Writing a fuse filesystem

Table of Contents

This is intended to be a step-by-step guide to writing a pseudo-filesystem using fuse.

Previous versions of this tutorial used `fusepy` for the python bindings to fuse. Unfortunately, `fusepy` has been suffering bitrot, and is no longer a reliable set of bindings, so this version now uses the C bindings for fuse. This requires the libs and devel components to be installed (fuse3-libs and fuse-devel in Fedora 37). The C bindings have both a low-level and a high-level interface, and we'll just be using the high-level interface here.

This filesystem is a silly use of fuse and filesystems in general, but should give an idea of what writing a filesystem entails.

circlefs - our example filesystem

Our example filesystem will be a simple in-memory pseudo-filesystem named circlefs. circlefs will store one piece of information, `radius`, which can be changed as desired.

In addition to radius, the filesystem will provide files containing the `circumference` and `diameter` of a circle with the stored radius, as well as a file containing the value of `pi`. All files will be read-write (except for `pi`, which will be read-only for obvious reasons). Reading from any file will return either the value of the radius, the value of pi, or the computed value associated with the name of the file. Modifying any file will result in a corresponding change to radius.

Excuse me… is that a skeleton in your closet?

We start out with nothing but the skeleton C code, most of which we'll put into a header file:

circlefs.h

```
#ifndef __CIRCLEFS_H__
#define __CIRCLEFS_H__
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <inttypes.h>
#include <math.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>

#define FUSE_USE_VERSION 31

#include <fuse3/fuse.h>

struct circlefs_data {
        long double radius;
};

#endif
```

circlefs.c

```
#include "circlefs.h"

struct circlefs_data circlefs_data = {
        .radius = 0,
};

static const struct fuse_operations circlefs_ops = {
};

int main(int argc, char *argv[]) {
    return fuse_main(argc, argv, &circlefs_ops, NULL);
}
```

Houston, we have a filesystem (well, sort of)

Let's compile it and test it out to see what it does (hint: nothing yet):

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3
$ mkdir mnt
$ ./circlefs mnt

$ ls -al mnt
ls: cannot access 'mnt': Function not implemented
$ ls -ald mnt
ls: cannot access 'mnt': Function not implemented
```

So our filesystem really does nothing at this point. But hey... we did it. I guess.

Let's unmount before we forget:

```
$ fusermount -u mnt
```

So what's wrong?

Well, our `circlefs_ops` struct contains pointers to functions in our userspace program that implement the various kernel-side functions of a filesystem. In this case, all of our pointers are zero, so we haven't told our program what to do when listing the contents of a directory or getting the attributes of a directory entry (such as the root of the filesystem).

Start implementing some functions

In our code, we'll implement two functions: `readdir` and `getattr`

`readdir` is like the getdents() syscall, and will return a directory listing (the entries `.` and `..`).

`getattr` is similar to the stat() syscall, and will return specific information about a particular directory entry.

We'll also add some defines and helper functions to our `circlefs.h` as we go (just add the new pieces at the end of the existing content).

circlefs.h

```
--- circlefs.h  2023-10-28 10:12:09.325608215 -0500
+++ circlefs.h  2023-10-28 10:12:46.544019437 -0500
@@ -21,4 +21,27 @@
     long double radius;
 };

+struct circlefs_dirent {
+    char *name;     // entry name
+    struct stat st; // permissions, inode number, etc.
+};
+#define ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
+
+#define BLOCK_SIZE      42
+#define FILE_SIZE       4096
+#define DEVICE_MAJOR    42
+#define DEVICE_MINOR    42
+
+#define NUM_BLOCKS(size, bsize) ( (size + bsize - 1) / bsize )
+
+// helper function to fill a directory entry's 'struct stat'
+static void fill_statbuf(struct circlefs_dirent *ent) {
+    ent->st.st_uid = getuid();
+    ent->st.st_gid = getgid();
+    ent->st.st_size = FILE_SIZE;
+    ent->st.st_blksize = BLOCK_SIZE;
+    ent->st.st_blocks = NUM_BLOCKS(FILE_SIZE, BLOCK_SIZE),
+    ent->st.st_dev = makedev(DEVICE_MAJOR, DEVICE_MINOR);
+}
+
 #endif
```

And the `circlefs.c` additions just need to be between the definitions for `circlefs_data` and `circlefs_ops`

circlefs.c

```
--- circlefs.c  2023-10-28 10:14:37.401244284 -0500
+++ circlefs.c  2023-10-28 10:18:35.636876519 -0500
@@ -4,6 +4,47 @@
    .radius = 0,
 };

+struct circlefs_dirent circlefs_dirents[] = {
+   { .name = ".",      .st = { .st_mode = S_IFDIR | 0555, .st_ino = 1, } },
+   { .name = "..",     .st = { .st_mode = S_IFDIR | 0555, .st_ino = 2, } },
+   { .name = "radius", .st = { .st_mode = S_IFREG | 0644, .st_ino = 3, } },
+};
+#define DIRENT_COUNT (ARRAY_SIZE(circlefs_dirents))
+
+// path doesn't matter, since we don't have any subdirs
+static int circlefs_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
+       off_t start_offset, struct fuse_file_info *ffi,
+       enum fuse_readdir_flags readdir_flags) {
+
+   int offset;
+
+   // The 'start_offset' is used in case our directory listing needs to call
+   //     into readdir() more than once.  However, our filesystem is very
+   //     small, so we'll probably always start at the beginning of the dir.
+
+   for (offset = start_offset ; offset < DIRENT_COUNT ; offset++) {
+       fill_statbuf(&circlefs_dirents[offset]);
+       filler(buf, circlefs_dirents[offset].name,
+           &circlefs_dirents[offset].st, 0, FUSE_FILL_DIR_PLUS);
+   }
+   return 0;
+}
+static int circlefs_getattr(const char *path, struct stat *st,
+       struct fuse_file_info *ffi) {
+   int i;
+   if (!strcmp("/", path))
+       path = ".";
+   else if (*path == '/') // paths start with '/', so advance to fix that
+       path++;
+   for (i = 0 ; i < DIRENT_COUNT ; i++)
+       if (!strcmp(path, circlefs_dirents[i].name)) {
+           fill_statbuf(&circlefs_dirents[i]);
+           memcpy(st, &circlefs_dirents[i].st, sizeof(struct stat));
+           return 0;
+       }
+   return -ENOENT;
+}
+
 static const struct fuse_operations circlefs_ops = {
 };
```

And we'll plug the two new functions into the `circlefs_ops` structure:

circlefs.c

```
--- circlefs.c  2023-10-28 10:18:35.636876519 -0500
+++ circlefs.c  2023-10-28 10:21:23.428730431 -0500
@@ -46,6 +46,8 @@
 }

 static const struct fuse_operations circlefs_ops = {
+   .readdir        = circlefs_readdir,
+   .getattr        = circlefs_getattr,
 };

 int main(int argc, char *argv[]) {
```

Compile and run again:

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3
$ ./circlefs mnt

$ ls -alnd mnt
dr-xr-xr-x. 0 1000 1000 4096 Dec 31  1969 mnt

$ ls -aln mnt
total 102
dr-xr-xr-x. 0 1000 1000 4096 Dec 31  1969 .
drwxrwxr-x. 3 1000 1000 4096 Oct 23 12:15 ..
-rw-r--r--. 0 1000 1000 4096 Dec 31  1969 radius
```

Okay, that's a bit more interesting. Let's see what radius contains:

```
$ cat mnt/radius
cat: mnt/radius: Function not implemented

$ fusermount -u mnt
```

Ah. Well, that makes sense, since we haven't implemented the function to allow reading yet.

What's in the box?

Okay, now that we've got a directory listing, we need to be able to read the radius, so we'll add the function to allow reading a file:

circlefs.c

```
--- circlefs.c  2023-10-28 10:39:57.625041042 -0500
+++ circlefs.c  2023-10-28 10:42:56.801020735 -0500
@@ -44,6 +44,43 @@
        }
    return -ENOENT;
 }
+static int circlefs_read(const char *path, char *buf,
+   size_t size, off_t off, struct fuse_file_info *ffi) {
+
+   char localbuf[128] = { 0 };
+   long double val = NAN;
+   int copied;
+
+   // paths start with '/', so advance to fix that
+   if (*path == '/')
+       path++;
+
+   if (!strcmp(path, "radius"))
+       val = circlefs_data.radius;
+
+   if (isnan(val)) // not a real file
+       return -EBADF;
+
+   // try to check whether our value can be expressed as an integer
+   uint64_t int_val = val;
+   long double tmp = val - int_val;
+
+   if (tmp > 0)
+       snprintf(localbuf, sizeof(localbuf) - 1, "%.36Lf", val);
+   else
+       snprintf(localbuf, sizeof(localbuf) - 1, "%" PRIu64, int_val);
+
+   if (off > strlen(localbuf))
+       copied = 0;
+   else
+       copied = strlen(localbuf) - off;
+
+   if (copied > size)
+       copied = size;
+   if (copied)
+       memcpy(buf, localbuf + off, copied);
+   return copied;
+}

 static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
```

And add the new function into the `circlefs_ops` structure:

circlefs.c

```
--- circlefs.c  2023-10-28 10:42:56.801020735 -0500
+++ circlefs.c  2023-10-28 10:43:13.936210060 -0500
@@ -85,6 +85,7 @@
 static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
    .getattr        = circlefs_getattr,
+   .read       = circlefs_read,
 };

 int main(int argc, char *argv[]) {
```

Compile and run:

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3
$ ./circlefs mnt

$ cat mnt/radius
0

$ echo 1 > mnt/radius
bash: echo: write error: Function not implemented

$ fusermount -u mnt
```

Okay, so now we can read, but still can't write until we add a write function,

But I want a purple pony!

We need to be able to change the radius, so let's allow writing to the `radius`:

circlefs.c

```
--- circlefs.c  2023-10-28 10:50:23.303962588 -0500
+++ circlefs.c  2023-10-28 11:03:50.522904649 -0500
@@ -81,6 +81,33 @@
        memcpy(buf, localbuf + off, copied);
    return copied;
 }
+static int circlefs_write(const char *path, const char *buf,
+   size_t size, off_t off, struct fuse_file_info *ffi) {
+
+   char *endptr = NULL;
+   long double val;
+
+   if (off != 0) // makes no sense to write anywhere but 0
+       return -EINVAL;
+
+   errno = 0;
+   val = strtold(buf, &endptr);
+
+       if (errno == ERANGE || // out-of range
+           endptr == buf || // empty write or bad value
+           val == HUGE_VAL || val == -HUGE_VAL)
+               return -EINVAL;
+
+   // paths start with '/', so advance to fix that
+   if (*path == '/')
+       path++;
+
+   if (!strcmp(path, "radius"))
+       circlefs_data.radius = val;
+   else // what file is this?
+       return -EBADF;
+
+   return size;
+}

 static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
```

And add the new function into the `circlefs_ops` structure:

circlefs.c

```
--- circlefs.c  2023-10-28 11:03:50.522904649 -0500
+++ circlefs.c  2023-10-28 11:05:06.739745844 -0500
@@ -114,6 +114,7 @@
    .readdir        = circlefs_readdir,
    .getattr        = circlefs_getattr,
    .read       = circlefs_read,
+   .write      = circlefs_write,
 };

 int main(int argc, char *argv[]) {
```

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3
$ ./circlefs mnt

$ cat mnt/radius
0

$ echo 1 > mnt/radius
$ cat mnt/radius
1

$ fusermount -u mnt
```

Okay, now we're talking!

If only we had a wheelbarrow

Now that we can read/write our `radius`, how about listing, reading, and writing the other files: `pi`, `diameter`, `circumference`, and `area`?

We'll add the files to our list of directory entries:

circlefs.c - changes to circlefs_dirents

```
--- circlefs.c  2023-10-28 11:08:27.991967040 -0500
+++ circlefs.c  2023-10-28 11:16:01.636973864 -0500
@@ -8,6 +8,10 @@
     { .name = ".",      .st = { .st_mode = S_IFDIR | 0555, .st_ino = 1, } },
     { .name = "..",     .st = { .st_mode = S_IFDIR | 0555, .st_ino = 2, } },
     { .name = "radius", .st = { .st_mode = S_IFREG | 0644, .st_ino = 3, } },
+    { .name = "pi",     .st = { .st_mode = S_IFREG | 0444, .st_ino = 4, } },
+    { .name = "diameter",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 6, } },
+    { .name = "circumference", .st = { .st_mode = S_IFREG | 0644, .st_ino = 7, } },
+    { .name = "area",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 8, } },
 };
 #define DIRENT_COUNT (ARRAY_SIZE(circlefs_dirents))
```

And add code to read and write them into the `circlefs_read` and `circlefs_write` functions:

circlefs.c - changes to circlefs_read

```
--- circlefs.c  2023-10-28 11:16:01.636973864 -0500
+++ circlefs.c  2023-10-28 11:20:47.068124800 -0500
@@ -61,6 +61,14 @@

     if (!strcmp(path, "radius"))
         val = circlefs_data.radius;
+    else if (!strcmp(path, "pi"))
+        val = M_PIf128;
+    else if (!strcmp(path, "diameter"))
+        val = 2.0 * circlefs_data.radius;
+    else if (!strcmp(path, "circumference"))
+        val = 2.0 * M_PIf128 * circlefs_data.radius;
+    else if (!strcmp(path, "area"))
+        val = M_PIf128 * circlefs_data.radius * circlefs_data.radius;

     if (isnan(val)) // not a real file
         return -EBADF;
```

circlefs.c - changes to circlefs_write

```
--- circlefs.c  2023-10-28 11:20:47.068124800 -0500
+++ circlefs.c  2023-10-28 11:25:44.751415301 -0500
@@ -116,6 +116,13 @@

     if (!strcmp(path, "radius"))
         circlefs_data.radius = val;
+    else if (!strcmp(path, "diameter"))
+        circlefs_data.radius = val / 2.0;
+    else if (!strcmp(path, "circumference"))
+        circlefs_data.radius = val / (2.0 * M_PIf128);
+    else if (!strcmp(path, "area"))
+        circlefs_data.radius = powl(val / M_PIf128, 0.5);
+
     else // what file is this?
         return -EBADF;
```

Compile and run (need to include the math lib now):

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3 -lm
$ ./circlefs mnt

$ for f in {radius,diameter,circumference,area,pi} ; do echo "$f - $(cat mnt/$f)" ;
done
radius - 0
diameter - 0
circumference - 0
area - 0
pi - 3.14159265358979323851280895940618620 4

$ echo 2 > mnt/diameter

$ for f in {radius,diameter,circumference,area,pi} ; do echo "$f - $(cat mnt/$f)" ;
done
radius - 1
diameter - 2
circumference - 6.28318530717958647702561791881237240 9
area - 3.14159265358979323851280895940618620 4
pi - 3.14159265358979323851280895940618620 4

$ fusermount -u mnt
```

But I don’t want to be the ugly stepsister!

You may have noticed that the dates for all the files in our filesystem are set to the epoch, 'zero' in unix time:

```
$ TZ=UTC ls -aln mnt
total 298
dr-xr-xr-x. 0 1000 1000 4096 Jan  1  1970 .
drwxrwxr-x. 3 1000 1000 4096 Oct 24 15:35 ..
-rw-r--r--. 0 1000 1000 4096 Jan  1  1970 area
-rw-r--r--. 0 1000 1000 4096 Jan  1  1970 circumference
-rw-r--r--. 0 1000 1000 4096 Jan  1  1970 diameter
-r--r--r--. 0 1000 1000 4096 Jan  1  1970 pi
-rw-r--r--. 0 1000 1000 4096 Jan  1  1970 radius

$ TZ=UTC stat mnt | grep -E 'Access|Modify|Change|Birth'
Access: (0555/dr-xr-xr-x)  Uid: ( 1000/sorenson)   Gid: ( 1000/sorenson)
Access: 1970-01-01 00:00:00.000000000 +0000
Modify: 1970-01-01 00:00:00.000000000 +0000
Change: 1970-01-01 00:00:00.000000000 +0000
 Birth: -
```

We don't know the actual date that `pi` was first determined, and probably can't represent that date as a timestamp anyway, but we can do something useful with the rest of these. Let's set all the timestamps to the mount time, then update the modify time whenever we change a value. We can also update the access times whenever we read any of the files.

Another ugly thing in our code is that we call `getuid()` and `getgid()` for every directory entry, every time:

```
        ent->st.st_uid = getuid();
        ent->st.st_gid = getgid();
```

Since that's pretty inefficient and unnecessary (it's not going to change), let's just call those functions once while mounting, and then set them to our stored values.

This takes us all the way back to the top of our `circlefs.h` file, where we'll add a few timestamps and uid/gid to our circlefs_data:

circlefs.h - updates to circlefs_data

```
--- circlefs.h  2023-10-28 10:12:46.544019437 -0500
+++ circlefs.h  2023-10-28 11:32:56.387186470 -0500
@@ -19,7 +19,13 @@

 struct circlefs_data {
    long double radius;
+   struct timespec mount_time;
+   struct timespec modify_time;
+   struct timespec access_time;
+   uid_t uid;
+   gid_t gid;
 };
+extern struct circlefs_data circlefs_data;

 struct circlefs_dirent {
    char *name;     // entry name
```

circlefs.h - updates to fill_statbuf

```
--- circlefs.h  2023-10-28 11:32:56.387186470 -0500
+++ circlefs.h  2023-10-28 11:36:32.822578880 -0500
@@ -42,12 +42,16 @@

 // helper function to fill a directory entry's 'struct stat'
 static void fill_statbuf(struct circlefs_dirent *ent) {
-   ent->st.st_uid = getuid();
-   ent->st.st_gid = getgid();
+   ent->st.st_uid = circlefs_data.uid;
+   ent->st.st_gid = circlefs_data.gid;
    ent->st.st_size = FILE_SIZE;
    ent->st.st_blksize = BLOCK_SIZE;
    ent->st.st_blocks = NUM_BLOCKS(FILE_SIZE, BLOCK_SIZE),
    ent->st.st_dev = makedev(DEVICE_MAJOR, DEVICE_MINOR);
+   ent->st.st_ctim = circlefs_data.mount_time;
+   ent->st.st_mtim = (!strcmp(ent->name, "pi")) ? circlefs_data.mount_time :
+       circlefs_data.modify_time;
+   ent->st.st_atim = circlefs_data.access_time;
 }

 #endif
```

and update the 'circlefs_read', 'circlefs_write', and 'main' functions:

circlefs.c - updates to circlefs_read

```
--- circlefs.c  2023-10-28 11:25:44.751415301 -0500
+++ circlefs.c  2023-10-28 12:08:50.001035129 -0500
@@ -91,6 +91,7 @@
        copied = size;
    if (copied)
        memcpy(buf, localbuf + off, copied);
+   clock_gettime(CLOCK_REALTIME, &circlefs_data.access_time);
    return copied;
 }
 static int circlefs_write(const char *path, const char *buf,
```

circlefs.c - updates to circlefs_write

```
--- circlefs.c  2023-10-28 12:08:50.001035129 -0500
+++ circlefs.c  2023-10-28 12:09:38.349570648 -0500
@@ -127,6 +127,7 @@
    else // what file is this?
        return -EBADF;

+   clock_gettime(CLOCK_REALTIME, &circlefs_data.modify_time);
    return size;
 }
```

circlefs.c - updates to main

```
--- circlefs.c  2023-10-28 12:09:38.349570648 -0500
+++ circlefs.c  2023-10-28 12:10:34.127188453 -0500
@@ -139,6 +139,11 @@
 };

 int main(int argc, char *argv[]) {
+   circlefs_data.uid = getuid();
+   circlefs_data.gid = getgid();
+   clock_gettime(CLOCK_REALTIME, &circlefs_data.mount_time);
+   circlefs_data.modify_time = circlefs_data.access_time = circlefs_data.mount_time;
+
    return fuse_main(argc, argv, &circlefs_ops, NULL);
 }
```

Now, how do the timestamps look?

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3 -lm
$ ./circlefs mnt

$ TZ=UTC ls -aln mnt
total 298
dr-xr-xr-x. 0 1000 1000 4096 Oct 24 16:34 .
drwxrwxr-x. 3 1000 1000 4096 Oct 24 16:34 ..
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:34 area
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:34 circumference
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:34 diameter
-r--r--r--. 0 1000 1000 4096 Oct 24 16:34 pi
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:34 radius

$ echo 10 > mnt/radius
$ cat mnt/diameter
20

$ TZ=UTC stat mnt/diameter | grep -E 'Access|Modify|Change|Birth'
Access: (0644/-rw-r--r--)  Uid: ( 1000/sorenson)   Gid: ( 1000/sorenson)
Access: 2023-10-24 16:38:08.199070682 +0000
Modify: 2023-10-24 16:38:00.302121398 +0000
Change: 2023-10-24 16:34:05.665096412 +0000
 Birth: -

$ fusermount -u mnt
```

That wasn't enough? You want more?

Just for kicks, let's add a symlink for `π`, pointing to `pi`. For this, we'll add the entry to the directory listing, and implement a `readlink` function:

circlefs.c - updates to circlefs_dirents

```
--- circlefs.c  2023-10-28 12:10:34.127188453 -0500
+++ circlefs.c  2023-10-28 12:17:46.769982004 -0500
@@ -9,6 +9,7 @@
    { .name = "..",     .st = { .st_mode = S_IFDIR | 0555, .st_ino = 2, } },
    { .name = "radius", .st = { .st_mode = S_IFREG | 0644, .st_ino = 3, } },
    { .name = "pi",     .st = { .st_mode = S_IFREG | 0444, .st_ino = 4, } },
+   { .name = "π",      .st = { .st_mode = S_IFLNK | 0777, .st_ino = 5, } },
    { .name = "diameter",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 6, } },
    { .name = "circumference", .st = { .st_mode = S_IFREG | 0644, .st_ino = 7, } },
    { .name = "area",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 8, } },
```

circlefs.c - add circlefs_readlink

```
--- circlefs.c  2023-10-28 12:17:46.769982004 -0500
+++ circlefs.c  2023-10-28 12:21:32.309481105 -0500
@@ -131,6 +131,24 @@
    clock_gettime(CLOCK_REALTIME, &circlefs_data.modify_time);
    return size;
 }
+static int circlefs_readlink(const char *path, char *buf, size_t size) {
+   int i;
+
+   // paths start with '/', so advance to fix that
+   if (*path == '/')
+       path++;
+
+   if (!strcmp(path, "π")) {
+       strncpy(buf, "pi", size - 1);
+       if (strlen(buf) < 2)
+           return -ENAMETOOLONG;
+       return 0;
+   }
+   for (i = 0 ; i < ARRAY_SIZE(circlefs_dirents) ; i++)
+       if (! strcmp(path, circlefs_dirents[i].name))
+           return -EINVAL; // not a symlink
+   return -ENOENT; // no such file
+}

 static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
```

circlefs.c - updates to circlefs_ops

```
--- circlefs.c  2023-10-28 12:21:32.309481105 -0500
+++ circlefs.c  2023-10-28 12:23:18.714660133 -0500
@@ -155,6 +155,7 @@
    .getattr        = circlefs_getattr,
    .read       = circlefs_read,
    .write      = circlefs_write,
+   .readlink   = circlefs_readlink,
 };

 int main(int argc, char *argv[]) {
```

compile and test

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3 -lm
$ ./circlefs mnt

$ TZ=UTC ls -aln mnt
total 347
dr-xr-xr-x. 0 1000 1000 4096 Oct 24 16:56 .
drwxrwxr-x. 3 1000 1000 4096 Oct 24 16:55 ..
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:56 area
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:56 circumference
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:56 diameter
-r--r--r--. 0 1000 1000 4096 Oct 24 16:56 pi
-rw-r--r--. 0 1000 1000 4096 Oct 24 16:56 radius
lrwxrwxrwx. 0 1000 1000 4096 Oct 24 16:56 π -> pi

$ fusermount -u mnt
```

What I wouldn’t give for a holocaust cloak!

Of course, no filesystem would be complete without `df` causing panic over the filesystem being full, so let’s make it happen. We’ll make our filesystem always full:

circlefs.c - add circlefs_statfs

```
--- circlefs.c  2023-10-28 12:23:18.714660133 -0500
+++ circlefs.c  2023-10-28 12:27:23.478372247 -0500
@@ -149,6 +149,24 @@
            return -EINVAL; // not a symlink
    return -ENOENT; // no such file
 }
+// 'path' is really irrelevant... our filesystem doesn't vary based on the path
+static int circlefs_statfs(const char *path, struct statvfs *stbuf) {
+   struct statvfs stvfs = {
+       .f_bsize = BLOCK_SIZE,
+       .f_bfree = 0,
+       .f_bavail = 0,
+       .f_blocks = DIRENT_COUNT,
+       .f_files = DIRENT_COUNT,
+       .f_ffree = 0,
+       .f_favail = 0,
+       .f_flag = ST_NODEV | ST_NOEXEC | ST_NOSUID,
+       .f_frsize = 0,
+       .f_namemax = 255,
+   };
+   memcpy(stbuf, &stvfs, sizeof(stvfs));
+
+   return 0;
+}

 static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
```

circlefs.c - update circlefs_ops

```
--- circlefs.c  2023-10-28 12:27:23.478372247 -0500
+++ circlefs.c  2023-10-28 12:30:46.587623627 -0500
@@ -174,6 +174,7 @@
    .read       = circlefs_read,
    .write      = circlefs_write,
    .readlink   = circlefs_readlink,
+   .statfs     = circlefs_statfs,
 };

 int main(int argc, char *argv[]) {
```

```
$ gcc -Wall circlefs.c -o circlefs -g -lfuse3 -lm
$ ./circlefs mnt

$ stat -f mnt
  File: "mnt"
    ID: 0        Namelen: 255     Type: fuseblk
Block size: 42         Fundamental block size: 42
Blocks: Total: 8          Free: 0          Available: 0
Inodes: Total: 8          Free: 0

$ df mnt
Filesystem     1K-blocks  Used Available Use% Mounted on
circlefs               1     1         0 100%
/home/sorenson/projects/misc/training/circlefs-v2/mnt

$ df -i mnt
Filesystem     Inodes IUsed IFree IUse% Mounted on
circlefs            8     8     0  100%
/home/sorenson/projects/misc/training/circlefs-v2/mnt

$ fusermount -u mnt
```

Excellent!

I did all my chores! Can I go out and play now?

Our filesystem is 'complete'. Give it a test, and see if we're missing anything.

Think you're a wise-guy, eh? Let's see you do this, then!

some possible exercises for the reader:

- make the filesystem size dependent on the radius (or area?) of the circle
- add:
  - surface area (of a sphere): 4 * pi * radius^2
  - volume (of a sphere): 4/3 * pi * radius^3
  - allow creation of directories, with each directory containing its own radius (i.e. 'mkdir basketball && echo 4.7 > basketball/radius' or 'mkdir golfball && echo 0.84 > golfball/radius')
  - alternately, read a file containing a list of ball types and their sizes, and create read-only directories (or even store the balls in an sqlite db)
  - units file, so reading from `radius` and other files will convert (e.g. 'echo cm > units'), and/or

accept units when writing to the files (e.g. 'echo 21.35mm > golfball/radius')

- implement some other silly filesystem (squarefs, rectanglefs, ?)
- implement multiple silly filesystems, and have the shape type alterable via mount option (options are processed in main() before calling fuse_main)
- implement multiple silly filesystems, and have the shape type alterable on-the-fly via a read-write `shape` file
- create some less-silly filesystem
- re-implement circlefs inside a real kernel module

The source

circlefs.h

```
#ifndef __CIRCLEFS_H__
#define __CIRCLEFS_H__
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <inttypes.h>
#include <math.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>

#define FUSE_USE_VERSION 31

#include <fuse3/fuse.h>

struct circlefs_data {
    long double radius;
    struct timespec mount_time;
    struct timespec modify_time;
    struct timespec access_time;
    uid_t uid;
    gid_t gid;
};
extern struct circlefs_data circlefs_data;

struct circlefs_dirent {
    char *name;     // entry name
    struct stat st; // permissions, inode number, etc.
};
#define ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
```

```
#define BLOCK_SIZE      42
#define FILE_SIZE       4096
#define DEVICE_MAJOR    42
#define DEVICE_MINOR    42

#define NUM_BLOCKS(size, bsize) ( (size + bsize - 1) / bsize )

// helper function to fill a directory entry's 'struct stat'
static void fill_statbuf(struct circlefs_dirent *ent) {
    ent->st.st_uid = circlefs_data.uid;
    ent->st.st_gid = circlefs_data.gid;
    ent->st.st_size = FILE_SIZE;
    ent->st.st_blksize = BLOCK_SIZE;
    ent->st.st_blocks = NUM_BLOCKS(FILE_SIZE, BLOCK_SIZE),
    ent->st.st_dev = makedev(DEVICE_MAJOR, DEVICE_MINOR);
    ent->st.st_ctim = circlefs_data.mount_time;
    ent->st.st_mtim = (!strcmp(ent->name, "pi")) ? circlefs_data.mount_time :
        circlefs_data.modify_time;
    ent->st.st_atim = circlefs_data.access_time;
}

#endif
```

circlefs.c

```
#include "circlefs.h"

struct circlefs_data circlefs_data = {
    .radius = 0,
};

struct circlefs_dirent circlefs_dirents[] = {
    { .name = ".",      .st = { .st_mode = S_IFDIR | 0555, .st_ino = 1, } },
    { .name = "..",     .st = { .st_mode = S_IFDIR | 0555, .st_ino = 2, } },
    { .name = "radius", .st = { .st_mode = S_IFREG | 0644, .st_ino = 3, } },
    { .name = "pi",     .st = { .st_mode = S_IFREG | 0444, .st_ino = 4, } },
    { .name = "π",      .st = { .st_mode = S_IFLNK | 0777, .st_ino = 5, } },
    { .name = "diameter",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 6, } },
    { .name = "circumference", .st = { .st_mode = S_IFREG | 0644, .st_ino = 7, } },
    { .name = "area",   .st = { .st_mode = S_IFREG | 0644, .st_ino = 8, } },
    // sphere attributes
    { .name = "surface_area", .st = { .st_mode = S_IFREG | 0644, .st_ino = 9, } },
    { .name = "volume", .st = { .st_mode = S_IFREG | 0644, .st_ino = 10, } },
};
#define DIRENT_COUNT (ARRAY_SIZE(circlefs_dirents))

// path doesn't matter, since we don't have any subdirs
static int circlefs_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
        off_t start_offset, struct fuse_file_info *ffi,
        enum fuse_readdir_flags readdir_flags) {
```

```
    int offset;

    // The 'start_offset' is used in case our directory listing needs to call
    //     into readdir() more than once.  However, our filesystem is very
    //     small, so we'll probably always start at the beginning of the dir.

    for (offset = start_offset ; offset < DIRENT_COUNT ; offset++) {
        fill_statbuf(&circlefs_dirents[offset]);
        filler(buf, circlefs_dirents[offset].name,
            &circlefs_dirents[offset].st, 0, FUSE_FILL_DIR_PLUS);
    }
    return 0;
}
static int circlefs_getattr(const char *path, struct stat *st,
        struct fuse_file_info *ffi) {
    int i;
    if (!strcmp("/", path))
        path = ".";
    else if (*path == '/') // paths start with '/', so advance to fix that
        path++;
    for (i = 0 ; i < DIRENT_COUNT ; i++)
        if (!strcmp(path, circlefs_dirents[i].name)) {
            fill_statbuf(&circlefs_dirents[i]);
            memcpy(st, &circlefs_dirents[i].st, sizeof(struct stat));
            return 0;
        }
    return -ENOENT;
}
static int circlefs_read(const char *path, char *buf,
    size_t size, off_t off, struct fuse_file_info *ffi) {

    char localbuf[128] = { 0 };
    long double val = NAN;
    int copied;

    // paths start with '/', so advance to fix that
    if (*path == '/')
        path++;

    if (!strcmp(path, "radius"))
        val = circlefs_data.radius;
    else if (!strcmp(path, "pi"))
        val = M_PIf128;
    else if (!strcmp(path, "diameter"))
        val = 2.0 * circlefs_data.radius;
    else if (!strcmp(path, "circumference"))
        val = 2.0 * M_PIf128 * circlefs_data.radius;
    else if (!strcmp(path, "area"))
        val = M_PIf128 * circlefs_data.radius * circlefs_data.radius;
    else if (!strcmp(path, "surface_area"))
        val = 4.0 * M_PIf128 * powl(circlefs_data.radius, 2.0);
```

```
    else if (!strcmp(path, "volume"))
        val = (4.0 / 3.0) * M_PIf128 * powl(circlefs_data.radius, 3.0);

    if (isnan(val)) // not a real file
        return -EBADF;

    // try to check whether our value can be expressed as an integer
    uint64_t int_val = val;
    long double tmp = val - int_val;

    if (tmp > 0)
        snprintf(localbuf, sizeof(localbuf) - 1, "%.36Lf", val);
    else
        snprintf(localbuf, sizeof(localbuf) - 1, "%" PRIu64, int_val);

    if (off > strlen(localbuf))
        copied = 0;
    else
        copied = strlen(localbuf) - off;

    if (copied > size)
        copied = size;
    if (copied)
        memcpy(buf, localbuf + off, copied);
    clock_gettime(CLOCK_REALTIME, &circlefs_data.access_time);
    return copied;
}
static int circlefs_write(const char *path, const char *buf,
    size_t size, off_t off, struct fuse_file_info *ffi) {

    char *endptr = NULL;
    long double val;

    if (off != 0) // makes no sense to write anywhere but 0
        return -EINVAL;

    errno = 0;
    val = strtold(buf, &endptr);

        if (errno == ERANGE || // out-of range
            endptr == buf || // empty write or bad value
            val == HUGE_VAL || val == -HUGE_VAL)
                return -EINVAL;

    // paths start with '/', so advance to fix that
    if (*path == '/')
        path++;

    if (!strcmp(path, "radius"))
        circlefs_data.radius = val;
    else if (!strcmp(path, "diameter"))
```

```
        circlefs_data.radius = val / 2.0;
    else if (!strcmp(path, "circumference"))
        circlefs_data.radius = val / (2.0 * M_PIf128);
    else if (!strcmp(path, "area"))
        circlefs_data.radius = powl(val / M_PIf128, 0.5);
    else if (!strcmp(path, "surface_area"))
        circlefs_data.radius = sqrtl(val / (M_PIf128 * 4));
    else if (!strcmp(path, "volume"))
        circlefs_data.radius = cbrtl((val * 3.0) / (M_PIf128 * 4.0));
    else // what file is this?
        return -EBADF;

    clock_gettime(CLOCK_REALTIME, &circlefs_data.modify_time);
    return size;
}
static int circlefs_readlink(const char *path, char *buf, size_t size) {
    int i;

    // paths start with '/', so advance to fix that
    if (*path == '/')
        path++;

    if (!strcmp(path, "π")) {
        strncpy(buf, "pi", size - 1);
        if (strlen(buf) < 2)
            return -ENAMETOOLONG;
        return 0;
    }
    for (i = 0 ; i < ARRAY_SIZE(circlefs_dirents) ; i++)
        if (! strcmp(path, circlefs_dirents[i].name))
            return -EINVAL; // not a symlink
    return -ENOENT; // no such file
}
// 'path' is really irrelevant... our filesystem doesn't vary based on the path
static int circlefs_statfs(const char *path, struct statvfs *stbuf) {
    struct statvfs stvfs = {
        .f_bsize = BLOCK_SIZE,
        .f_bfree = 0,
        .f_bavail = 0,
        .f_blocks = DIRENT_COUNT,
        .f_files = DIRENT_COUNT,
        .f_ffree = 0,
        .f_favail = 0,
        .f_flag = ST_NODEV | ST_NOEXEC | ST_NOSUID,
        .f_frsize = 0,
        .f_namemax = 255,
    };
    memcpy(stbuf, &stvfs, sizeof(stvfs));

    return 0;
}
```

```
static const struct fuse_operations circlefs_ops = {
    .readdir        = circlefs_readdir,
    .getattr        = circlefs_getattr,
    .read       = circlefs_read,
    .write      = circlefs_write,
    .readlink   = circlefs_readlink,
    .statfs     = circlefs_statfs,
};

int main(int argc, char *argv[]) {
    circlefs_data.uid = getuid();
    circlefs_data.gid = getgid();
    clock_gettime(CLOCK_REALTIME, &circlefs_data.mount_time);
    circlefs_data.modify_time = circlefs_data.access_time = circlefs_data.mount_time;

    return fuse_main(argc, argv, &circlefs_ops, NULL);
}
```