

Operating Systems

Week 10. Assignment 7 Discussion

Paul Krzyzanowski
Rutgers University
Spring 2015

March 31, 2015 © 2015 Paul Krzyzanowski 1

File Systems & *Everything is a File*

- File systems
 - Typically used to store & organize data
 - Implemented over block devices (disks and flash memory)
- “Everything is a file”
 - Evolved through the history of UNIX (& BSD, Plan 9, Linux)
 - Devices appear as files
 - Names in the file system name space
 - inode contains major & minor device number
 - Requests are sent to the device driver

March 31, 2015 © 2015 Paul Krzyzanowski 2

Pseudo devices and files

- Device files can refer to software drivers
 - No underlying device
 - `/dev/zero` – read an infinite # of 0 bytes
 - `/dev/random` – return random bytes
- File systems can be software drivers too
 - No underlying block device
 - File system driver under VFS presents something that looks like a file system
 - Example:
 - `/proc` – process file system: get kernel & process information

March 31, 2015 © 2015 Paul Krzyzanowski 3

File Systems as a Name Space

- File system name space is a powerful abstraction
 - Easy to understand: users & programs know how to browse, read, and write files
 - Easy to work with: GUI tools, command-line utilities, scripts, and programming language interfaces
- Example
 - Change the maximum # of file handles the kernel will allocate
 - `echo 8192 > /proc/sys/fs/file-max`
 - Look at the computer's name
 - `cat /proc/sys/kernel/hostname`
 - Change it
 - `echo myname.pk.org > /proc/sys/kernel/hostname`
 - *No need for extra commands or system calls!*

March 31, 2015 © 2015 Paul Krzyzanowski 4

FUSE: Filesystem in USErspace

FUSE enables a file system to run as a normal user process

- FUSE file system module
 - Conduit to pass data between VFS and the user process that implements the file system

March 31, 2015 © 2015 Paul Krzyzanowski 5

Getting information

FUSE is maintained at fuse.sourceforge.net

- Source code
- Documentation
- Examples

March 31, 2015 © 2015 Paul Krzyzanowski 6

FUSE components

- The program that implements the file system links with the FUSE library (`libfuse`)
- FUSE consists of:
 - Kernel module (`fuse.ko`)
 - File system module (`fusefs`) and character device (`/dev/fuse`)
 - User-space library (`libfuse.so`)
 - Mount utility (`fusermount`) to mount the file system onto the namespace
- Your user-level file system is linked with the FUSE library (`libfuse.so`)

March 31, 2015

© 2015 Paul Krzyzanowski

7

How it works (at the simplest level!)

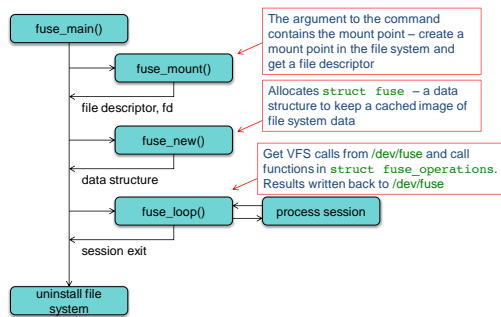
- FUSE kernel module
 - Redirects VFS calls to the user-level library via the `/dev/fuse` character device
- The main program in your file system (`fuse_main`):
 - Parses arguments and calls `fuse_mount()`
 - Opens `/dev/fuse`
 - Each process that opens `/dev/fuse` gets a different file descriptor
 - Reads VFS file system calls from `/dev/fuse`
 - Calls file system functions stored in `fuse_operations` struct
 - These are functions you write to implement the file system
 - Results written back to `/dev/fuse` via the file descriptor

March 31, 2015

© 2015 Paul Krzyzanowski

8

The life of a user-level file system



March 31, 2015

© 2015 Paul Krzyzanowski

9

FUSE Operations

- Defined in `struct fuse_operations`
 - Not all of these need to be implemented – depends on what the file system needs to do
 - This is a high-level overview & not a complete list
 - Read the documentation!
- Operations: file system
 - `void *(*init) (struct fuse_conn_info *conn);`
 - Initialize your file system
 - `int (*statfs) (const char *, struct statvfs *);`
 - Provide file system statistics
 - `void (*destroy) (void *);`
 - Clean up your file system – free any allocated data

March 31, 2015

© 2015 Paul Krzyzanowski

10

File & Directory create/move/delete

- `int (*mknod) (const char *, mode_t, dev_t);`
 - Create a file node (device file or named pipe)
- `int (*mkdir) (const char *, mode_t);`
 - Create a directory
- `int (*unlink) (const char *);`
 - Remove a file
- `int (*rmdir) (const char *);`
 - Remove a directory
- `int (*symlink) (const char *, const char *);`
 - Create a symbolic link (pointer to a file or directory)
- `int (*rename) (const char *, const char *);`
 - Rename a file or directory
- `int (*link) (const char *, const char *);`
 - Create a hard link to a file (alias)

March 31, 2015

© 2015 Paul Krzyzanowski

11

Directory data operations

- `int (*opendir) (const char *, struct fuse_file_info *);`
 - Open directory
 - Unless the `'default_permissions'` mount option is given, this method should check if `opendir` is permitted for this directory
 - `opendir` may return an arbitrary `filehandle` in the `fuse_file_info` structure
 - This will be passed to `readdir`, `closedir` and `fsyncdir`.
- `int (*readdir) (const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);`
 - Read directory
- `int (*releasedir) (const char *, struct fuse_file_info *);`
 - Release directory
- `int (*fsyncdir) (const char *, int, struct fuse_file_info *);`
 - Synchronize directory contents

March 31, 2015

© 2015 Paul Krzyzanowski

12

File attribute operations

```
-int (*getattr) (const char *, struct stat *);
  • Get file attributes
-int (*setattr) (const char *, const char *, const char *, size_t, int);
  • Set extended attributes
-int (*getxattr) (const char*, const char*, char*, size_t);
  • Get extended attributes
-int (*listxattr) (const char *, char *, size_t);
  • List extended attributes
-int (*removexattr) (const char *, const char *);
  • Remove extended attributes
-int (*readlink) (const char *, char *, size_t);
  • Read the target of a symbolic link
-int (*chmod) (const char *, mode_t);
  • Change the permission bits of a file
-int (*chown) (const char *, uid_t, gid_t);
  • Change the owner and group of a file
```

March 31, 2015

© 2015 Paul Krzyzanowski

13

File operations

```
-int (*open) (const char *, struct fuse_file_info *);
  • Open a file
-int (*flush) (const char *, struct fuse_file_info *);
  • Flush any cached data for an open file
  • Called when a file is closed
-int (*fsync) (const char *, int, struct fuse_file_info *);
  • Synchronize file contents
-int (*create) (const char *, mode_t, struct fuse_file_info *);
  • Create and open a file. If the file does not exist, first create it with the specified mode, and then open it.
```

March 31, 2015

© 2015 Paul Krzyzanowski

14

File data operations

```
-int (*truncate) (const char *, off_t);
  • Change the file size to a given offset
-int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
  • Read bytes of data from an open file
-int (*write) (const char*, const char*, size_t, off_t,
              struct fuse_file_info *);
  • Write bytes of data to an open file
-int (*flush) (const char *, struct fuse_file_info *);
  • Flush any cached data for an open file
  • Called when a file is closed
```

March 31, 2015

© 2015 Paul Krzyzanowski

15

Assignment 7 Overview

- Create a user-level **math file system (mathfs)**
 - Runs via FUSE
- The root of mathfs comprises seven directories
- Each directory represents a mathematical function:
 1. `/factor` - Computes the prime factors of a number.
 2. `/fib` - Computes the first *n* fibonacci numbers.
 3. `/add` - Adds two numbers
 4. `/sub` - Subtracts two numbers.
 5. `/mul` - Multiplies two numbers.
 6. `/div` - Divides two numbers.
 7. `/exp` - Raises a number to a given exponent.

March 31, 2015

© 2015 Paul Krzyzanowski

16

Assignment 7 Overview

- Suppose you mount your file system on `/tmp/math`
 - Create a directory `/tmp/math`: `mkdir /tmp/math`
 - Run the program, giving it the mount point: `./mathfs /tmp/math`
- The command


```
cat /tmp/math/factor/12782
```

 will produce the prime factors of 12782:


```
2
7
11
83
```
- The command


```
cat /tmp/math/add/6/4
```

 will produce the sum of 6+4


```
10
```

March 31, 2015

© 2015 Paul Krzyzanowski

17

First, run the demo: get it

Before starting the assignment, be sure that you can use FUSE and run the “hello, world” demo

- See fuse.sourceforge.net

March 31, 2015

© 2015 Paul Krzyzanowski

18

Running the demo: compile it

- The “hello, world” file system is < 100 lines long
- Download hello.c
 - <http://fuse.sourceforge.net/helloworld.html>
- Compile it:


```
cc -o hello hello.c -D_FILE_OFFSET_BITS=64 -lfuse
```

Executable file:
hello

Source file: hello.c

You need to explicitly define
64-bit file offsets

Link with the fuse
library

March 31, 2015

© 2015 Paul Krzyzanowski

19

Running the demo: run it

- Create a mount point: any directory


```
mkdir hitest
```
- Run the hello file system, telling it to use hitest as the mount point


```
./hello hitest
```
- hello runs in the background
 - Be aware of this when debugging your program!
 - You have to remember to *unmount* the file system when done!


```
./fusermount -u hitest
```

March 31, 2015

© 2015 Paul Krzyzanowski

20

Running the demo: test it

We now have the file system running. Test it out:

```
$ ls -l hitest
total 0
-r--r--r-- 1 root root 13 Dec 31 1969 hello
```

There's just one file in there called hello. Let's look at it:

```
$ cat hitest/hello
Hello World!
```

It doesn't do much but it works!

We can see the process *hello* is still running

```
$ ps x |grep hello
15806 ?        Ssl    0:00 ./hello hitest
```

March 31, 2015

© 2015 Paul Krzyzanowski

21

Running the demo: stop it

When we're done, unmount the file system

```
$ fusermount -u hitest ← this is our mount point
```

This causes the process to exit

March 31, 2015

© 2015 Paul Krzyzanowski

22

Running the demo: debugging

Running *hello* with a *-d* flag enables debug logging

```
$ ./hello hitest -d ← this is our mount point
```

- Use another window for typing commands since log output goes to the screen
- Great way to see what functions are being called

ls calls:

- *getxattr* (not implemented)
- *readdir*
- *releasedir*

cat hello calls:

- *lookup*
- *open*
- *read*
- *getattr*
- *flush* (not implemented)
- *release*

March 31, 2015

© 2015 Paul Krzyzanowski

23

Minimal implementation

- FUSER passes in dozens of VFS functions
- You don't need to implement those you don't use
- The “Hello, World!” demo implements only four!

```
static struct fuse_operations hello_oper = {
    .getattr = hello_getattr,
    .readdir = hello_readdir,
    .open = hello_open,
    .read = hello_read,
};
```

March 31, 2015

© 2015 Paul Krzyzanowski

24

Minimal implementation

- Some implementations can be hard-coded
 - Everything in the “Hello, World!” demo is
 - Example, `readdir` returns a directory listing
 - The demo supports only one directory with one file

```
static const char *hello_path = "/hello";
static int hello_readdir(const char *path, void *buf,
                        fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)
{
    (void) offset;
    (void) fi;

    if (strcmp(path, "/") != 0)
        return -ENOENT;

    filler(buf, ".", NULL, 0);
    filler(buf, "..", NULL, 0);
    filler(buf, hello_path + 1, NULL, 0);

    return 0;
}
```

March 31, 2015

© 2015 Paul Krzyzanowski

25

Assignment 7 Implementation

Implement & debug each of the 7 math functions

- Make sure they work before plugging them into the file system
- Handle ALL errors: overflow, divide by 0, bad data
- You can always return an error message but don't die!

March 31, 2015

© 2015 Paul Krzyzanowski

26

Assignment 7 Implementation

- Then, create a basic file system that doesn't implement the operations but parses pathnames & returns dummy data
- At a minimum, you will need to implement
 - `getattr`: get attribute of a file; don't bother with timestamps
 - `readdir`
 - At the top level, you should show these directories
 - `factor fib add sub mul div exp`
 - Within each directory, you should show just one directory
 - `doc`: contains usage info for that function
 - `open`
 - Parses the pathname to get the operation & numbers and produce the results

March 31, 2015

© 2015 Paul Krzyzanowski

27

Assignment 7 Implementation

- Finally, tie the implementation of the functions into the file system and test everything!

March 31, 2015

© 2015 Paul Krzyzanowski

28

The End

March 31, 2015

© 2015 Paul Krzyzanowski

29