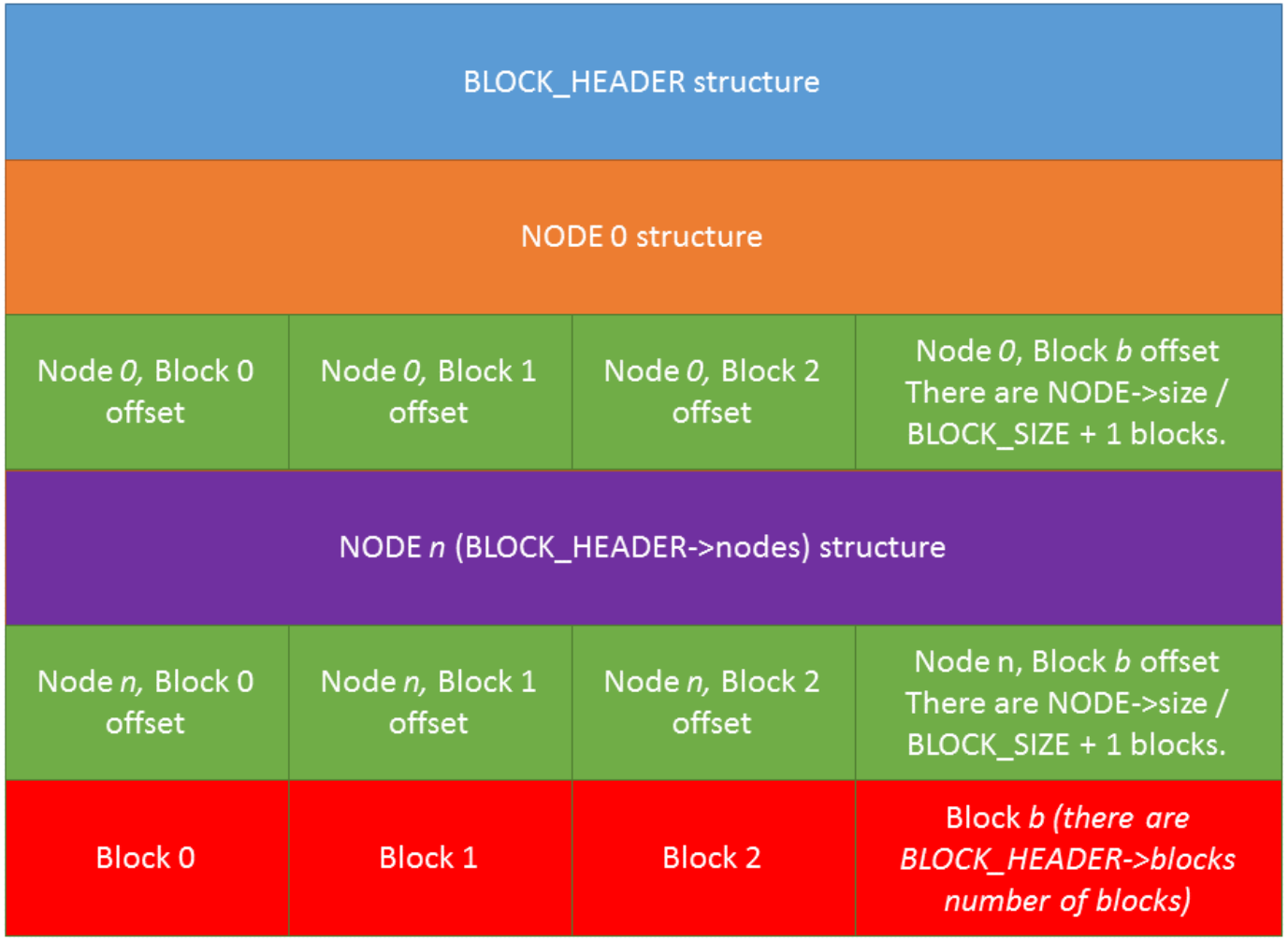
COSC 361
Filesystem Functions

Stephen Marz

Topics

- Filesystem Functions
  - create
  - open
  - read
  - write
  - getattr
  - readdir
  - opendir
  - mkdir
  - rmdir
  - chmod
  - chown
  - unlink
  - truncate
  - rename

COSC 361 Project 4 File System



Create Function

- Given: path and mode (permissions and type)
  - Make sure path and node do not already exist!
- Allocates a new inode and block
- Size will be 0 (increasing this will be done with write)
- Everything else is "defaulted"
  - Default uid = getuid()
  - Default gid = getgid()
  - Default mode given by passed parameter (or this with S_IFREG)
  - Default time = time(NULL) // There are 3 of these!!

Open Function

- For files only (not directories)
- Checks to see if file exists
- OS' job is to associate the file with a descriptor number
- Openable? return 0
- Not found? return –ENOENT

Read Function

- Reads data up to “size” starting with “offset”

- Step 1: Find the file given by the path
- Step 2: Find the blocks associated with this file
- Step 3: Move to “offset”
- Step 4: Start writing the data into *buf
  - (you should write “size” bytes up to the size of the file)
- Step 5: Return number of bytes you read into *buf

Write Function

- Similar to read

- Step 1: Find the file given by the path (should exist since if it doesn't create is called)
- Step 2: Find the blocks associated with the file
- Step 3: Allocate new blocks or write the rest of current blocks (you will need at least size+offset blocks).
- Step 4: Move to offset
- Step 5: Write *data into the new blocks (may span more than 1 block!)
- Step 6: Update the block offsets for this file
- Step 7: Return number of bytes written

- Do NOT truncate using write. If the current size is > size+offset, then don't change the size. fs_write can only grow a file. fs_truncate is used to shrink them.

Getattr Function

- Gets attributes about a file or directory
- Uses the “stat” data structure
- struct stat;
  - Fields you need to populate:
  - st_mode – matches node->mode
  - st_nlink – Always 1 (our FS doesn’t support hard links)
  - st_uid – matches node->uid
  - st_gid – matches node->gid
  - st_size – matches node->size
  - st_*x*time – matches node->*x*time (where x $\in \{a, m, c\}$)

Readdir Function

- Reads the nodes that belong to a directory
- Uses a function pointer “filler” to fill void *buf

- filler(buf, “.”, 0, 0);
- filler(buf, “..”, 0, 0);
  - You must ALWAYS have . and .. (current and parent)
- Make sure you NEVER put any forward slashes in this!
  - filler(buf, file_name_without_slashes, 0, 0);
  - One filler call for each node inside of a directory

Opendir Function

- Analogous to the Open function
- Checks to see if the directory exists
  - If yes: return 0
  - If no: return –ENOENT (no entry)

Mkdir Function

- Creates a new, empty directory at the given path
- path contains the directory that needs to be created (it’ll be the last name)
- Step 1: Allocate a new node
- Step 2: Set default metadata (mode, uid, gid, size, etc.)
  - size must be 0 for all directories
  - Set the time by using time(NULL)
  - Set the uid and gid by using getuid() and getgid()
  - Set the mode by S_IFDIR | mode
- Step 3: Add the new node to the node list.

Rmdir Function

- Removes a directory node
- Step 1: Find the node
- Step 2: Make sure it is a directory (check for S_IFDIR bit)
- Step 3: Remove the node
- Step 4: Free resources used (if used)
- Step 5: return 0 if successful

Chmod Function

- Changes the mode of a node
- Step 1: Search for the node given by path
- Step 2: Change the mode to given mode
- Step 3: return 0 if successful, error code if not

Chown Function

- Changes the uid and/or the gid of a node (directory or file)

- Step 1: Search for file or directory
- Step 2: Set the uid to given uid and gid to given gid
- Step 3: Return 0 if successful, error code if not

Unlink Function

- Deletes a file only (not directories!)
- Step 1: Find the node given by “path”
- Step 2: Ensure it is NOT a directory (if it is, return –EISDIR)
  - Directories are removed by rmdir
- Step 3: Delete the blocks associated with this node
- Step 4: Delete the node
- Step 5: Return 0 if successful, error code if not

Truncate Function

- Step 1: Find the node given by the “path”
- Step 2: End the file at the given size. Any block past the size must be deleted.
- You may need to reduce the number of blocks in the node for this function.
- Take care that you remove the block properly and update the # of blocks.

Rename Function

- Renames “path” to “new_name”
- The new name might contain a new path!!
- Step 1: Search for “path” node (may be file or directory)
- Step 2: Check to make sure “new_name” contains a valid path
- Step 3: Rename the node to the new path (change node.name)
- Step 4: Return 0 if successful, error code if not

fs_drive: Loading the Block Drive

- You will load the entire drive into memory by implementing fs_drive function.

- Keep track of nodes and blocks separately.
  - You may implement whatever data structures you need
  - You may even use globals

- Create a fresh block drive by using "makeblock"

fs_destroy: Saving the Block Drive

- This function is called when your program exits
- It should write the new nodes/blocks back to the hard drive file
  - This saves anything I did with your file system
- Refer to the format above

- If you screw this up, use makeblock to create a fresh, new drive

Project Hints

1. Implement fs_drive
2. Implement getattr
3. Implement open and opendir
4. Implement readdir and read
5. These functions are all that are required to get a very basic file system going.

Potential Problems

- The attached Makefile uses the libraries in my home folder
  - This will not work if you're using your own system!
  - Must be on a Hydra machine

- If you segfault or any other failure, you may have to manually unmount your filesystem
  - fusermount –u */path/to/your/mountpoint*
    - -u means "unmount"

Interacting

- Run ./fs
- This will mount your file system into the “mnt” folder
- Open a new terminal
- cd into the mnt folder
- You should see README.txt and README.too if you created a new hard_drive using makeblock.
- To test you filesystem, interact with it much like you would a normal filesystem.

```
smarz@qdev:~/mzfs $ ./fs
fs_drive: hard_drive
fs_drive: sizeof(BLOCK_HEADER) = 48, sizeof(NODE) = 584, sizeof(BLOCK) = 8
fs_drive: BlockSize:  1024, Nodes: 6, Blocks: 3
NODE: / [0]
NODE: /README.txt [1]
   block 0 -> 0
NODE: /README.too [2]
   block 0 -> 1
NODE: /some_subdir [3]
NODE: /some_subdir/subdir_2 [4]
NODE: /some_subdir/subdir_2/data [5]
   block 0 -> 2
Press CONTROL-C to quit

fs_getattr: /
fs_getattr: /
fs_opendir: /
fs_readdir: /
fs_getattr: /
fs_getattr: /README.txt
fs_getattr: /README.too
fs_getattr: /some_subdir
fs_getattr: /hello.world
fs_create: /hello.world
fs_getattr: /hello.world
fs_write: /hello.world
fs_opendir: /
fs_getattr: /
fs_readdir: /
fs_getattr: /README.txt
fs_getattr: /README.too
fs_getattr: /some_subdir
fs_getattr: /hello.world
fs_getattr: /
fs_getattr: /hello.world
fs_open: /hello.world
fs_read: /hello.world
fs_read: Node: '/hello.world', Blocks: 1, Block Start: 0, Size: 6
fs_read: Bytes = 6, Size = 6
fs_getattr: /hello.world
```

Terminal 1

Terminal 2

```
smarz@qdev:~/mzfs/mnt $ ls -la
total 0
drwxr-xr-x 0 smarz smarz   0 Apr 10 10:36 .
drwx------ 1 smarz smarz 272 Apr 12 11:42 ..
-rw-r--r-- 0 smarz smarz  24 Apr 10 10:36 README.too
-rw-r--r-- 0 smarz smarz  41 Apr 10 10:36 README.txt
drwxr-xr-x 0 smarz smarz   0 Apr 10 10:36 some_subdir
smarz@qdev:~/mzfs/mnt $ echo hello > hello.world
smarz@qdev:~/mzfs/mnt $ ls
hello.world  README.too  README.txt  some_subdir
smarz@qdev:~/mzfs/mnt $ cat hello.world
hello
smarz@qdev:~/mzfs/mnt $
```