File system allocation and implementation – disk allocation and stuff.

File organizations:

File System Monitoring:

* File system must be mounted before it can be accessed.
* Unmounted file system is mounted at mount-point directory.

Access Lists:

* Read, write, and execute
* 3 Classes of users on unix/linux:
  + Owner (can read, write, execute) – RWX (111) – 7 in decimal
  + Group (can read, write) – RWX (110) – 6 in decimal
  + Public – (can execute) – RWX (001) – 1 in decimal
* Define access to a file: *chmod 761 nameoffile*
* Attach group to a file (let’s say group ‘G’): *chgrp G nameoffile*

EXAMPLE: we are to assign the “students” group into the file “assignment.txt”, owners can read, write, and execute, group can read and execute, and public can read only.

The commands are:

* *chgrp students assignment.txt* (assign students to file)
* *chmod 751 (or 754???) assignment.txt (assign permissions)*

File Types Based on Access Methods:

* Sequential access methods – read next 🡪 write next 🡪 reset
* Direct access files: fixed length logical records – read n 🡪 write n 🡪 position to n 🡪 read next 🡪 write next 🡪 rewrite n

File System Layers:

* Device drivers layer manage I/O devices at the I/O control layer.
  + Gives commands like which drive to read and which memory location to; outputs hardware commands to hardware controller.
* Basic file system layer given commands like which block to retrieve memory from. Also manages buffer and cache.
* File organizational module layer: maps logical address to physical address. Also manages freespace and disk allocation.
* Logical file system layer: manages metadata information.
  + Translates file name into file number, file handle, location by maintaining FCB (file control blocks, or inodes in UNIX)
  + Manages directory
  + Protection

File Systems:

* File Systems, sometimes many within an OS.
* Different OS’s have different formats: Unix has UFS; Windows has FAT, NTFS, FAT32, etc…

Google File System:

* Used to manage big data
* Files divided into 64MB chunks
* Files are usually appended to or read
* Master data server doesn’t store the ‘chunks’, but the metadata.

UNIX UFS:

* No hierarchy
* Files stored in disk
* Disk is arranged as logical collection of disk blocks
* Our trees, are flattened onto these blocks
* Unix file systems has to 2 components:
  + Inodes (stores the info about file allocation)
  + Data blocks (stores file contents)

The Real View:

* 3 areas on the ODS (on disk structure)
  + The superblock
  + The ilist (inode table)
  + Data area
* Data area itself comprised of used and unused blocks
  + Unused blocks are known as free list.
* Superblocks store info about the file system.
  + Size of each area, unused data blocks, etc…

What’s an inode?

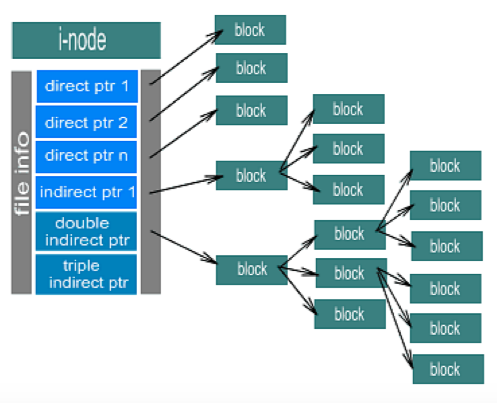
* Index node… it’s a struct that each file has.
* Stores info about a file (size, who the owner is, etc.)
* Records where file contents are stored in the disk. Which blocks in data area.
* ID’d by position in inode table… ‘inode 0’ is first in table, ‘inode 1’ is second.

How all this shit works:

* The command 🡪 *who > userlist*
* This executes the user command “who” and stores the output in the file “userlist” (which means we need to create a new file)
* The kernel creates the “userlist” file by:
  + Finding free inode & stores file properties in it
  + Find enough free data blocks to store file contents
  + Record block indices in the inode
  + Stores data in the blocks.
  + Add “userlist” to current directory

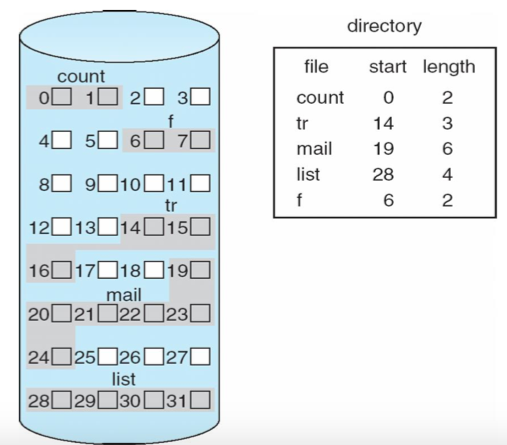
Big Files:

* Sinces inodes are fixed-size structs, how do we store big files?
* Each inode has 13 data block addresses
* Inode 11 contains the address of a block that stores more addresses (indirect block)
* Inode 12 contains address of a block that contains address of a block that stores more addresses (double indirect block)
* Inode 13 is a triple indirect block.

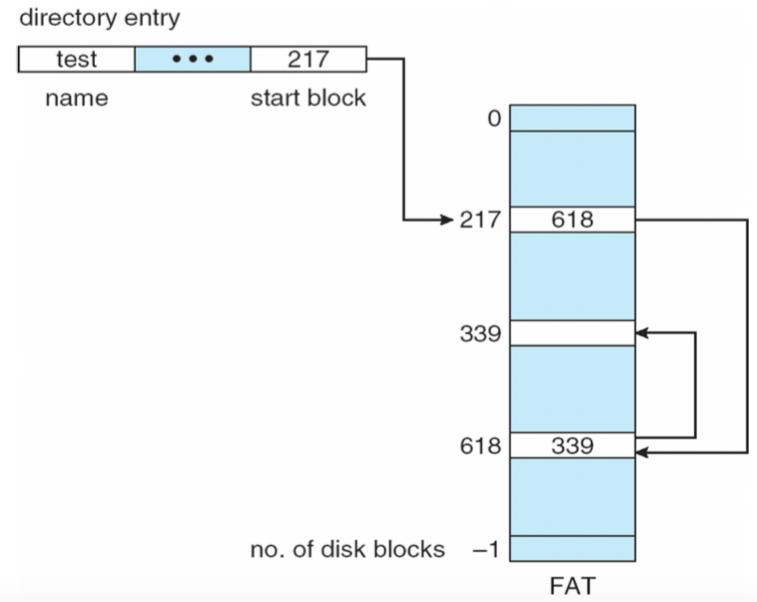


Disk allocation methods:

* Contiguous allocation – each file occupies set of contiguous blocks
  + Best performances in most cases.
  + Simple – only starting location (block number) and length (number of blocks) required.
  + Problems with it: Need for compaction offline (downtime) or online, external fragmentation
  + Mapping from logical to physical
  + Assume block size is 512
  + Logical address (Quotient)/512 (Remainder)
  + Block to be accessed = Q + starting address
  + Displacement into block = R



* Linked allocation
  + Each file a linked list of blocks
  + File ends at nil pointer
  + No external fragmentation
  + Each block contains pointer to next block
  + No external fragmentation or compaction
  + Free space management system called when new block needed
  + Improves efficiency by clustering blocks into groups but increases internal fragmentation.
  + Problems: not reliable (if one link is broken), locating a block takes many I/O’s and disk seeks.



do the rest later.