Exam:

- Dec 13th Thursday BRKI 165 3:30 PM

(90-120) minutes exam (you can spend up to 180)

- 3 two sided hand written cheat sheet allowed
- Comprehensive w/ emphasize on after exam 2 and its labs.

Review:

- File System Interface:
  - file attributes
  - file types
  - Access methods: 1) Sequential
    2) Direct Access (Random)
    3) Indexing
  - Directory Structure (in super block)
  - Disk Structure (. . .) - Partitions: break down disk space
    - Raw disk
    - volumes: entity containing FS
  - Mapping between name ⟷ inode
    - search
    - create
    - delete
    - list
    - Rename
    - Traverse

* Directory structure:

1) Single-level

2) Two-level

3) Tree-structured : absolute vs relative pathname

4) Acyclic Graph directories

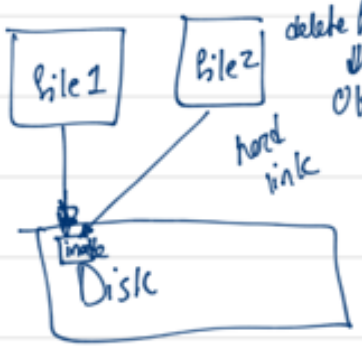
5) General Graph

Links: A) Hard link: within FS, not to directories (Aliases) ← for directories.
another file with link to the same file content

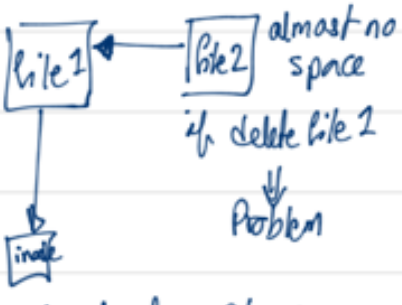
```
ln file1 hlink1      ls -i
```

B) Symbolic link (soft link): to directories, to files on different file systems. (on diff partitions). disadv: if target deleted ⇒ unusable.

link to another name in FS



Symbolic — pointing to another file



keep track of soft links

* Protection (as a requirement of sharing) — top secret

Mandatory Access Control (MAC): public, restricted, confidential, classified

files & processes ⇒ Process read and write to

OS limits accesses from subjects to Objects (policy)
- at each level ⇒ Process read/write to same level

One policy

- read : from files below
- write : to files above

- Discretionary Access Control (DAC)
  owner is able to assign permissions.
  ↳ hard to set for all files & all users
- Role-Based Access Control (RBAC)
  Define a set of roles
- Access lists and Groups:

owner group public

--- --- ---

RWX

3 classes of users 1) owner ---

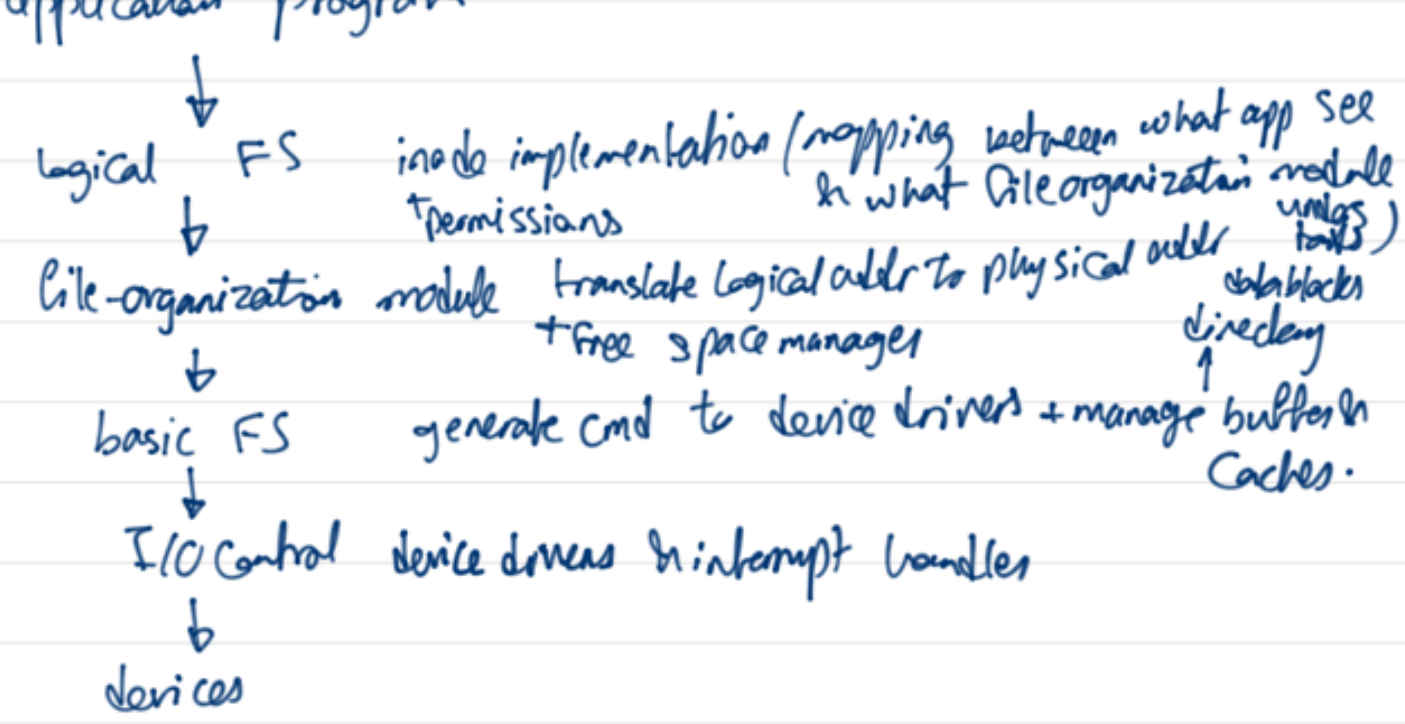
2) group

3) public

File System Implementation:

1) User's Convenience (interface design)

2) Efficient mapping of logical file system to physical disk.



- system-wide open table: Contains inode data.
- per-process " " : pointer to system-wide open-file table.
- Directory structure: name, inode # mapping

open

1) Directory Implementation: (name ↔ inode #)

1) Linear list O(log n) – O(n)

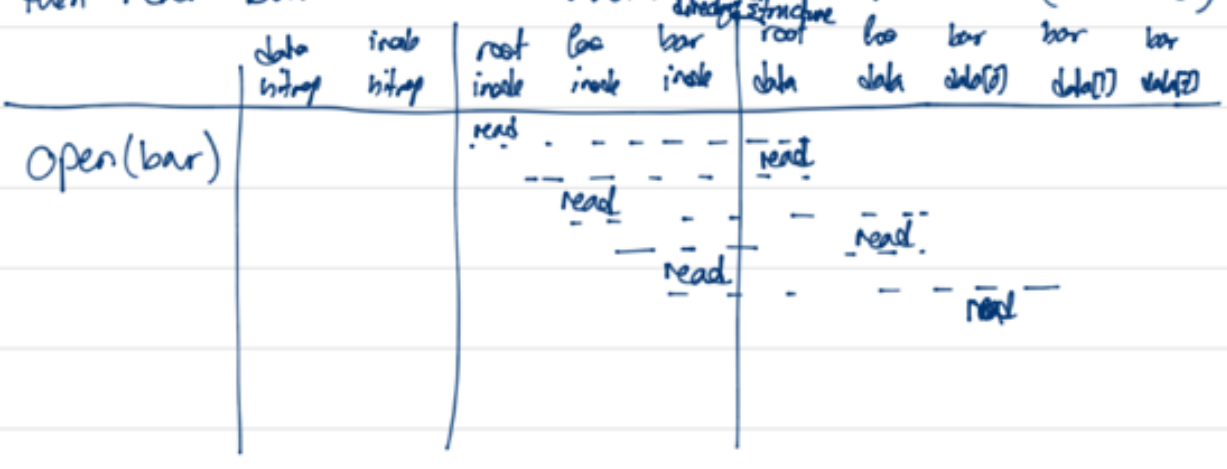
2) Hash table O(1)

Why important?

- we want to open a file /foo/bar read it & the close it

open("/foo/bar", O_RDONLY)

each directory has an inode itself, usually the root directory has a well-known inode # (2 in Linux) ⟹ after accessing inode 2 we have to load the directory of foo to read its inode val then read bar file inode from data within foo (data block)



⟹ Linear vs Hashed ⟹ Faster access

2) Allocation Methods

2-A) Contiguous

2-B) Linked (FAT)

2-C) Indexed : a) Linked Indexes

b) Multilevel Index

C) Combined (Unix Inode)

64 bit file pointers (8 bytes)

System w/ 4KB blocks →

only direct ⇒ 4X12=48KB filesize

- 12 pointers: direct data block pointers
- 1 single indirect
- 1 double "
- 1 triple "

Single indirect + direct: $\frac{4KB}{8} = 512$ addr per block

direct

$512 \times 4KB = 2MB$ file $+ 48KB$

double indirect: 512x512 addr

$512 \times 512 \times 4KB = 2^{10} \times 2^{10} \times 2^{10} B = 1GB$

triple indirect: $512 \times 512 \times 512 \times 4KB = 2^{10} \times 2^{10} \times 2^{10} \times 512B = 512GB$

3) Free-space Management:

3-A) bit vector free: 1
allocated: 0

3-B) Linked list

3-C) Grouping

3-D) Counting