# 添加一个加密文件系统

# 实验目的

文件系统是操作系统中最直观的部分，因为用户可以通过文件直接地和操作系统交互，操作系统也必须为用户提供数据计算、数据存储的功能。本实验通过添加一个文件系统，进一步理解Linux中的文件系统原理及其实现。

* 深入理解操作系统文件系统原理
* 学习理解Linux的VFS文件系统管理技术
* 学习理解Linux的ext2文件系统实现技术
* 设计和实现加密文件系统

# 实验内容

添加一个类似于ext2，但对磁盘上的数据块进行加密的文件系统myext2。实验主要内容：

* + 添加一个类似ext2的文件系统myext2
  + 修改myext2的magic number
  + 添加文件系统创建工具
  + 添加加密文件系统操作，包括read\_crypt, write\_crypt，使其增加对加密数据的读写。

# 实验指导

### 1. 问题描述

本实验的内容是要添加一个类似于ext2的自定义文件系统myext2。myext2文件系统的描述如下：

1、myext2文件系统的物理格式定义与ext2基本一致，但 myext2的magic number是0x6666，而ext2的magic number是0xEF53。

2、myext2是ext2的定制版本，它不但支持原来ext2文件系统的部分操作，还添加了用加密数据进行读写的操作。

### 2. 实验步骤

提示：下面的操作步骤以4.6.0版本的内核为例，其它版本内核可能会有所区别。下面的操作用户需要root权限

#### 2.1 添加一个类似ext2的文件系统myext2

要添加一个类似ext2的文件系统myext2，首先是确定实现ext2文件系统的内核源码是由哪些文件组成。Linux源代码结构很清楚地告诉我们：fs/ext2目录下的所有文件是属于ext2文件系统的。再检查一下这些文件所包含的头文件，可以初步总结出来Linux源代码中属于ext2文件系统的有：

fs/ext2/acl.c

fs/ext2/acl.h

fs/ext2/balloc.c

fs/ext2/bitmap.c

fs/ext2/dir.c

fs/ext2/ext2.h

fs/ext2/file.c

……

include/linux/ext2\_fs.h

接下来开始添加myext2文件系统的源代码到Linux源代码。把ext2部分的源代码克隆到myext2去，即复制一份以上所列的ext2源代码文件给myext2用。按照Linux源代码的组织结构，把myext2文件系统的源代码存放到fs/myext2下，头文件放到include/linux下。在Linux的shell下，执行如下操作：

#cd /usr/src/linux /\*内核源代码目录，假设内核源代码解压在主目录的linux子目录\*/

#cd fs

#cp –R ext2 myext2

#cd /usr/src/linux/fs/myext2

#mv ext2.h myext2.h

#cd /lib/modules/$(uname -r)/build/include/linux

#cp ext2\_fs.h myext2\_fs.h

#cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops

#cp ext2-atomic.h myext2-atomic.h

#cp ext2-atomic-setbit.h myext2-atomic-setbit.h

这样就完成了克隆文件系统工作的第一步——源代码复制。对于克隆文件系统来说，这样当然还远远不够，因为文件里面的数据结构名、函数名、以及相关的一些宏等内容还没有根据myext2改掉，连编译都通不过。

下面开始克隆文件系统的第二步：修改上面添加的文件的内容。为了简单起见，做了一个最简单的替换：将原来“EXT2”替换成“MYEXT2”；将原来的“ext2”替换成“myext2”。

对于fs/myext2下面文件中字符串的替换，也可以使用下面的脚本：

#!/bin/bash

SCRIPT=substitute.sh

for f in \*

do

if [ $f = $SCRIPT ]

then

echo "skip $f"

continue

fi

echo -n "substitute ext2 to myext2 in $f..."

cat $f | sed 's/ext2/myext2/g' > ${f}\_tmp

mv ${f}\_tmp $f

echo "done"

echo -n "substitute EXT2 to MYEXT2 in $f..."

cat $f | sed 's/EXT2/MYEXT2/g' > ${f}\_tmp

mv ${f}\_tmp $f

echo "done"

done

把这个脚本命名为substitute.sh，放在fs/myext2下面，加上可执行权限，运行之后就可以把当前目录里所有文件里面的“ext2”和“EXT2”都替换成对应的“myext2”和“MYEXT2”。

特别提示：

* 不要拷贝word文档中的substitute.sh脚本，在Linux环境下重新输入一遍，substitute.sh脚本程序只能运行一次。ubuntu环境：sudo bash substitute.sh。
* 先删除fs/myext2目录下的 \*.o文件，再运行脚本程序。
* 在下面的替换或修改内核代码时可以使用gedit编辑器，要注意大小写。

用编辑器的替换功能，把/lib/modules/$(uname -r)/build/include/linux/myext2\_fs.h,和/lib/modules/$(uname -r)/build/include/asm-generic/bitops/下的myext2-atomic.h与myext2-atomic-setbit.h文件中的“ext2”、“EXT2”分别替换成“myext2”、“MYEXT2”

在/lib/modules/$(uname -r)/build/include/asm-generic/bitops.h文件中添加：

#include <asm-generic/bitops/myext2-atomic.h>

在/lib/modules/$(uname -r)/build/arch/x86/include/asm/bitops.h文件中添加：

#include <asm-generic/bitops/myext2-atomic-setbit.h>

在/lib/modules/$(uname -r)/build/include/uapi/linux/magic.h 文件中添加：#define MYEXT2\_SUPER\_MAGIC 0xEF53

源代码的修改工作到此结束。接下来就是第三步工作—一把myext2编译成内核模块。

要编译内核模块，首先要生成一个Makefile文件。我们可以修改myext2/Makefile文件，

修改后的Makefile文件如下：

#

# Makefile for the linux myext2-filesystem routines.

#

obj-m := myext2.o

myext2-y := balloc.o dir.o file.o ialloc.o inode.o \

ioctl.o namei.o super.o symlink.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

default:

make -C $(KDIR) M=$(PWD) modules

编译内核模块的命令是make，在myext2目录下执行命令：

#make

编译好模块后，使用insmod命令加载模块：

#insmod myext2.ko

查看一下myext2文件系统是否加载成功：

#cat /proc/filesystems |grep myext2

确认myext2文件系统加载成功后，可以对添加的myext2文件系统进行测试了，输入命令cd先把当前目录设置成主目录。

对添加的myext2文件系统测试命令如下：

#dd if=/dev/zero of=myfs bs=1M count=1

#/sbin/mkfs.ext2 myfs

#mount -t myext2 -o loop ./myfs /mnt

#mount

……

…… on /mnt type myext2 (rw)

#umount /mnt

#mount -t ext2 -o loop ./myfs /mnt

#mount

……

…… on /mnt type ext2 (rw)

#umount /mnt

#rmmod myext2 /\*卸载模块\*/

#### 2.2 修改myext2的magic number

在上面做的基础上。找到myext2的magic number，并将其改为0x6666：

4.6.0内核版本，这个值在include/uapi/linux/magic.h文件中。

- #define MYEXT2\_SUPER\_MAGIC 0xEF53

+ #define MYEXT2\_SUPER\_MAGIC 0x6666

改动完成之后，再用make重新编译内核模块，使用命令insmod安装编译好的myext2.ko内核模块。

在我们测试这个部分之前，我们需要写个小程序changeMN.c，来修改我们创建的myfs文件系统的magic number。因为它必须和内核中记录myext2文件系统的magic number匹配，myfs文件系统才能被正确地mount。

changeMN.c程序可以在课程网站中下载。这个程序经过编译后产生的可执行程序名字为changeMN。

下面我们开始测试：

#dd if=/dev/zero of=myfs bs=1M count=1

#/sbin/mkfs.ext2 myfs

#./changeMN myfs

#mount -t myext2 -o loop ./fs.new /mnt

**这里与书上不一样的**

#mount

…… on /mnt type myext2 (rw)

#sudo umount /mnt

# sudo mount -t ext2 -o loop ./fs.new /mnt

mount: wrong fs type, bad option, bad superblock on /dev/loop0, …

# rmmod myext2

#### 2.3 修改文件系统操作

myext2只是一个实验性质的文件系统，我们希望它只要能支持简单的文件操作即可。因此在完成了myext2的总体框架以后，我们来修改掉myext2支持的一些操作，来加深对操作系统对文件系统的操作的理解。下面以裁减myext2的mknod操作为例，了解这个过程的实现流程。

Linux将所有的对块设备、字符设备和命名管道的操作，都看成对文件的操作。mknod操作是用来产生那些块设备、字符设备和命名管道所对应的节点文件。在ext2文件系统中它的实现函数如下：

fs/ext2/namei.c, line 141

static int ext2\_mknod (struct inode \* dir, struct dentry \*dentry, int mode, dev\_t rdev)

{

struct inode \* inode;

int err;

if (!new\_valid\_dev(rdev))

return -EINVAL;

inode = ext2\_new\_inode (dir, mode);

err = PTR\_ERR(inode);

if (!IS\_ERR(inode)) {

init\_special\_inode(inode, inode->i\_mode, rdev);

#ifdef CONFIG\_EXT2\_FS\_XATTR

inode->i\_op = &ext2\_special\_inode\_operations;

#endif

mark\_inode\_dirty(inode);

err = ext2\_add\_nondir(dentry, inode);

}

return err;

}

它定义在结构ext2\_dir\_inode\_operations中：

fs/ext2/namei.c, line 428

struct inode\_operations ext2\_dir\_inode\_operations = {

.create = ext2\_create,

.lookup = ext2\_lookup,

.link = ext2\_link,

.unlink = ext2\_unlink,

.symlink = ext2\_symlink,

.mkdir = ext2\_mkdir,

.rmdir = ext2\_rmdir,

.mknod = ext2\_mknod,

.rename = ext2\_rename,

#ifdef CONFIG\_EXT2\_FS\_XATTR

.setxattr = generic\_setxattr,

.getxattr = generic\_getxattr,

.listxattr = ext2\_listxattr,

.removexattr = generic\_removexattr,

#endif

.setattr = ext2\_setattr,

.permission = ext2\_permission,

};

当然，从ext2克隆过去的myext2的myext2\_mknod，以及myext2\_dir\_inode\_operations和上面的程序是一样的。对于mknod函数，我们在myext2中作如下修改：

fs/myext2/namei.c

static int myext2\_mknod (struct inode \* dir, struct dentry \*dentry, int mode, int rdev)

{

printk(KERN\_ERR “haha, mknod is not supported by myext2! you’ve been cheated!\n”);

return -EPERM;

/\*

…..

把其它代码注释

\*/

}

添加的程序中：

第一行 打印信息，说明mknod操作不被支持。

第二行 将错误号为EPERM的结果返回给shell，即告诉shell，在myext2文件系统中，mknod不被支持。

修改完毕，再用make重新编译内核模块，使用命令insmod安装编译好的myext2.ko内核模块。我们在shell下执行如下测试程序：

#mount –t myext2 –o loop ./fs.new /mnt

#cd /mnt

#mknod myfifo p

mknod: `myfifo': Operation not permitted

#

第一行命令：将fs.new mount到/mnt目录下。

第二行命令：进入/mnt目录，也就是进入fs.new这个myext2文件系统。

第三行命令：执行创建一个名为myfifo的命名管道的命令。

第四、五行是执行结果：第四行是我们添加的myext2\_mknod函数的printk的结果；第五行是返回错误号EPERM结果给shell，shell捕捉到这个错误后打出的出错信息。需要注意的是，如果你是在图形界面下使用虚拟控制台，printk打印出来的信息不一定能在你的终端上显示出来，但是可以通过命令dmesg|tail来观察。

可见，我们的裁减工作取得了预期的效果。

#### 2.4. 添加文件系统创建工具

文件系统的创建对于一个文件系统来说是首要的。因为，如果不存在一个文件系统，所有对它的操作都是空操作，也是无用的操作。

其实，前面的第一小节《添加一个类似ext2的文件系统myext2》和第二小节《修改myext2的magic number》在测试实验结果的时候，已经陆陆续续地讲到了如何创建myext2文件系统。下面工作的主要目的就是将这些内容总结一下，制作出一个更快捷方便的myext2文件系统的创建工具：mkfs.myext2（名称上与mkfs.ext2保持一致）。

首先需要确定的是该程序的输入和输出。为了灵活和方便起见，我们的输入为一个文件，这个文件的大小，就是myext2文件系统的大小。输出就是带了myext2文件系统的文件。

我们在主目录下编辑如下的程序：

~/mkfs.myext2

#!/bin/bash

/sbin/losetup -d /dev/loop2

这里与教材上不一样的，以本实验指导为准。

/sbin/losetup /dev/loop2 $1

/sbin/mkfs.ext2 /dev/loop2

dd if=/dev/loop2 of=./tmpfs bs=1k count=2

./changeMN $1 ./tmpfs

dd if=./fs.new of=/dev/loop2

/sbin/losetup -d /dev/loop2

rm -f ./tmpfs

第一行 表明是shell程序。

第三行 如果有程序用了/dev/loop2了，就将它释放。

第四行 用losetup将第一个参数代表的文件装到/dev/loop2上

第五行 用mkfs.ext2格式化/dev/loop2。也就是用ext2文件系统格式格式化我们的文件系统。

第六行 将文件系统的头2K字节的内容取出来，复制到tmpfs文件里面。

第七行 调用程序changeMN读取tmpfs，复制到fs.new，并且将fs.new的magic number改成0x6666

第八行 再将2K字节的内容写回去。

第九行 把我们的文件系统从loop2中卸下来。

第十行 将临时文件删除。

我们发现mkfs.myext2脚本中的changeMN程序功能，与2.2节的changeMN功能不一样，请修改changeMN.c程序，以适合本节mkfs.myext2和下面测试的需要。

编辑完了之后，做如下测试：

# dd if=/dev/zero of=myfs bs=1M count=1

# ./mkfs.myext2 myfs (或 sudo bash mkfs.myext2 myfs )

#sudo mount –t myext2 –o loop ./myfs /mnt

# mount

/dev/loop on /mnt myext2 (rw)

#### 2.5 修改加密文件系统的read和write操作

在内核模块myext2.ko中修改file.c的代码，添加两个函数new\_sync\_read\_crypt和new\_sync\_write\_crypt，将这两个函数指针赋给myext2\_file\_operations结构中的read和write操作。在new\_sync\_write\_crypt中增加对用户传入数据buf的加密，在new\_sync\_read\_crypt中增加解密。可以使用DES等加密和解密算法。

对new\_sync\_read\_cryp函数，可以做如下修改：

ssize\_t new\_sync\_write\_crypt(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*ppos)

{

char\* mybuf = buf;

//在此处添加对长度为len的buf数据进行加密（简单移位密码，将每个字符值+25）

printk("haha encrypt %ld\n", len);

return new\_sync\_write(filp, mybuf, len, ppos);//调用默认的写函数，把加密数据写入

}

对new\_sync\_read\_cryp函数，可以做如下修改：

ssize\_t new\_sync\_read\_crypt(struct file \*filp, char \_\_user \*buf, size\_t len, loff\_t \*ppos)

{

int i;

//先调用默认的读函数读取文件数据

ssize\_t ret = new\_sync\_read(filp, buf, len, ppos);

//此处添加对文件的解密（简单移位解密，将每个字符值-25）

printk("haha encrypt %ld\n", len);

return ret;

}

上述修改完成后，再用make重新编译myext2模块，使用命令insmod安装编译好的myext2.ko内核模块。重新加载myext2内核模块，创建一个myext2文件系统，并尝试往文件系统中写入一个字符串文件。

mount -t myext2 -o loop ./fs.new /mnt/

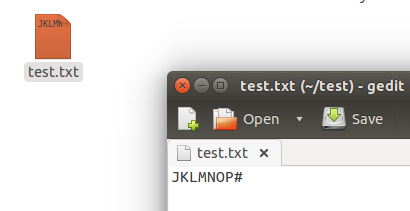
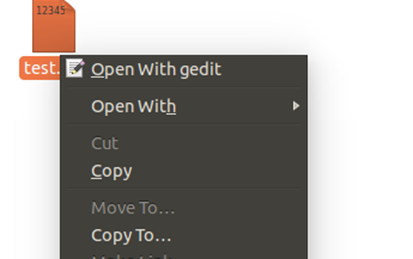
cd /mnt/

新建文件test.txt并写入字符串“1234567”，再查看test.txt文件内容：cat test.txt 。

把test.txt文件复制到主目录下：cp test.txt ~ 。

在主目录下打开test.txt文件，查看test.txt文件内容的结果？

使用文件管理器的复制，再查看结果？



我们把之前的magic number改回0xEF53。重新编译myext2模块，安装myext2.ko后，执行下面命令：

dd if=/dev/zero of=myfs bs=1M count=1

/sbin/mkfs.ext2 myfs

mount -t myext2 -o loop ./myfs /mnt

cd /mnt

echo “1234567” > test.txt

cat test.txt

cd

umount /mnt

mount -t ext2 -o loop ./myfs /mnt

cd /mnt

cat test.txt

查看实验结果，此时即使使用ext2文件系统的magic number，在myext2文件系统中创建的文件都是加密文件。

至此，文件系统部分的实验已经全部完成了。通过本实验，你对Linux整个文件系统的运作流程，如何添加一个文件系统，以及如何修改Linux对文件系统的操作，有了比较深的了解。在本实验的基础上，你完全可以发挥自己的创造性，构造出自己的文件系统，然后将它添加到Linux中。

# 撰写实验报告的要求

1. 按照实验报告模板格式撰写；
2. 整个实验过程的截图；
3. 源程序的修改部分，运行结果的截图；
4. 必须撰写实验讨论（即心得体会），内容为实验过程中遇到的问题及解决方法等。否则扣除本实验20%分数。