Project #5: FullFS

Hyeongtak Ji

Systems Software & Architecture Lab.

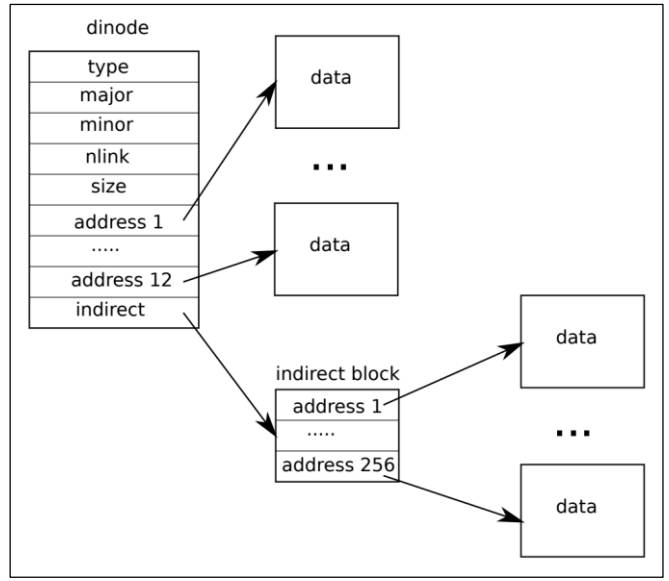
Seoul National University

2024.12.06

The xv6 file system

- The xv6 file system provides Unix-like files, directories, and pathnames, and stores its data on a virtio disk for persistence
  - B: Boot block (1 block)
  - S: Superblock (1 block)
  - L: Log blocks (30 blocks)
  - I: Inode blocks (13 blocks)
  - M: Free bitmap blocks (1 block)
  - D: Data blocks (1954 blocks)

< Structure of the xv6 file system>

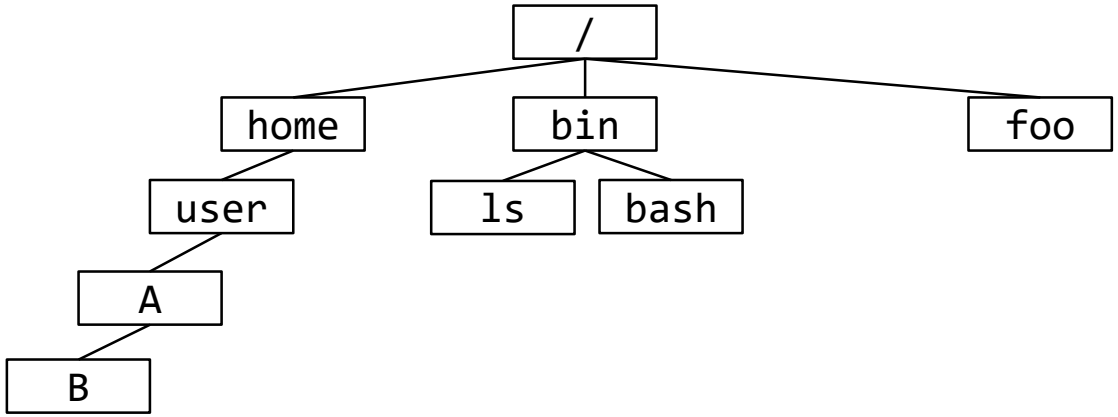


< Representation of a file on disk>

Traditional Pathname Lookup

- **Hierarchical traversal**: Path resolution starts at the root (‘/’) for absolute paths or the current directory for relative paths, traversing each component sequentially

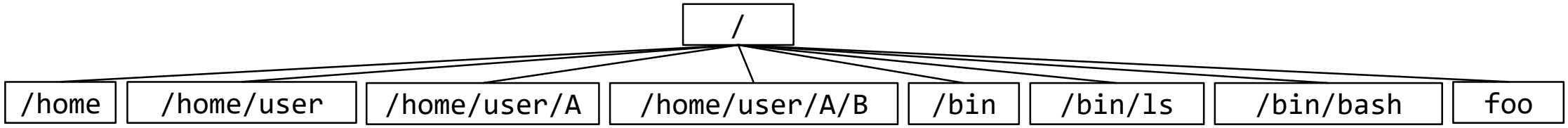
- **Step-by-step inode lookup**: At each step, the system accesses the current directory's inode, searches for the next component, and retrieves its inode until the final component is resolved.



Full-Path Indexing

- **Direct lookup**: Files and directories are indexed by their *full path*, enabling direct lookup without hierarchical traversal

- **Advantages**: Reduces the overhead of resolving deep or complex paths by eliminating intermediate steps



FullFS Design

- All files and directories are stored in the *single root directory*, eliminating the traditional hierarchical file system

- Directory entry design
  - Each entry stores the inode number and the full path
  - Each directory entry is fixed at 128 bytes in size

```
// @ kernel/param.h
#define MAXPATH   106

// @ kernel/fs.h
struct dirent {
  ushort inum;          // inode number
  char padding[20];     // reserved space for custom implementation
  char name[MAXPATH];   // full pathname
};
```

FullFS Design (cont'd)

- Subdirectories do not have their own directory entries
  - Again, the only directory that has directory entries is the "/"

- But… must still satisfy the following requirements:
  - A parent directory must exist to create files or subdirectories
    - e.g., creating "`/foo/bar.txt`" is allowed only if a directory named "`/foo`" exists
  - Support complex paths including '`.`' and '`..`'
    - `/home/user/../../bin/./ls, /..//../../bin/ls, …`
  - Fully support commands with relative path like
    - `$ cd ..`
    - `$ ../cat ../README`
  - Hint: run `$ usertests -q`

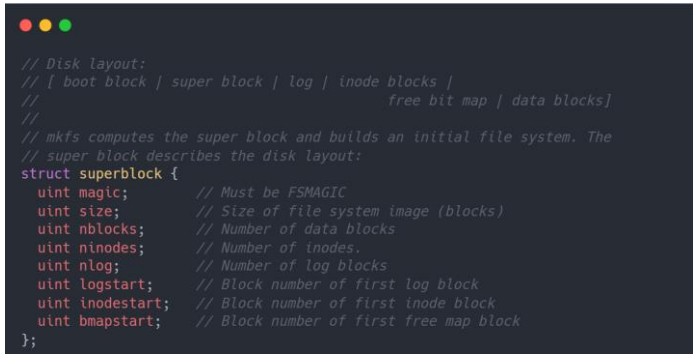
Project#5: FullFS

- In this project, you have to
  1. Modify the *mkfs* utility (10 points)
  2. Implement full-path indexing (60 points)
  3. Modify the *ls* command (20 points)
  4. Design document (10 points)
  5. And, there is a bonus (up to 20 points)

- Due date is 11:59 PM, December 22 (Sunday)

1. Modify the *mkfs* Utility

- You should modify `mkfs` to set up the FullFS
- You may need to consider disk layout changes
- About superblock…
  - You may add fields to the superblock
  - Do not change the location of superblock
  - Do not change the names of existing variables (e.g. `magic`, `size`, `inodestart`, …)
- The root directory should store all entries
  - ‘.’, ‘..’ should be excluded



```
// Disk layout:
// [ boot block | super block | log | inode blocks |
//                                          free bit map | data blocks]
//
// mkfs computes the super block and builds an initial file system. The
// super block describes the disk layout:
struct superblock {
  uint magic;        // Must be FSMAGIC
  uint size;         // Size of file system image (blocks)
  uint nblocks;      // Number of data blocks
  uint ninodes;      // Number of inodes.
  uint nlog;         // Number of log blocks
  uint logstart;     // Block number of first log block
  uint inodestart;   // Block number of first inode block
  uint bmapstart;    // Block number of first free map block
};
```

2. Implement Full-Path Indexing

- Modify the file system so that all system calls operate correctly with the new directory structure

- Implement path resolution logic that handles both absolute and relative paths correctly
  - Traditional ‘.’ and ‘..’ entries are not stored in the directory, though

2. Implement Full-Path Indexing (cont'd)

- Implement `pwd()` system call
  - The system call number of `pwd()` is already assigned to 23

- `int pwd(char *buf)`

- pwd() returns the absolute pathname of the current working directory for the calling process
  - The pathname is stored in the buffer provided by the argument `buf`

3. Modify the `ls` command

- Modify the `ls` command to display directory contents as if they exist in a hierarchical structure
- The `ls` command must correctly interpret and display the contents of directories as if they existed hierarchically
  - `$ ls /home` → should display the entries within the `/home` directory
- See the `README` file for detailed examples and expected output formats

```
$ ls
2 2 1 2403 README
2 3 1 35264 cat
2 4 1 34144 echo
2 5 1 16336 forktest
2 6 1 38696 grep
2 7 1 34616 init
2 8 1 34080 kill
2 9 1 33904 ln
...
```

4. Design Document

- You need to prepare and submit the design document for your implementation
- You should explain what you have considered, and what you have done
- Requirements
  - New data structures
  - Algorithm design
  - Testing and validation

5. Bonus

- Students with perfect scores on part 1, 2, and 3 in the grading server qualify for bonus points
- We will evaluate the average execution time of the `open()` system call
  - Using `rdtime()`, with the QEMU option "`-icount shift=0`"
- The top five fastest implementations will receive 20 bonus points, and the next five will earn 10 points — prove you've got the speed!

- Note: using hashing can be an option, but…
  - Ensure at least one byte-to-byte pathname comparison to avoid false positives
  - Any submissions without this comparison will NOT BE ELIGIBLE for bonus points

Restrictions

- Please use QEMU version 8.2.0 or later
- Your implementation should pass `usertests` on multi-processor RISC-V systems (i.e., CPUS > 1)
  - Some irrelevant test cases in the `usertests` suite have been disabled
- Please remove all the debugging outputs before you submit

Tips

- Read Chap. 8 of the xv6 book to understand the file system in xv6
- For your reference, the following roughly shows the amount of changes you need to make for this project assignment
- Each "+" symbol indicates 1~10 lines of code that should be added, deleted, or altered

```
kernel/defs.h       |  +
kernel/exec.c       |  +
kernel/fs.h         |  +
kernel/fs.c         |  ++++++
kernel/proc.c       |  +
kernel/sysfile.c    |  ++++++++++++
mkfs/mkfs.c         |  ++++
user/ls.c           |  ++++++++++++++++
```

Skeleton Code

- Skeleton Code
  - You should work on the pa5 branch of the xv6-riscv-snu repository as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ git checkout pa5
```

- The pa5 branch includes two user-level program, `pwd` and `fsperf`
  - `pwd` simply calls the `pwd()` system call and prints its result
  - `fsperf` is a program designed to evaluate file system performance
  - Note: A different program, not `fsperf`, will be used for bonus point evaluations

Notification

- Due
  - 11:59 PM, December 22 (Sunday)

- Any attempt to copy others' work will result in a heavy penalty

- Submission
  - Run the `make submit` command to generate a tarball named `xv6-pa5-{STUDENTID}.tar.gz` in the `xv6-riscv-snu` directory
  - Upload the compressed file to the submission server
  - The total number of submissions for this project will be limited to 50
  - Only the version marked FINAL will be considered for the project score

Thank you!