

CSCI 3753 Assignment 4: FFS

Assignment Out: March 20th
Assignment Due: Tuesday, April 10th, 11:59PM

Motivation

In this assignment, you will write a user-level program that examines a disk device formatted with the Berkeley Fast File System and provides information about the disk structure of files. You will then use your program to examine the structure of several different files, including large ones such as `/usr/lib/libcrypto.p.a`. By doing so, you will become familiar with FFS, and also learn a little bit about the device file system (`devfs`). You will learn how a fairly complex file system is structured, and how this structure is handled by a kernel to provide the abstraction of a stream of bytes.

Assignment

Your assignment, should you choose to accept it, is to decode the inode structure of any file and locate the individual disk blocks allocated for that file. You'll look at some details of the Berkeley Fast File System (`ffs`). Your task is to write a program that will show the on-disk structure of a file. You will then provide a write up in which you analyze your findings and note any interesting observations. You may use either C or C++, whichever is more comfortable for you.

This program doesn't have to reside in the kernel. It is more efficient to develop code in user-land because you have a number of tools at your disposal (`gdb`, etc) and you don't risk crashing a kernel.

As you'll be writing this program from scratch, you should follow good programming practices. Notably:

- Your program should be neat, clean, and well organized.
- Your program should be well commented and easy to follow.
- Your program should be reasonably efficient.

We are providing a series of steps that will guide you on completing the project; however, if you would rather pursue the goals in a different manner, go for it. To help focus your efforts and to help with consistent grading, we're providing point values of the sub-goals in brackets.

All coding and testing should be performed on the OpenBSD machines that CSOps has been provided (as they're the only machines with FFS in the CSEL, you have to). You can remotely log into them via `ssh` from other nodes, of course. The FFS `devfs` entries are read-only, so you can't possibly cause trouble for other people. As these machines are a limited resource, though, be sure to not leave runaway processes eating up CPU. Check that you don't have any periodically. The commands `xcpustate` or `top` help with this. The OpenBSD nodes are named: `perciatelli`, `fedelini`, `cavatelli`, and `gnocchi`.

Program Specification

Your program will take a filename and an optional device entry on the command line and print the file's inode number. You will print selected fields of the superblock and you will print parts of the inode including the disk block addresses that contain the file.

A sample version of the program exists: `/home/cia/csci3753/bin/fsinfo`.

You must provide Makefile for your program. Typing `'make clean;make'` should correctly compile your program. All programs will be tested on the OpenBSD machines in the CSEL, as they're the only machines with FFS partitions. Your executable must be named `fsinfo`. The syntax for `fsinfo` is as follows:

```
fsinfo [-f filesystem-device] filename
```

If the `-f` option is not specified, then you should determine the appropriate device with the `stats(2)` system call. Note that you want to use the character devices such as `/dev/rwd0e`, not the block devices such as `/dev/wd0e`.

The output from your program should look something like this:

```
-----  
  
INODE=359  
  
fs_fsize=1024 fs_bsize=8192 fs_size=1889216 fs_ncg=58 fs_fpg=32768  
  
MODE=100555 SIZE=1556332 BLKCNT=21  
  
129802 129807 129810 129815 129818 129823 129826 129831 129842 129847  
139802 139807 139810 139815 139818 139823 139826 139831 139842 139847  
139848  
  
-----
```

The fewer deviations you have from the above format, the less chance the TAs will misunderstand your output and deduct points accordingly. `man printf(3)` for how you can format the output properly (say, `"%7d"`).

Hints and Suggested Steps

Check the return values of system calls. Be very careful matching data types. If something is declared as a given type provided with a typedef, use that type. Otherwise, your code might not work if the actual type is different on another system. Some data types are interchangeable, some are not. You should know the difference between these:

- int
- unsigned int

CSCI 3753 Assignment 4: FFS

- long int
- unsigned long int
- int64_t
- quad_t
- off_t
- daddr_t

Use the flag `-Wall` when compiling. It will help you keep the data types proper. Your handin shouldn't have any compilation warnings.

The header files provide a large number of macros for accessing the file system. Read over them carefully; chances are, there's already one that does something you want to do involving the `fs` structure.

Be careful when using block numbers; they usually have to be translated from file system block numbers to disk block numbers. There are macros in `fs.h` that do this for you.

Be sure to read in blocks of data at a time. Trying to read an individual inode may not succeed.

The 8 Step Approach

Step 1: Read, think, question, read more

You'll need to understand the filesystem structure. You'll need to look at the following code (we showed these data structures in recitation):

- `/usr/include/ufs/ffs/fs.h`
- `/usr/include/ufs/ufs/dinode.h`

Here are some suggested online references: `man 2 stat`, `man 5 fs`.

Step 2: Get a feel for the filesystem

Run some provided commands to get a feel for the file system.

- `df -i`
- `ls -l -i`
- `fdisk /dev/rwd0d`
- `disklabel /dev/rwd0d`
- `dumpfs /dev/rwd0d`

CSCI 3753 Assignment 4: FFS

Step 3: Play with the file system debugger, fsdb

```
fsdb -r /dev/rwd0s2a
> ls
> cd
> inode 123
> help
> print
```

Check out "man fsdb". The filesystem debugger can help you verify that your program is working correctly before submittal time.

Step 4: Use the "stat" system call

Use `stat` to lookup and print the inode number of a file.

Step 5: Super-block

Read the superblock of a filesystem and print various values. Here are some suggested fields to print: `fs_ipg`, `fs_sblkno`, `fs_cblkno`, `fs_iblkno`, `fs_dblkno`, `fs_cgoffset`, `fs_size`, `fs_dsize`, `fs_bsize`, `fs_fsize`, `fs_frag`, `fs_csaddr`, `fs_cssize`, `fs_cgsize`, `fs_inopb`.

Eventually you'll need most of these include files. It won't hurt to enumerate them now:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>
#include <sys/queue.h>
#include <sys/lock.h>
#include <ufs/ffs/fs.h>
#include <ufs/ufs/quota.h>
#include <ufs/ufs/inode.h>
#include <ufs/ufs/dinode.h>
#include <fcntl.h>
#include <err.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/uio.h>
#include <unistd.h>
```

Here might be a handy block-reading function:

```
int bread(off_t off, void* buf, int cnt) {
    lseek(rfd, off, SEEK_SET);
    if ((nr = read(rfd, buf, cnt)) != cnt) {
        perror("failed read");
        return (0);
    }
}
```

CSCI 3753 Assignment 4: FFS

```
    return (1);  
}
```

It would be called as follows:

```
if ((rfd = open(file, O_RDONLY)) < 0) {  
    perror("open");  
    exit(5);  
}  
  
if (bread((off_t)SBOFF, &sbk, SBSIZE) == 0) {  
    (void)close(rfd);  
    exit(6);  
}
```

Step 6: Map the inode number

Next you want map the inode number to a cylinder group and eventually to the actual disk sectors containing your target inode. Use the `ino_to_cg` macro found in `fs.h`. Also, these macros may prove useful: `ino_to_fsba`, `ino_to_fsbo`, and `fsbtodb`.

Step 7: Then read the inode block

Then read the file system block containing your desired inode. Print out selected inode information as described in the program specification. Also, print the filesystem block numbers for your file.

Step 8: Collect Data

Collect some data on some medium-large files and analyze the file structure. Write up the results as described below.

Grading

[20] Proper tar/compress submission

Do NOT submit any binary or executable programs. FOLLOW THE PROGRAM SPECIFICATION. We will be compiling and running your program with automated scripts. Failure to following the specs will likely result in a failed run and you'll cause us tedious work that will lower your grade.

[15] Program compiles and correctly prints the inode number

[15] Program correctly reads the superblock and prints selected fields.

[15] Program correctly reads the on-disk inode for the specified file and prints the information correctly.

[10] Program correctly handles indirect block pointers.

[15] README

A short report describing this project (1 paragraph **introduction**), a **program** section, an **analysis** section, and a **summary**. Don't repeat the details of what is in man pages or in the include

CSCI 3753 Assignment 4: FFS

files - reference them if necessary. You should provide information on the structure of your program in the **program** section, along with any known bugs.

[10] Analysis

In the analysis section, answer these questions:

- Is the filesystem allowing the files to occupy contiguous blocks, or are there many discontinuities?
- Based on what you see, how many I/O operations would you estimate would be necessary to read your whole selected file?
- Is it optimal? If not, how could things be improved?

[EC] Extra Credit

For extra credit, enhance your program to automatically figure out what filesystem contains the specified file. Your program will need to figure out filesystem mount points. See "man 2 getfsstat" and "man 3 getmntinfo" for details.

Handin

Your program should include a makefile to compile your source files. Entering the directory your program exists in, typing `make clean; make fsinfo` should compile your program, which must be named `fsinfo`. Your program will be tested on the OpenBSD machines in the CSEL, so be sure they compile and run properly there.

Your program should include a README file. This file should outline the basic structure of your program. You should also state any known bugs. Reporting a bug will result in fewer points being taken off than if you do not report it; knowing what's wrong with your program but not fixing it is better than not knowing at all. Your README should also describe the naming convention you use in your code.

In addition, your README should contain the outputs of a few runs of your program, with observations and comments. Are there any interesting patterns that you see? What are they? What might be their cause? Look at the analysis section of the grading scheme for more detailed information.

You should take all of the files for your program (makefile included) and place them into a tar. `tar cf fsinfo.tar [file1 file2 ...]` accomplishes this. You should then run the handin programs in `/home/cia/csci3753/bin`. You should be in the directory your file is in when running this program. The syntax for handin is as follows:

```
% handin <assignment#> <file>
```

This is the fourth assignment, so if the tar of your files were named `fsinfo.tar`, you would type

```
% /home/cia/csci3753/bin/handin 4 fsinfo.tar
```

Only the latest handin is kept. If you discover a bug, feel free to hand in again before the due date. Even if you handed in a program before the due date, handing in one after it will result in your program being late. In other words, only the last handin matters.