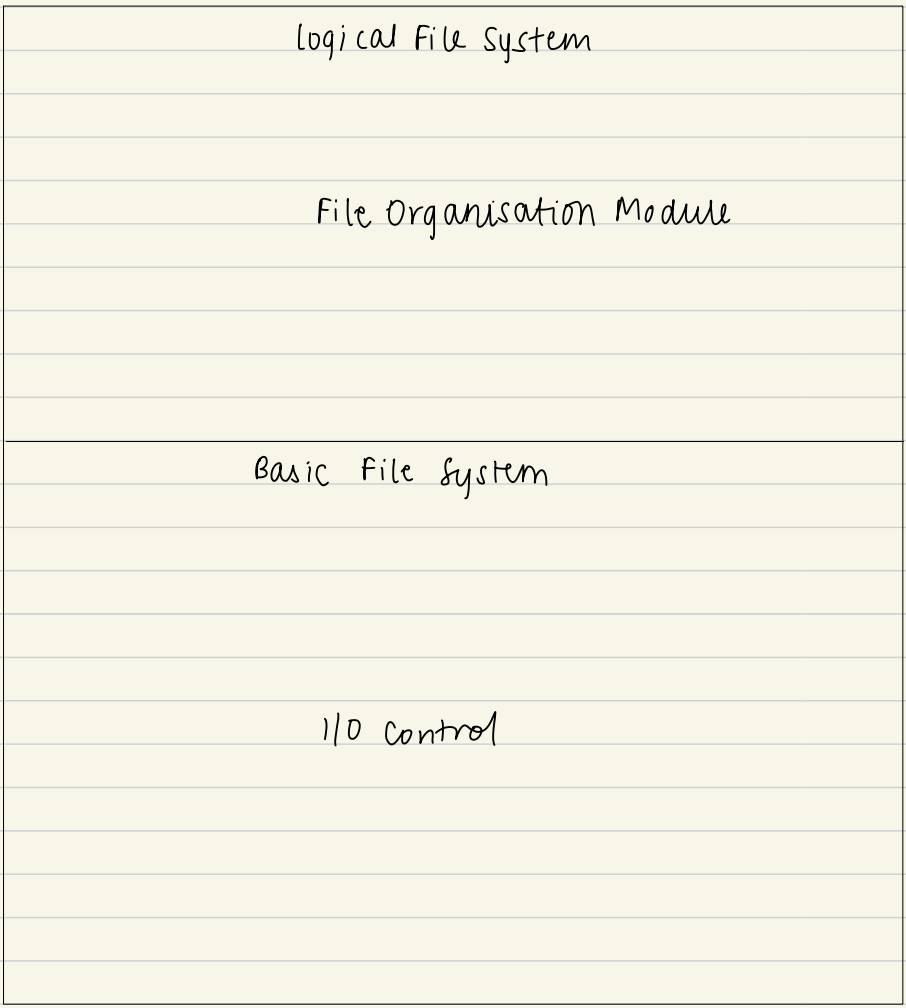
File System Structure

- resource abstractions typically associated with secondary storage
- long existing, shareable, organised heirarchically
- file system :



File Attributes

1. Name
2. Type
3. Location
4. Size
5. Protection
6. Time, Date
7. User Identification

} meta data of a file

↑ Stored in directory structure (↑ special type of file) maintained on disk (↑ non volatile memory)

File Types

- file has a contiguous address space which can store many different types of info

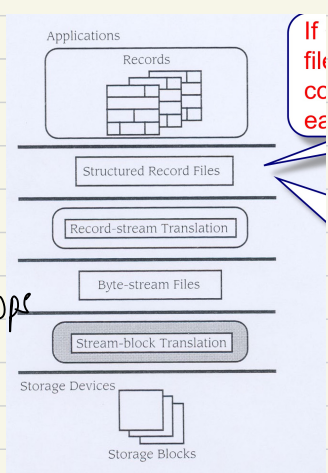
File Type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	complied, machine language, not linked
Source code	c, p, pas, 177, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data documents
Word processor	wp, tex, rrf, etc.	various word-processor formats
Library	lib, a	libraries of routines
Print or view	ps, dvi, gif	ASCII or binary file
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed.

- extension also communicates the type of operations one can perform + applications that support it

→ sequence of bytes

File Structure

- logical structuring of records as determined by the way they are accessed by application programs
- file is a set of records
  - ↘ can be stored in a certain way to facilitate file access
- storing a file physically on a secondary storage device needs
  1. record stream translation : map logical file struct to contiguous logical address space → sequence of bytes
  2. stream-block translation : divide the byte sequence into blocks
- retrieving file is the same but in opposite order
- each structure needs its own translation code in the OS ∴ many struct is cumbersome
  - ↑ ∴ the translation part is left to the application programs
- ∴ OS assumes all files are sequences of bytes and translation is left to app
  - ↑ UNIX, early windows



File Access Methods

- Sequential Access
- Direct Access : referenced using byte number
  - but still need to load entire block into the memory

File Operations

Commands	Explanation
Create	allocate disk space; create directory entry with file attributes.
Delete	delete the corresponding directory entry; deallocate disk space.
Open	search the directory structure for file entry; move the content to memory (put in the *open file table*). Information Associated with an open file: - *Current Position Pointer* - *File Open Count* - *Disk Location*
Close	move the content of directory entry in memory to directory structure on disk.
Write	search *open file table* to find the location of file; write data to the position pointed by *Current Position Pointer*.
Read	search *open file table* to find the location of file; read data from the position pointed by *Current Position Pointer*.

kernel data structure

how many opens

if 0 delete from OFT

file open count --;

step 1 → space allocate
step 2 → new entry in directory

step 1 → search directory, delete
step 2 → release all

→ before read/write
→ basically moving file info from directory (hard disk) to main memory.
→ ∴ file info comes to memory and is easier to access

lec 1: 43

Directory structure

- multiple entries → one per file
- directory and files all reside on disk

- 2 ways of storing

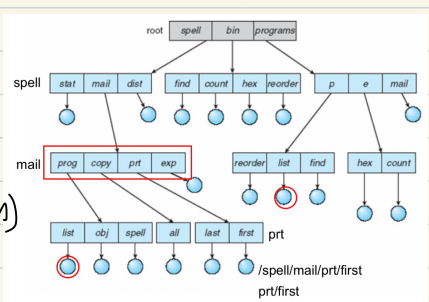
(1) each entry has the name and other metadata
(2) each entry has name + pointer to ds that stores metadata.
↑ used by UNIX
↑ file control block

- implemented using

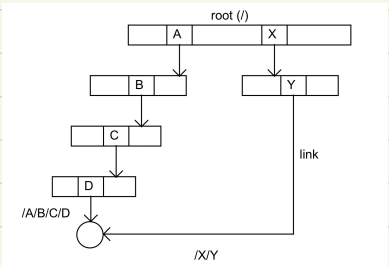
(1) linear list
- simple
- time-consuming to search

(2) Hash Table
- list w/ hash table to facilitate file look-up
- decreases search time
- collisions may occur

(3) Tree
- file is identified by path name
- also relative path based on working directory
- can have same name but diff path (under diff directories)
- easy search, easy group
- allows for file sharing using links : tree ⟶ acyclic graph



④ Acyclic Graph directory



- links allow sharing
  ↑
  only 1 physical file
- 2 ways of creating links

ⓐ Symbolic Link

- creates a redirect, directory file entry contains path
- to actual file.
- so the path is used to access the file
- time-consuming

ⓑ Hard Link

- duplicate
- new directory entry which contains all the info stored in the original entry
- faster
- ∵ duplicate → any time file attributes are modified the link entry must also be modified.

- Problems w/ sharing
→ traversing file system may result in visiting shared files more than once : soln → ignore link while traversing
→ deleting shared file may leave link dangling
soln: search for all links and remove
doesn't always work → leave them, delete when used
preserve file, delete only when all are deleted.

Directory Operations

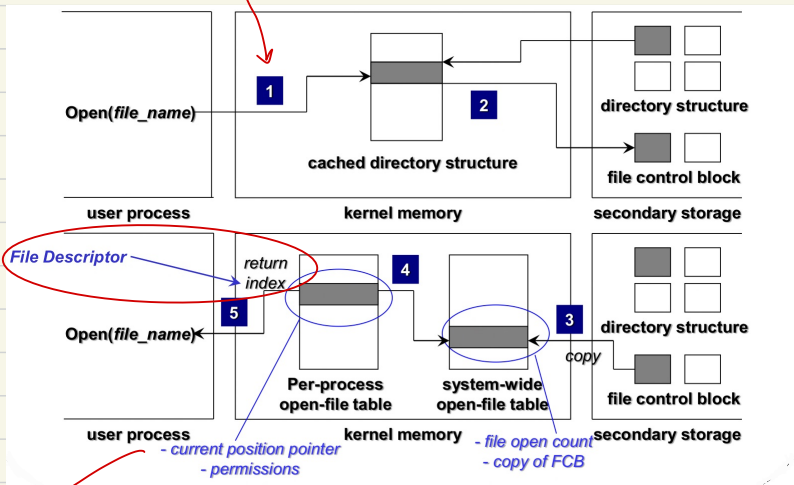
Commands	*Explanation*
Create	create a directory. In UNIX, two entries "." and ".." are automatically added when a directory is created. "." refers to the *current directory*; and ".." refers to its *parent*.
Delete	delete a directory. Only empty directory can be deleted (directory containing only "." and ".." is considered empty).
List	list all files (directories) and their contents of the directory entry in a directory
Search	search directory structure to find the entry for a particular file.
Traverse	access every directory and every file within a directory structure.

In-Memory File System Data Structure

- open() → to avoid constant search of directory
- given name + cached directory structure = OS can find FCB on hard disk

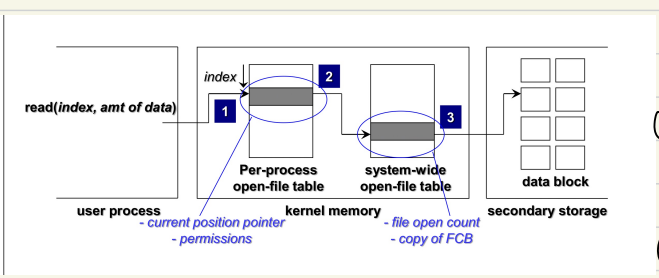
- if multiple processes want to open the same file simultaneously → its a problem
  - ↓
  - soln : 2 levels of open files

kernel instruct^n



where file is currently being accessed.

① open (file) system call
② cache directory entry
③ access FCB
④ copy FCB to system-wide open-file table
⑤ per-process open-file table contains current position pointer + permissions
⑥ index of entry in ppof is returned to user and application program.



① index gives current position pointer
② CPP + amt of data indicates which block of file to access
③ using FCB from swof the exact location of blocks on hard disk is found.

+ file allocation method

File Protection

- 3 actions : read, write, execute
- 3 groups :

3 protection bits

		R	W	X
owner access	7 :	1	1	1
group access	5 :	1	0	1
public access	1 :	0	0	1

File type	User access	Group access	Public access	Links	Userid	Size	Date	Time	File name
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
-	r w x	r - x	- - x	1	smith	58	Mar 3	3:04	game1

Directory Protection

- access directory means execute it.
- read permission → list the directory
- write permission → create or delete files from a directory

File type	User access	Group access	Public access	Links	Userid	Size	Date	Time	File name
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
d	r w x	r - x	- - -	0	smith	5	Feb 2	12:01	games

Allocation Methods

① Contiguous Allocation

- each file occupies a set of contiguous blocks on the disk
- from 'b' to 'b + n - 1' n = blocks in file
- supports random access → cause if you know to go to i^{th} block in file : 'b+i' is the place to be

PROBLEMS

int frag
- waste of space (as files are created and deleted, hard disk becomes fragmented)

ext frag
- find big enough hole : first fit, best fit ← external fragmentation
- size of file is limited by holes below the file

→ x —

- address mapping : block size : K
  LA : Q × K + R ← offset
  block to be acc → Q + starting address

$$\frac{LA}{K} = Q + R$$

Q ↑ quotient, R ↖ remainder

eg : LA = 2333
K : 512
starting = 19

LA = Q × 512 + R = 2333
Q ↑ 4, R ↑ 285

PA = 285th byte of 23rd block
23 ↑ 4 + 19

(2) Linked Allocation

- each file is a linked list of blocks that could be scattered anywhere on the disk

- no issue of external fragmentation
- no random access ← inefficient. first
- some waste of space ← pointer takes up 4 bytes.
- no constraint on size

- LA = Q x (size − 4) + R
  - Q: block no
  - R: offset

byte at : block Q
byte R+4

(3) Indexed Allocation

- every file has an index block which contains all pointers to the allocated blocks arranged in Logical block order
- directory entry has index block number

- big file ⇒ multiple index blocks
- supports random access
- dynamic storage w/o external fragmentation
- excess space for index block
- overhead of updating + mapping index block

- Logical to physical

file max size : 128 K bytes
block size : 512 bytes
∴ 2 blocks for index table (4 bytes for pointer)

$2^{17} / 2^{9} \rightarrow 2^{8}$ blocks

x4

2^{10} bytes ← index table size

$\lceil 2^{10} / 2^{9} \rceil = 2$

if LA = 2333
Q = 4
R = 285
PA = 285th byte of 4th block

directory : filename + inode number

- UNIX IMPLEMENTATION : inode (index node)

→ fixed location in disk

/usr/ast/mbox
root

a) file attributes
b) 12 ptrs to direct blocks
c) 3 ptrs to indirect blocks
   - single → index blk → blocks
   - double → index → index → blocks
   - triple → index → index → index → blocks

what file size will it support?
ptr : 4 bytes = 2^2 bytes
block : 4k bytes = 2^{12} bytes
direct : 4k x 12 = 48k bytes
direct + single indirect = 48k + 2^{22} bytes
double indirect = $2^{10} \times 2^{22} = 2^{32}$
triple indirect = $2^{10} \times 2^{10} \times 2^{10} \times 2^{12} = 2^{42}$

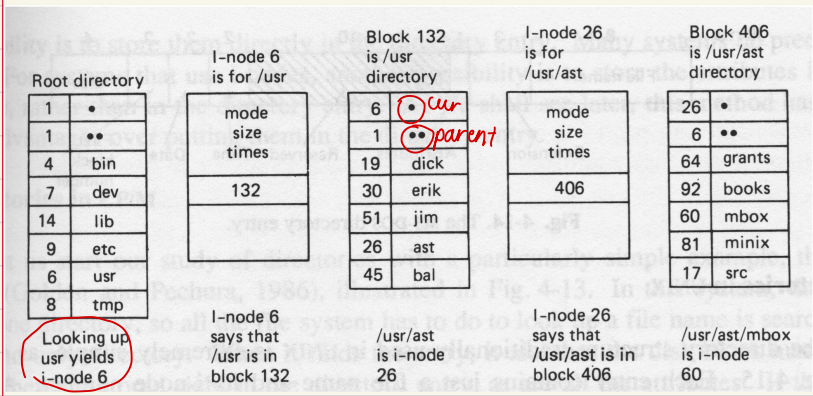
2^{12} block

$2^{12} / 2^2 = 2^{10}$ entries

$2^{10} \times 2^{12} = 2^{22}$

entry entry x block

file look-up in UNIX



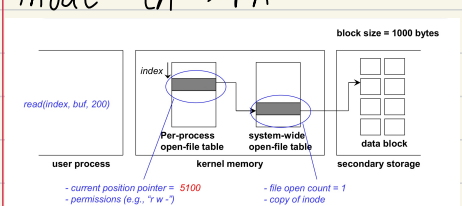
look up /urs/ast/mbox.

① currently only root is in the cache
② find i-node6, bring to mem
③ pts to block 132, bring it to mem
④ find ast : inode 26, bring it to mem
⑤ pts to block 406, bring to mem
⑥ find mbox : inode 60, bring it to mem

4 disk I/O to find mbox

5 to open mbox

inode LA → PA



- read 200 bytes from 5100
- logic block : $\frac{5100}{1000} = 5$
- same as

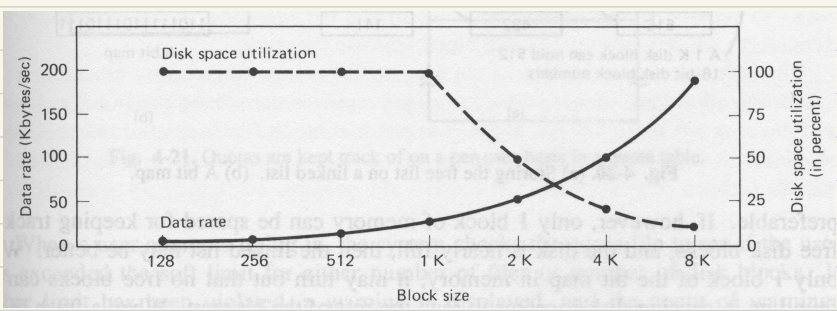
Disk Space Management

- determining block size
- block size affects both data rate and disk space utilisation

a) big block size : file fits into fewer blocks
- fast to fetch & transfer blocks
- space can be wasted if file doesn't occupy the entire last block (int-frag)

b) small block size : file may consist of many blocks
- slows data rate

- ? time space trade off issue.



- how to keep track of the empty blocks

① Bit map / vector
- each block is represented by 1 bit (0: free, 1: allocated)
- bit map size = $\frac{\text{disk size}}{\text{block size} \times 8}$ → no of blocks; ← bits to byte

Kernel updates it.
- kept in disk, bring to memory.

② Linked list
- Kernel maintains a list of all free blocks in order.
- so keeps track of only the first counter
- no extra space
- but not efficient

for contiguous memory allocation to find large enough hole kernel must traverse linked list

- deleting file will need the block to be inserted in the right place which will also need traversal.