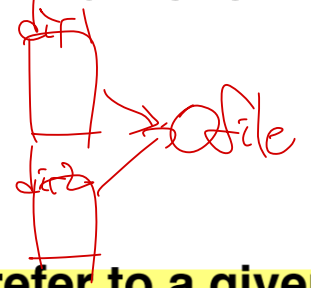
File Systems (part 2)

Instructor: Youngjin Kwon

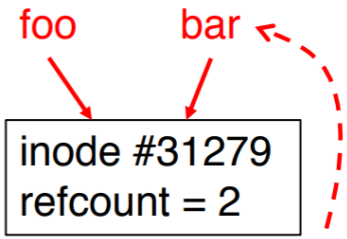
identical inodes

different inode

Hard and Soft Link

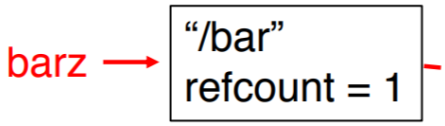


- **More than one dir entry can refer to a given file**
  - Unix stores count of pointers ("hard links") to inode
  - To make: "`ln foo bar`" creates a synonym (bar) for file foo



- **Soft/symbolic links = synonyms for names**
  - Point to a file/dir name, but object can be deleted from underneath it (or never exist).
  - Unix implements like directories: inode has special "symlink" bit set and contains name of link target

```
ln –s /bar barz
```



  - When the file system encounters a soft link it automatically translates it (if possible).

Other File System design issues

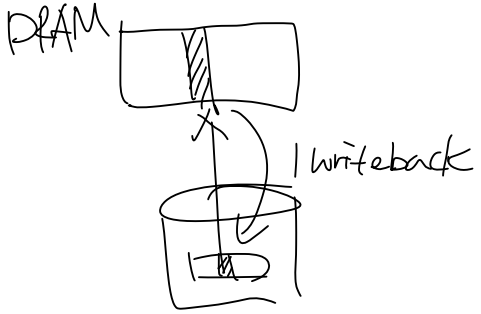
- Performance: DRAM caching of slow storage
- Sharing: File sharing
- ~~Protection: File Access control~~ skip

File buffer cache (aka page cache)

- Rationale for caching in DRAM?

- File buffer cache
  - Cache file blocks in memory to capture locality
  - Cache is system wide, used and shared by all processes
  - Reading from the cache makes a disk perform like memory
- Issues
  - File buffer cache competes with VM (tradeoff here), Like VM, it has limited size
  - Need replacement algorithms again (LRU usually used)

Caching Write



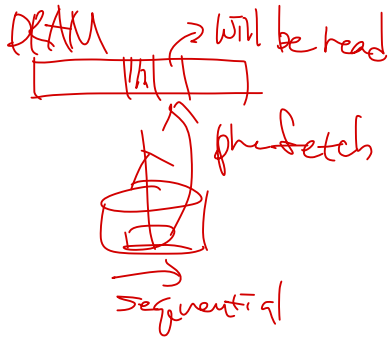
- On a write, some applications assume that data makes it through the buffer cache and onto the disk

- OSes typically do write back caching
  - Maintain a queue of uncommitted blocks
  - Periodically flush the queue to disk (30 second threshold)
  - If blocks changed many times in 30 secs, only need one I/O
  - If blocks deleted before 30 secs (e.g., /tmp), no I/Os needed

- Unreliable, but practical
  - On a crash, all writes within last 30 secs are lost
  - Modern OSes do this by default; too slow otherwise
  - How to make data persistent if needed?

linux

fsync API to flush to disk



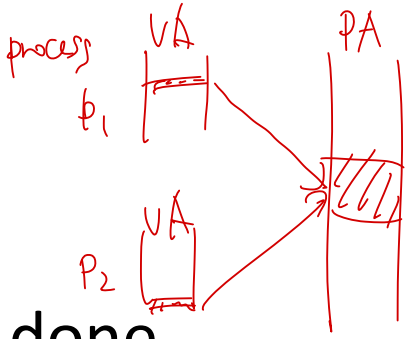
Sequential is more frequent than random (locality advantage)

Read Ahead (prefetch)

- Read ahead: File system *preloads* data blocks to the file cache
  - FS predicts that the process will request next block
  - FS goes ahead and requests it from the disk
  - This can happen while the process is computing on previous block
    - Overlap I/O with execution
  - When the process requests block, it will be in cache

- Typically what blocks to read ahead?
  - Hint. Spatial locality → (ex. sequential)

File Sharing



- File sharing is important for getting work done
  - Basis for communication and synchronization
    - E.g., Lock file: /var/lock/myLock.lock

Applications can share file by..

- Data on file cache, how to share?

page table mapping (diff VA but same PA)

- Semantics of concurrent access

Undecidable without synchronization
No guarantee

  - What happens when one process reads while another writes?
  - What happens when two processes open a file for writing?
  - What are we going to use to coordinate?

↳ still undecidable

Using file lock

File Sharing

- Semantics of concurrent access
  - What happens when one process reads while another writes?
  - What happens when two processes open a file for writing?
  - What are we going to use to coordinate?