File systems 2

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Admin

- Midterm grades are out
- Midterm solutions sketch on Piazza

Previous lecture

- Storage terminology
- File system terminology
- Design space
- inodes and metadata
- Typically FS interface and basic steps
  - Mounting file system
  - Open
  - Read/Write
  - Delete
  - Finding files

Recall: FS challenges and requirements

- Huge files
- Many files
- Support the backing devices: magnetic tapes, HDD, SDD, non-volatile memory, non-volatile
- Data may need to survive the process lifetime
- Consistency
- Data revisiting, check-pointing
- Distributed remote access

Recall: Storage terminology

- Disk: Non-volatile block-addressable storage
- Block: Smallest unit of IO on a disk
  - Common block size = 512 bytes or 4KB
- Partition: A set of contiguous blocks on a disk

Recall: File system terminology

- File: A unit of data managed by the file system
- Data: The user data associated with a file
  - Unstructured (byte stream) or structured (records)
- File name:
  - A textual name that identifies the file

Recall: File system terminology

- Metadata: Information about the file (e.g., owner, creation time, permission, length of file data, location of file data, etc.)
  - As opposed to the data in the file
- Directory (folder):
  - Container for the file names
  - Directories within directories provide a hierarchical name system

Recall: File system terminology

- Superblock:
  - A record of file system's properties, containing the key file system information
- Inode (file control block):
  - A structure that stores a file's metadata and location of file data

Recall: Design choices

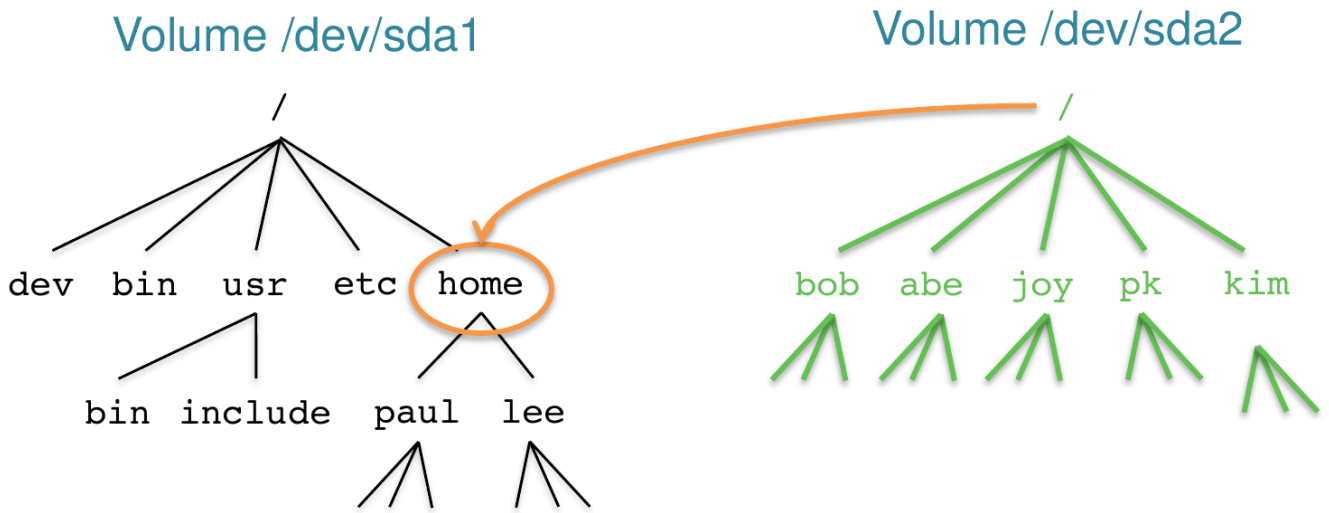
- Namespaces: flat, hierarchical, or other
- File system types: support one type of filesystem or multiple types
- File types: unstructured (byte streams) or structured (indexed files)
- Metadata: What kind of attribute are should be stored?
- Implementation: How is data laid out on disk?

Recall: Mounting

- Make file system available for use
- **Mount** system call (in Linux):
  - Args: file system type, block device, mount point
- Steps:
  - Access the raw disk (block device)
  - Read superblock and file system metadata
  - Prepare in-memory data structures:
    - In-memory version of the superblock
    - References to the root directory

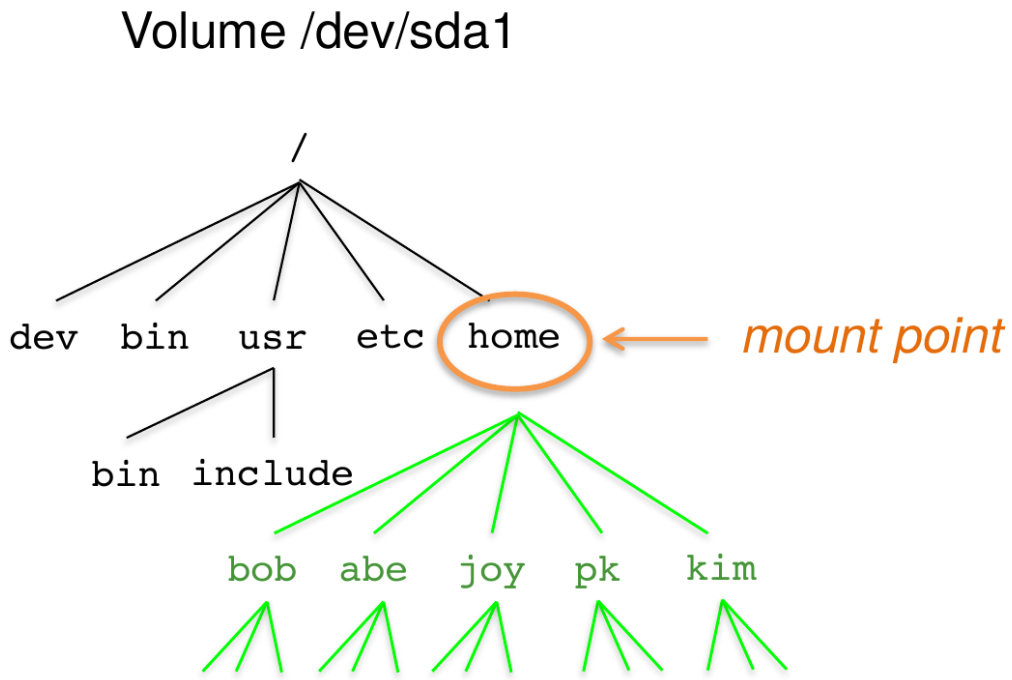
Recall: Mounting: building up a name space

- Combine multiple file systems into a single hierarchical name space
- The mounted file overlays (i.e., hides) anything in the file system under that mount point
- Looking up a pathname may involve traversing multiple mount points



Recall: Mounting: building up a name space

- Result



`/home/paul` and `/home/lee` are no longer visible

Creating a file

- Create an inode to hold info (metadata) about the file
  - Initialize timestamps
  - Set owner/permissions/modes
- Add a directory entry (dentry) for the new inode
  - Directory entry holds: a set of {filename, inode #}
  - Use current directory or pathname specified by filename

Recall: Creating a directory

- A directory is just like a file:
  - Contents = set of {name, inode} pairs
- Steps:
  - Create a new inode and initialize it
  - Initialize contents to contain:
    - A directory entry to the parent (name = "...")
    - A directory entry to itself (name = ".")

Recall: Delete a file

- Remove the file from its directory entry
  - This stops other programs from opening it: they won't see it
- If there are no program with open references to the file
  - Mark all the blocks used by the file as free
  - Mark the inode used by the file as free
  - Check this condition when closing a file (or existing a process)

Recall: Read a directory

- Directories are like files but contain a set of {name, inode} tuples
- The FS implementation parses this storage structure
- Operations (in Linux):
  - opendir
  - readdir
  - closedir

Recall: Read and write metadata

- Read inode information: stat Linux system call
- Write metadata: calls to change specific properties
  - chown
  - chgrp
  - chmod
  - utime
- Extended attributes (name-value sets)
  - listxattr
  - getxattr
  - setxattr
  - removexattr

Discussion

- SSD vs. HDD how does this affect a file system?
- What is the impact of keeping time information about file accesses?
  - What optimizations can we implement if we give on it?
- Is fragmentation a potential problem?

Break

File system design challenge

- How to organize the hierarchical file system on an array of blocks?
  - And make it space efficient and fast?
- See file system comparison:
  - https://en.wikipedia.org/wiki/Comparison_of_file_systems#Limits

Directory organization

- A directory is just a file containing names and references
- Linear list
  - Search can be slow for large directories
  - Cache frequently-used entries
- Hash table
  - Linear list but with hash structure
  - Hash(name)
- Tree structure
  - Balanced tree, constant depth
  - Great for a large number of files

Block allocation: contiguous

- Contiguous storage for file data would be great!
  - Minimized disk seeks when accessing a file
- Each file occupies a set of adjacent blocks
- You just need to know the starting block & file length

Problems with contiguous allocation

- Storage allocation is difficult:
  - External fragmentation
  - Internal fragmentation
  - Periodic fragmentation

Block allocation #1: Linked allocation

- A file's data is stored with a linked list of disk blocks
  - Directory contains a pointer to the first block of the file
  - Each block contains a pointer to the next block
- Problems:
  - Only good for sequential accesses
  - Each block uses space for the pointer to the next block

Block allocation: #2. Indexed allocation

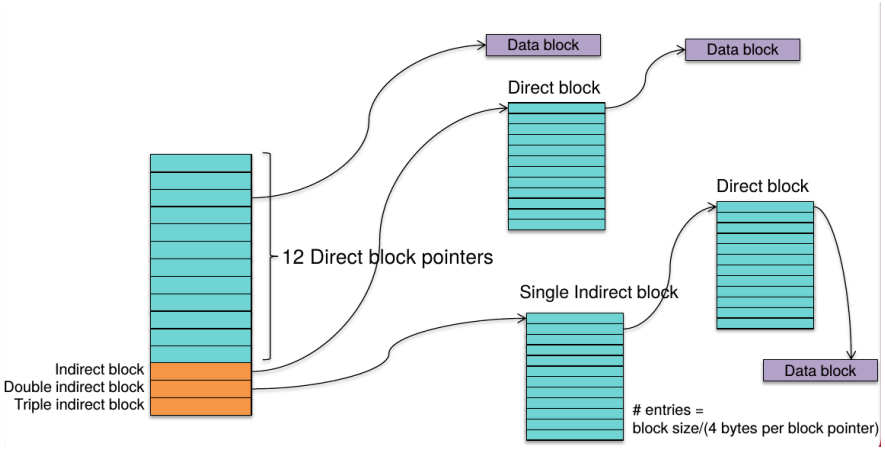
- Linked allocation is not efficient for random access
- Indexed allocation:
  - Store the entire list of block pointers for a file in one place
    - Disjoint metadata: decouple metadata and data
    - The index block (inode)

Block allocation: #2. Indexed allocation

- Directory entry contains name and inode number
- Inode contains file metadata (length, timestamps, etc.) and a block map
- On file open, read the inode to get the index map
- Q: How do you support a huge file?
  - Increase # of block pointers
  - Increase block size
  - A hierarchical structure like page tables?

Example: Linux ext2

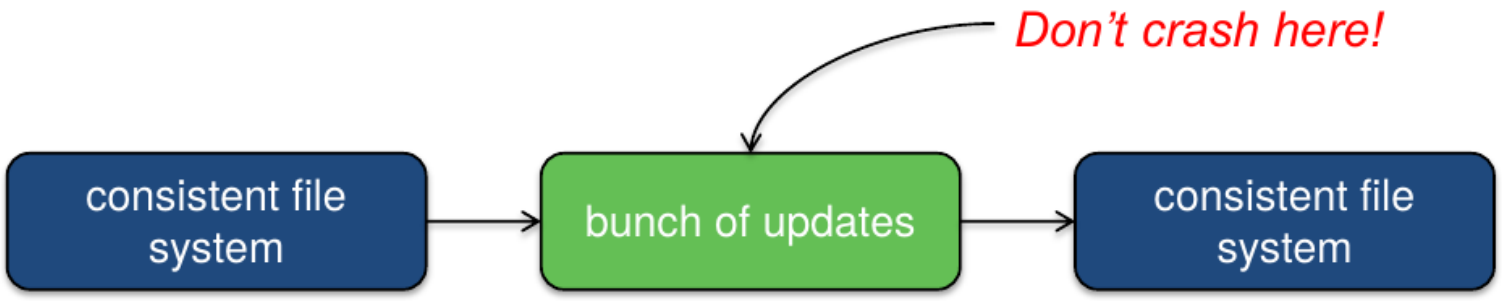
- Inodes with direct, indirect, double-indirect, and triple-indirect blocks



Journaling

Consistent update problem

- Example: writing a block to a file may require several operations:
  - Inode is:
    - Updated with a new block pointer
    - Updated with a new file size
  - Data free block bitmap is updated
  - Data block contents are written to disk
  - If any of these are not written to disk, we have a file system inconsistency



Journalling

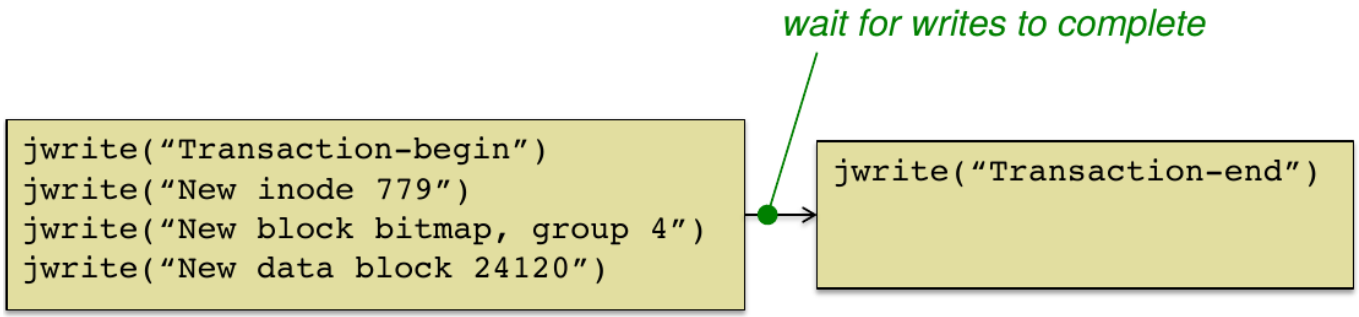
- Journaling == write-ahead logging
- Keep a transaction-oriented journal of changes:
  - Record what the FS is about to do (along with all the data)

```
Transaction-begin
  New inode 779
  New block bitmap, group 4
  New data block 24120
Transaction-end
```

- Once the log entry has been committed to disk then overwrite the real data in disk
  - If all goes well, no need for this entry
  - If there is a crash after the log has been committed:
    - Replay the log on reboot (*redo logging*)
- This scheme is called full data journaling

Writing the journal

- Writing the journal all at once would be risky
  - We don't know the order the disk will schedule the individual block writes
    - E.g., don't want to risk having a "transaction end" written before the transaction contents
- Steps:
  - Write all blocks, except "transaction end" tags
  - Wait for all the writes to complete
  - Then write "transaction end" tag
- If the log is replayed and a transaction-end is missing, ignore the log



Cost of journaling

- We’re writing everything twice and constantly seeks to the journal area of the disk
- Optimization (metadata journalling, aka ordered journalling):
  - Do not write user-data to journal
  - Only metadata

```
Transaction-begin
  New inode 779
  New block bitmap, group 4
Transaction-end
```

More discussion

- What can go wrong with a file system?
  - Power loss
  - Disk failure?
  - What if disk failure is partial? How can a file system design help / mitigate such problems?
- How to implement file deletion?
  - What is the impact of such implementations?
- Concurrency

Discussion

- When is data guaranteed to be written to disk?

Summary

- Terminology
- FS design space
- Basic FS:
  - Inodes and metadata
  - Typically FS interface and basic steps
- Block allocation
- Journalling