COS 316 Precept: Unix file system & Git’s content addressable storage

Outline

- Unix File System
- Git's content addressable storage

Recap of UNIX File System Layers

- Block layer
  - Names: integer block numbers
  - Values: fix-sized “blocks” of contiguous persistent memory
  - Purpose: Organize persistent storage into fix-sized blocks
- File layer
  - Names: Inode struct
  - Values: Arrays of bytes up to size N
  - Purpose: Organizes blocks into arbitrary-length files
- Inode number layer
  - Names: Inode numbers
  - Values: Inode struct
  - Purpose: Name files as uniquely numbered inodes
- Directory layer
  - Names: Human readable names with “directory”
  - Values: Inode numbers
  - Purpose: Human-readable names for files in a directory
- Absolute path name
  - Purpose: a global root directory
  - Usually assigned with a hardcoded inode number

```
typedef block uint8_t[4096]

# There is some hardware-specific translation from
# blocks to, e.g., plate number and offset
struct device {
  block blocks[N]
}
```

```
struct inode {
  int32_t block_numbers[N];
  int32_t filesize
}
```

```
struct dirent {
  char[MAX_NAME_LENGTH] filename;
  int    inode_number;
}
```

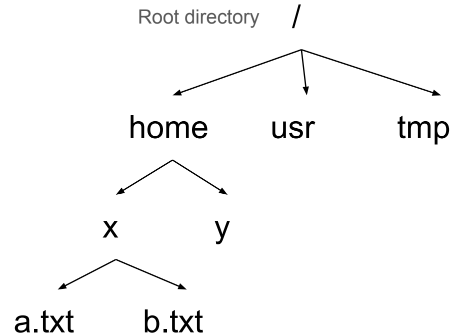
- Recap from lecture:
-
- Block layer: we can reach the physical block by the block number. All the contents are stored as byte arrays in blocks.
- File layer: we can get the list of allocated physical blocks number from the inode struct
- Inode number layer: we can get the inode struct by the provided inode number
- Directory layer: we can get the corresponding inode number given the directory name

Example:

Task 1: Check if “a.txt” is in “/home/x/” (lookup)

Suppose we’ve already known that the inode number of “x” is #1000. (**Directory layer**)

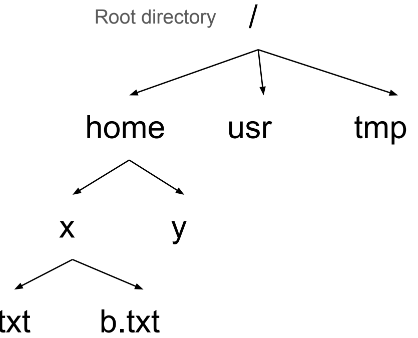
1. Investigate the inode struct whose inode number is #1000
   a. #1000 → Inode struct #1000
   b. **Inode number layer**
2. Get list of physical block numbers in #1000
   a. **File layer**
3. Read the byte array stored in those blocks based on the given block numbers
   a. …| a.txt #1101 | b.txt #1102 |...
   b. **Block layer**



Task 1: lookup task. Find if there is a file called “a.txt” in “/home/x/”. The directory tree is on the upper right corner.
Given an inode number, we can get an ordered byte array (stored in the physical blocks) as follows:

- Assumption: We already know the inode number of “/home/x”
- First: Inode number layer – Get inode struct for the given inode number
- Second: File layer – Get the list of physical block numbers in this inode struct
- Finally: Block layer – Read the blocks by the provided block numbers.
    - For each file/subdir in this directory, the filename and inode number is one entry of the block(s).

Example:



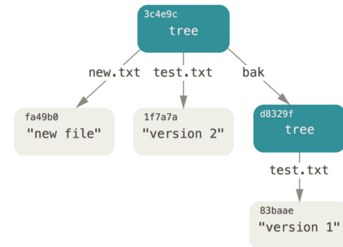
Task 2: Read /home/x/a.txt (read)

Assume inode number of “/” is #1. (**Absolute path name**)

1. Start from “/”, get the byte array from the blocks following the previous procedure shown in task 1
   a. #1 -> …**|** home #2 **|** usr #4 **|** tmp #6 **|**...
2. The sub-dir “home” is in “/”. Its inode number is 2. Repeat the previous procedure
   a. #2 -> …**|** x #20 **|** y #24 **|**...
3. The sub-dir “x” is in “/home/. It is #20. Repeat..
   a. #20 -> …**|** a.txt #35 **|** b.txt #2484 **|**...
4. Finally, we find “a.txt”
   a. #35 -> …**|** b”welcome to” **|** b”the precept” **|**...

Recap of Git

- Object
  - Name: SHA-1 hash value of the given contents
  - Values: Blobs (Binary large objects), Trees, Commits
  - Purpose: All data is stored as objects
  - Similar to Blocks
- Tree
  - Name: Human-readable strings, like UNIX dirs
  - Values: Object name, type, permission
  - Similar to Directories
- Commit
  - Name: SHA-1 hash of the value
  - Values: Object name of the tree, object name of parent commits, committer info, message…
  - Purpose: A way to express a version history of source code tree. Each commit is like a “snapshot”.
- Reference
  - Name: human readable names
  - Values: a commit name
  - Purpose: Provide global, human readable names for objects



Recap from lecture:

- Blobs, trees and commits are all objects. Objects is named with its hash value of the given contents.
- Trees organizes blobs into a directory-like hierarchy
- Commit versions the tree layer like a “snapshot”. It also record the parent commit.
- References; not shown in this figure.

Example

Task: Create a file “hello.txt” with some contents. Then update to the remote repo.

1. “git add hello.txt”
   a. Git creates an object (blob) where name is the hash of the contents and value is the contents. (Object layer)
2. “git commit -m ‘add a new file’”
   a. Git constructs a new tree object to represent the structure of the repo. (Tree layer)
   b. Git creates a commit object which includes reference to the tree, metadata of tree(message and committer info) and reference of the previous commit.
3. “git push”

(Optional) You can also give reference to a commit by

“git tag <ref_name> <commit_hash>”