Lecture 19: recovery

- fsck
- Journaling
- Log-structured FS
- Quiz (really!)
Two data structures:
- inodes & files
- free list

Invariants:
- every block is part of a file (no orphan) block with no references to it: orphan
- no block is both free & in a file (leads to same block being in two files: stomp on each other)

Can lose power/shut down unexpectedly at any time: what state is disk in.
- allocate(): find a free block
- mark it not free
- shut: add it to inodes of new file

How to remain consistent?
- prevent power outages
  - universal power supply (batteries)
  - raise interrupt when power is about to be lost
  - could respond by not starting any new writes, just waiting for existing to complete
FS recovery

- when we boot, check to see if the disk is in a consistent state, if not, fix it.
  (fsck: filesystem check)

  - traverse entire FS. (both files & free list)
    checking variants.
    - if a block is in neither: add to free list
    - if in both, remove from free list.

  - read entire disk! (only if shut down unexpectedly).
Journaling: improve (drastically) speed of fsck
- before writing to FS (structure)
  (allocation/dealloc)
  make a note indicating what you're about to do.
  then do the operation
  later: mark it as done.

```
crash:
  alloc():
    OK → Find a free block b
    write to journal: “I’m about to allocate block b as inode for file /home/uid/new.txt”
    Journal indicates need to ensure:
    b not in FL, b is in inode structure.
    OK →
```

- Need journal write to complete before
  start FS writes.
- `sync()`: wait for indicated
  writes to complete.
Log structured Filesystem
idea: everything goes in journal (journal is filesystem).
advantage: only need to append to end of FS,
never need to go back to do new writes.
- always a prefix that is consistent.

If we assume everything cached in memory
never read from disk, only write new
data.
- all writes sequential: fast.
---

**Filesystem Details**

- Each complete segment represents a consistent state.
- Need to know what is the most recent complete segment.
  (Footer in 'superblock' only updated after seg. completely written)

---

**How to write to a file?**

1. Need to overwrite Fl, B1,
2. Can't go back & change
3. ⇒ Create new copy in current segment,
   make that (in memory)
4. ⇒ Also need to update Inode (and indirect blocks, ...)
5. ⇒ Inode map: a table mapping node#s
to disk addresses

(Updates in an in-memory buffer [cached copy of]
new segment)

---

**Checkpoint:**
- Write buffer to disk in empty segment.
- Update super block

Lose everything saved between checkpoints.

---