文件系统是一种用来存储和组织计算机文件、目录及其包含的数据的方法，它使文件、目录以及数据的查找和访问得到简化。如果您正在使用一台计算机，很可能使用了多个文件系统。文件系统能提供丰富的扩展能力。它可以编写成底层文件系统的一个封装程序，从而对其中的数据进行管理，并提供一个增强的、具有丰富特性的文件系统（例如 cvsfs-fuse，它为 CVS 提供了一个文件系统的接口；或 Wayback 文件系统，它提供了一种用于保留原始数据文件的文件备份机制）。

在用户空间的文件系统出现之前，文件系统的开发曾是内核开发人员的工作。创建文件系统需要了解内核编程和内核技术（例如 vfs）方面的知识。调试则需要 C 和 C++ 方面的专业技能。但是其他开发人员需要熟练地操作文件系统以添加个性化特性（例如添加历史记录或转发缓存）及对其改进。

一、FUSE 简介

使用 FUSE 您可以开发功能完备的文件系统：其具有简单的 API 库，可以被非特权用户访问，并可以安全的实施。更重要的是，FUSE 以往的表现充分证明了其稳定性。

使用 FUSE，您可以像可执行二进制文件一样来开发文件系统，它们需要链接到 FUSE 库上 —— 换言之，这个文件系统框架并不需要您了解文件系统的内幕和内核模块编程的知识。

就文件系统来说，用户空间的文件系统就不再是新奇的设计了。用户空间文件系统的商业实现与学术实现的实例包括：

* LUFS 是一个混合用户空间的文件系统框架，它对用于任何应用程序无数的文件系统提供透明支持。大部分LUFS 包括一个内核模块和一个用户空间的守护进程。从根本上来说，它将大部分 VFS 调用都委托给一个专用的守护进程来处理。
* UserFS 让用户进程可以像普通的文件系统一样进行加载。这种概念性的原型提供了 ftpfs，这可以使用文件系统接口提供匿名 FTP 访问。
* Ufo Project 是为 Solaris 提供的一个全局文件系统，它允许用户将远程文件真正当作本地文件一样对待。
* OpenAFS 是 Andrew FileSystem 的一个开源版本。
* CIFS 是 Common Internet FileSystem 的简称。

与这些商业实现和学术实现不同，FUSE 将这种文件系统的设计能力带到了 Linux 中来。由于 FUSE 使用的是可执行程序（而不像 LUFS 一样使用的是共享对象），因此可以简化程序的调试和开发。FUSE 可以在linux 的内核上使用，现在可以支持 Java™ 绑定，因此您可以不必限定于使用 C 和 C++ 来编写文件系统了。（有关更多使用 FUSE 的用户层的文件系统的内容，请参阅 [参考资料](https://www.ibm.com/developerworks/cn/linux/l-fuse/#artrelatedtopics)。）

要在 FUSE 中创建一个文件系统，您需要安装一个 FUSE 内核模块，然后使用 FUSE 库和 API 来创建自己的文件系统。

**二、FUSE operations**

FUSE使用fuse\_operations来给用户提供编程结构，让用户通过注册自己编写的函数到该结构体来实现自己的文件系统。

struct fuse\_operations {

int (\*getattr) (const char \*, struct stat \*, struct fuse\_file\_info \*fi);

int (\*readlink) (const char \*, char \*, size\_t);

int (\*mknod) (const char \*, mode\_t, dev\_t);

int (\*mkdir) (const char \*, mode\_t);

int (\*unlink) (const char \*);

int (\*rmdir) (const char \*);

int (\*symlink) (const char \*, const char \*);

int (\*rename) (const char \*, const char \*, unsigned int flags);

int (\*link) (const char \*, const char \*);

int (\*chmod) (const char \*, mode\_t, struct fuse\_file\_info \*fi);

int (\*chown) (const char \*, uid\_t, gid\_t, struct fuse\_file\_info \*fi);

int (\*truncate) (const char \*, off\_t, struct fuse\_file\_info \*fi);

int (\*open) (const char \*, struct fuse\_file\_info \*);

int (\*read) (const char \*, char \*, size\_t, off\_t,

struct fuse\_file\_info \*);

int (\*write) (const char \*, const char \*, size\_t, off\_t,

struct fuse\_file\_info \*);

int (\*statfs) (const char \*, struct statvfs \*);

int (\*flush) (const char \*, struct fuse\_file\_info \*);

int (\*release) (const char \*, struct fuse\_file\_info \*);

int (\*fsync) (const char \*, int, struct fuse\_file\_info \*);

int (\*setxattr) (const char \*, const char \*, const char \*, size\_t, int);

int (\*getxattr) (const char \*, const char \*, char \*, size\_t);

int (\*listxattr) (const char \*, char \*, size\_t);

int (\*removexattr) (const char \*, const char \*);

int (\*opendir) (const char \*, struct fuse\_file\_info \*);

int (\*readdir) (const char \*, void \*, fuse\_fill\_dir\_t, off\_t,

struct fuse\_file\_info \*, enum fuse\_readdir\_flags);

int (\*releasedir) (const char \*, struct fuse\_file\_info \*);

int (\*fsyncdir) (const char \*, int, struct fuse\_file\_info \*);

void \*(\*init) (struct fuse\_conn\_info \*conn,

struct fuse\_config \*cfg);

void (\*destroy) (void \*private\_data);

int (\*access) (const char \*, int);

int (\*create) (const char \*, mode\_t, struct fuse\_file\_info \*);

int (\*lock) (const char \*, struct fuse\_file\_info \*, int cmd,

struct flock \*);

int (\*utimens) (const char \*, const struct timespec tv[2],

struct fuse\_file\_info \*fi);

int (\*bmap) (const char \*, size\_t blocksize, uint64\_t \*idx);

int (\*ioctl) (const char \*, int cmd, void \*arg,

struct fuse\_file\_info \*, unsigned int flags, void \*data);

int (\*poll) (const char \*, struct fuse\_file\_info \*,

struct fuse\_pollhandle \*ph, unsigned \*reventsp);

int (\*write\_buf) (const char \*, struct fuse\_bufvec \*buf, off\_t off,

struct fuse\_file\_info \*);

int (\*read\_buf) (const char \*, struct fuse\_bufvec \*\*bufp,

size\_t size, off\_t off, struct fuse\_file\_info \*);

int (\*flock) (const char \*, struct fuse\_file\_info \*, int op);

int (\*fallocate) (const char \*, int, off\_t, off\_t,

struct fuse\_file\_info \*);

};

**三、hello示例**文件系统**分析**

  FUSE在实例目录example下有一些示例文件系统，通过阅读这些示例文件系统可以掌握FUSE用户态文件系统的编写规范。下面以hello.c为示分析FUSE的编写规范：

/\*

FUSE: Filesystem in Userspace

Copyright (C) 2001-2007 Miklos Szeredi <miklos@szeredi.hu>

This program can be distributed under the terms of the GNU GPL.

See the file COPYING.

\*/

/\*\* @file

\*

\* minimal example filesystem using high-level API

\*

\* Compile with:

\*

\* gcc -Wall hello.c `pkg-config fuse3 --cflags --libs` -o hello

\*

\* ## Source code ##

\* \include hello.c

\*/

#define FUSE\_USE\_VERSION 31

#include <fuse.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <fcntl.h>

#include <stddef.h>

#include <assert.h>

/\*

\* Command line options

\*

\* We can't set default values for the char\* fields here because

\* fuse\_opt\_parse would attempt to free() them when the user specifies

\* different values on the command line.

\*/

static struct options {

const char \*filename;

const char \*contents;

int show\_help;

} options;

#define OPTION(t, p) \

{ t, offsetof(struct options, p), 1 }

static const struct fuse\_opt option\_spec[] = {

OPTION("--name=%s", filename),

OPTION("--contents=%s", contents),

OPTION("-h", show\_help),

OPTION("--help", show\_help),

FUSE\_OPT\_END

};

static void \*hello\_init(struct fuse\_conn\_info \*conn,

struct fuse\_config \*cfg)

{

(void) conn;

cfg->kernel\_cache = 1;

return NULL;

}

static int hello\_getattr(const char \*path, struct stat \*stbuf,

struct fuse\_file\_info \*fi)

{

(void) fi;

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+1, options.filename) == 0) {

stbuf->st\_mode = S\_IFREG | 0444;

stbuf->st\_nlink = 1;

stbuf->st\_size = strlen(options.contents);//文件长度

} 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,

enum fuse\_readdir\_flags flags)

{

(void) offset;

(void) fi;

(void) flags;

if (strcmp(path, "/") != 0)

return -ENOENT;

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

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

filler(buf, options.filename, NULL, 0, 0);

return 0;

}

 /-\* fill的定义：  
     typedef int (\*fuse\_fill\_dir\_t) (void \*buf, const char \*name, const struct stat \*stbuf, off\_t off);  
     其作用是在readdir函数中增加一个目录项  
 \*-/

static int hello\_open(const char \*path, struct fuse\_file\_info \*fi) //用于打开hello文件

{

if (strcmp(path+1, options.filename) != 0) //不是hello文件

return -ENOENT;

if ((fi->flags & O\_ACCMODE) != O\_RDONLY)

return -EACCES;

return 0;

}

//读取hello文件时的操作，

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+1, options.filename) != 0)

return -ENOENT;

len = strlen(options.contents);

if (offset < len) {

if (offset + size > len)

size = len - offset;

memcpy(buf, options.contents + offset, size);

} else

size = 0;

return size;

}

static struct fuse\_operations hello\_oper = {

.init = hello\_init,

.getattr = hello\_getattr,

.readdir = hello\_readdir,

.open = hello\_open,

.read = hello\_read,

};

static void show\_help(const char \*progname)

{

printf("usage: %s [options] <mountpoint>\n\n", progname);

printf("File-system specific options:\n"

" --name=<s> Name of the \"hello\" file\n"

" (default: \"hello\")\n"

" --contents=<s> Contents \"hello\" file\n"

" (default \"Hello, World!\\n\")\n"

"\n");

}

int main(int argc, char \*argv[])

{

int ret;

struct fuse\_args args = FUSE\_ARGS\_INIT(argc, argv);

/\* Set defaults -- we have to use strdup so that

fuse\_opt\_parse can free the defaults if other

values are specified \*/

options.filename = strdup("hello");

options.contents = strdup("Hello World!\n");

/\* Parse options \*/

if (fuse\_opt\_parse(&args, &options, option\_spec, NULL) == -1)

return 1;

/\* When --help is specified, first print our own file-system

specific help text, then signal fuse\_main to show

additional help (by adding `--help` to the options again)

without usage: line (by setting argv[0] to the empty

string) \*/

if (options.show\_help) {

show\_help(argv[0]);

assert(fuse\_opt\_add\_arg(&args, "--help") == 0);

args.argv[0] = (char\*) "";

}

ret = fuse\_main(args.argc, args.argv, &hello\_oper, NULL);

fuse\_opt\_free\_args(&args);

return ret;

}

**终端运行：**  
   
 ~/fuse/example$ mkdir /tmp/fuse

~/fuse/example$ ./hello /tmp/fuse

~/fuse/example$ ls -l /tmp/fuse hello

-r--r--r-- 1 root root 13 1970-01-01 07:00 hello

~/fuse/example$ cat /tmp/fuse/hello Hello World!

~/fuse/example$ fusermount -u /tmp/fuse

  通过上述的分析可以知道，使用FUSE必须要自己实现对文件或目录的操作， 系统调用也会最终调用到用户自己实现的函数。

  用户实现的函数需要在结构体fuse\_operations中注册。而在main()函数中，用户只需要调用fuse\_main()函数就可以了，剩下的复杂工作可以交给FUSE。