CSE 415 - Operating Systems
Project #3 - RAM Disk Filesystem
Spring 2012 - Prof. Butler[1]

Due date: June 8, 2012

In this project, you will implement filesystem functions on a RAM disk file system that is provided. The ideas behind this project are discussed in the lecture on filesystem implementaion and are described in the OSC book in Chapter 11. While this code is not based directly on any OS implementation, it does share the common concepts from a UNIX file system. You will be allowed to work in groups of two for this project.

1 Specification

You are going to be required to implement four functions in the file system implementation:

- **fileSeek(unsigned int fd, unsigned int offset):** Set the offset in the per-process file structure (`fstat_t`) for the file specified by the file descriptor `fd`. You will need to look up the file for the file descriptor. Your implementation must be able to seek to any location in the file.
- **fileRead(unsigned int fd, char *buf, unsigned int bytes):** This function reads bytes bytes from the current position of the file offset (e.g., set by `fileSeek` above). Read must find the file blocks to read and collect the data into the buffer. The function diskRead is provided to retrieve the bytes from the block when the correct block is found. Your file read must be able to read the entire file or any subset. Files may span multiple data blocks.
- **diskWrite(fcb_t *fcb, unsigned int block, char *buf, ...):** Writes the data from the buffer buf into the RAM disk at the appropriate block for the file referenced by the file control block `fcb`. You need the fcb to maintain the correct size of the file on the disk. The function `diskRead` is similar, so please use that as a hint. This file only writes to one block at a time.
- **fileWrite(unsigned int fd, char *buf, unsigned int bytes):** This function writes bytes of data from buffer buf into the file from the current position of the file offset (e.g., set by `fileSeek` above). Your function `diskWrite` performs the actual writing to the RAM disk. This function and `fileRead` share a number of similarities: both require finding the appropriate blocks (to write in this case) and call the disk-level version to update the RAM disk. Your file write must be able to write to any location in the file and beyond the current end of the file (i.e., can make the file larger). Files may span multiple data blocks.

You need data to test `fileRead`, so you may build a simplified version of `fileWrite` first (e.g., writes to only one block) before going to the full versions necessary.

The challenge in this project is to understand the concepts in the file system. The file system structures are defined in the file `cis415-filesys.h`. Please study these structures – most map to the structures discussed in Chapter 11. The full tarball for the project will be attached.

The other challenge is to understand the layout of these structures in blocks on the RAM disk. Figure 1 gives a diagram outlining the blocks in the on-disk file system.

The first block (block 0) is the file system (partition) control block. All the blocks have a bit of block information at the beginning (`dblock_t`), but after that are the block contents. Block 0 specifies the number

[1]Many thanks to Dr. Trent Jaeger for the design of this assignment.

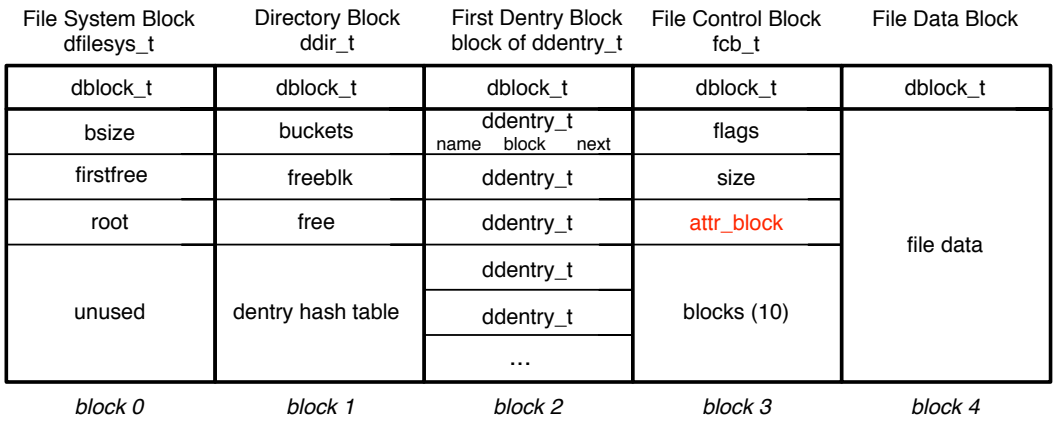

Figure 1: Structure of the file system.

of blocks in the file system (`bsize`), the current first free block (`firstfree`), and the block number for the root directory (only directory). There is only one file system block.

Block 1 is the block that stores the directory. A directory refers to its hash table of directory entries (i.e., `dentry`) by its number of buckets which fill the remainder of the block and the location of the next free spot for a `dentry` (`freeblk` and `free (slot)`). Only the heads of the hash table are stored in the hash table. Each `dentry` has a next reference that is used to traverse the hash table lists. There are usually multiple directory blocks, but in this project there is only one.

Block 2 is the first `dentry` block, and it contains a set of dentries which each refer to a specific file by its name and (first) block. A next reference specifies the next entries to access for the dentry hash table. There can be multiple of these.

Block 3 is the file control block (FCB) which refers to the file metadata and actual data blocks. There are only 10 blocks in a file currently. There is one FCB per file.

Block 4 is a typical file data block. Other than the block header these blocks contain only file data. There should be lots of these.

In the assignment, an output file shows the sequence of commands and responses for your filesystem. You will run 5 commands to generate this output: `./cis415-p3 your_fs cmdi` where `cmdi` is the *ith* command for (e.g., cmd1 for the first). This program is deterministic, so your output should match the given output, modulo any bugs.

As a hint, you should make copies of the `fs` file as once you have created files within the RAM disk, any subsequent creates will cause file errors, since they will already exist. This is not a problem for when you are working on the file seek/write/read etc, but for the purposes of showing a clean run for your submitted solution, you should show your answers against a new instance of the RAM filesystem.

2 Submission Instructions

You may develop your code on any machine with a C compiler, but we will be testing on Ubuntu. You should probably use the VM that you have been working with throughout the course.

You **must** turn in the following:

1. A `README` file. The `README` will contain the following:
   - Your name and your parter's name if you work in a group of 2
   - A list of submitted source files

- Compilation instructions
- Overview of work accomplished and what each group member worked on
- Description of code and code layout
- General comments and anything that can help us grade your code

Use the existing makefile. **cd** up a directory so that `proj3` is a subdirectory of your working directory, and run the following command to create your submission tarball, but replacing "UOEMAIL" with your cs.uoregon.edu email account name.

```
tar cvzf proj3_submission_UOEMAIL.tar.gz proj3
```

For example, since my UO CIS email account is "butler", I would run the command:

```
tar cvzf proj3_submission_butler.tar.gz proj3
```

Use the turnin script, located at `http://systems.cs.uoregon.edu/apps/turnin.cgi`, to submit your tarball. If you work in a group of 2, only one submission needs to be made for the group. Make sure both group members are identified in the README.

3 Grading Guidelines

This programming project will be graded as follows:

- 10% Documentation
- 10% Correct operation of `fileSeek`
- 20% Correct operation of `diskWrite`
- 20% Correct operation of `fileRead`
- 20% Correct operation of `fileWrite`

Note that general deductions may occur for a variety of programming errors including memory violations, lack of error checking, poor code organization, etc. Also, do not take the documentation lightly, as it provides those evaluating your project with a general roadmap of your code; without good documentation, it can be quite difficult to grade the implementation. Once again, these will be graded on machines running the Ubuntu 10 Linux distribution.

4 Attribution

We expect that you will do this assignment *yourself/in your group*. In order to avoid issues of plagiarism and cheating in this project, you must attribute any outside sources that you use. This includes both resources used to help you understand the concepts, and resources containing sample code that you may have borrowed as a template for your shell. The course textbook only needs to be cited in this latter case; all other sources must be attributed. You should use external code sparingly. For example, using most of an example from the man page for `pipe(2)` would be reasonable. Using a function such

as `ParseInputandCreateJobandHandleSignals()` from a *Write Your Own Shell in Four Easy Steps* site would not be OK, either using text verbatim or closely following the structure for your own shell design. If you have technical concept questions, you should consider sending email to the mailing list. You should also email the admin list if you have questions.