

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 3.18.24 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 myext 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 ~/linux-3.18.24 /* 内核源代码目录，假设内核源代码解压在主目录的 Linux-3.18.24 子目录中*/
#cd fs
#cp -R ext2 myext2
#cd ~/linux-3.18.24/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——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-file system routines.
#
obj-m := myext2.o
myext2-y := alloc.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/filesystem |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:

3.18.24 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
#mount
```

这里与书上不一样的

```
..... 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 144

```
144 static int ext2_mknod (struct inode * dir, struct dentry *dentry, int mode, dev_t rdev)
145 {
146     struct inode * inode;
147     int err;
148
149     if (!new_valid_dev(rdev))
150         return -EINVAL;
151
152     inode = ext2_new_inode (dir, mode);
153     err = PTR_ERR(inode);
154     if (!IS_ERR(inode)) {
155         init_special_inode(inode, inode->i_mode, rdev);
156 #ifdef CONFIG_EXT2_FS_XATTR
157         inode->i_op = &ext2_special_inode_operations;
158 #endif
159         mark_inode_dirty(inode);
160         err = ext2_add_nondir(dentry, inode);
161     }
162     return err;
163 }
```

它定义在结构 ext2_dir_inode_operations 中：

fs/ext2/namei.c, line 400

```
392 struct inode_operations ext2_dir_inode_operations = {
393     .create      = ext2_create,
394     .lookup      = ext2_lookup,
395     .link        = ext2_link,
396     .unlink      = ext2_unlink,
397     .symlink     = ext2_symlink,
398     .mkdir       = ext2_mkdir,
399     .rmdir       = ext2_rmdir,
400     .mknod       = ext2_mknod,
401     .rename      = ext2_rename,
402 #ifdef CONFIG_EXT2_FS_XATTR
403     .setxattr    = generic_setxattr,
404     .getxattr    = generic_getxattr,
405     .listxattr   = ext2_listxattr,
406     .removexattr = generic_removexattr,
407 #endif
}
```

```

408         .setattr      = ext2_setattr,
409         .permission    = ext2_permission,
410 };

```

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，maknod 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《添加一个和类似 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
```

```
/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
```

这里与教材上不一样的，以本实验指导为准。

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。（//以 3.18 内核为例）

对 new_sync_write_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;
}
```

//4.x 的内核可以这样实现的

把 fs/read_write.c 中的 new_sync_write 和 new_sync_read 两个函数中复制 file.c 中，并添加头文件 #include <linux/uio.h> 。//4.x 的内核，ext2_file_operations 的 read 和 write 操作函数不一样了

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/
```

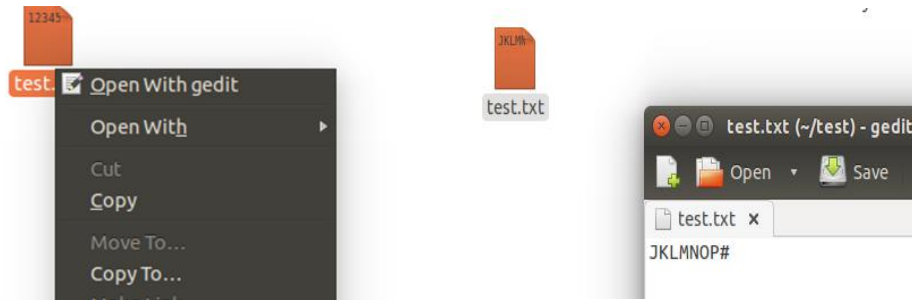
```
cd /mnt/
```

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. 按照下面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。
5. 实验报告文件格式为 word 或 pdf，你编写的源代码以文本形式附在实验报告所在的文件中，不要把 pdf 文件制作为图像格式，实验报告文件上传到“学在这里”中。

本实验评分参考：

1. 按时提交一个完整和规范的实验报告得 20 分：
 - 延迟一天扣 5 分，直至扣完 20 分，延迟一周以上本实验记 0 分。
 - 实验报告的格式规范完整，包括有实验过程中完整的截图。
2. 实验内容占 80 分：
 - 添加文件系统内核代码修改、编译内核模块成功占 10 分
 - 四个测试完成 20 分
 - 修改 read、write 完成加密 30 分
 - 讨论心得（实验过程中遇到的问题及解决方法）20 分

浙江大学实验报告

课程名称：____操作系统____实验类型：____综合型____

实验项目名称：_____

学生姓名：_____学号：_____

电子邮件地址：_____

实验日期：__年__月__日

一、实验环境

填写您的计算机配置，操作系统环境，Linux 版本

二、实验内容和结果及分析

实验设计思路

实验步骤及截图

测试程序运行结果截图

结果分析

源程序

三、讨论、心得（20 分）

在这里写：实验过程中遇到的问题及解决的方法，您做本实验体会