**一、FUSE简介**

FUSE（用户空间文件系统）是这样一个框架：它使得FUSE用户在用户态下编写文件系统成为可能，而不必和内核打交道。

FUSE由三个部分组成：linux内核模块、FUSE库 以及mount 工具。

用户关心的只是FUSE库和mount工具，内核模块仅仅提供kernel的接入口，给了文件系统一个框架，而文件系统本身的主要实现代示位于用户空间中。FUSE库给用户提供了编程的接口，而mount工具则用于挂在用户编写的文件系统。

FUSE起初是为了研究AVFS(A Virtual Filesystem)而设计的，而现在已经成为 SourceForge的一个独立项目，目前适用的平台有Linux, FreeBSD, NetBSD, OpenSolaris和 Mac OS X。

官方的linux kernel版本到 2.6.14 才添加了FUSE模块，因此 2.4 的内核模块下，用户如果要在FUSE中创建一个文件系统，需要先安装一个FUSE内核模块，然后使用 FUSE库和API来创建。

**二、FUSE特性**

1 库文件和API简单，极大地方便了用户的使用

2 安装简便，不需要加补丁或者重新编译 kernel

3 执行安全，使用稳定

4 高效，相对于其它用户态文件系统实现

5 非特权用户可以使用

6 基于 linux2.4.x 和 2.6.x 内核，现在可以支持JavaTM 绑定，不必限定使用C和C++来 编写文件系统。

**三、源代码目录**

  1 ./doc 包含FUSE相关文档  
  2 ./include 包含了FUSE API头，对创建文件系统有用，主要用fuse.h  
  3 ./lib 存放FUSE库  
  4 ./util 包含了FUSE工具库  
  5 ./example 参考的例子

**四、安装**

  FUSE的示示安装类似于其他软件，只需要在FUSE的目录下以root身份执行执行如下命令即可，如：

./configure --disable-kernel-module

make

make install

如果在配置时提示

configure: error: no acceptable C compiler found in $PATH

则需要安装gcc:

yum install gcc

**五、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 \*);  
     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 \*);  
     int (\*link) (const char \*, const char \*);  
     int (\*chmod) (const char \*, mode\_t);  
     int (\*chown) (const char \*, uid\_t, gid\_t);  
     int (\*truncate) (const char \*, off\_t);  
     int (\*utime) (const char \*, struct utimbuf \*);  
     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 \*);  
     int (\*releasedir) (const char \*, struct fuse\_file\_info \*);  
     int (\*fsyncdir) (const char \*, int, struct fuse\_file\_info \*);  
     void \*(\*init) (struct fuse\_conn\_info \*conn);  
     void (\*destroy) (void \*); int (\*access) (const char \*, int);  
     int (\*create) (const char \*, mode\_t, struct fuse\_file\_info \*);  
     int (\*ftruncate) (const char \*, off\_t, struct fuse\_file\_info \*);  
     int (\*fgetattr) (const char \*, struct stat \*, 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]);  
     int (\*bmap) (const char \*, size\_t blocksize, uint64\_t \*idx);  
 };

**六、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：

 #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;  
}

 /-\* 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) {  
    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;  
}  
   
 /-\*注册上面定义的函数\*-/  
 static struct fuse\_operations hello\_oper =  
 {  
   .getattr = hello\_getattr,  
   .readdir = hello\_readdir,  
   .open    = hello\_open,  
   .read    = hello\_read,  
 };  
   
 /-\*用户只需要调用fuse\_main()，剩下的事就交给FUSE了\*-/  
 int main(int argc, char \*argv[])  
 {  
     return fuse\_main(argc, argv, &hello\_oper, NULL);  
 }

**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。