5/15/24 Crash Consistency

→ FS Operations

→ create, read, write, rename ...

modify multiple filesys metadata & disk blocks (bitmap, inodes)

① allocate a new inode
(inode bitmap write)
② update new file's inode
(file inode write)
③ add new dirent to parent dir
(parent data write)
④ update parent dir file size
(parent inode write)

① allocate data blocks
(data bitmap write)
② update file inode w/ size & new data layout
(file inode write)
③ write new data
(file data block write)

→ Filesys Persistence Model

→ fs use cache to speed up performance : inode cache, block cache

→ fs operate on cached blocks → when cached blocks change, when should we write them back?

→ write through cache

↳ persist changes immediately, every op persists right away

↳ problem: slow! lots of disk writes, frequently modified blocks keep getting written to disk

→ let user decide

↳ fs ops are not persistent by default (on cached blocks)

↳ user request for persistence via

① sync: flush all cached blocks to disk

② fsync: persist changes associated w/ the file

→ doesn't flush dependencies (parent dir)

* Kernel periodically flush cached blocks (10 - 30s)

Crash Consistency

- → fs op generates multiple disk writes
  - → file write : inode, data bitmap, data block ✲ may crash after any write
  - → concurrent disk requests can be reordered
- → crash before all requests completed ... → interrupted by disk I/O completion
  - → no writes made to disk? fine! nothing changes
  - → only data bitmap made it?
    - → bitmap thinks a block is used when it's not!
    - → leaks blocks ∴ inconsistent filesys metadata
  - → only file inode is written?
    - → inode thinks the data block is allocated
    - → while bitmap can allocate it to someone else ✲ leak data content

      inconsistent fs metadata ∴
  - → only data block is written?
    - → fs metadata is consistent, all good
    - → user lost data (but also haven't heard back from fsync!)

What to do ?

→ Resolve Inconsistency : filesys checker (fsck)

→ start from superblock, check its validity & walk through the fs structures on disk

→ inconsistent bitmap & inodes

→ blocks allocated by bitmap but not tracked by any inode

→ resolve by marking them free (why is this safe?)

→ blocks free in bitmap but tracked by an inode

→ resolve by marking them allocated

→ check for consistent namespace

→ scan through directory entries (starting from root) to see if all allocated inodes are referenced by dirent

→ move valid yet not referenced inodes to lost+found folder

→ Avoid Inconsistency : Journaling / Logging

→ mismatching granularity of atomicity

→ disk supports single block atomic update

→ fs wants single atomic operation (multi-block updates)

→ transaction : abstraction that groups arbitrary # of updates into an atomic unit

write to log

```
tx_begin
tx_write(b1)
tx_write(b2)
tx_write(b3)
tx_commit
```

Logging : reserves log space on disk, write txn into log first, after log is persisted, apply changed blocks to their actual location

* changes are written twice, once to log & once to their actual locations.

* apply committed txns upon recovery