Hello World

From fuse

Contents

Hello World example

This is a Hello World example program with extensive comments.

Definitions and libraries

First thing to do, is to declare what version of the interface you want to use, the default version as of today (24th Jun 2007) is 2.1, we will use 2.6, the newest one.

```
#define FUSE_USE_VERSION 26
```

To be able to program a FUSE filesystem you need to include a series of headers:

```
#include <stdlib.h>
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
```

Apart from `fuse.h`, `fcntl.h` and `errno.h` you should know them, if it is not the case, i'd suggest going back to your C book or reading system documentation.

- `fuse.h` contains definitions for basic functions needed to implement a filesystem
- `errno.h` contains definitions of error numbers
- `fcntl.h` contains definitions of file options (the ones used with fcntl() and open())

Global variables

The next thing to do in our example will be the the definition of global variables containing path and data of a `/hello` file:

```
static const char *hello_str = "Hello World!\n";
static const char *hello_path = "/hello";
```

getattr() function

This function returns metadata concerning a file specified by `path` in a special stat structure. It has to be declared `static`, as all functions passed in the fuse_operations structure to fuse_main(), to work properly.

```
static int hello_getattr(const char *path, struct stat *stbuf)
{
    int res = 0; /* temporary result */
```

Reset memory for the `stat` structure (set it's contents to 0's)

```
  memset(stbuf, 0, sizeof(struct stat));
```

Now we have to check which file attributes we have to return, our filesystem will have only two files (one regular file and one directory) so we can use simple if here.

```
  if(strcmp(path, "/") == 0) {
```

st_mode contains file attributes, S_IFDIR is equal to 0040000 and it marks a file as a directory, then it is binary added to 0755 value, being normal file permissions (http://en.wikipedia.org/wiki/File_permissions#Notation_of_traditional_Unix_permissions) written in octal format.

```
    stbuf->st_mode = S_IFDIR | 0755;
```

st_nlink contains number of hardlinks, this file being a directory and not having another directories inside will have 2 links

```
    stbuf->st_nlink = 2;
  }
```

if the user requests permissions for `/hello` file

```
  else if(strcmp(path, hello_path) == 0) {
```

set the file attributes to a regular file (S_IFREG) and give it 0444 permissions

```
    stbuf->st_mode = S_IFREG | 0444;
```

the `/hello` file is one of two files in this filesystem so it will have only one hard link pointing to it

```
    stbuf->st_nlink = 1;
```

file size is equal to the size of the `hello_str`

```
    stbuf->st_size = strlen(hello_str);
  }
```

if the user asks for permissions for any other file, return negated ENOENT value. ENOENT is declared in the `errno.h` and means that "A component of the path `path` does not exist, or the `path` is an empty string.", it is negated to identify it as a FUSE error.

```
  else
     res = -ENOENT;
  return res;
```

}

readdir() function

Next very important function is used to read directory contents. Because our implementation is very simple, we will be intrested only in `path`, `buf` and `filler` arguments passed to it. `Path` is the path to the directory from which we will have to read our contents, `buf` will hold them, and `filler` is a fuse_fill_dir_t function which we will use to add contents to directory. `Offset` and `fi` are not important now.

```
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;
```

check if users is asking for the content's of the root directory, if it's not the case, return "Directory does not exists" error number.

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

Add "." and ".." directory entries.

```
   filler(buf, ".", NULL, 0);
   filler(buf, "..", NULL, 0);
```

Add `hello` file, because it's name should show as "`hello`", not "`/hello`" we have to ommit first character in `hello_path`.

```
   filler(buf, hello_path + 1, NULL, 0);
   return 0;
}
```

open() function

This function checks whatever user is permitted to open the `/hello` file with flags given in the fuse_file_info structure.

```
static int hello_open(const char *path, struct fuse_file_info *fi)
{
```

if user is asking for anything besides `/hello` we tell him, that such file does not exist.

```
   if(strcmp(path, hello_path) != 0)
      return -ENOENT;
```

if the user wants to open the file for anything else than reading only, we tell him that he does not have sufficient permissions.

```
   if((fi->flags & 3) != O_RDONLY )
      return -EACCES;
```

if neither of the above things happend we tell the user that he can open the file

```
   return 0;
```

```
}
```

read() function

Read function is used to feed the user with data from the file. `Path` is the path to the file to read, `buf` is a buffer to which we will copy information from the file, `size` is at the same time the size of the buffer and tells how much of the data we have to read, `offset` is the place from which we have to start reading and `fi` contains additional information about the type of access. Since this procedure will be called only after `open()` (and because this particular `open` allows only read-only operations on `/hello` file) we don't have to pay any attention to it's contents.

```
static int hello_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi)
{
   (void) fi;
```

`len` will contain number of read bytes

```
   size_t len;
```

user cannot open any file besides `/hello`

```
   if(strcmp(path, hello_path) != 0)
     return -ENOENT;
```

set the number of chars to read as length of `hello_str`

```
   len = strlen(hello_str);
```

check if the amount of data to read does not exceed the length of data, if yes, return this smaller size, then copy requested amount of data from hello_str

```
   if (offset < len) {
      if (offset + size > len)
         size = len - offset;
      memcpy(buf, hello_str + offset, size);
   } else
      size = 0;
   return size;
}
```

declaring structure containing implemented fuse operations

Functions you have implemented must be contained in fuse_operations structure

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

main() function

`main()` function can have only one function, fuse_main():

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

Running

To run this example, compile it with `gcc`:

```
gcc -Wall `pkg-config fuse --cflags --libs` hello.c -o hello
```

`pkg-config fuse --cflags --libs` should return something similar to "`-D_FILE_OFFSET_BITS`64 -I/usr/include/fuse -pthread -lfuse -lrt=", you can try to pass those arguments if you have some problems with compilation.

and run it with a single parameter - a mount point:

```
$mkdir tmp
$./hello tmp/
$
```

after running the program you should have a single file in the `tmp/` directory named `hello`, its content should be `Hello World!`:

```
$ ls tmp/
$ ./hello tmp/
$ ls tmp/
hello
$ cat tmp/hello
Hello World!
$
```

Congratulations! You made your's first file system!

Whole example program

```
 #define FUSE_USE_VERSION  26

 #include <fuse.h>
 #include <stdio.h>
 #include <string.h>
 #include <errno.h>
 #include <fcntl.h>

 static const char *hello_str = "Hello World!\n";
 static const char *hello_path = "/hello";

 static int hello_getattr(const char *path, struct stat *stbuf)
 {
     int res = 0;
     memset(stbuf, 0, sizeof(struct stat));
     if(strcmp(path, "/") == 0) {
         stbuf->st_mode = S_IFDIR | 0755;
         stbuf->st_nlink = 2;
     }
     else if(strcmp(path, hello_path) == 0) {
         stbuf->st_mode = S_IFREG | 0444;
         stbuf->st_nlink = 1;
         stbuf->st_size = strlen(hello_str);
     }
     else
         res = -ENOENT;

     return res;
 }

 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;
}

static int hello_open(const char *path, struct fuse_file_info *fi)
{
    if(strcmp(path, hello_path) != 0)
        return -ENOENT;

    if((fi->flags & 3) != O_RDONLY)
        return -EACCES;

    return 0;
}

static int hello_read(const char *path, char *buf, size_t size, off_t offset,
                      struct fuse_file_info *fi)
{
    size_t len;
    (void) fi;
    if(strcmp(path, hello_path) != 0)
        return -ENOENT;

    len = strlen(hello_str);
    if (offset < len) {
        if (offset + size > len)
            size = len - offset;
        memcpy(buf, hello_str + offset, size);
    } else
        size = 0;

    return size;
}

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

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

Retrieved from "http://sourceforge.net/apps/mediawiki/fuse/index.php?title=Hello_World"

- This page was last modified on 14 November 2008, at 12:59.