

## File System

Function: main secondary data storage; also permanent

Extreme speed bottleneck!

Capacity not a problem nowadays: 2 TB disks even for PC.  
But backup becoming a problem.

Logical view (view of programmer):

Have a tree structure of files together with read/write operation and creation of directories

Physical view:

Just a sequence of blocks, which can be read and written

OS has to map logical view to physical view

must impose tree structure and assign blocks for each file

## Caching

Disk blocks used for storing directories or recently used files cached in main memory

Blocks periodically written to disk

⇒ Big efficiency gain

Inconsistency arises when system crashes

Reason why computers must be shutdown properly

Two possibilities:

- **Linked list:** Each block contains pointer to next  
⇒ Problem: random access costly: have to go through whole file.
- **Indexed allocation:** Store pointers in one location: so-called index block. (cf. page table).  
To cope with vastly differing file sizes, may introduce indirect index blocks.

## Journaling File Systems

To minimise data loss at system crashes, ideas from databases are used:

- Define **Transaction points:** Points where cache is written to disk  
  
⇒ Have consistent state
- Keep log-file for each write-operation

Log enough information to unravel any changes done after latest transaction point

## RAID-Arrays

RAID: Redundant Array of Independent Disks

Main purpose: Increase reliability

- **Mirroring**: Store same data on different disks
- **Parity Schemes** Store data on  $n$  disks. Use disk  $n + 1$  to contain parity blocks

⇒ can recover from single disk failure

Disadvantage: Parity bit needs to be re-computed for each write operation

## Disk access

Disk access contains three parts:

- **Seek**: head moves to appropriate track
- **Latency**: correct block is under head
- **Transfer**: data transfer

Time necessary for Seek and Latency dwarfs transfer time

⇒ Distribution of data and scheduling algorithms have vital impact on performance

## Disk scheduling algorithms

Standard algorithms apply, adapted to the special situation:

1.) **FCFS**: easiest to implement, but: may require lots of head movements

2.) **Shortest Seek Time First**: Select job with minimal head movement

Problems:

- may cause starvation
- Tracks in the middle of disk preferred

Algorithm does not minimise number of head movements

3.) **SCAN-scheduling**: Head continuously scans the disk from end to end (lift strategy)

⇒ solves the fairness and starvation problem of SSTF

Improvement: **LOOK-scheduling**:

head only moved as far as last request (lift strategy).

Particular tasks may require different disk access algorithms

Example : Swap space management

Speed absolutely crucial ⇒ different treatment:

- Swap space stored on **separate partition**
- **Indirect access methods not used**
- **Special algorithms used for access of blocks**  
Optimised for speed at the cost of space (eg increased internal fragmentation)

## Linux Implementation

Interoperability with Windows and Mac requires support of different file systems (eg vfat)

⇒ Linux implements common interface for all filesystems

Common interface called **virtual file system**

virtual file system maintains

- inodes for files and directories
- caches, in particular for directories
- superblocks for file systems

All system calls (eg open, read, write and close) first go to virtual file system

If necessary, virtual file system selects appropriate operation from real file system

## Disk Scheduler

Kernel makes it possible to have different schedulers for different file systems

Default scheduler (Completely Fair Queuing) based on lift strategy  
have in addition separate queue for disk requests for each process  
queues served in Round-Robin fashion

Have in addition No-op scheduler: implements FIFO

Suitable for SSD's where access time for all sectors is equal