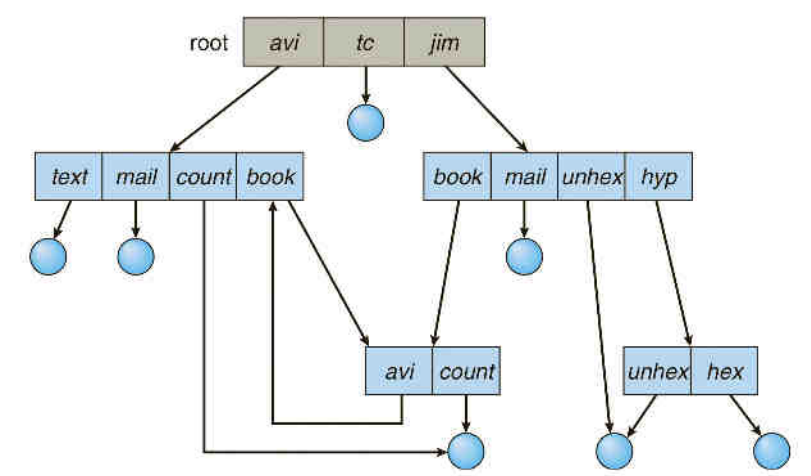
Acyclic Graph DIRECTORY

- Shared subdirectories and files among multiple users
- Links (pointers to files or directories)



General Graph DIRECTORY

- Cycles allowed
- Self-referencing cycles problem
- Garbage collection to clear dangling pointers; time-consuming
- Cycle detection algorithm for new entries



File System Mounting

- File system must be mounted before processes can access it
- OS given device name and mount point
- OS verifies device's file system (device driver to read files from directory and check format)
- OS notes in its directory structure that new device mounted at location (mount point)

File Sharing

- Multiple users, multiple file systems
- Resolving conflicts

MULTIPLE USERS

- Access control & protection between users & files
- File/directory owner/user and group
- Owner has all permissions and decides permissions for other users and groups
- When user requests access to a file, user ID compared with file owner attribute (also group ID)

REMOTE FILE SYSTEMS

- ftp — manually transferring files between machines
- Distributed file system — remote directories visible from local system
- Cloud computing
- WWW — anonymous file exchange with wrapper for ftp

Client-Server Model

- Server serves file to client that requested it
- Server serves to client based on IP address
- Encrypted client for security — unsecure simpler and more commonly used
- File operation requests via DFS protocol

- File open request sent with user ID of client
- Server checks credentials and checks if user has authority to access files
- NFS: UNIX standard

Distributed Information Systems

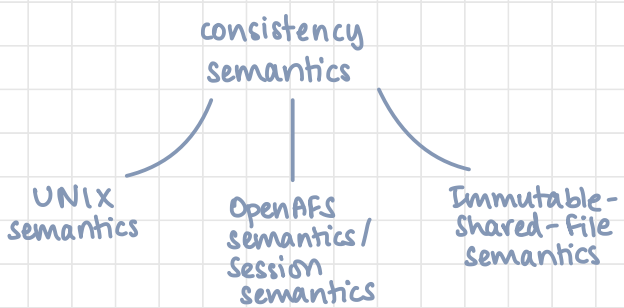
- Distributed Naming Services
- Hosts identify each other via domain name (DNS)
- Windows: CIFS
- DIS — NIS, LDAP, DNS Active Directory

Failure Modes

- Disk failure, network issues
- DFS protocols allow delaying of file operations on remote hosts instead of deleting commands
- To reduce loss, state information of client & server can be stored
- NFS4: NFS is stateful
- NFS3: information in request; less secure

CONSISTENCY SEMANTICS

- How multiple users access shared file simultaneously
- Process synchronisation algorithms
- File session: series of accesses between open() and close()



UNIX Semantics

- Writes to an open file by a user are immediately visible to other users who have opened the file
- One mode has a single shared current location pointer of the file; any advances made by one user are reflected in all users' files
- File: single physical image and contention causes delays

Andrew File System - OpenAFS Semantics

- Writes to open file not visible immediately to other users who have the file open
- Changes in a session only visible after starting a new session and closing the file
- Can be compared to Git — local changes take place independently and are only reflected after push (close) and pull (starting new session), but for a single file

Immutable-Shared-File Semantics

- All shared files are declared as read-only
- Filenames cannot be reused and file is immutable
- Simple implementation

File Protection

- Reliability: physically reliable; safe from damage
- Duplication of files (see - RAID)
- Protection; proper access by users
- Access types — limited to different extents for different users

TYPES of ACCESS

- Read
- Write
- Execute
- Append
- Delete
- List (file attributes)

Access Control

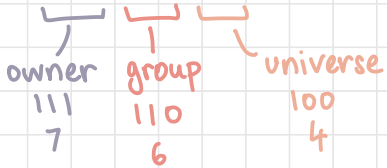
- User-dependent access
- Each file/directory has associated Access Control List (ACL)
- ACL specifies accessec for each user
- User job denied access if protection violation occurs
- List with every user tedious and not scalable

Classification of Users

1) Owner: user who created the file
2) Group: set of users sharing the file (work group)
3) Universe: all other users in the system

- Each file associated with group
- UNIX systems: 3 letters for each class

- Eg: rwx rw- r--



Current mode = 764

- Add group to file test.c

```
chgrp new_group test.c
```

- Change access to rwx for all

```
chmod 777 test.c
```

- Sample UNIX directory listing

```
-rwxrwxr-x  1 vibhamasti vibhamasti 16744 Mar  9 09:50 a.out
drwxrwxr-x 10 vibhamasti vibhamasti  4096 Mar 21 04:21 flow
-rw-rw-r--  1 vibhamasti vibhamasti    12 Mar  9 07:47 hello.txt
-rwxrwxr-x  1 vibhamasti vibhamasti 17240 Mar 11 09:49 scope
-rw-rw-r--  1 vibhamasti vibhamasti  1303 Mar 11 09:48 scope.c
-rw-rw-r--  1 vibhamasti vibhamasti   251 Mar  9 09:49 test.c
```

subdirectory — owner — group

```
-rw-r--r--   1 vibhamasti  staff  4597 Jul  9  2020 README.md
-rw-r--r--   1 vibhamasti  staff  9624 Jul  9  2020 analysis_options.yaml
drwxr-xr-x   8 vibhamasti  staff   256 Jul  9  2020 bin
-rw-r--r--   1 vibhamasti  staff   852 Jul  9  2020 dartdoc_options.yaml
drwxr-xr-x  16 vibhamasti  staff   512 Jul  9  2020 dev
drwxr-xr-x  13 vibhamasti  staff   416 Jul  9  2020 examples
-rw-r--r--   1 vibhamasti  staff  1731 Jul  9  2020 flutter_console.bat
drwxr-xr-x  13 vibhamasti  staff   416 Jul  9  2020 packages
```

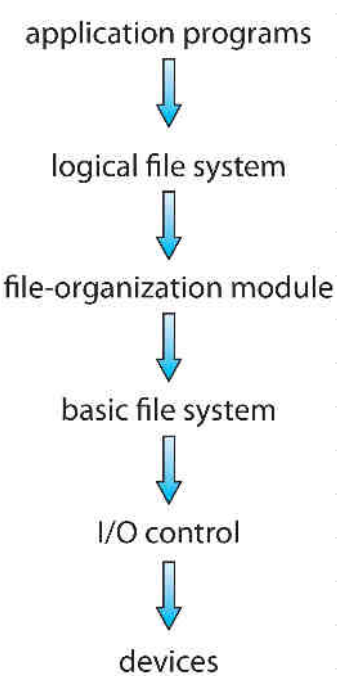
owner — group — subdirectory

Other Protection

- Password-protected files & directories

File System STRUCTURE

- Efficient access to secondary devices with easy retrieval
- Provides UI to disk storage
- Map logical file system to physical storage devices (algorithm)
- I/O control level: device drivers & interrupt handlers to transfer data between disk and main memory
- Device driver: translator from high-level instructions to low-level hardware-specific exams
- Basic file system: sends commands to device drivers to read/write to disk, manages memory buffers & caches
- File organisation module: translate file's logical block to physical block, free space manager to provide unallocated blocks to file organisation module
- Logical file system: manages metadata, directory structure, file control via File Control Blocks (FCBs) — contains info on ownership, permissions etc.



Implementation of File Systems

- Several on-disk & in-memory structures used to implement a file system

On-Disk Structures

1) Boot Control Block

- Information needed by system to boot OS from a volume
- One per volume
- Block is empty if volume/disk has no OS
- Usually first block of a volume
- UFS: boot block
- NTFS: partition boot sector
- More: Operating Systems Concepts, Boot Block, pg 480 (Galvin et al.)

2) Volume Control Block

- Contains info on volume/partition
- No. of blocks in volume, size of blocks in volume
- Free block count, free block pointers
- Free FCB count, free FCB pointers
- UFS: superblock
- NTFS: stored in master file table (relational DB, one row / file)

3) Directory Structure

- Orgnise files
- UFS: file names and associated inode numbers (FCB)
- NTFS: stored in master file table

4) File-Control Block

- Per file
- Unique identifier number
- UFS: inode number
- NTFS: stored as row in master file table

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

FCB

In-Memory Structures

- Data loaded at mount time
- Updated during file-system operations
- Discarded at dismount

1) Mount Table
   - Information about each mounted volume

2) Directory Structure Cache
   - Directory information on recently accessed directories
   - For directories where volumes are mounted, pointer to volume table

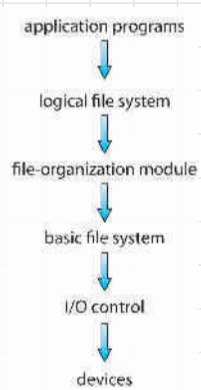
3) System-Wide Open File Table
   - Contains copy of FCB of each open file
   - Additional info for each file (no. of processes that have it open)

4) Per-Process Open File Table
   - Contains pointer to entry in System-Wide Open File Table
   - Additional info (pointer to current location in file, access mode)

5) Buffers
   - File system blocks being read from / written to disk

CREATE NEW FILE

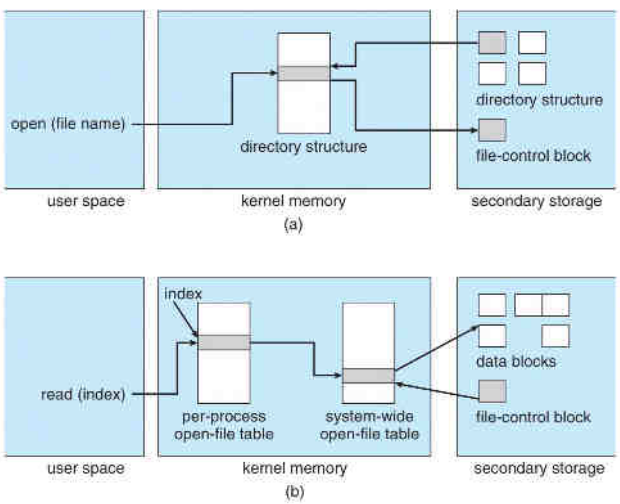
- Application program calls logical file system
- Depending on file system implementation, either a new FCB is allocated or an FCB is allocated from the set of free FCBs
- System reads directory into memory, updates it with new filename and FCB and writes it back to the disk
- Logical file system calls file organisation module to map directory I/O to disk-block numbers, which are passed onto basic file system and I/O control system
- Recall hierarcy:



OPEN FILE

- open() system call from application program passes filename to logical file system
- First searches system-wide open file table to see if file in use by another process
- If already in use, new entry created in per-process open file table, pointing to system-wide open file table entry

- If not already open, directory structure searched for given filename and once found, FCB copied onto system-wide open file table
- Entry made in per-process open file table with pointer to entry in system-wide open file table
- open() call returns appropriate pointer to entry in per-process open file table (UNIX: file descriptor, Windows: file handler)



In memory file system structures
(a) File open (b) File read

CLOSE FILE

- Per-process table entry removed
- System-wide count decremented
- If system-wide count is 0, updated metadata copied back to disk directory structure and system-wide table entry removed

PARTITIONS & MOUNTING

- Disk can be sliced into different partitions
- Each partition may be raw (no file system) or cooked (containing file system)
- Raw partitions used when file system not appropriate (eg: swap space in UNIX, databases, info needed by RAID systems etc); Operating Systems Concepts, Disk Formatting, pg 479 (Galvin et al.)

Boot Partition

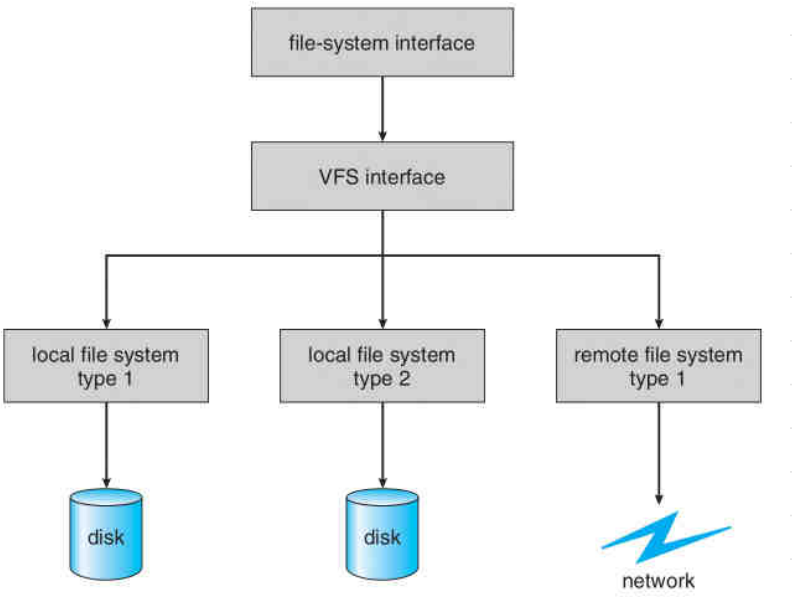
- Boot information can be stored in its own partition in its own format
- Boot info is sequence of blocks loaded as an image into memory
- Execution of image starts at predefined location
- Boot loader finds and loads kernel and starts executing (can understand multiple file systems and OSes)
- Read: Operating Systems Concepts, Boot Block, pg 480 (Galvin et al.)

Root Partition

- Contains OS kernel and other system files
- Mounted at boot time
- Other volumes either mounted automatically or manually based on OS
- OS verifies if device has valid file system
- OS updates in-memory mount table

virtual file system

- File system implementation consists of three major layers



- File System Interface: based on open(), read(), write() and close() calls and involving file descriptors

- Virtual File System Interface: two main functionalities

1. Separates file system generic operations from their implementation by defining a VFS interface; different VFS implementations can exist on the same machine to access different file system types locally

2. Provides mechanism for uniquely representing a file in a network (vnode: unique number network-wide)

- VFS distinguishes local & remote files

- VFS calls NFS protocol procedures for remote requests and file system specific operations on local requests

- Object types defined by Linux VFS
  (a) inode object: represents individual file
  (b) file object: represents open file
  (c) superblock object: represents an entire file system (volume)
  (d) dentry object: represents individual directory entry

- Each object type has a set of functions & each object has a pointer to a function table containing all functions of an object type (addresses of implementation)

- Eg: file object APIs:
  - `int open(...)` – open a file
  - `int close(...)` – close an opened file
  - `ssize_t read(...)` – read from file
  - `ssize_t write(...)` – write to file
  - `int mmap(...)` – memory-map a file

DIRECTORY IMPLEMENTATION

- Different algorithms & data structures affect performance

1) Linear List
   - Linear list of file names with pointers to data blocks
   - Time-consuming to scan directory before creating new file
   - New files added to end of list
   - Various deletion implementations
   - Linked list, Array

- Linear retrieval time
- Sorted: binary search is quicker but creation & deletion is complicated

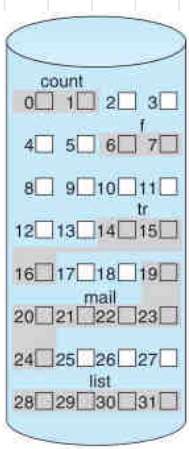
2) Hash Table

- Table maps filename to index in a list
- Constant retrieval time
- Must reorganise when hash table needs to be expanded
- Can instead use chaining for collisions and large no. of entries, but linked list will be quite slow

Disk Space Allocation

1) CONTIGUOUS ALLOCATION

- Each file occupies contigious blocks of a disk
- Minimum seek time as all blocks are contiguous
- Defined by disk address and length (blocks)
- Eg: file at block b of length n occupies b, b+1, ..., b+n-1 blocks
- Directory entry for each file indicates disk address & length
- Supports both sequential & direct access



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Shortcomings

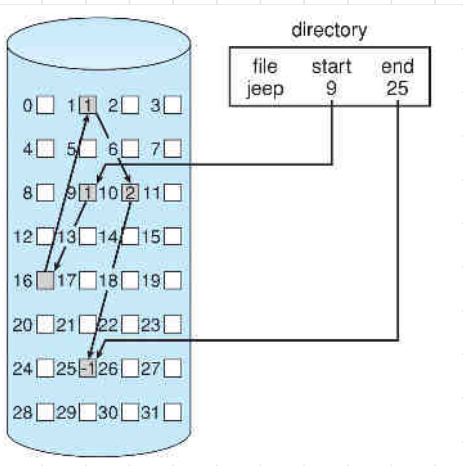
1. Difficulty in finding contiguous blocks for a new file (managed by management systems for free space)

2. External fragmentation — free space broken and scattered
   - One solution: copy entire file system to another disk, free original disk, copy files back by allocating contiguous space — extremely time-consuming (compacting)
   - compaction done when disk unmounted (off-time)

3. Determining space required for a file beforehand
   - Some systems use extents when a contiguous chunk is exhausted
   - Pointer to next extent and size of block recorded
   - Used by Veritas file system (high performance replacement for standard UNIX UFS)

2) LINKED ALLOCATION

- Each file linked list of disk blocks (located anywhere)
- Directory contains pointer to first & last blocks of file



- New file: new entry in directory
- Directory entry has pointer to start & end blocks, initialised to NULL
- Write causes free-space management system to find free block, perform write and linked to eof
- Read: read blocks by following pointers
- File size need not be defined

Shortcomings

1. Only sequential-access files
   - Each block requires disk read

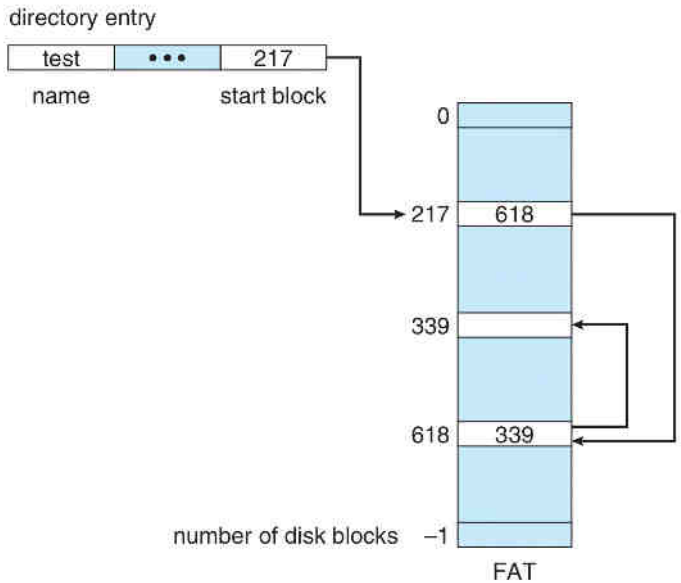
2. Space allocated to pointers increases file size
   - Solution: use clusters instead of blocks
   - Cluster: multiple blocks together, reducing pointer space
   - Increases internal fragmentation

3. Reliability low if a pointer in the middle is lost/damaged
   - Solution: doubly linked list

3) FILE ALLOCATION TABLE (FAT)

- Variation of linked allocation
- Used by MS-DOS
- Section of disk at beginning of each volume set aside for FAT
- Indexed by block number, one entry for each block
- Directory entry contains block number of first block of file
- Table entry of block number contains block number of next file

- Last block has special eof entry in table
- Unused block indicated by value 0 in table

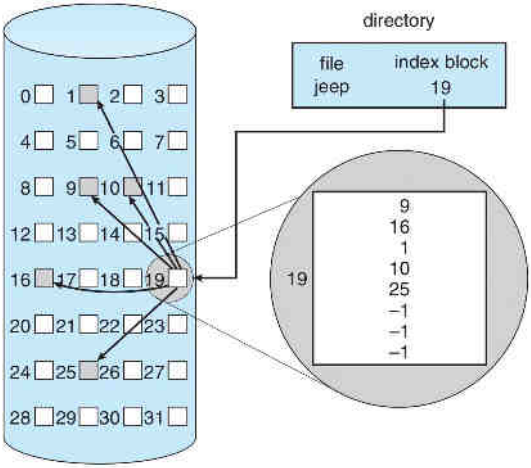


Shortcomings

1. Significant number of disk head seeks for FAT reading
   - Solution: cache FAT

4) INDEXED ALLOCATION

- Without FAT, direct access in linked allocation very inefficient
- All pointers together in one location: index block
- Each file has own index block (array of disk-block addresses)
- i^{th} entry of index block is i^{th} block
- Direct access without external fragmentation (any free block can satisfy request for more space)



- Pointer overhead of index block greater than linked allocation (entire index block needs to be allocated)
- Every file has index block; size of index block needs to be decided
- Index block size limits size of file.

Solutions

1. Linked Scheme

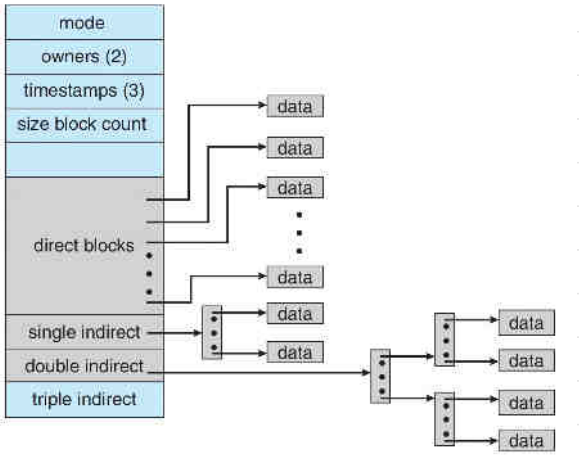
- index block : one disk block
- large files: several index blocks

2. Multilevel index

- first level index block to point to second level index block

3. Combined Scheme

- Used in UNIX File Systems
- First x pointers of index block stored in file inode
- First m (of the x) pointers point to direct blocks
- Next x−m pointers point to indirect blocks
- First is single indirect, then double indirect



Performance

- Most efficient method depends on access type (sequential or direct)
- Some systems: access method to be defined at file creation time and method chosen based on that
- Many layers of indexing can be quite slow