CS330: Operating Systems

Filesystem: caching and consistency

Recap: file system organization

- File systems maintain several meta-data structures like super blocks, inodes, directory entries to provide a file system abstractions like files, directories
- How to search/lookup files/directories in a given path?
- Read the content of the root inode and search the next level dir using the name and find out its inode number
- Read the inode to check permissions and repeat the process
- Inode contains the index structures to deduce the disk block address given an logical offset

}

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access
- Examples:
  - Opening a file

```
fd = open("/home/user/test.c", O_RDWR);
```

  - Normal shell operations

```
/home/user$ ls
```

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access
- Executables, configuration files, library etc. are accessed frequently
- Many directories containing executables, configuration files are also accessed very frequently. Metadata blocks storing inodes, indirect block pointers are also accessed frequently

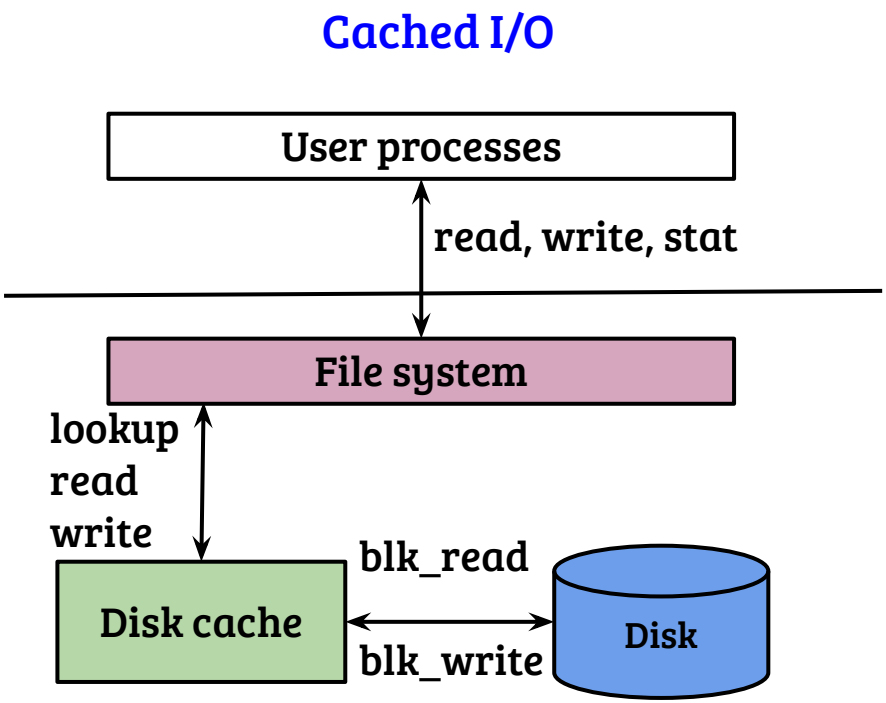
/home/user$ ls

File system and caching

- Accessing data and metadata from disk impacts performance
- Can we store frequently accessed disk data in memory?
  - What is the storage and lookup mechanism? Are the data and metadata caching mechanisms same?
  - Are there any complications because of caching?
  - How the cache managed? What should be the eviction policy?

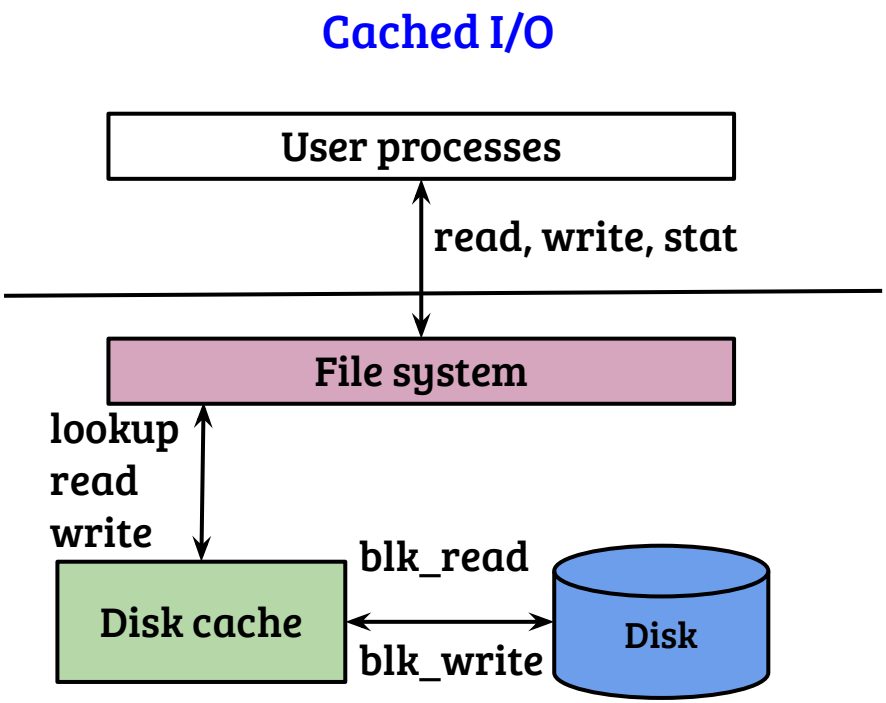
/home/user$ ls

Block layer caching



- Lookup memory cache using the block number as the key
- How does the scheme work for data and metadata?
- For data caching, file offset to block address mapping is required before using the cache
- Works fine for metadata as they are addressed using block numbers

File layer caching (Linux page cache)



- Store and lookup memory cache using {inode number, file offset} as the key
- For data, index translation is not required for file access
- Metadata may not have a file association, should be handled differently (using a special inode may be!)

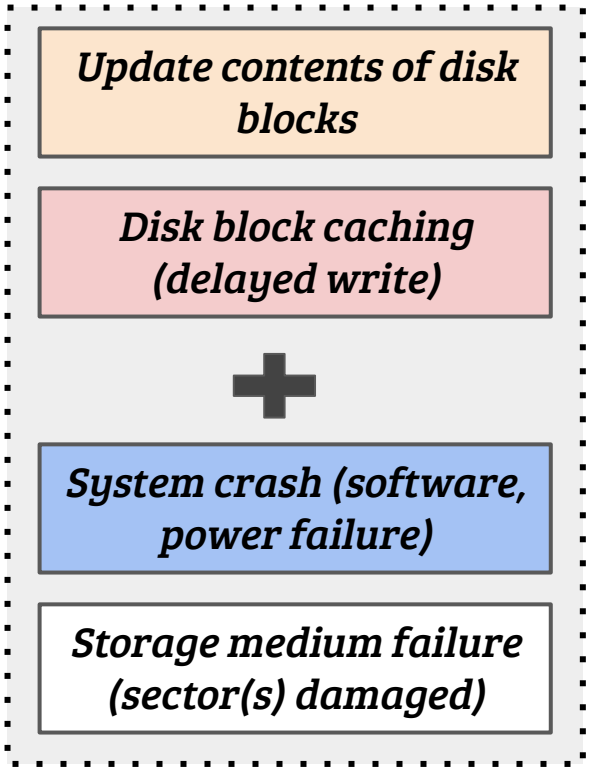
File system and caching

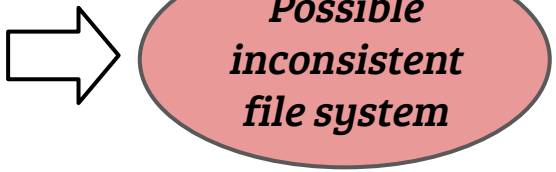
- Accessing data and metadata from disk impacts performance
- Can we store frequently accessed disk data in memory?
  - What is the storage and lookup mechanism? Are the data and metadata caching mechanisms same?
  - File layer caching is desirable as it avoids index accesses on hit, special mechanism required for metadata.
  - Are there any complications because of caching?
  - How the cache managed? What should be the eviction policy?

Caching and consistency

- Caching may result in inconsistency, but what type of consistency?
- System call level guarantees
  - Example-1: If a write( ) system call is successful, data must be written
  - Example-2: If a file creation is successful then, file is created.
  - Difficult to achieve with asynchronous I/O
- Consistency w.r.t. file system invariants
  - Example-1: If a block is pointed to by an inode data pointers then, corresponding block bitmap must be set
  - Example-2: Directory entry contains an inode, inode must be valid
  - Possible, require special techniques

File system inconsistency: root causes

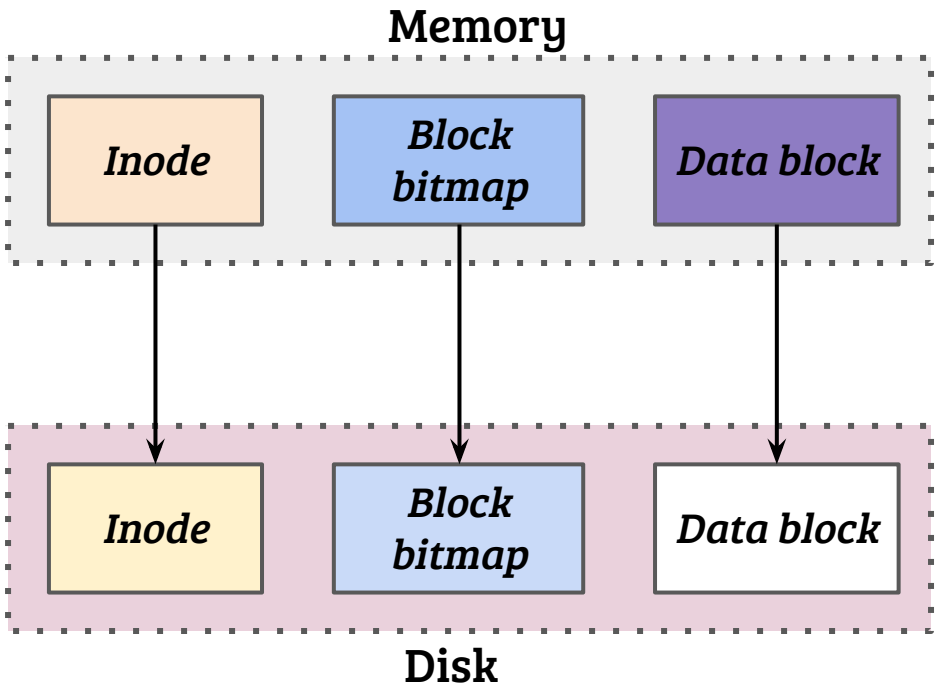




- No consistency issues if user operation translates to read-only operations on the disk blocks
- Device level atomicity guarantees matter!

Example: Append to a file

- Steps: (i) seek to the end of file, (ii) allocate a new block, (iii) write user data
- Inode modifications: size and block pointers
- Block bitmap update: set used block bit for the newly allocated block(s)
- Data update: data block content is updated



Three write operations reqd. to complete the operation, what if some of them are incomplete?

Failure scenarios and implications

Written	Yet to be written	Implications
Data block	Inode, Block bitmap	File system is consistent (Lost data)
Inode	Block bitmap, Data block	File system is inconsistent (correctness issues)
Block bitmap	Inode, Data block	File system is inconsistent (space leakage)

- All failure scenarios may not result in consistency issues!

Failure scenarios and implications

Written	Yet to be written	Implications
Data block, Block bitmap	Inode	File system is inconsistent (space leakage)
Inode, Data block	Block bitmap	File system is inconsistent (correctness issues)
Inode, Block bitmap	Data block	File system is consistent (Incorrect data)

- Careful ordering of operations may reduce the risk of inconsistency
- But, how to ensure correctness?

File system consistency with *fsck*

- Strategy: Do not worry about consistency, recover after abrupt failures
- During FS mount, check if it had been cleanly unmounted when it was last used, How to know?
  - Maintain the last unmount information on superblock
- If the FS was not cleanly unmounted, perform sanity checks at different levels: *superblock*, *block bitmap*, *inode*, *directory content*
- Sanity checks and verifying invariants across metadata. Examples,
  - Block bitmap vs. Inode block pointers
  - Used inodes vs. directory content