FUSE
Filesystem in User space

Chian yu Tseng
2012-6-11

Outline

- Introduction
- The FUSE structure
- 如何運作
- Struct fuse_operations
- Example

Introduction(1/2)

- FUSE is a loadable kernel module for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code.
- This is achieved by running file system code in user space while the FUSE module provides only a "**bridge**" to the actual kernel interfaces.

- FUSE is particularly useful for writing **virtual file systems**. Unlike traditional file systems that essentially save data to and retrieve data from disk, virtual filesystems do not actually store data themselves. They act as a view or translation of an existing file system or storage device.

Introduction(2/2)

- The FUSE system was originally part of A Virtual Filesystem (AVFS), but has since split off into its own project on SourceForge.net.
- FUSE is available for Linux, FreeBSD, NetBSD, OpenSolaris, and Mac OS X. It was officially merged into the mainstream Linux kernel tree in kernel version 2.6.14.

Examples(1/2)

- **ExpanDrive**: A commercial filesystem implementing SFTP/FTP/FTPS using FUSE.
- **GlusterFS**: Clustered Distributed Filesystem having capability to scale up to several petabytes.
- **SSHFS**: Provides access to a remote filesystem through SSH.
- **GmailFS**: Filesystem which stores data as mail in Gmail
- **EncFS**: Encrypted virtual filesystem

Examples(2/2)

- **NTFS-3G和Captive NTFS**: allowing access to NTFS filesystem.
- **WikipediaFS** : View and edit Wikipedia articles as if they were real files.
- Sun Microsystems's **Lustre** cluster filesystem
- Sun Microsystems's **ZFS**
- **HDFS**: FUSE bindings exist for the open source Hadoop distributed filesystem.

FUSE Installation

- http://fuse.sourceforge.net/

- ./configure
- make
- make install

FUSE source code

- **./doc**: contains FUSE-related documentation. Ex: **how-fuse-works**
- **./include**: contains the FUSE API headers, which you need to create a file system. The only one you need now is **fuse.h**.
- **./lib**: holds the source code to create the FUSE libraries that you will be linking with your binaries to create a file system.
- **./util**: has the source code for the FUSE utility library.
- **./example:** contains samples for your reference.

FUSE structure

- FUSE kernel module (fuse.ko)
  - inode.c, dev.c, control.c, dir.c, file.c

- LibFUSE module (libfuse.*)
  - helper.c, fuse_kern_chan.c, fuse_mt.c, fuse.c, fuse_lowlevel.c, fuse_loop.c, fuse_loop_mt.c, fuse_session.c

- Mount utility(fusermount)
  - fusermount, mount.fuse.c, mount_util.c, mount.c, mount_bsd.c,

FUSE Library

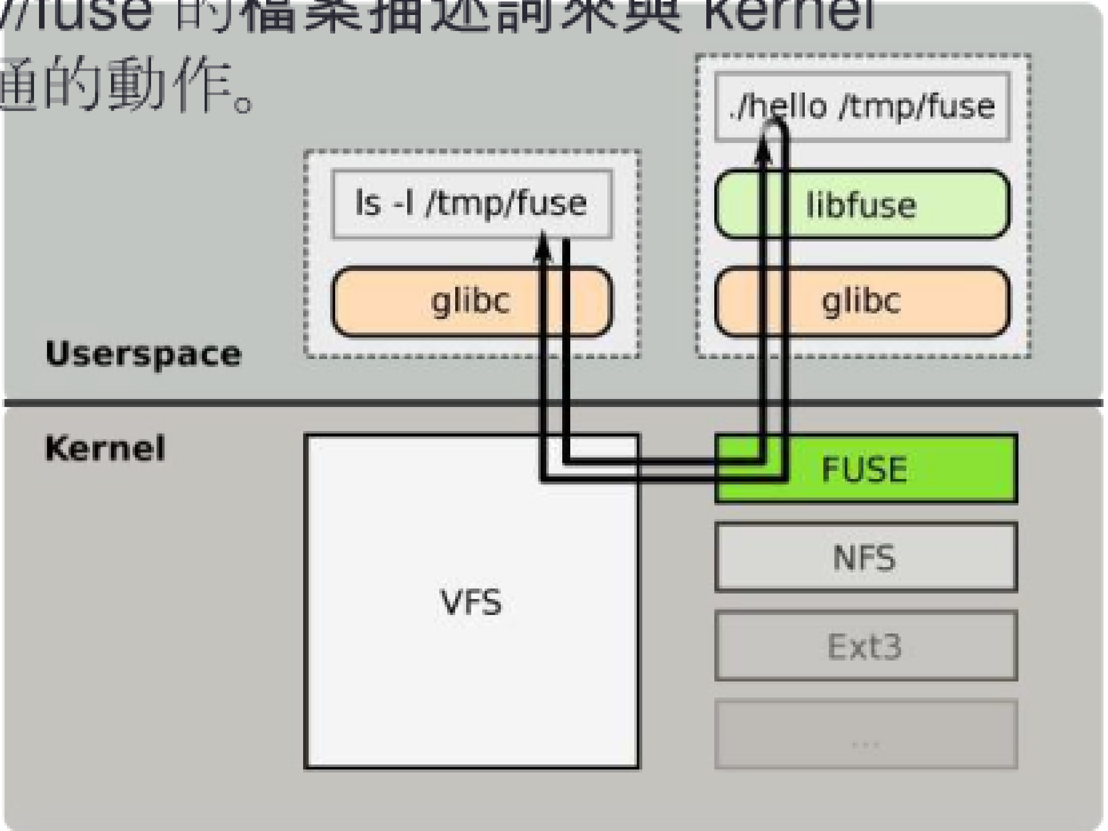
- include/fuse.h → the library interface of FUSE （HighLevel）
- include/fuse_common.h → common
- include/fuse_lowlevel.h → Lowlevel API
- include/fuse_opt.h → option parsing interface of FUSE

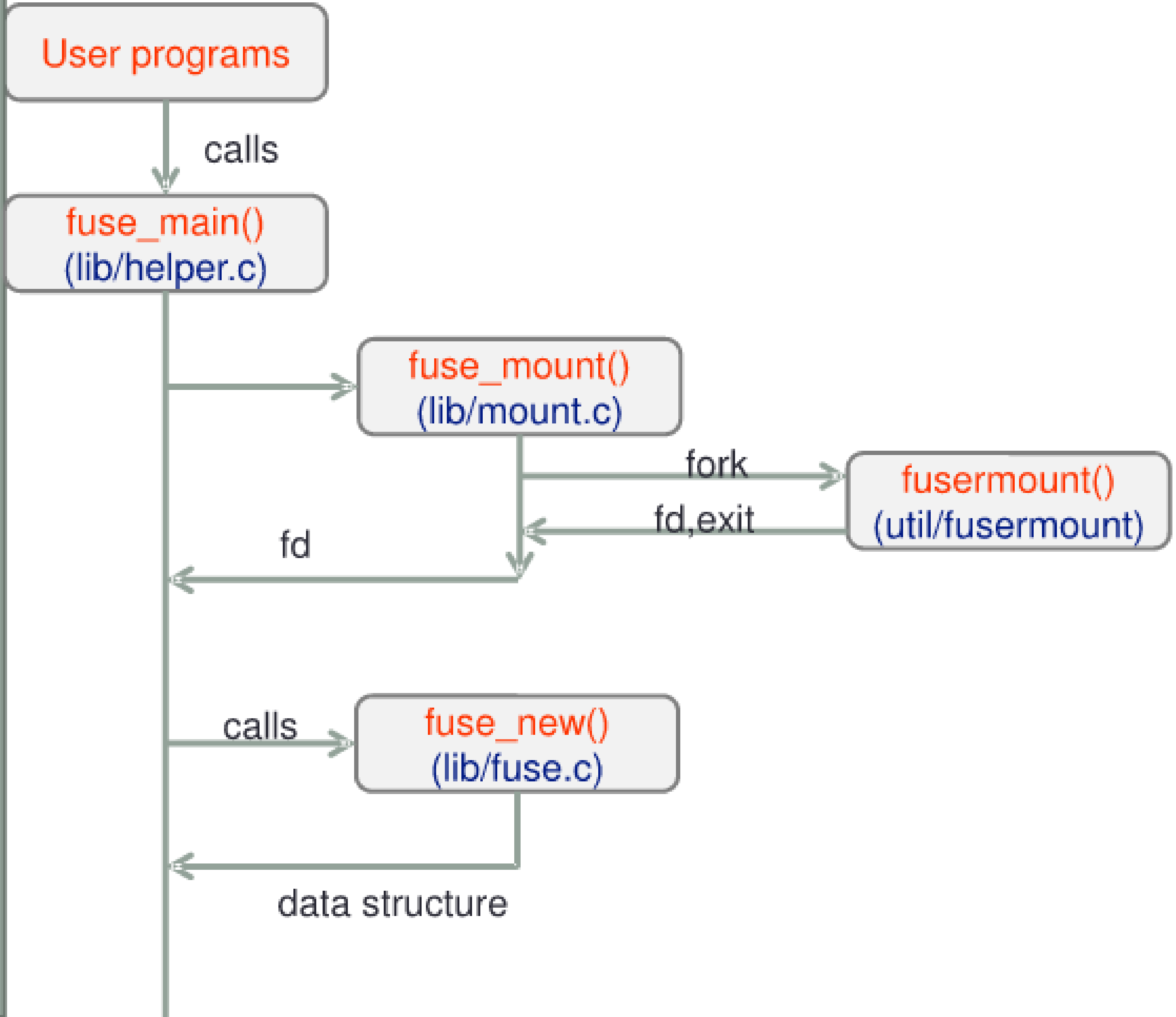
如何運作

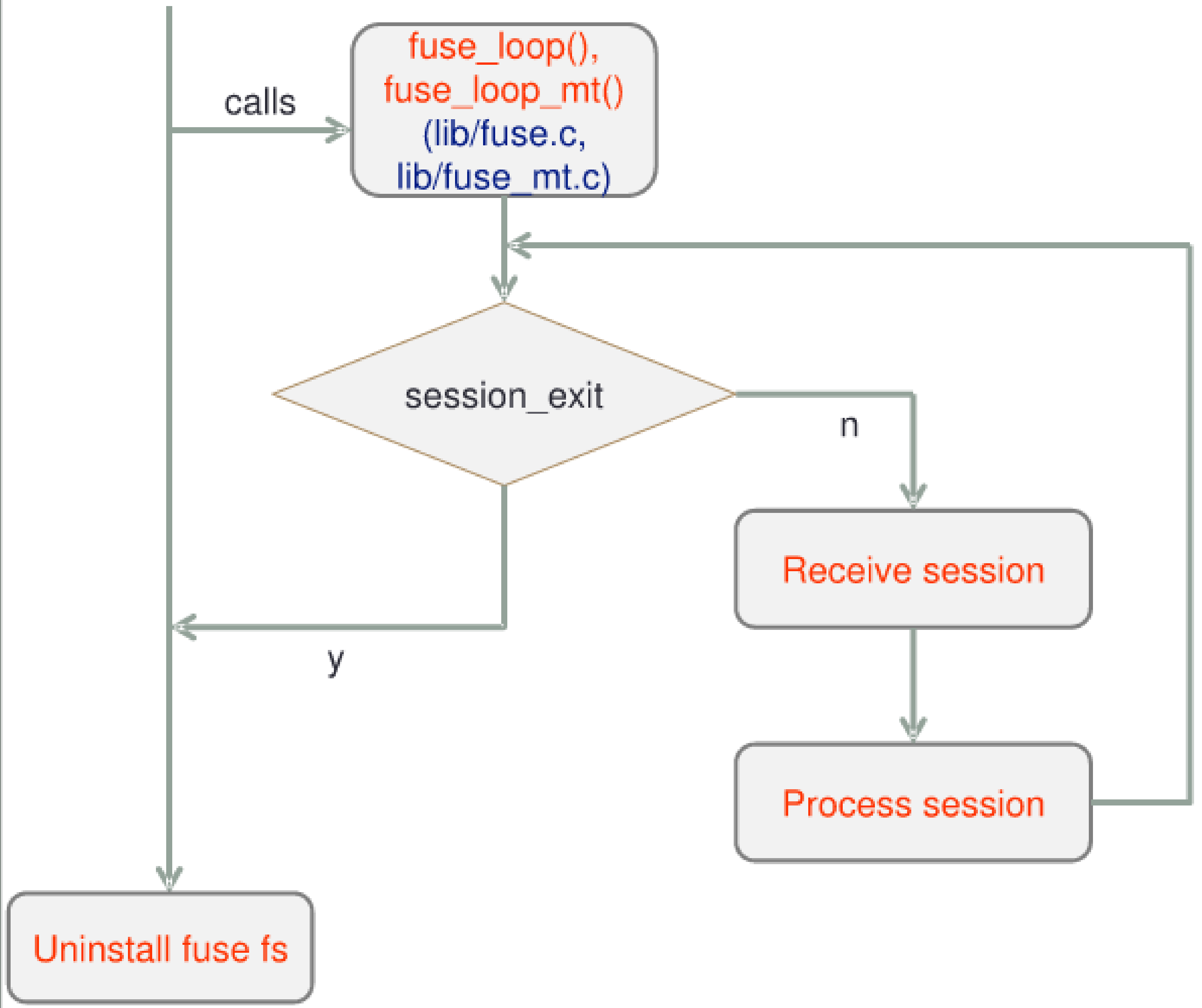
- 在 FUSE daemon 啟動的時候，會先進行掛載的動作，將 /dev/fuse 掛載到指定的目錄底下，並回傳/dev/fuse 的檔案描述詞(file descriptor)，而 FUSE daemon 在預設上會使用 multi-thread 的方式，透過/dev/fuse 的檔案描述詞來接收requests，再根據 requests 的類別來進行處理，最後透過 replies，將結果傳回去。

如何運作

- ls : FUSE daemon會接收到 OPENDIR、READDIR 等 requests，並採用 userspace library(libfuse.*)的函式，讀取 file 目錄的資訊，並將此資訊傳回去，其中 FUSE daemon 就是透過/dev/fuse 的檔案描述詞來與 kernel module(fuse.ko)作溝通的動作。




User programs
calls
fuse_main()
(lib/helper.c)
fuse_mount()
(lib/mount.c)
fork
fusermount()
(util/fusermount)
fd,exit
fd
calls
fuse_new()
(lib/fuse.c)
data structure


calls
fuse_loop(),
fuse_loop_mt()
(lib/fuse.c,
lib/fuse_mt.c)
session_exit
n
y
Receive session
Process session
Uninstall fuse fs

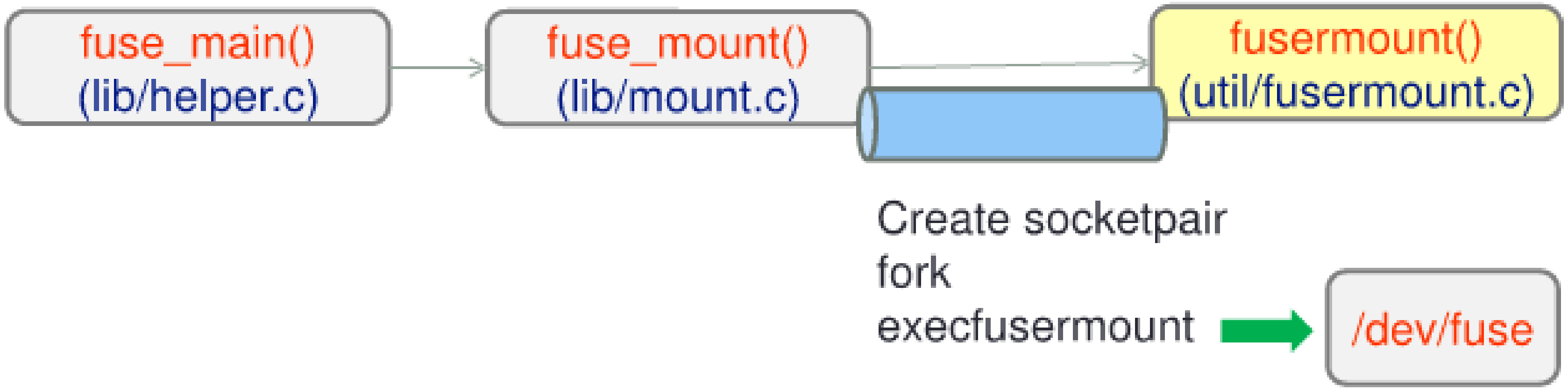
The fuse library(1/5)

- When your user mode program calls fuse_main() (lib/helper.c),fuse_main() parses the arguments passed to your user mode program, then calls fuse_mount() (lib/mount.c).

fuse_main()
(lib/helper.c)
fuse_mount()
(lib/mount.c)

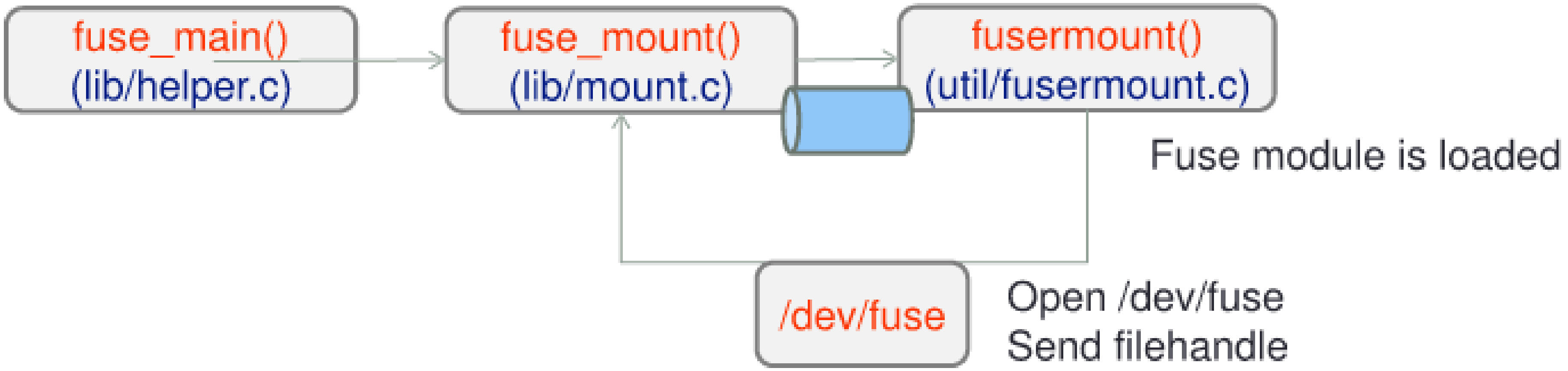
The fuse library(2/5)

- fuse_mount() creates a UNIX domain socket pair, then forks and execsfusermount (util/fusermount.c) passing it one end of the socket in the FUSE_COMMFD_ENV environment variable.



The fuse library(3/5)

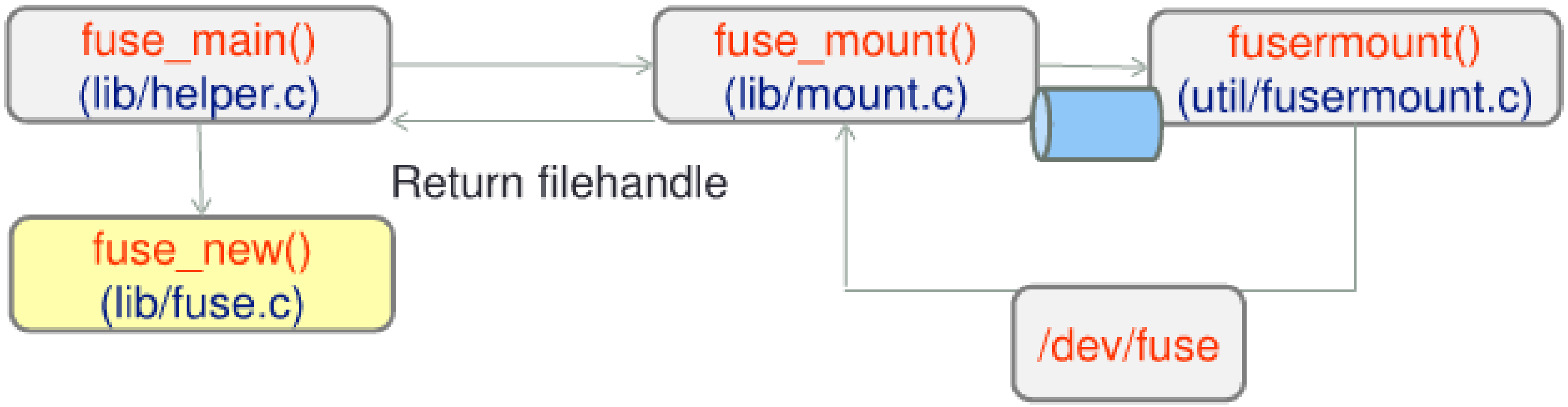
- fusermount (util/fusermount.c) makes sure that the fuse module is loaded. fusermount then open /dev/fuse and send the file handle over a UNIX domain socket back to fuse_mount().



The fuse library(4/5)

- fuse_mount() returns the file handle for /dev/fuse to fuse_main().

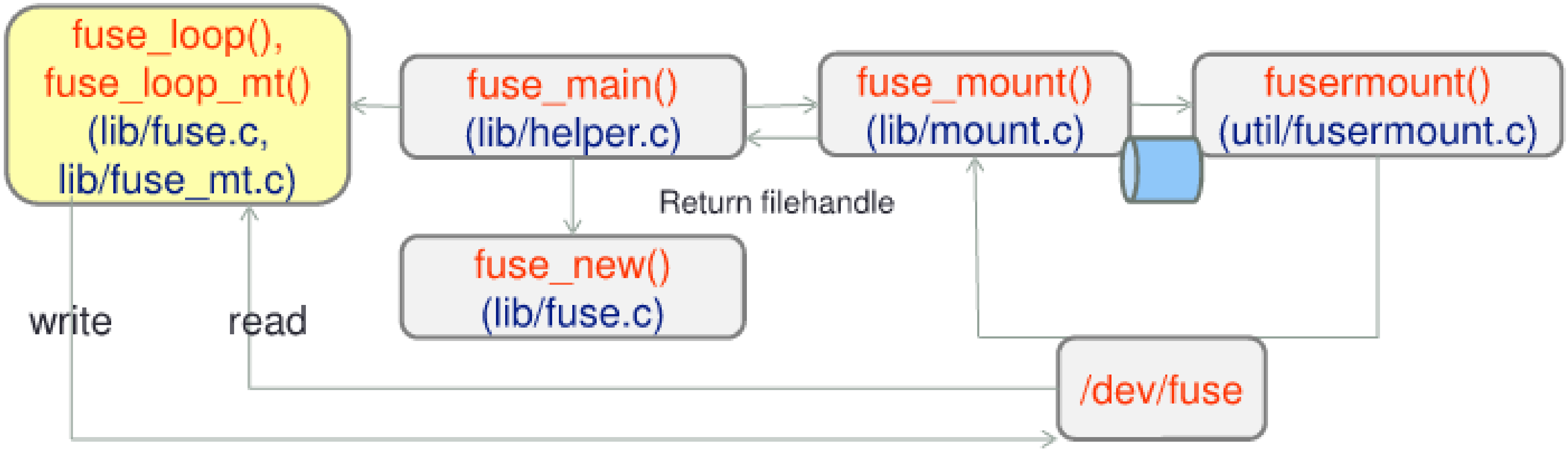
- fuse_main() calls fuse_new() (lib/fuse.c) which allocates the struct fuse data structure that stores and maintains a cached image of the filesystem data.



The fuse library (5/5)

- Lastly, fuse_main() calls either fuse_loop() (lib/fuse.c) or fuse_loop_mt() (lib/fuse_mt.c) which both start to read the file system system calls from the /dev/fuse, call the user mode functions stored in **struct fuse_operations** data structure before calling fuse_main().

- The results of those calls are then written back to the /dev/fuse file where they can be forwarded back to the system calls.



fuse_loop(), fuse_loop_mt() (lib/fuse.c, lib/fuse_mt.c)
fuse_main() (lib/helper.c)
fuse_mount() (lib/mount.c)
fusermount() (util/fusermount.c)
Return filehandle
fuse_new() (lib/fuse.c)
write
read
/dev/fuse

Struct fuse_operations (1/9)

- int (*getattr) (const char *, struct stat *);
  - Get file attributes.
- int (*readlink) (const char *, char *, size_t);
  - Read the target of a symbolic link
- int (*mknod) (const char *, mode_t, dev_t);
  - Create a file node.
- int (*mkdir) (const char *, mode_t);
  - Create a directory. Note that the mode argument may not have the type specification bits set, i.e. S_ISDIR(mode) can be false. To obtain the correct directory type bits use mode | S_IFDIR

Struct fuse_operations (2/9)

- int (*unlink) (const char *);
  - Remove a file
- int (*rmdir) (const char *);
  - Remove a directory
- int (*symlink) (const char *, const char *);
  - Create a symbolic link
- int (*rename) (const char *, const char *);
  - Rename a file
- int (*link) (const char *, const char *);
  - Create a hard link to a file

Struct fuse_operations (3/9)

- 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
- int (*truncate) (const char *, off_t);
  - Change the size of a file
- int (*open) (const char *, struct fuse_file_info *);
  - File open operation.

Struct fuse_operations (4/9)

- int (*read) (const char *, char *, size_t, off_t, struct fuse_file_info *);
  - Read data from an open file.

- int (*write) (const char *, const char *, size_t, off_t, struct fuse_file_info *);
  - Write data to an open file

- int (*statfs) (const char *, struct statvfs *);
  - Get file system statistics
- int (*flush) (const char *, struct fuse_file_info *);
  - Possibly flush cached data

Struct fuse_operations (5/9)

- int (*release) (const char *, struct fuse_file_info *);
  - Release an open file. Release is called when there are no more references to an open file: all file descriptors are closed and all memory mappings are unmapped.

- int (*fsync) (const char *, int, struct fuse_file_info *);
  - Synchronize file contents
- int (*setxattr) (const char *, const char *, const char *, size_t, int);
  - Set extended attributes
- int (*getxattr) (const char *, const char *, char *, size_t);
  - Get extended attributes

Struct fuse_operations (6/9)

- int (*listxattr) (const char *, char *, size_t);
  - List extended attributes
- int (*removexattr) (const char *, const char *);
  - Remove extended attributes
- 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. Optionally opendir may also return an arbitrary **filehandle** in the **fuse_file_info** structure, which will be passed to readdir, closedir and fsyncdir.

Struct fuse_operations (7/9)

- 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
- void *(*init) (struct fuse_conn_info *conn);
  - Initialize file system.

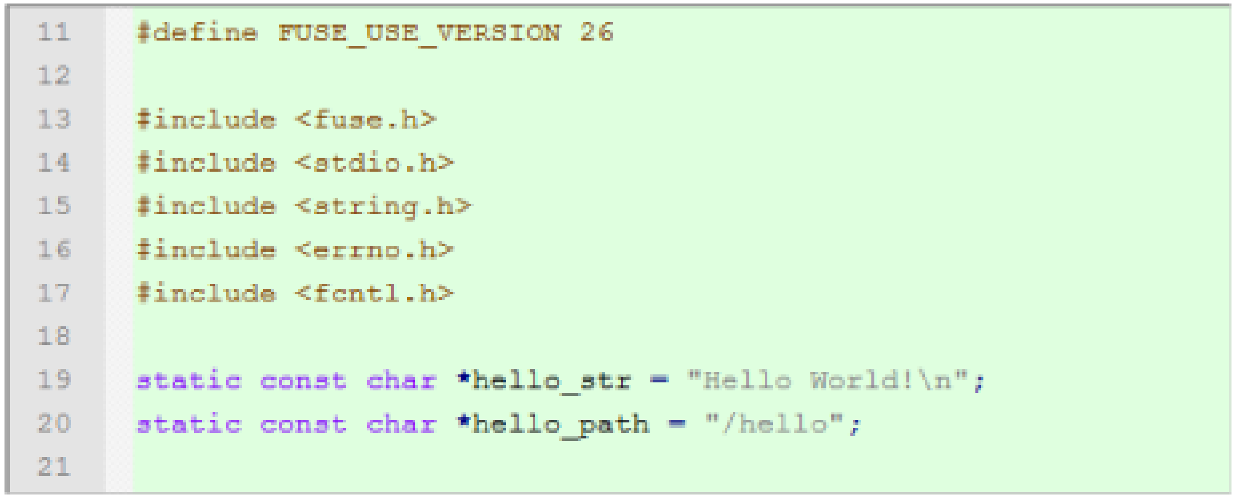
Struct fuse_operations (8/9)

- void (*destroy) (void *);
  - Clean up filesystem
- int (*access) (const char *, int);
  - Check file access permissions
- 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.
- int (*ftruncate) (const char *, off_t, struct fuse_file_info *);
  - Change the size of an open file
- int (*fgetattr) (const char *, struct stat *, struct fuse_file_info *);
  - Get attributes from an open file

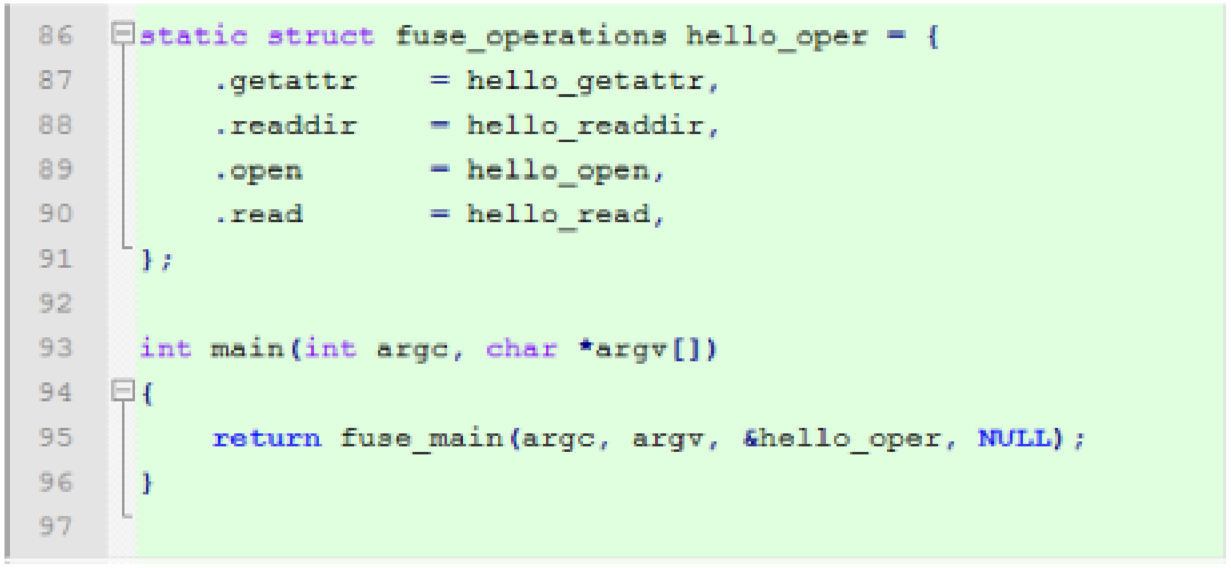
Struct fuse_operations(9/9)

- int (*lock) (const char *, struct fuse_file_info *, int cmd, struct flock *);
  - Perform POSIX file locking operation
- int (*utimens) (const char *, const struct timespec tv[2]);
  - Change the access and modification times of a file with nanosecond resolution
- int (*bmap) (const char *, size_t blocksize, uint64_t *idx);
  - Map block index within file to block index within device

Example1: Hello.c

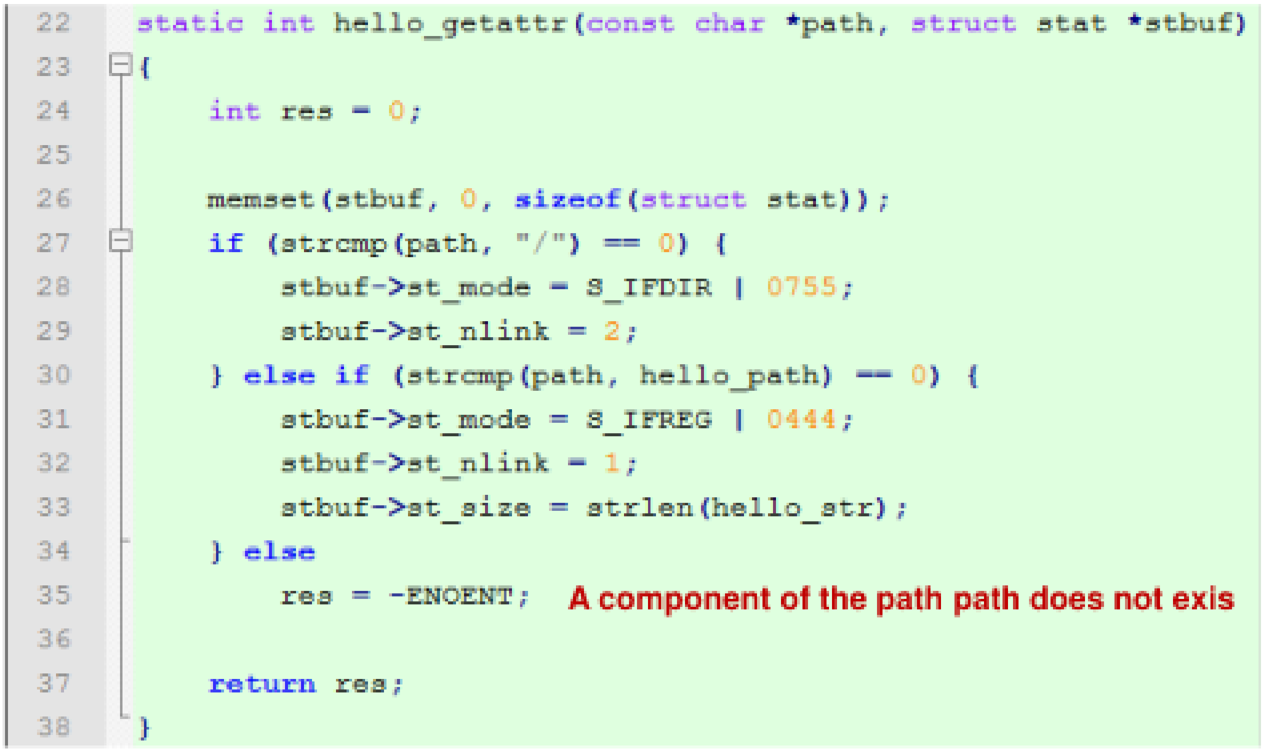


```c
#define FUSE_USE_VERSION 26

#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";

```



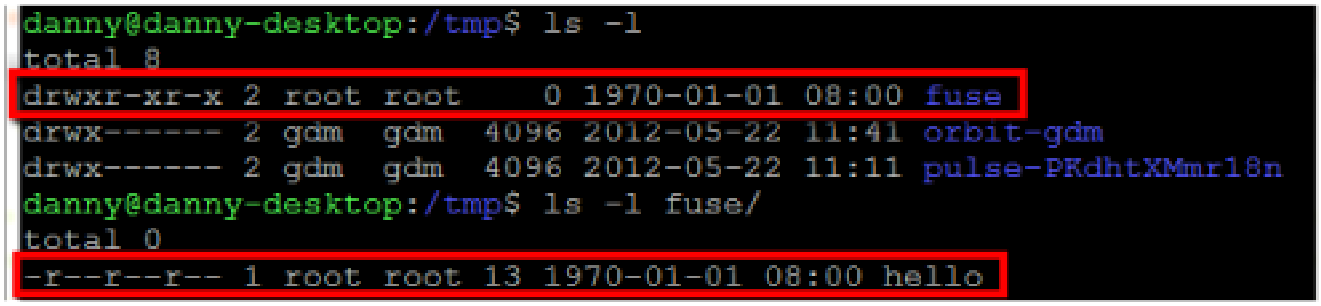
```c
static struct fuse_operations hello_oper = {
    .getattr    = hello_getattr,
    .readdir    = hello_readdir,
    .open       = hello_open,
    .read       = hello_read,
};

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

```

hello-getattr()



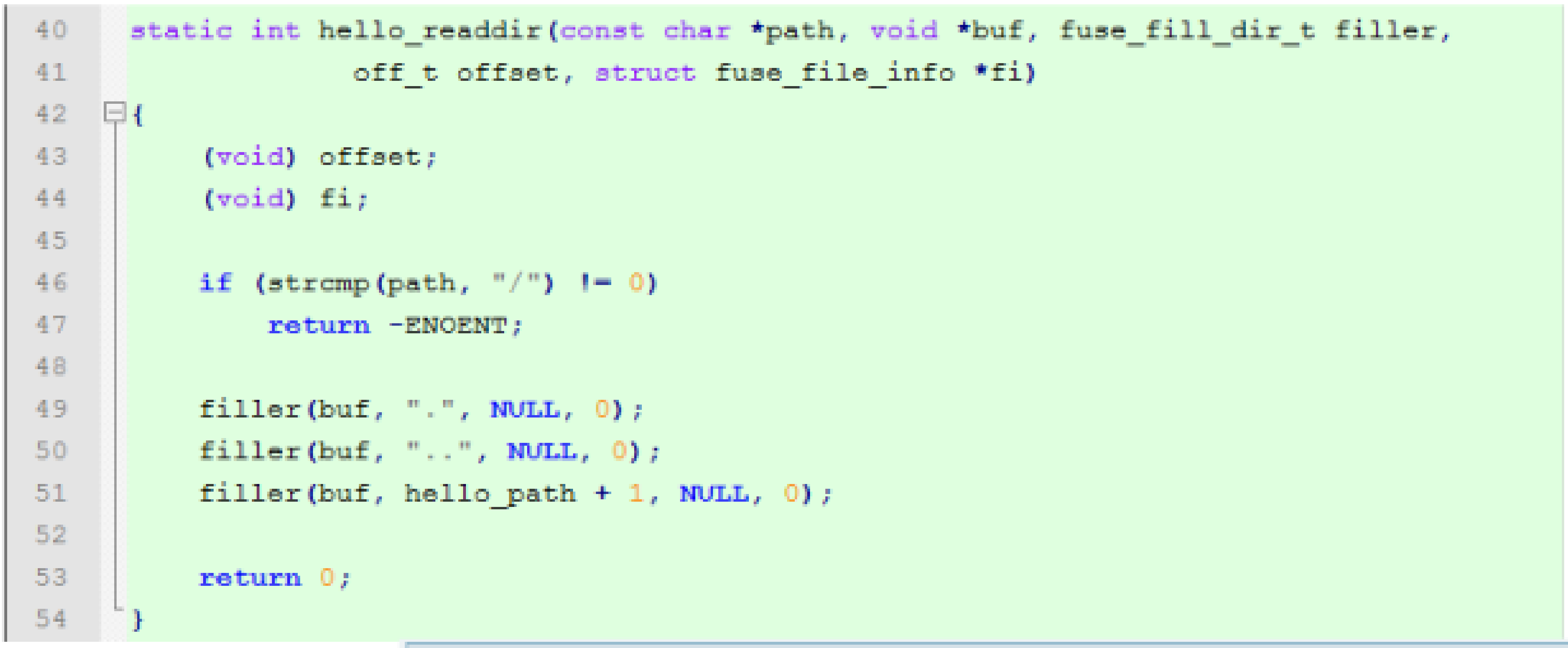
```c
static int hello_getattr(const char *path, struct stat *stbuf)
{
    int res = 0;

    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;
}
```

A component of the path path does not exis

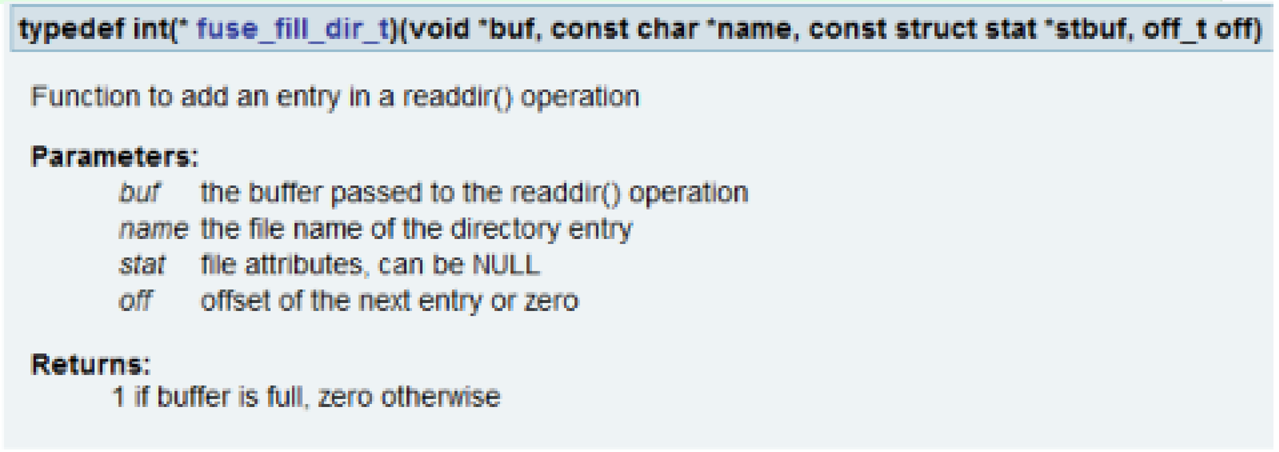


```
danny@danny-desktop:/tmp$ ls -l
total 8
drwxr-xr-x 2 root root     0 1970-01-01 08:00 fuse
drwx------ 2 gdm  gdm   4096 2012-05-22 11:41 orbit-gdm
drwx------ 2 gdm  gdm   4096 2012-05-22 11:11 pulse-PKdhtXMmr18n
danny@danny-desktop:/tmp$ ls -l fuse/
total 0
-r--r--r-- 1 root root 13 1970-01-01 08:00 hello
```

hello_readdir()



```c
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;
}
```



typedef int(* fuse_fill_dir_t)(void *buf, const char *name, const struct stat *stbuf, off_t off)

Function to add an entry in a readdir() operation

Parameters:

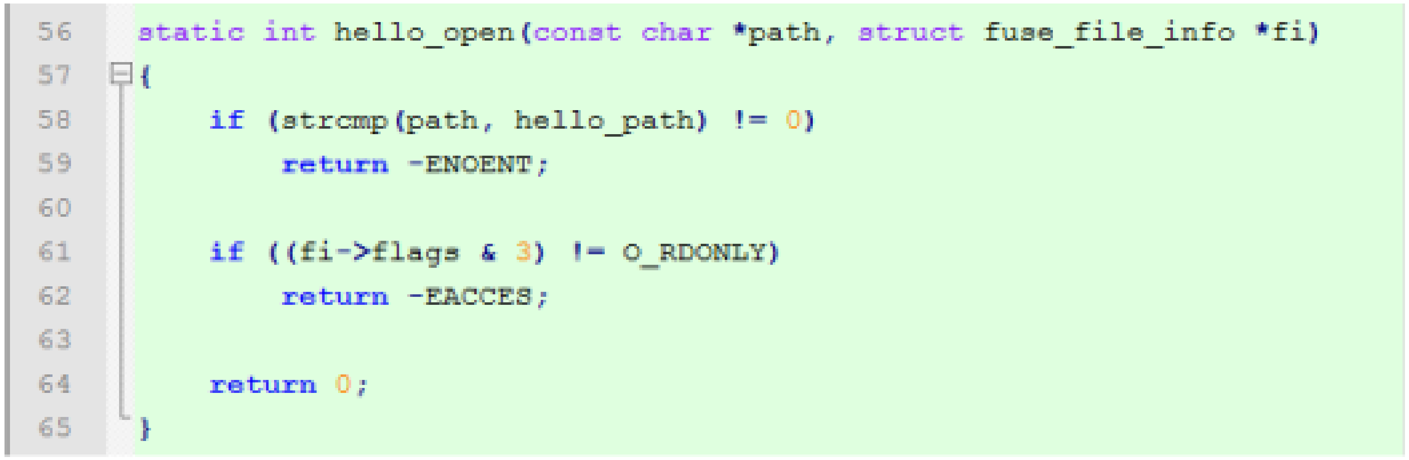
buf	the buffer passed to the readdir() operation
name	the file name of the directory entry
stat	file attributes, can be NULL
off	offset of the next entry or zero

Returns:

1 if buffer is full, zero otherwise

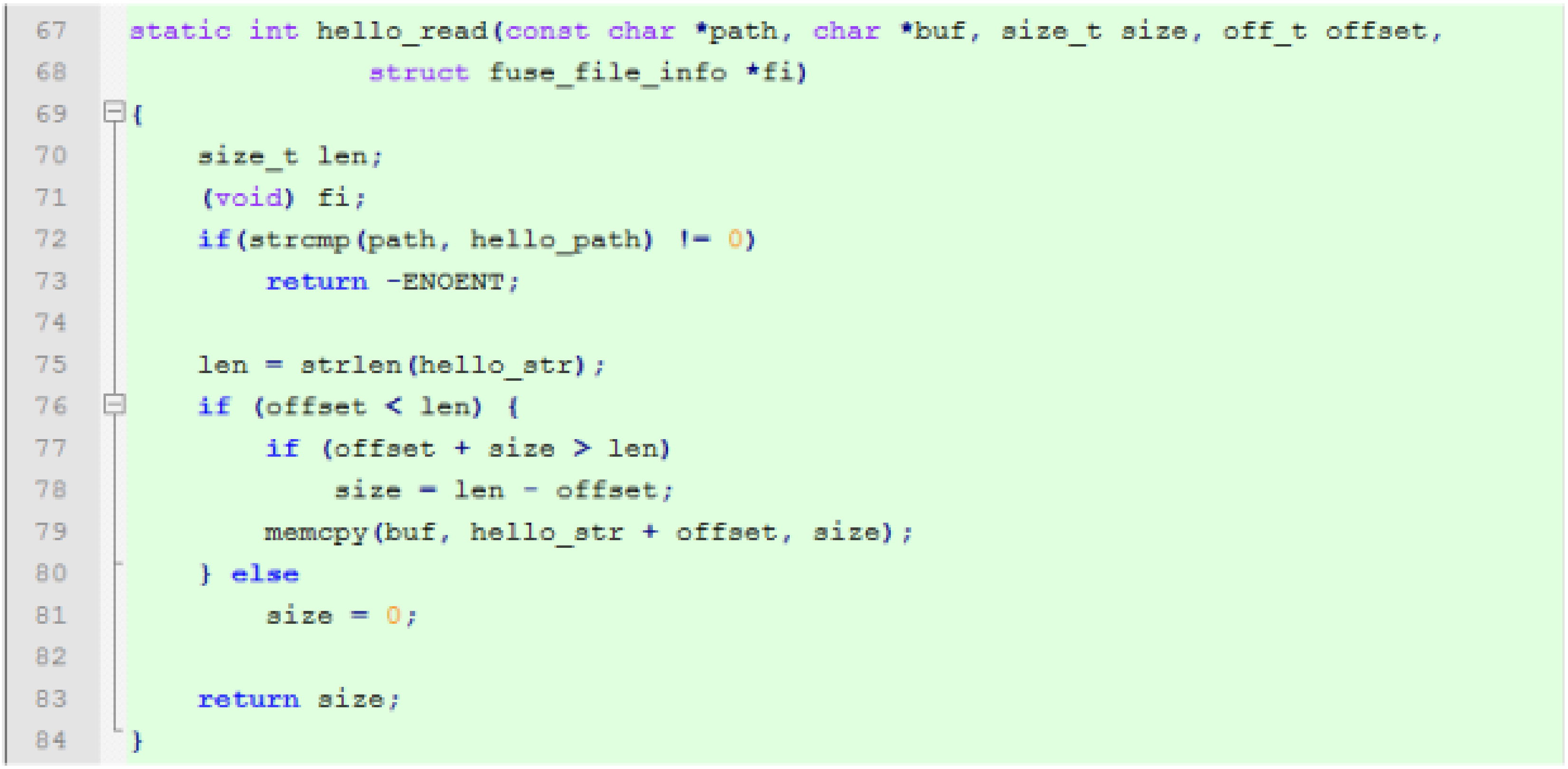
hello_open()

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



```c
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;
}
```

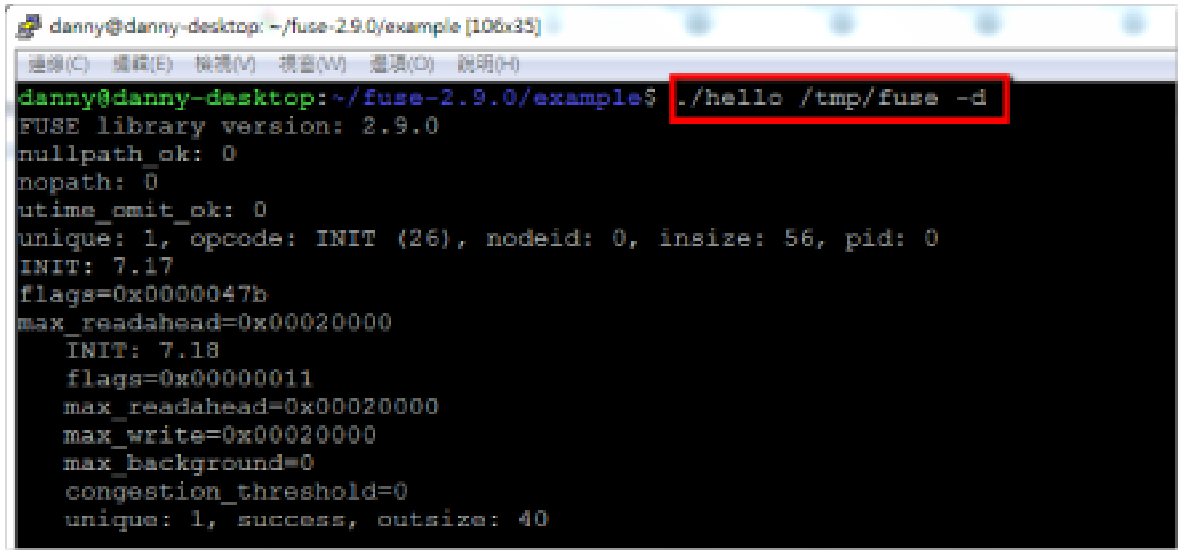
hello_read()



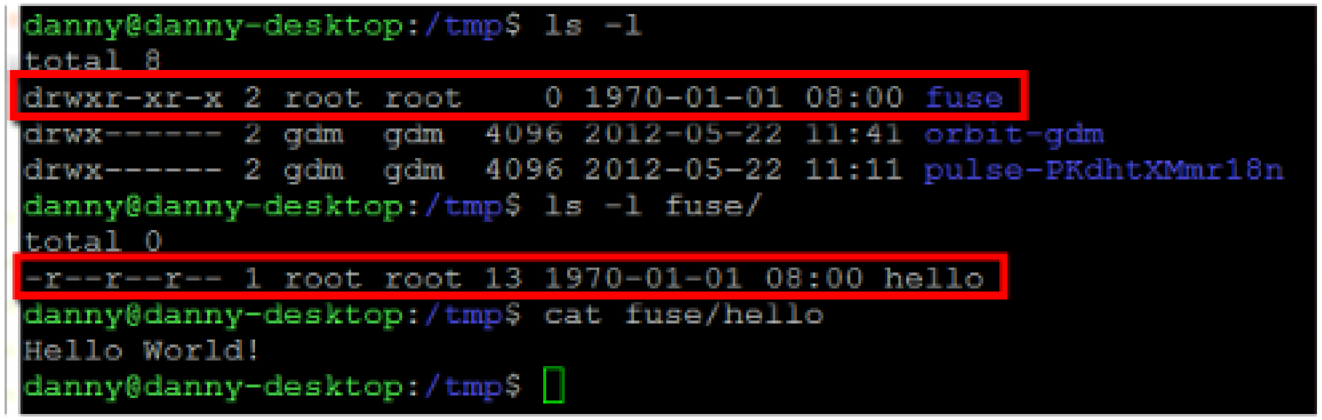
```
static int hello_read(const char *path, char *buf, size_t size, off_t offset,
                      struct fuse_file_info *fi)
{
    size_t len;
    (void) fi;
    if(strcmp(path, hello_path) != 0)
        return -ENOENT;

    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;
}
```

Example1: Hello.c 執行

- ./hello /tmp/fuse -d

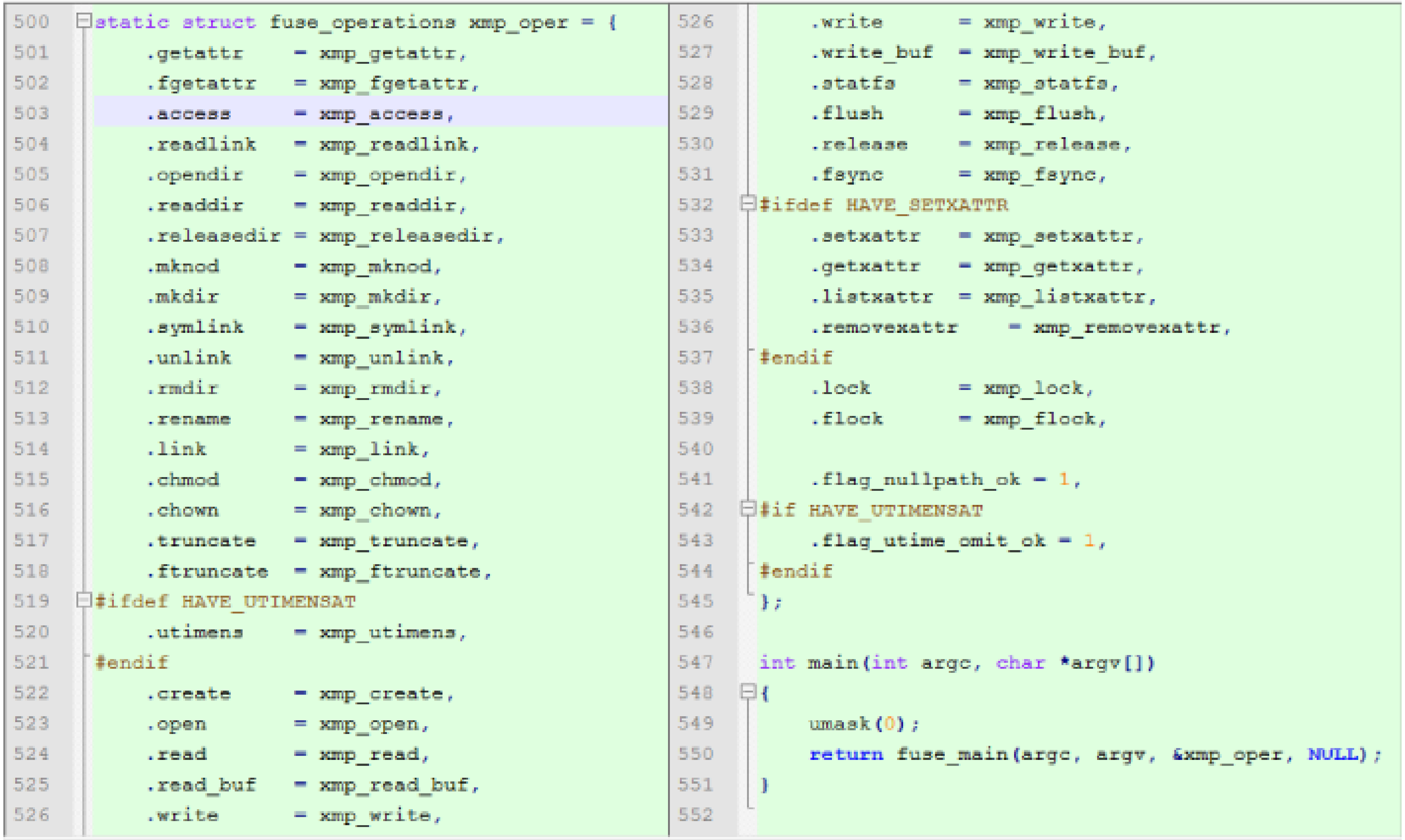

danny@danny-desktop: ~/fuse-2.9.0/example [106x35]
連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

```
danny@danny-desktop:~/fuse-2.9.0/example$ ./hello /tmp/fuse -d
FUSE library version: 2.9.0
nullpath_ok: 0
nopath: 0
utime_omit_ok: 0
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.17
flags=0x0000047b
max_readahead=0x00020000
   INIT: 7.18
   flags=0x00000011
   max_readahead=0x00020000
   max_write=0x00020000
   max_background=0
   congestion_threshold=0
   unique: 1, success, outsize: 40
```



```
danny@danny-desktop:/tmp$ ls -l
total 8
drwxr-xr-x 2 root root    0 1970-01-01 08:00 fuse
drwx------ 2 gdm  gdm  4096 2012-05-22 11:41 orbit-gdm
drwx------ 2 gdm  gdm  4096 2012-05-22 11:11 pulse-PKdhtXMmr18n
danny@danny-desktop:/tmp$ ls -l fuse/
total 0
-r--r--r-- 1 root root 13 1970-01-01 08:00 hello
danny@danny-desktop:/tmp$ cat fuse/hello
Hello World!
danny@danny-desktop:/tmp$ 
```

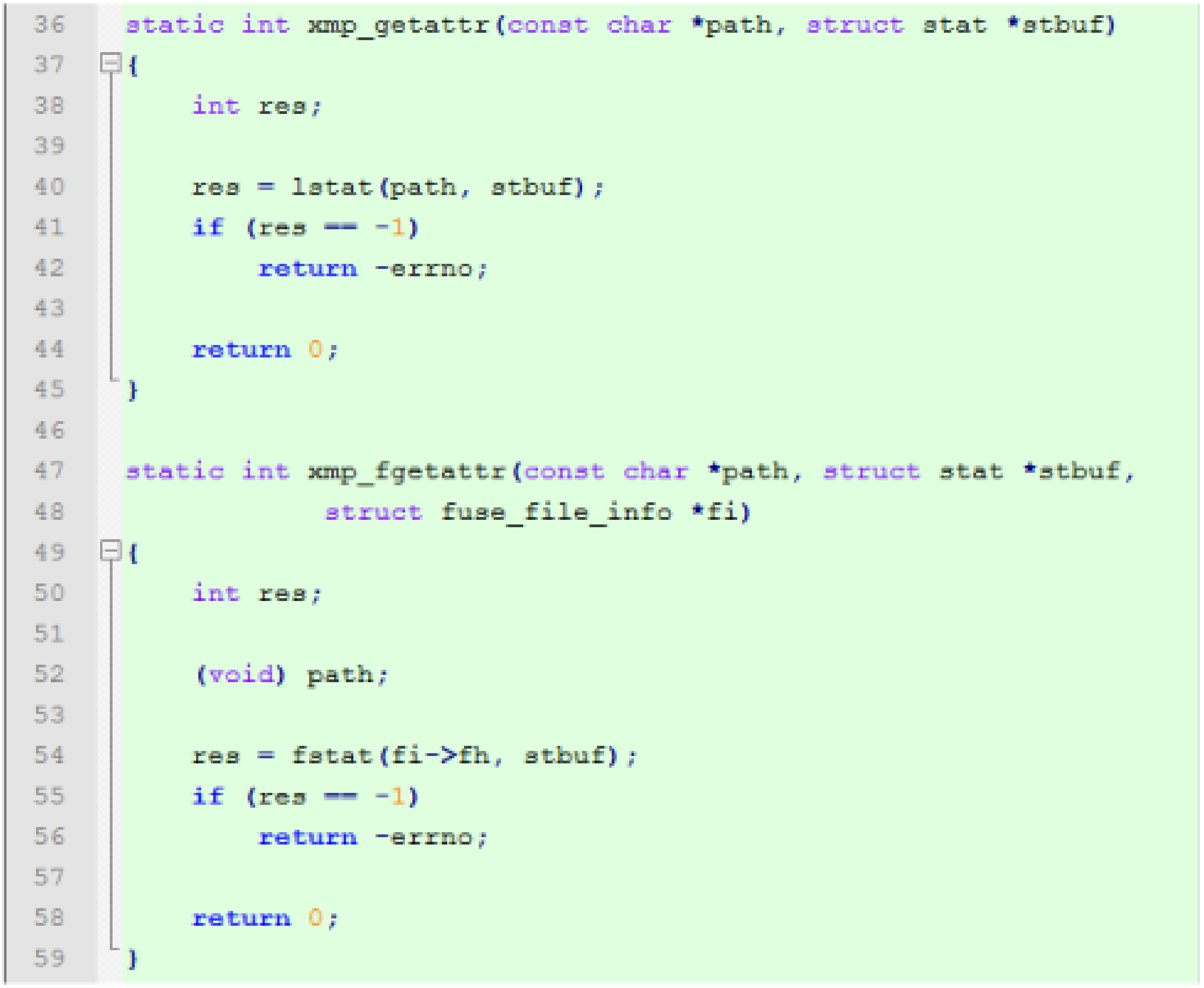
Example2: fusexmp_fh.c



```
static struct fuse_operations xmp_oper = {
	.getattr	= xmp_getattr,
	.fgetattr	= xmp_fgetattr,
	.access		= xmp_access,
	.readlink	= xmp_readlink,
	.opendir	= xmp_opendir,
	.readdir	= xmp_readdir,
	.releasedir = xmp_releasedir,
	.mknod		= xmp_mknod,
	.mkdir		= xmp_mkdir,
	.symlink	= xmp_symlink,
	.unlink		= xmp_unlink,
	.rmdir		= xmp_rmdir,
	.rename		= xmp_rename,
	.link		= xmp_link,
	.chmod		= xmp_chmod,
	.chown		= xmp_chown,
	.truncate	= xmp_truncate,
	.ftruncate	= xmp_ftruncate,
#ifdef HAVE_UTIMENSAT
	.utimens	= xmp_utimens,
#endif
	.create		= xmp_create,
	.open		= xmp_open,
	.read		= xmp_read,
	.read_buf	= xmp_read_buf,
	.write		= xmp_write,
```

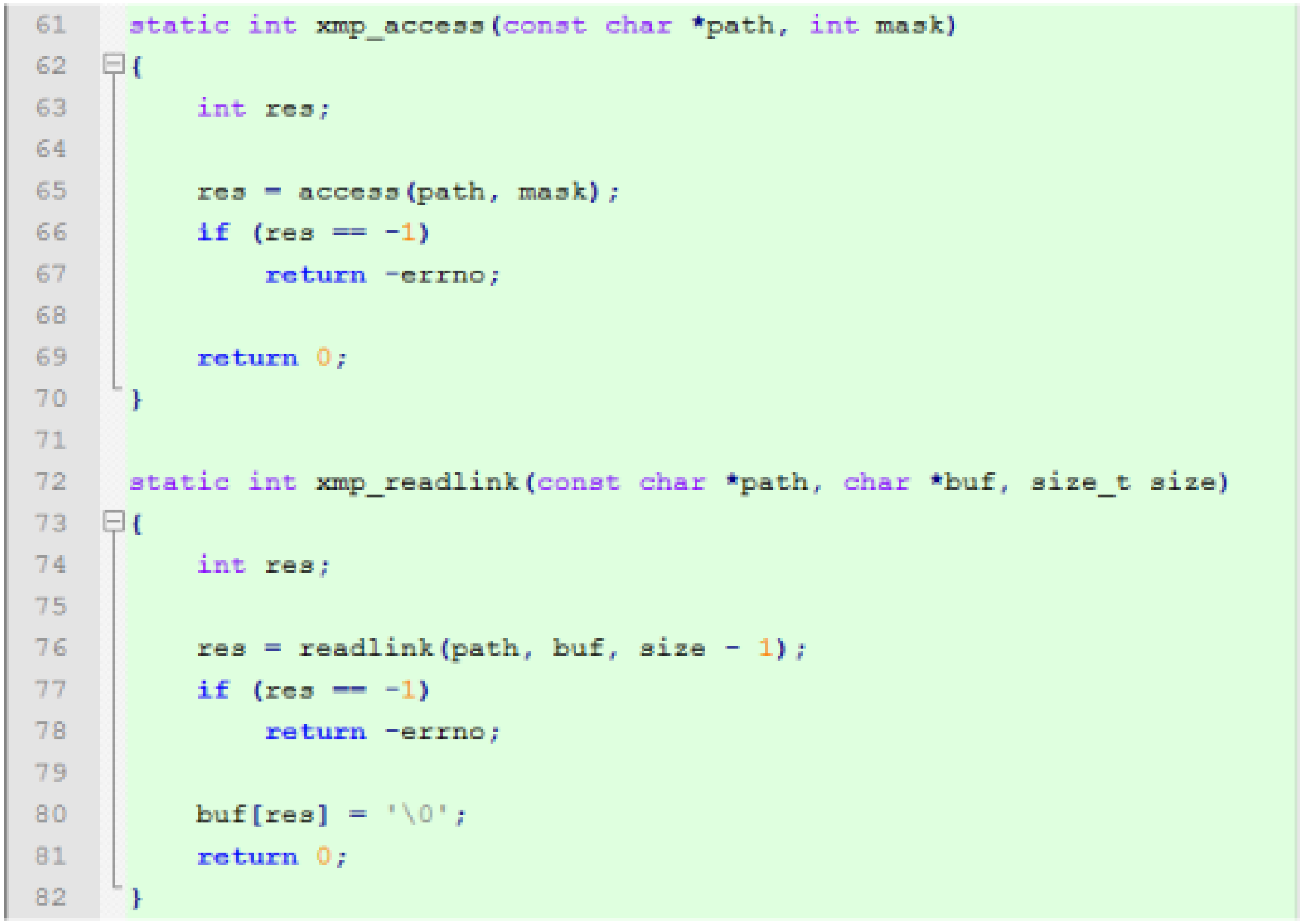
```
	.write		= xmp_write,
	.write_buf	= xmp_write_buf,
	.statfs		= xmp_statfs,
	.flush		= xmp_flush,
	.release	= xmp_release,
	.fsync		= xmp_fsync,
#ifdef HAVE_SETXATTR
	.setxattr	= xmp_setxattr,
	.getxattr	= xmp_getxattr,
	.listxattr	= xmp_listxattr,
	.removexattr	= xmp_removexattr,
#endif
	.lock		= xmp_lock,
	.flock		= xmp_flock,

	.flag_nullpath_ok = 1,
#if HAVE_UTIMENSAT
	.flag_utime_omit_ok = 1,
#endif
};

int main(int argc, char *argv[])
{
	umask(0);
	return fuse_main(argc, argv, &xmp_oper, NULL);
}

```

xmp_getattr(), xmp_fgetattr()



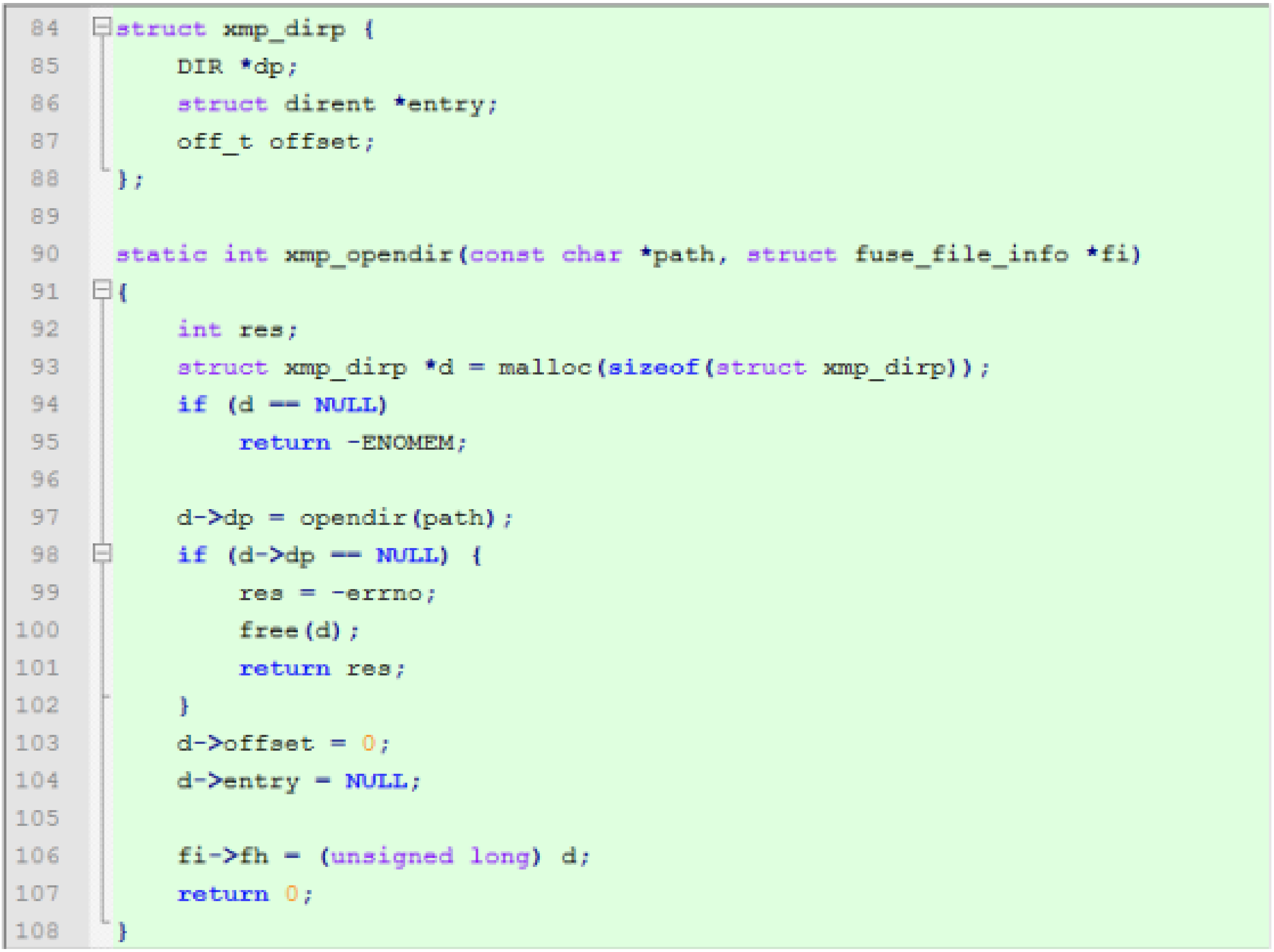
```
static int xmp_getattr(const char *path, struct stat *stbuf)
{
    int res;

    res = lstat(path, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_fgetattr(const char *path, struct stat *stbuf,
            struct fuse_file_info *fi)
{
    int res;

    (void) path;

    res = fstat(fi->fh, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_access(), xmp_readlink()



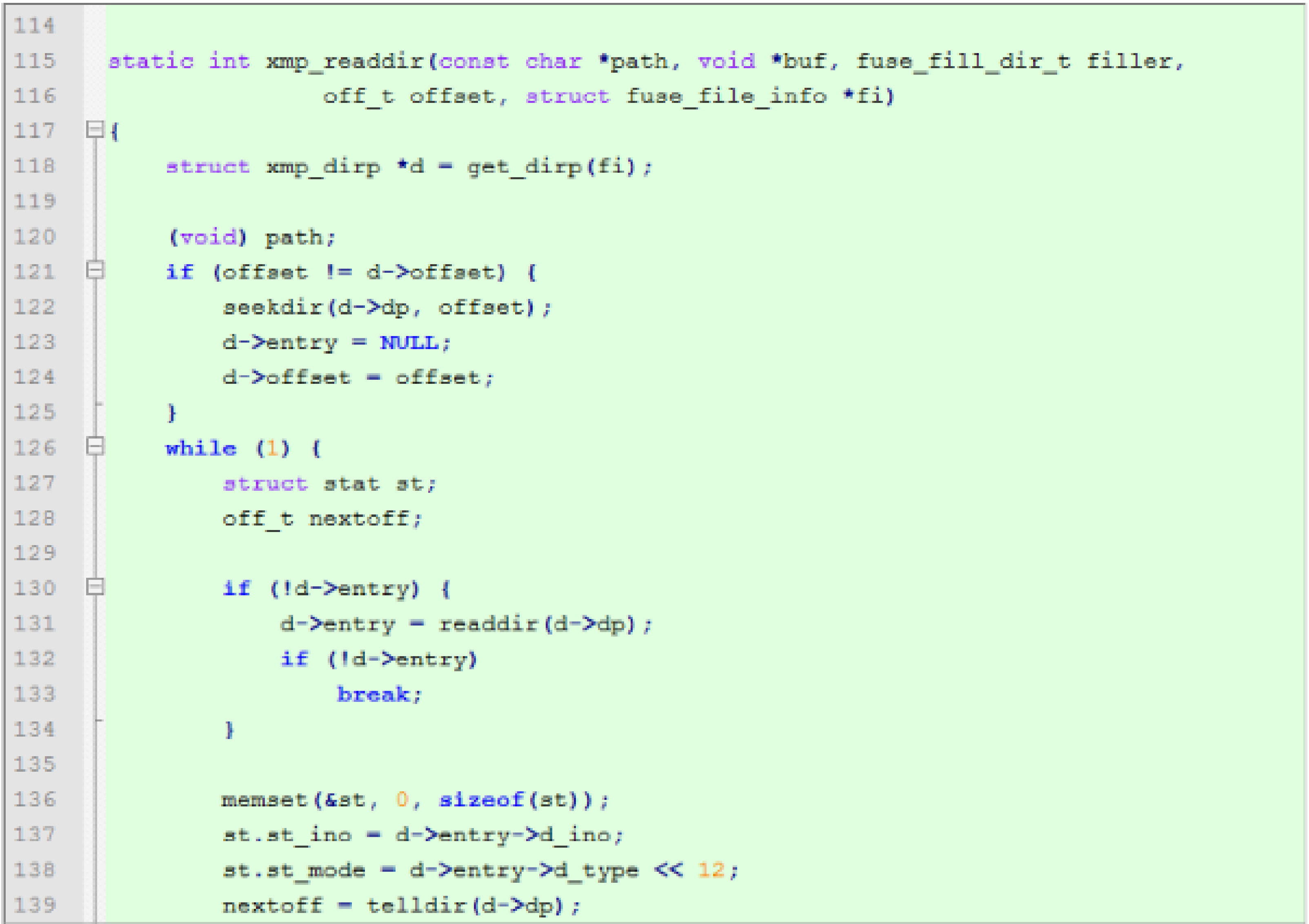
```c
static int xmp_access(const char *path, int mask)
{
    int res;

    res = access(path, mask);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_readlink(const char *path, char *buf, size_t size)
{
    int res;

    res = readlink(path, buf, size - 1);
    if (res == -1)
        return -errno;

    buf[res] = '\0';
    return 0;
}
```

Struct xmp_dirp, xmp_opendir()



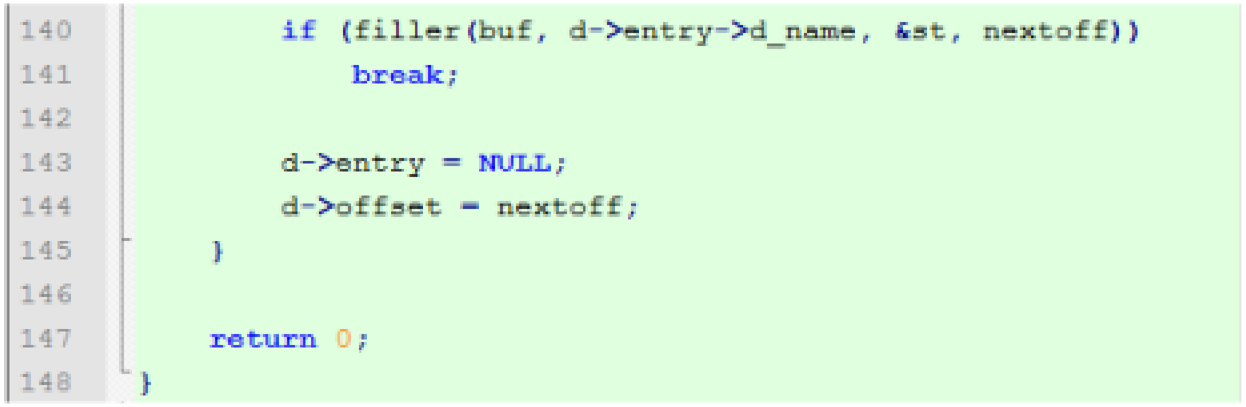
```c
struct xmp_dirp {
    DIR *dp;
    struct dirent *entry;
    off_t offset;
};

static int xmp_opendir(const char *path, struct fuse_file_info *fi)
{
    int res;
    struct xmp_dirp *d = malloc(sizeof(struct xmp_dirp));
    if (d == NULL)
        return -ENOMEM;

    d->dp = opendir(path);
    if (d->dp == NULL) {
        res = -errno;
        free(d);
        return res;
    }
    d->offset = 0;
    d->entry = NULL;

    fi->fh = (unsigned long) d;
    return 0;
}
```

xmp_readdir() (1/2)



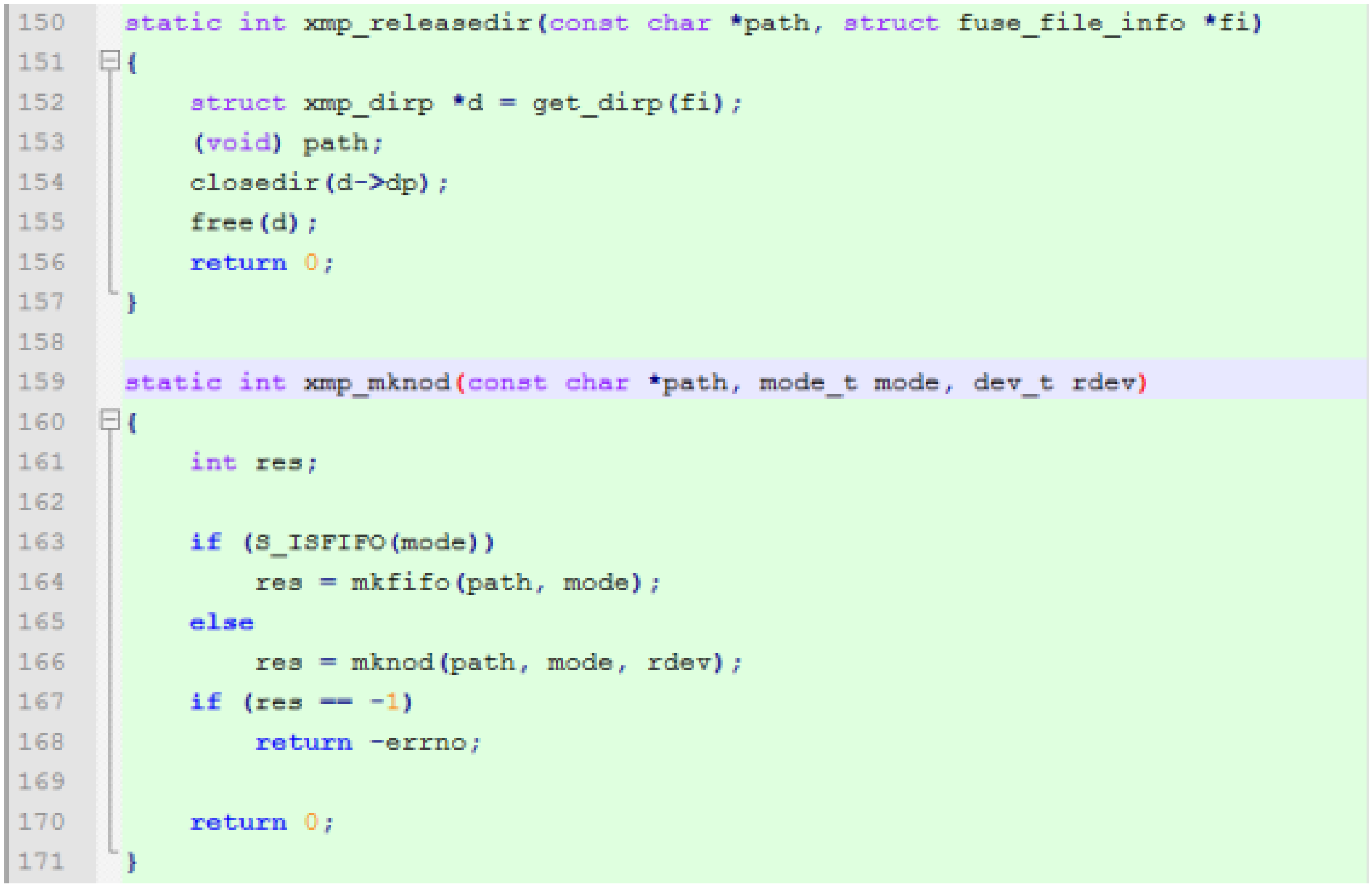
```c
static int xmp_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
               off_t offset, struct fuse_file_info *fi)
{
    struct xmp_dirp *d = get_dirp(fi);

    (void) path;
    if (offset != d->offset) {
        seekdir(d->dp, offset);
        d->entry = NULL;
        d->offset = offset;
    }
    while (1) {
        struct stat st;
        off_t nextoff;

        if (!d->entry) {
            d->entry = readdir(d->dp);
            if (!d->entry)
                break;
        }

        memset(&st, 0, sizeof(st));
        st.st_ino = d->entry->d_ino;
        st.st_mode = d->entry->d_type << 12;
        nextoff = telldir(d->dp);
```

xmp_readdir() (2/2)



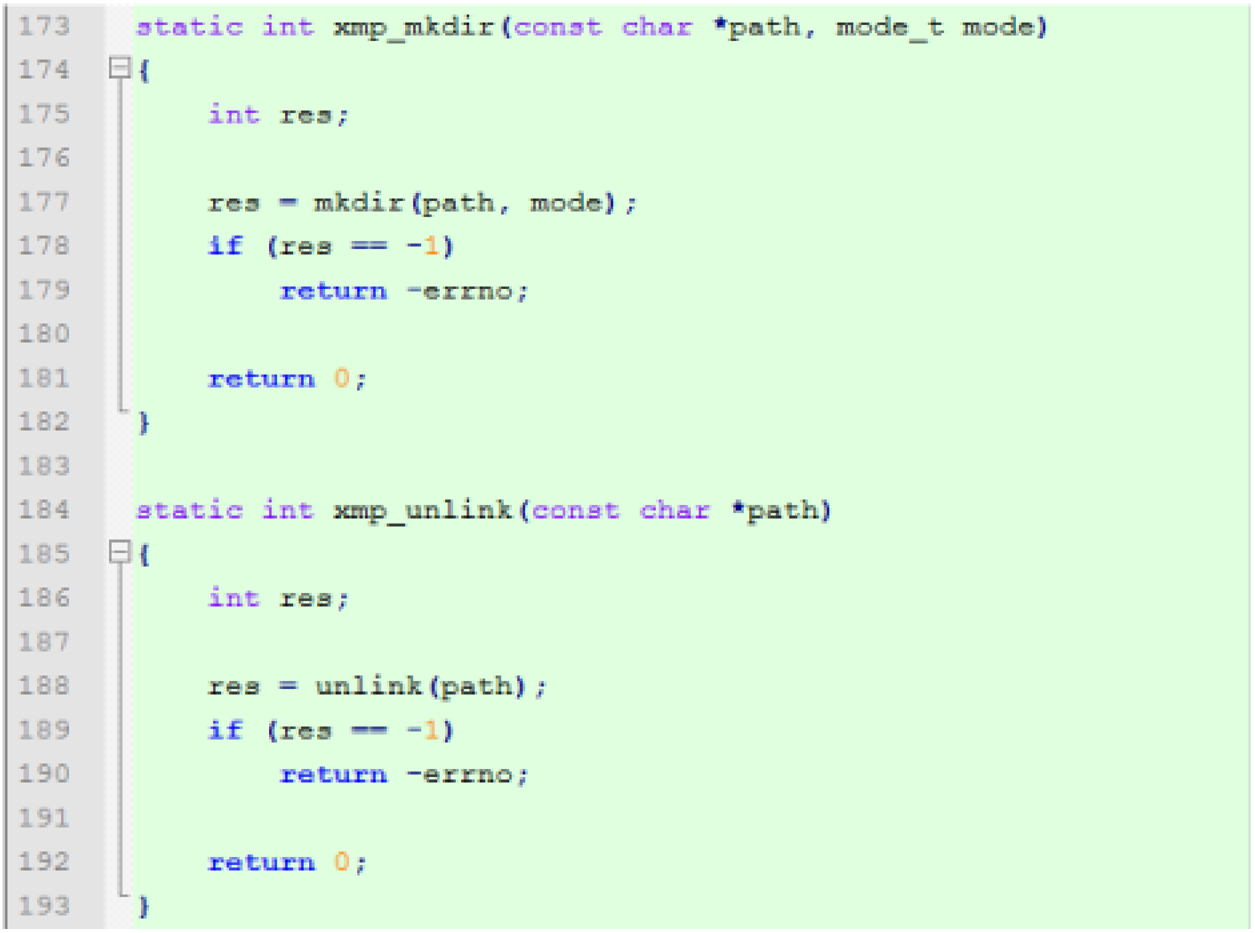
```
            if (filler(buf, d->entry->d_name, &st, nextoff))
                break;

            d->entry = NULL;
            d->offset = nextoff;
        }

    return 0;
}
```

xmp_releasedir(), xmp_mknod()



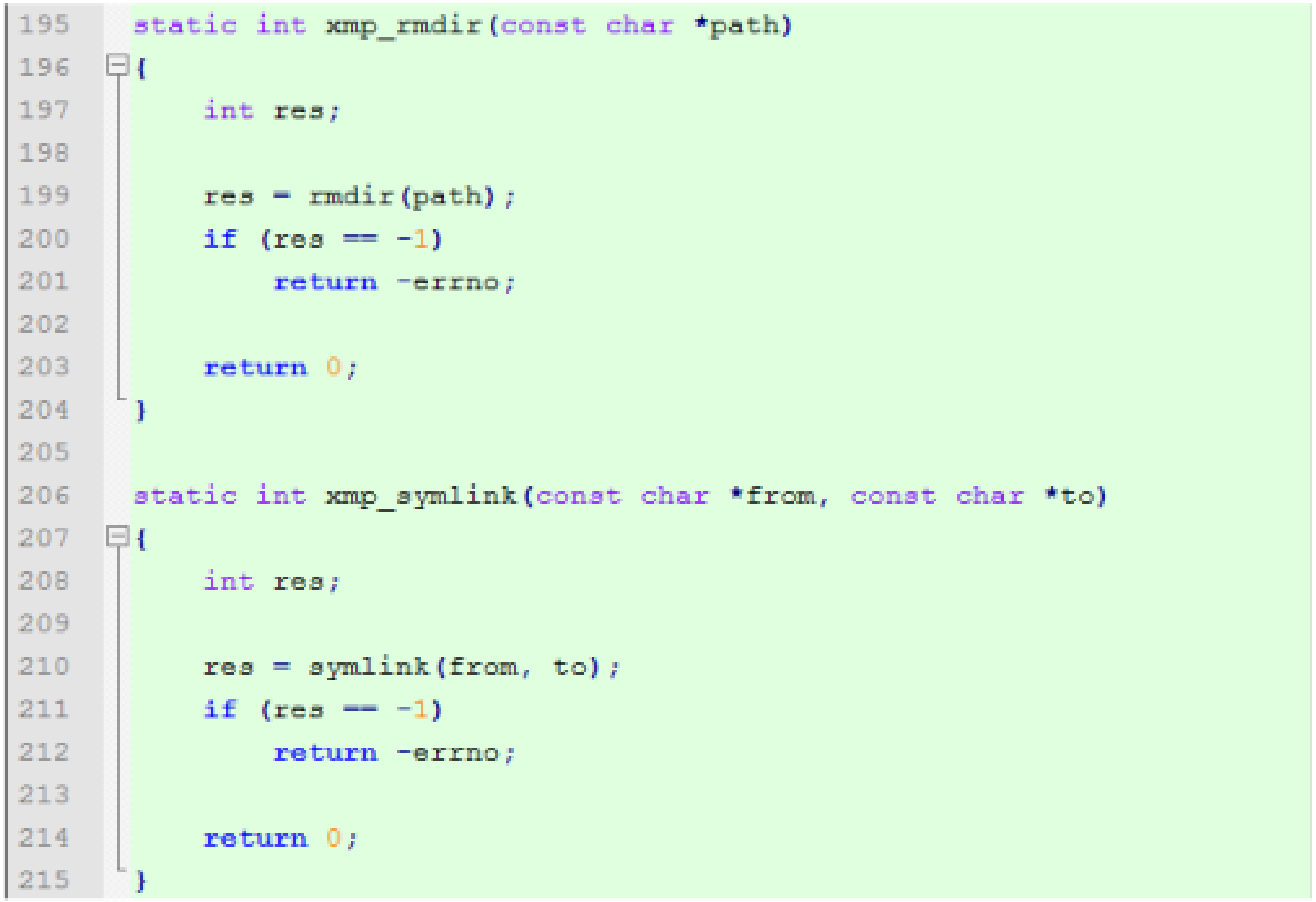
```c
static int xmp_releasedir(const char *path, struct fuse_file_info *fi)
{
    struct xmp_dirp *d = get_dirp(fi);
    (void) path;
    closedir(d->dp);
    free(d);
    return 0;
}

static int xmp_mknod(const char *path, mode_t mode, dev_t rdev)
{
    int res;

    if (S_ISFIFO(mode))
        res = mkfifo(path, mode);
    else
        res = mknod(path, mode, rdev);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_mkdir(), xmp_unlink()



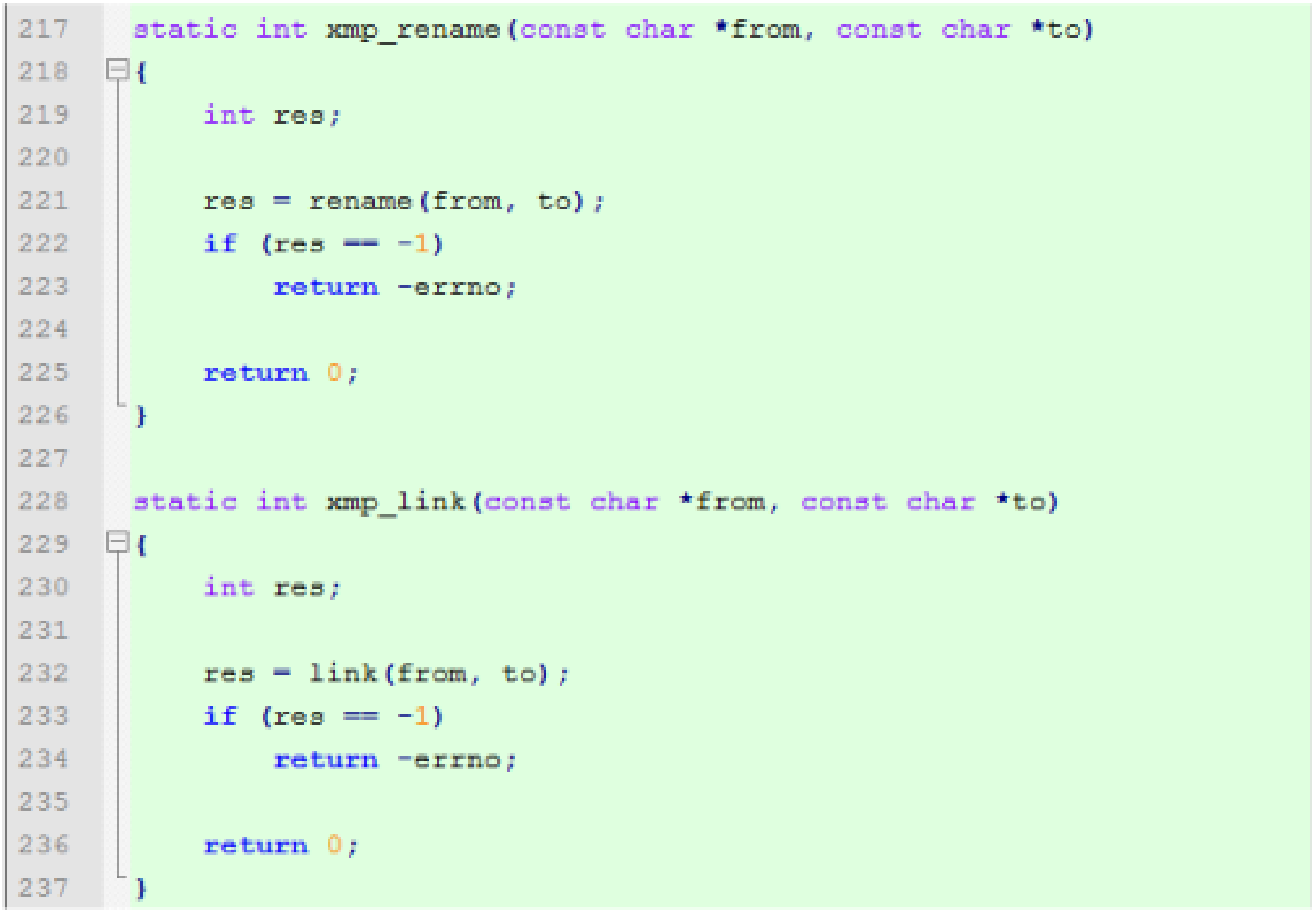
```
static int xmp_mkdir(const char *path, mode_t mode)
{
    int res;

    res = mkdir(path, mode);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_unlink(const char *path)
{
    int res;

    res = unlink(path);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_rmdir(), xmp_symlink()



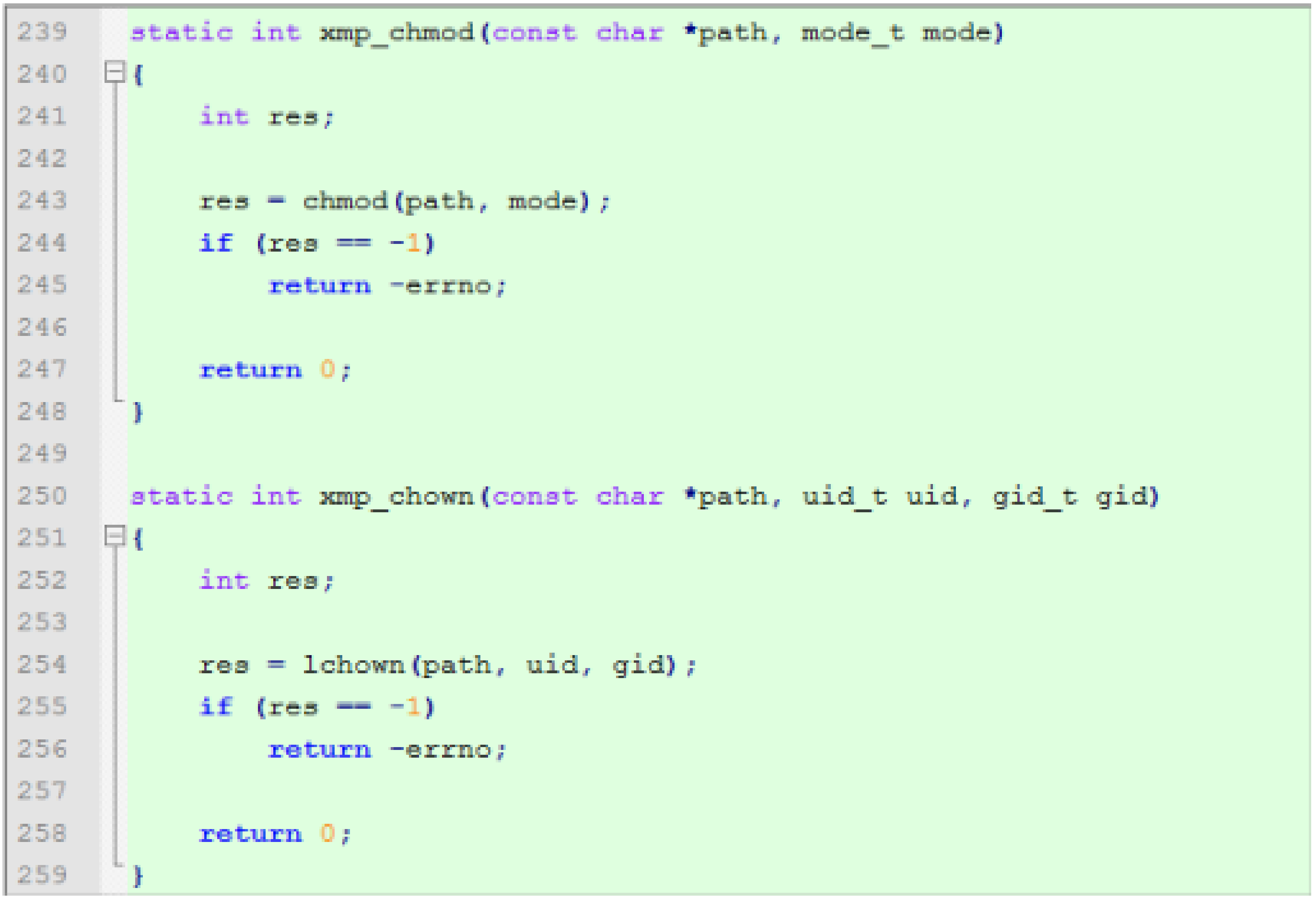
```c
static int xmp_rmdir(const char *path)
{
    int res;

    res = rmdir(path);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_symlink(const char *from, const char *to)
{
    int res;

    res = symlink(from, to);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_rename(), xmp_link()



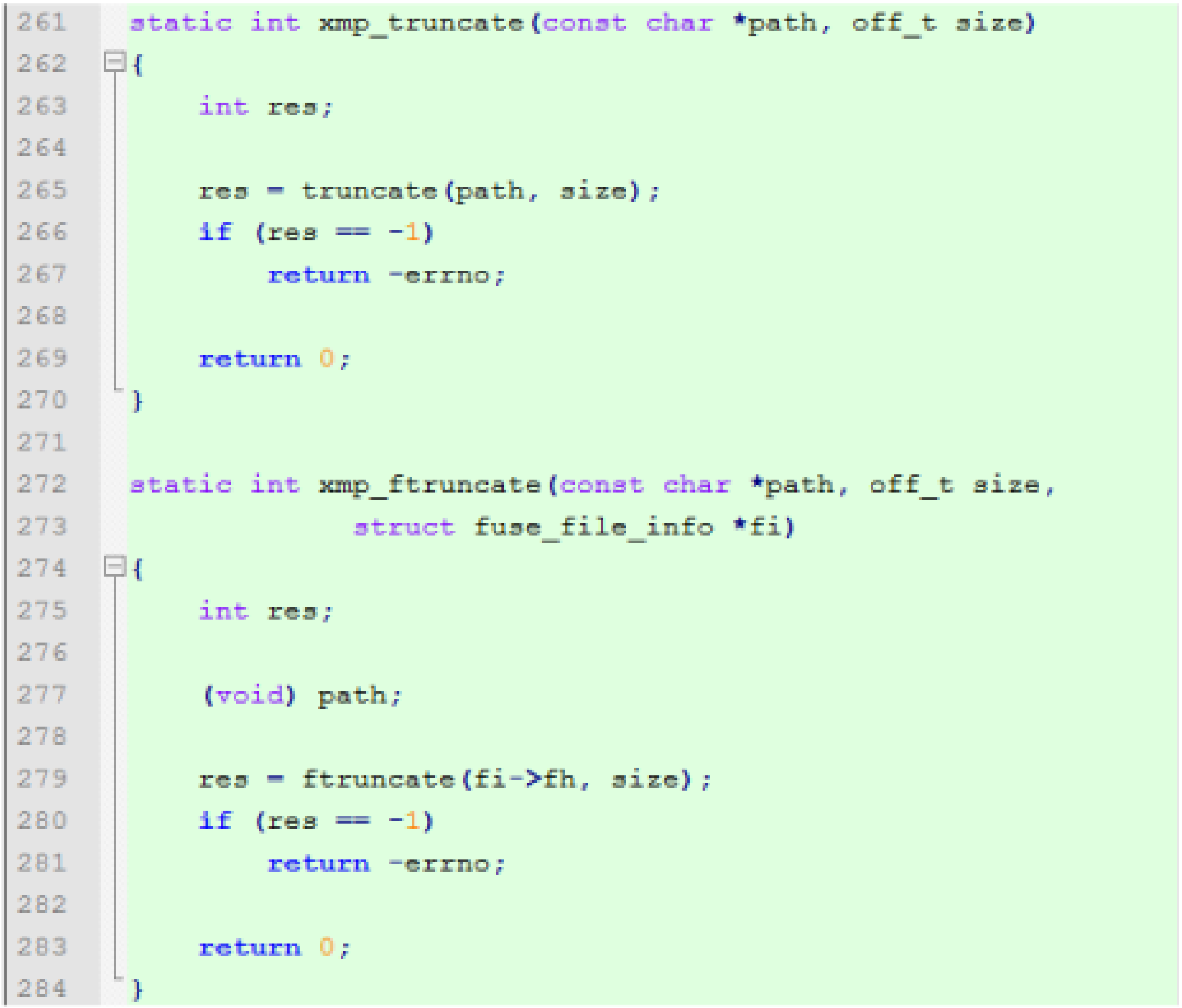
```c
static int xmp_rename(const char *from, const char *to)
{
    int res;

    res = rename(from, to);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_link(const char *from, const char *to)
{
    int res;

    res = link(from, to);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_chmod(), xmp_chown()



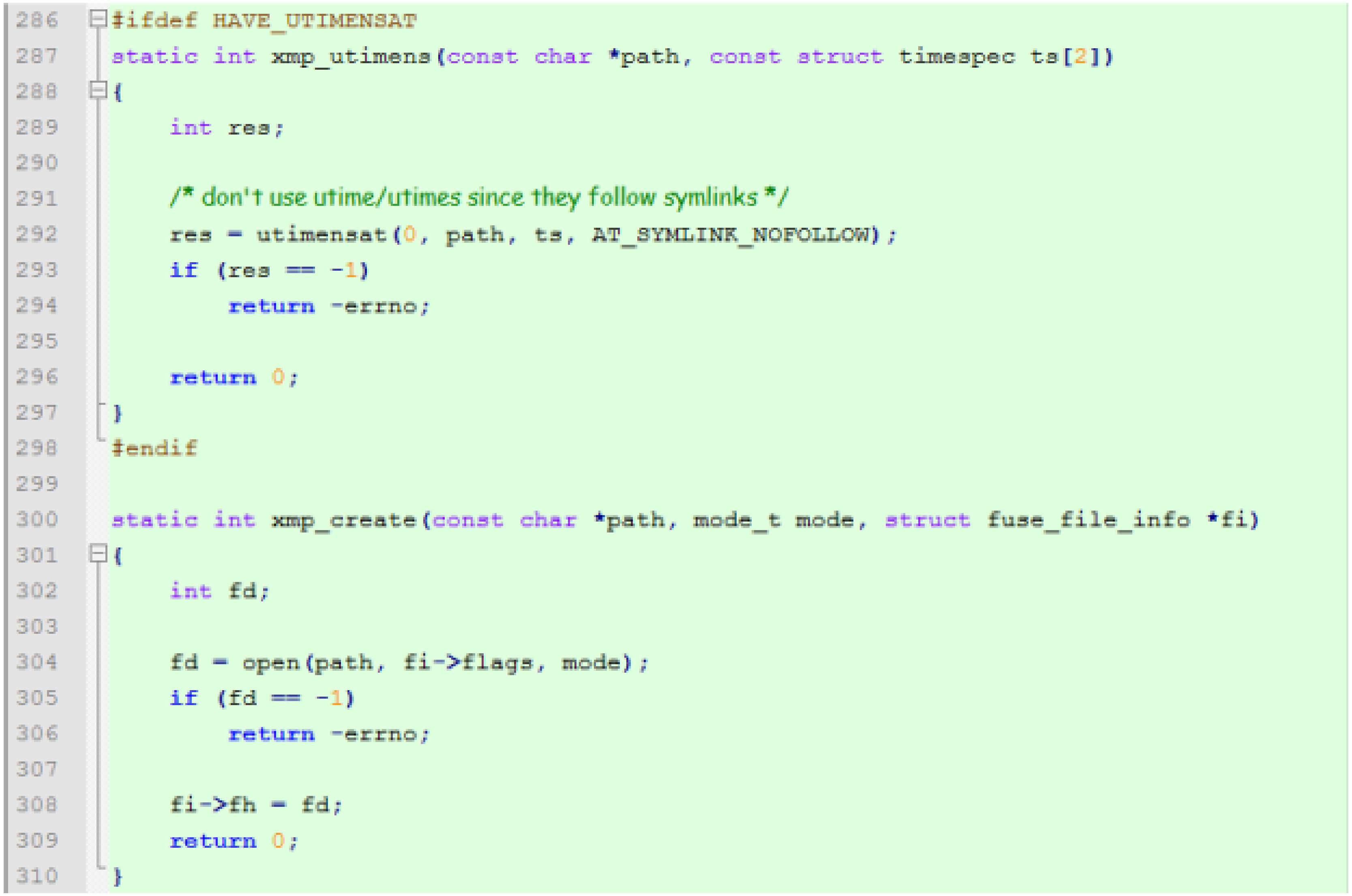
```c
static int xmp_chmod(const char *path, mode_t mode)
{
    int res;

    res = chmod(path, mode);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_chown(const char *path, uid_t uid, gid_t gid)
{
    int res;

    res = lchown(path, uid, gid);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_truncate(), xmp_ftruncate()



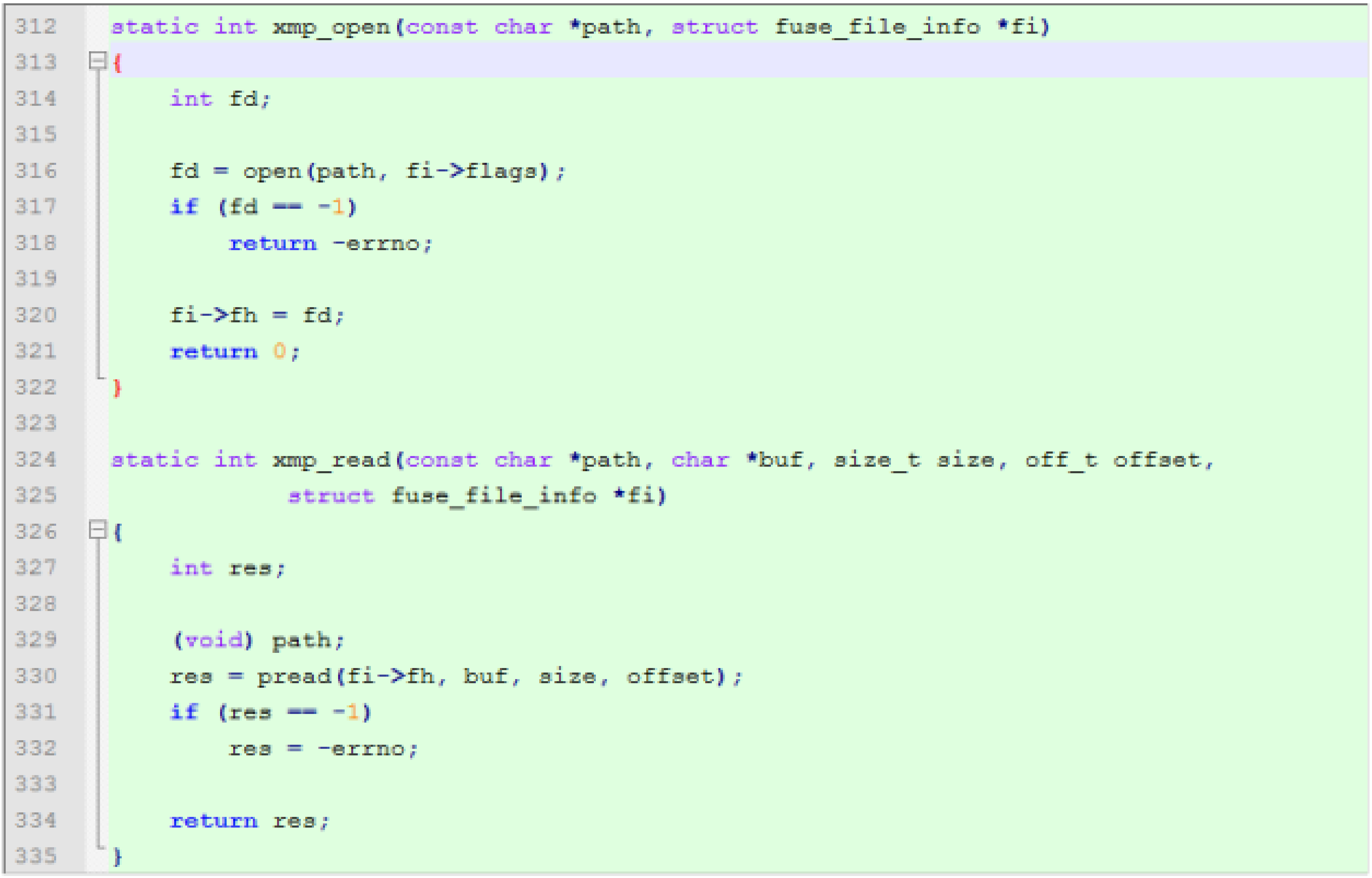
```c
static int xmp_truncate(const char *path, off_t size)
{
    int res;

    res = truncate(path, size);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_ftruncate(const char *path, off_t size,
             struct fuse_file_info *fi)
{
    int res;

    (void) path;

    res = ftruncate(fi->fh, size);
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_utimens(), xmp_create()



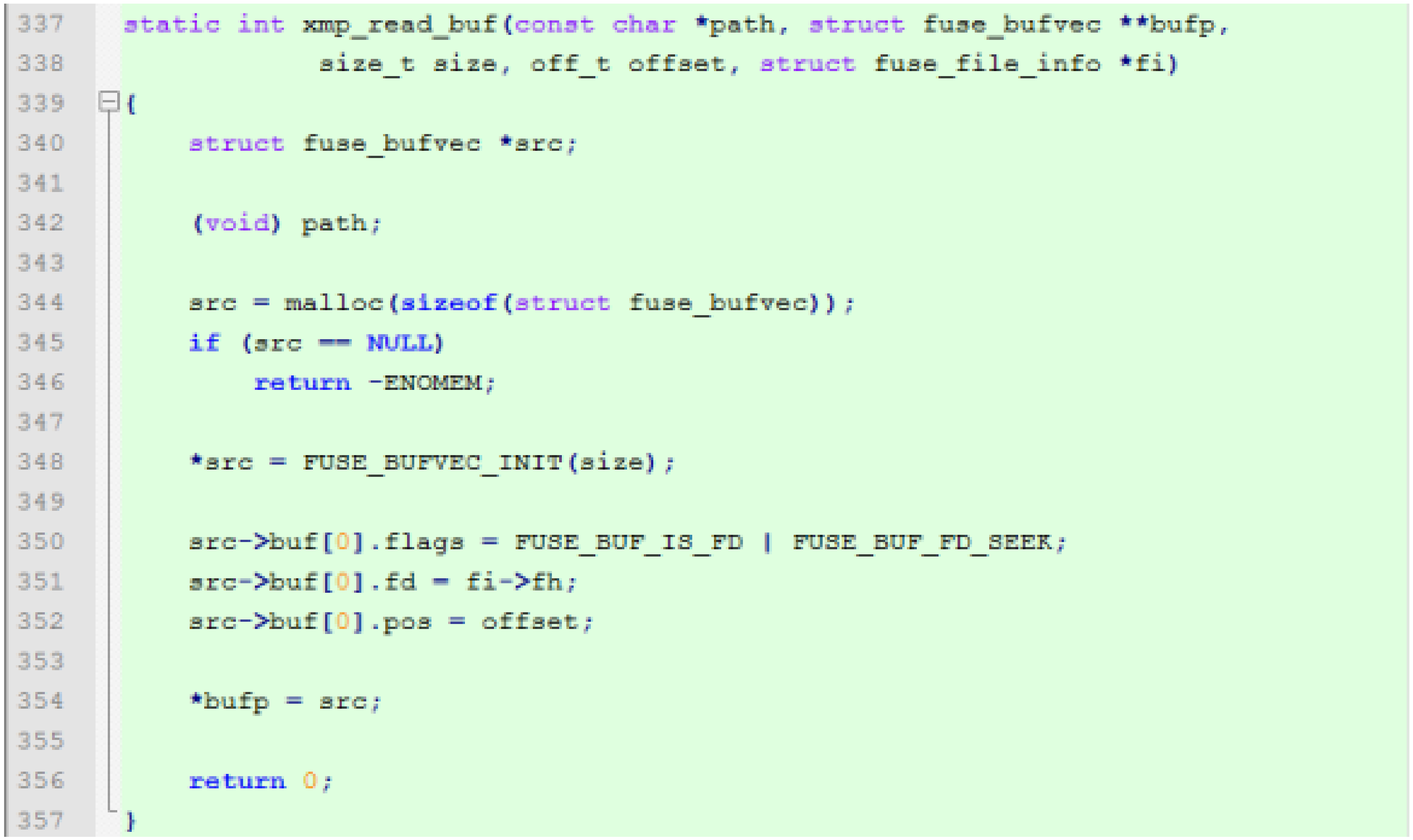
```c
#ifdef HAVE_UTIMENSAT
static int xmp_utimens(const char *path, const struct timespec ts[2])
{
	int res;

	/* don't use utime/utimes since they follow symlinks */
	res = utimensat(0, path, ts, AT_SYMLINK_NOFOLLOW);
	if (res == -1)
		return -errno;

	return 0;
}
#endif

static int xmp_create(const char *path, mode_t mode, struct fuse_file_info *fi)
{
	int fd;

	fd = open(path, fi->flags, mode);
	if (fd == -1)
		return -errno;

	fi->fh = fd;
	return 0;
}
```

xmp_open(), xmp_read()



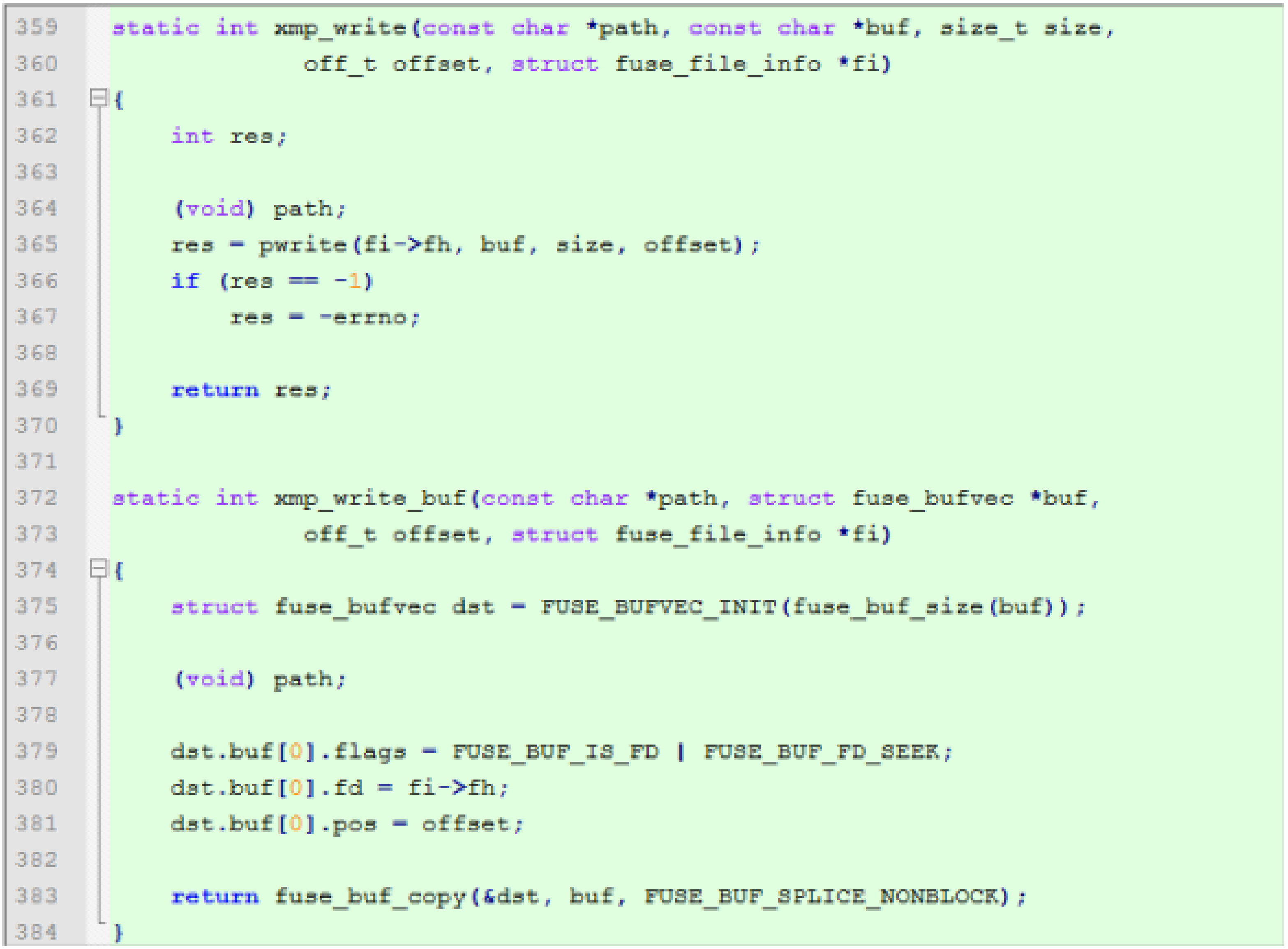
```c
static int xmp_open(const char *path, struct fuse_file_info *fi)
{
    int fd;

    fd = open(path, fi->flags);
    if (fd == -1)
        return -errno;

    fi->fh = fd;
    return 0;
}

static int xmp_read(const char *path, char *buf, size_t size, off_t offset,
            struct fuse_file_info *fi)
{
    int res;

    (void) path;
    res = pread(fi->fh, buf, size, offset);
    if (res == -1)
        res = -errno;

    return res;
}
```

xmp_read_buf()



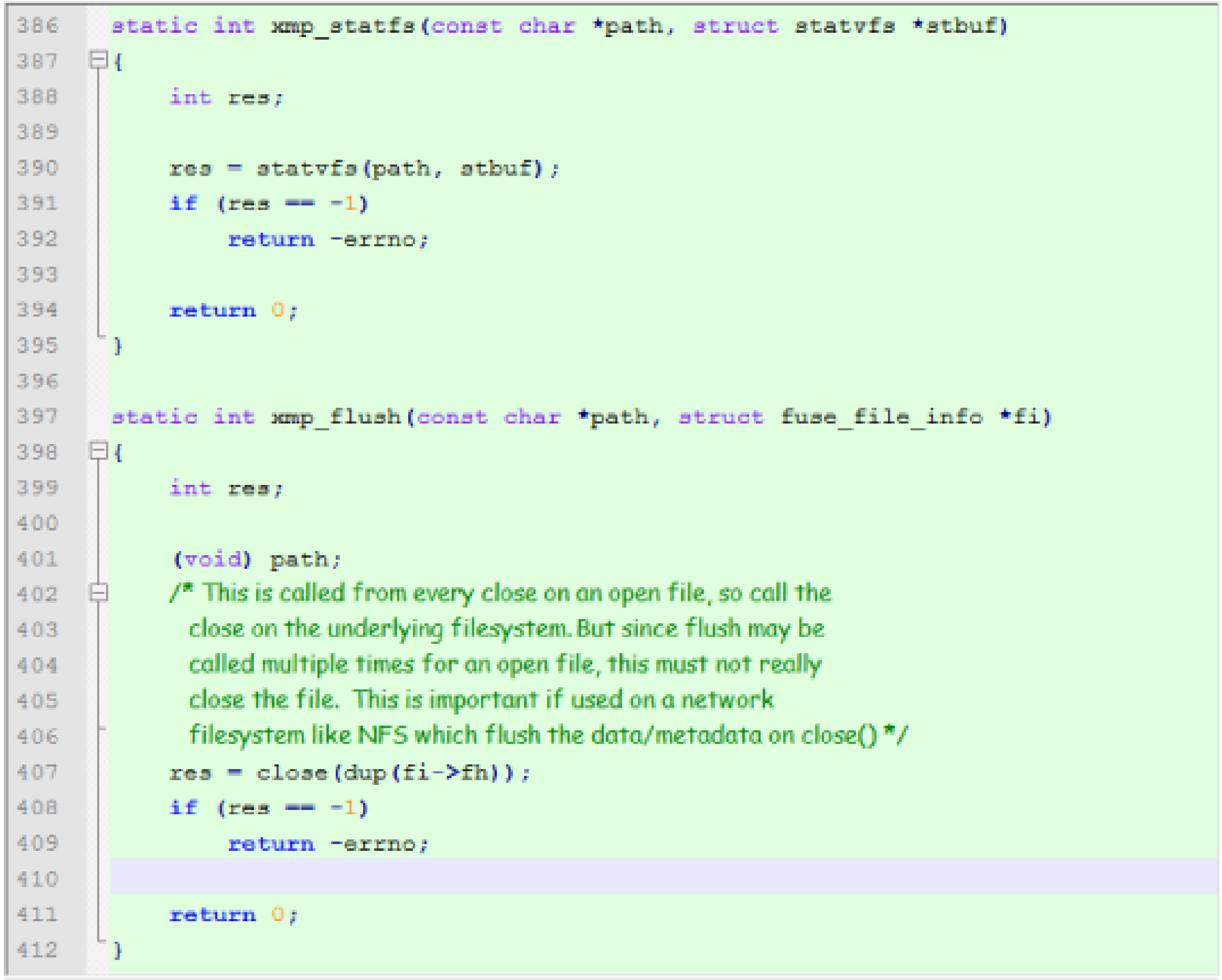
```c
static int xmp_read_buf(const char *path, struct fuse_bufvec **bufp,
            size_t size, off_t offset, struct fuse_file_info *fi)
{
    struct fuse_bufvec *src;

    (void) path;

    src = malloc(sizeof(struct fuse_bufvec));
    if (src == NULL)
        return -ENOMEM;

    *src = FUSE_BUFVEC_INIT(size);

    src->buf[0].flags = FUSE_BUF_IS_FD | FUSE_BUF_FD_SEEK;
    src->buf[0].fd = fi->fh;
    src->buf[0].pos = offset;

    *bufp = src;

    return 0;
}
```

xmp_write(), xmp_write_buf()



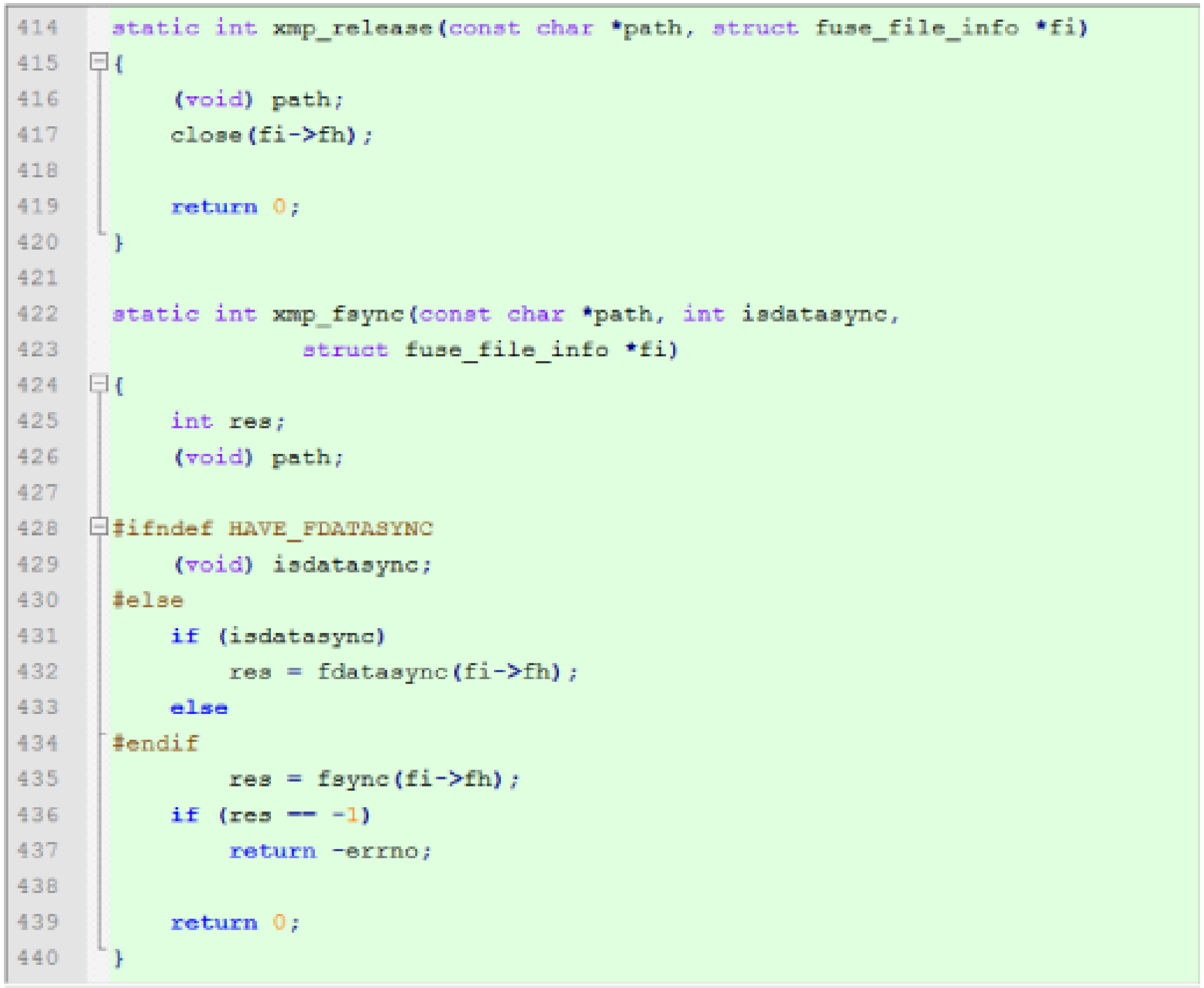
```
static int xmp_write(const char *path, const char *buf, size_t size,
             off_t offset, struct fuse_file_info *fi)
{
    int res;

    (void) path;
    res = pwrite(fi->fh, buf, size, offset);
    if (res == -1)
        res = -errno;

    return res;
}

static int xmp_write_buf(const char *path, struct fuse_bufvec *buf,
             off_t offset, struct fuse_file_info *fi)
{
    struct fuse_bufvec dst = FUSE_BUFVEC_INIT(fuse_buf_size(buf));

    (void) path;

    dst.buf[0].flags = FUSE_BUF_IS_FD | FUSE_BUF_FD_SEEK;
    dst.buf[0].fd = fi->fh;
    dst.buf[0].pos = offset;

    return fuse_buf_copy(&dst, buf, FUSE_BUF_SPLICE_NONBLOCK);
}
```

xmp_statfs(), xmp_flush()



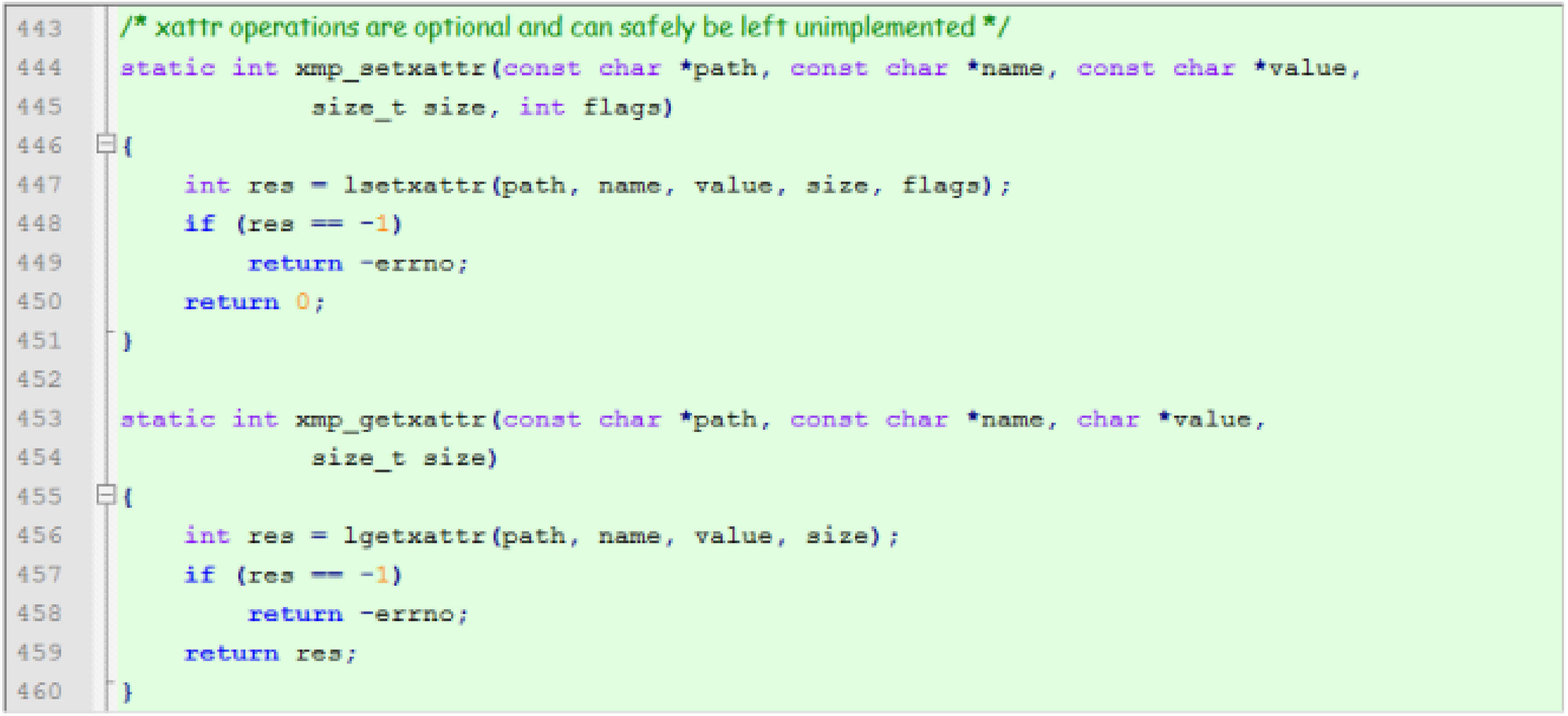
```
static int xmp_statfs(const char *path, struct statvfs *stbuf)
{
    int res;

    res = statvfs(path, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}

static int xmp_flush(const char *path, struct fuse_file_info *fi)
{
    int res;

    (void) path;
    /* This is called from every close on an open file, so call the
       close on the underlying filesystem. But since flush may be
       called multiple times for an open file, this must not really
       close the file.  This is important if used on a network
       filesystem like NFS which flush the data/metadata on close() */
    res = close(dup(fi->fh));
    if (res == -1)
        return -errno;

    return 0;
}
```

xmp_release(), xmp_fsync()



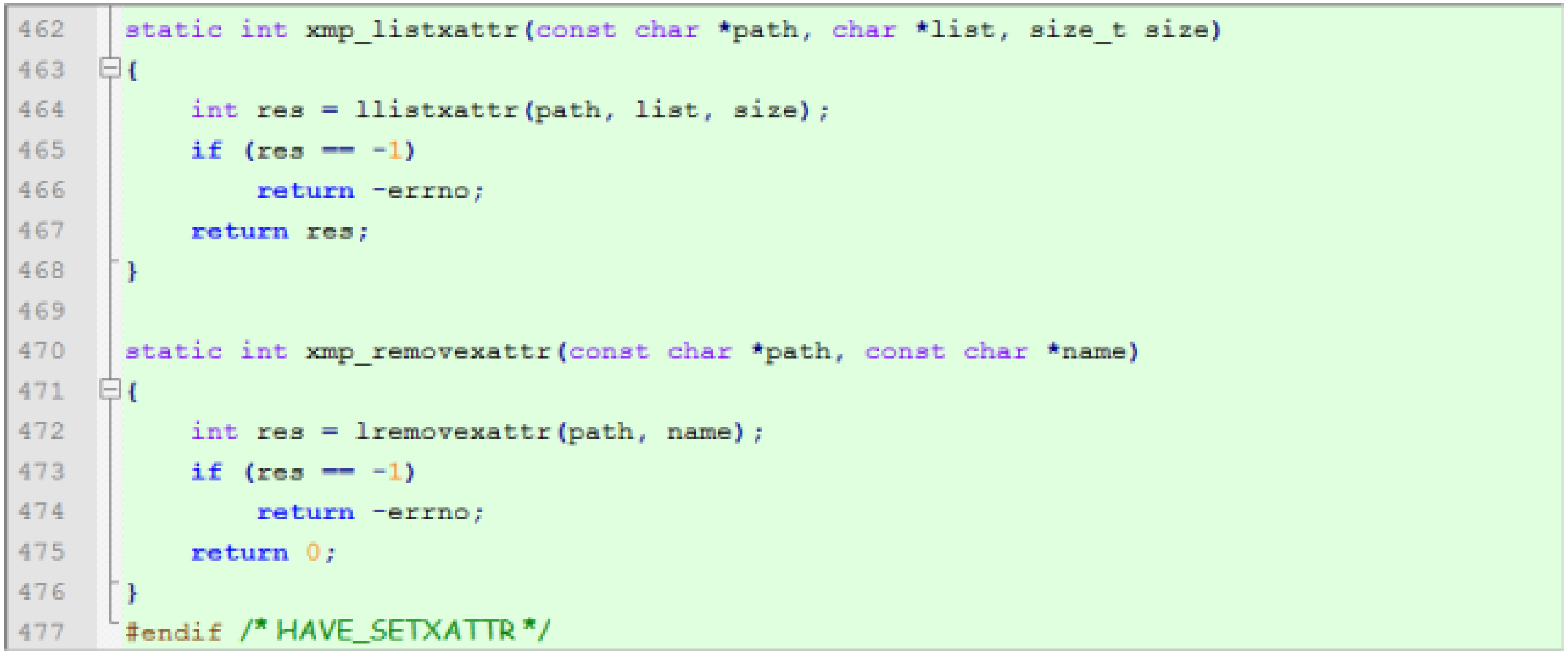
```
static int xmp_release(const char *path, struct fuse_file_info *fi)
{
	(void) path;
	close(fi->fh);

	return 0;
}

static int xmp_fsync(const char *path, int isdatasync,
		     struct fuse_file_info *fi)
{
	int res;
	(void) path;

#ifndef HAVE_FDATASYNC
	(void) isdatasync;
#else
	if (isdatasync)
		res = fdatasync(fi->fh);
	else
#endif
		res = fsync(fi->fh);
	if (res == -1)
		return -errno;

	return 0;
}
```

xmp_setattr(), xmp_getattr()



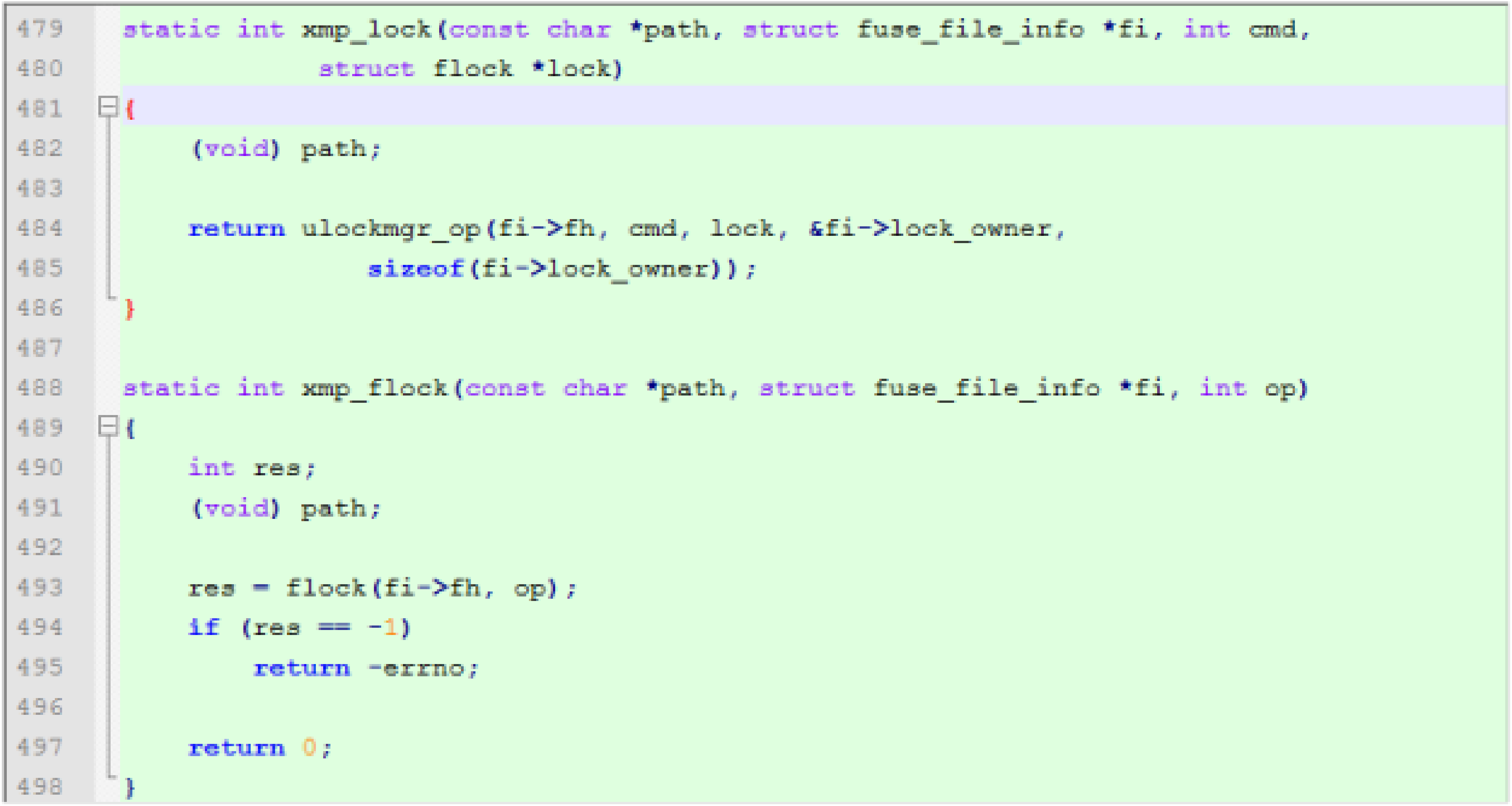
```
/* xattr operations are optional and can safely be left unimplemented */
static int xmp_setxattr(const char *path, const char *name, const char *value,
            size_t size, int flags)
{
    int res = lsetxattr(path, name, value, size, flags);
    if (res == -1)
        return -errno;
    return 0;
}

static int xmp_getxattr(const char *path, const char *name, char *value,
            size_t size)
{
    int res = lgetxattr(path, name, value, size);
    if (res == -1)
        return -errno;
    return res;
}
```

xmp_listattr(), xmp_removexattr()



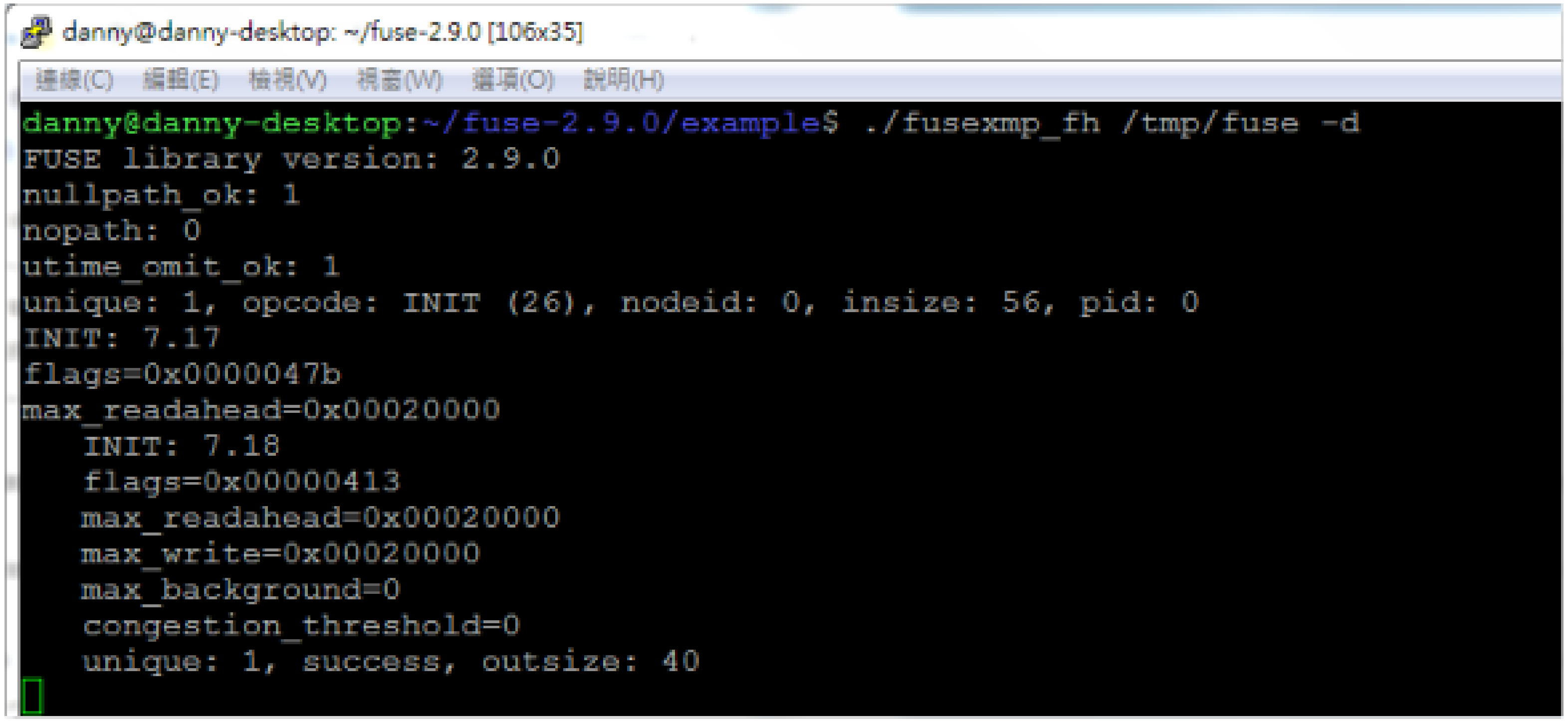
```
static int xmp_listxattr(const char *path, char *list, size_t size)
{
    int res = llistxattr(path, list, size);
    if (res == -1)
        return -errno;
    return res;
}

static int xmp_removexattr(const char *path, const char *name)
{
    int res = lremovexattr(path, name);
    if (res == -1)
        return -errno;
    return 0;
}
#endif /* HAVE_SETXATTR */
```

xmp_lock(), xmp_flock()

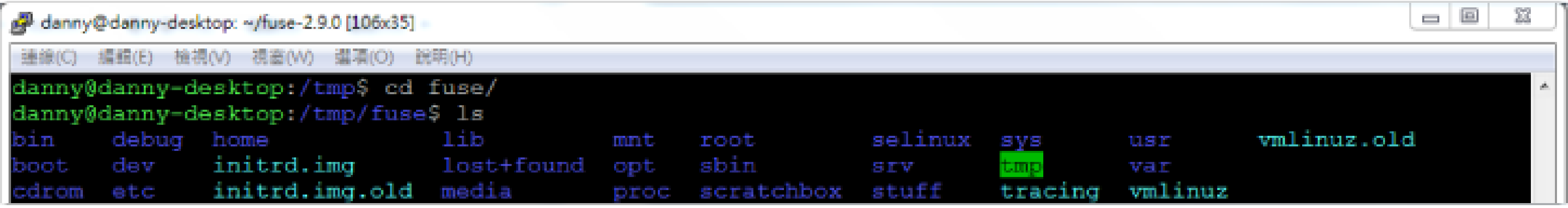


```c
static int xmp_lock(const char *path, struct fuse_file_info *fi, int cmd,
            struct flock *lock)
{
    (void) path;

    return ulockmgr_op(fi->fh, cmd, lock, &fi->lock_owner,
               sizeof(fi->lock_owner));
}

static int xmp_flock(const char *path, struct fuse_file_info *fi, int op)
{
    int res;
    (void) path;

    res = flock(fi->fh, op);
    if (res == -1)
        return -errno;

    return 0;
}
```

Example2: fusexmp_fh.c 執行


danny@danny-desktop: ~/fuse-2.9.0 [106x35]
連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

```
danny@danny-desktop:~/fuse-2.9.0/example$ ./fusexmp_fh /tmp/fuse -d
FUSE library version: 2.9.0
nullpath_ok: 1
nopath: 0
utime_omit_ok: 1
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.17
flags=0x0000047b
max_readahead=0x00020000
   INIT: 7.18
   flags=0x00000413
   max_readahead=0x00020000
   max_write=0x00020000
   max_background=0
   congestion_threshold=0
   unique: 1, success, outsize: 40
```


danny@danny-desktop: ~/fuse-2.9.0 [106x35]
連線(C) 編輯(E) 檢視(V) 視窗(W) 選項(O) 說明(H)

```
danny@danny-desktop:/tmp$ cd fuse/
danny@danny-desktop:/tmp/fuse$ ls
bin     debug  home            lib          mnt   root         selinux  sys      usr      vmlinuz.old
boot    dev    initrd.img      lost+found   opt   sbin         srv      tmp      var
cdrom   etc    initrd.img.old  media        proc  scratchbox   stuff    tracing  vmlinuz
```

The End

Thank you for your listening