File System

Function: main secondary data storage; also permanent

Extreme speed bottleneck!

Capacity not a problem nowadays: 20 GB 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

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.

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 (*e.g.*,increased internal fragmentation)

Standard interface important for plethora of device types
Have a few basic operations:

- open
- read
- write
- close

Example: UNIX devices listed in /dev
different types implement different operations:
sequential vs. random access
character-stream vs. block device

Input/Output

Start with Logical View:
Devices can be classified according to possible operations:

- Character devices: transfer bytes one by one
- Block devices: transfer blocks of bytes as units
- Memory mapped devices: OS interpretes memory access as access to device
- Network devices: Receive packets over the network

Have also different system calls:
blocking vs. non-blocking (system call returns after completion / immediately)

Physical View

Interaction between device and CPU:

- Polling: works for fast operations (eg graphics)
- Interrupts: standard way, priorities important
- Direct Memory access: implements the memory mapping without burdening the CPU

OS support for I/O:

- Buffers: need intermediate storage during transfer
- Caches: fast memory
- Support for spool files

I/O can be major performance bottleneck
Reason: enormous number of context and state switches
Ways out:

- Hardware support: DMA (Direct Memory Access) chips
- OS support: re-implementing telnet daemon using in-kernel threads (Solaris)
- Buffers: cope with speed mismatch (modem, hard disk), different data-transfer size (networks)
- Caching: keep disk blocks in free main memory