浙江大学实验报告

课程名称：操作系统分析及实验 实验类型：综合型/设计性

实验项目名称：实验 4 添加一个文件系统

学生姓名：胡亮泽 专业：计算机科学与技术 学号：3120102116

电子邮件地址（必须）：huliangze@yeah.net 手机：18868103152

实验日期：2014 年 1 月 8 日

一、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
- 设计和实现自定义文件系统

二、 实验内容

添加一个类似于 ext2 的自定义文件系统 myext2。实验主要内容：

a) 添加一个和 ext2 完全相同的文件系统 myext2
b) 修改 myext2 的 magic number
c) 修改文件系统操作
d) 添加文件系统创建工具

三、 主要仪器设备（必填）

VAIO E 系列，Ubuntu 14.04，

3.13.0-40-generic #69-Ubuntu SMP Thu Nov 13 17:56:26 UTC 2014 i686 i686 i686 GNU/Linux

四、操作方法和实验步骤

4.1 添加一个和 ext2 完全相同的文件系统 myext2

首先，由于自定义的 myext2 文件系统是基于 ext2 文件系统的，所以需要先复制一份 ext2 文件系统的源码作为 myext2 源码的基础。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：

1. 进入内核源代码的目录后，进入文件系统源码所在的目录 fs/ ,利用 cp –R ext2 myext2 命令将 ext2 文件系统的源码复制并命名为 myext2。
2. 进入该目录，使用 mv ext2_fs.h myext2_fs.h 将头文件重命名为 myext2_fs.h，这两个步骤如下图所示

```
root@Wholanz:/home/lk/linux# cd fs
root@Wholanz:/home/lk/linux/fs# cp -R ext2 myext2
root@Wholanz:/home/lk/linux/fs# cd myext2
root@Wholanz:/home/lk/linux/fs/myext2# mv ext2.h myext2.h
root@Wholanz:/home/lk/linux/fs/myext2#
```

3. 进入操作系统的 C 库，将一些 ext2 文件系统的文件复制并将其中和 ext2 有关的字符段重命名为 myext2.这些文件包括 ext2_fs.h, ext2-atomic.h, ext2-atomic-setbit.h. 具体操作步骤如下图所示：

```
root@Wholanz:/lib/modules/3.17.6/build/include/linux# cp ext2_fs.h myext2_fs.h
root@Wholanz:/lib/modules/3.17.6/build/include/linux#
```

复制 ext2_fs.h 文件并命名为 myext2_fs.h

```
root@Wholanz:/lib/modules/3.17.6/build/include/linux# cd ..
root@Wholanz:/lib/modules/3.17.6/build/include# cd asm-generic/bit
bitops/         bitops.h        bitsperlong.h
root@Wholanz:/lib/modules/3.17.6/build/include# cd asm-generic/bitops
root@Wholanz:/lib/modules/3.17.6/build/include/asm-generic/bitops#
```

```
root@Wholanz:/lib/modules/3.17.6/build/include# cd asm-generic/bitops
root@Wholanz:/lib/modules/3.17.6/build/include/asm-generic/bitops# cp ext2-atomi
c.h myext2-atomic.h
root@Wholanz:/lib/modules/3.17.6/build/include/asm-generic/bitops# cp ext2-atomi
c-setbit.h  myext2-atomic-setbit.h
```

复制 ext2-atomic.h 和 ext2-atomic-setbit.h 文件并命名为 myext2-atomic.h 和 myext2-atomic-setbit.h .至此源代码的克隆已经全部完成了

4. 对于克隆的源代码，需要修改源代码中 ext2 字段为 myext2。下面，使用一个 shell 脚本文件 substitute.sh 去执行一些 shell 命令来完成 fs/myext2 目录下的文件的字段替换。脚本内容如下图所示：

```
#!/bin/bash

SCRIPT=substitude.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
```

这段脚本的功能很容易看出就是将 ext2 替换成 myext2 并且将 EXT2 替换成 MYEXT2，但是这段脚本只能够运行一次，原因是当脚本执行过一次之后，脚本文件中的字段也会被替换，这样就变成了：

```
#!/bin/bash

SCRIPT=substitude.sh

for f in *
do
    if [$f = $SCRIPT]
    then
        echo "skip $f"
        continue
    fi

    echo -n "substitute myext2 to mymyext2 in $f..."
    cat $f | sed 's/myext2/mymyext2/g' > ${f}_tmp
    mv ${f}_tmp $f
    echo "done"

    echo -n "substitute MYEXT2 to MYMYEXT2 in $f..."
    cat $f | sed 's/MYEXT2/MYMYEXT2/g' > ${f}_tmp
    mv ${f}_tmp $f
    echo "done"

    done
```

可以看出，其功能为将 myext 替换为 mymyext2，以此类推，每次执行脚本后文件文件系统中的这些字段前面都会多出一个“my”.

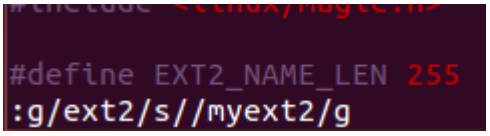
5. 执行脚本文件，如下图所示：

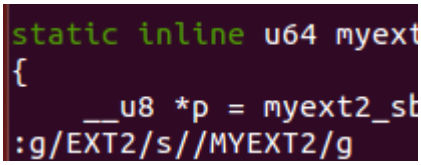
```
lk@Wholanz:~/linux/fs/myext2$ sudo vi substitute.sh
lk@Wholanz:~/linux/fs/myext2$ sudo bash substitute.sh
substitute.sh: 行 7: [acl.c: 未找到命令
substitute ext2 to myext2 in acl.c...done
substitute EXT2 to MYEXT2 in acl.c...done
substitute.sh: 行 7: [acl.h: 未找到命令
substitute ext2 to myext2 in acl.h...done
substitute EXT2 to MYEXT2 in acl.h...done
substitute.sh: 行 7: [balloc.c: 未找到命令
substitute ext2 to myext2 in balloc.c...done
substitute EXT2 to MYEXT2 in balloc.c...done
substitute.sh: 行 7: [dir.c: 未找到命令
substitute ext2 to myext2 in dir.c...done
substitute EXT2 to MYEXT2 in dir.c...done
substitute.sh: 行 7: [file.c: 未找到命令
substitute ext2 to myext2 in file.c...done
substitute EXT2 to MYEXT2 in file.c...done
substitute.sh: 行 7: [ialloc.c: 未找到命令
substitute ext2 to myext2 in ialloc.c...done
substitute EXT2 to MYEXT2 in ialloc.c...done
substitute.sh: 行 7: [inode.c: 未找到命令
```

执行后打开 myext2.h 头文件，查看其中的字段，发现 ext2 已经全部被替换为 myext2，如图：

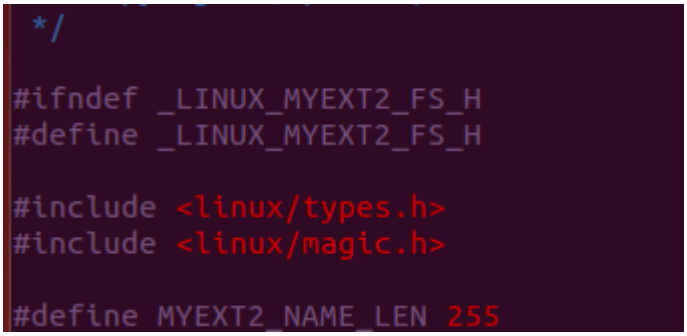
```
typedef int myext2_grpblk_t;

/* data type for filesystem-wide blocks number */
typedef unsigned long myext2_fsblk_t;

#define E2FSBLK "%lu"

struct myext2_reserve_window {
    myext2_fsblk_t      _rsv_start; /* First byte reserved */
    myext2_fsblk_t      _rsv_end;   /* Last byte reserved or 0
};

struct myext2_reserve_window_node {
    struct rb_node      rsv_node;
    __u32           rsv_goal_size;
    __u32           rsv_alloc_hit;
    struct myext2_reserve_window    rsv_window;
};
```

6. 同理，替换 myext2_fs.h, myext2-atomic.h, myext2-atomic-setbit.h 这其他三个文件中的字段。这里直接使用编辑器自带的替换功能替换，我使用的是 vim 编辑器，在命令模式下输入 g/ext2/s//myext2/g 和 g/EXT2/s//MYEXT2/g,将大小写的 ext2 全部替换为 myext2。

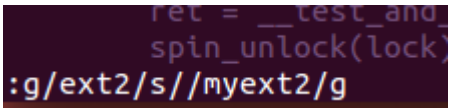




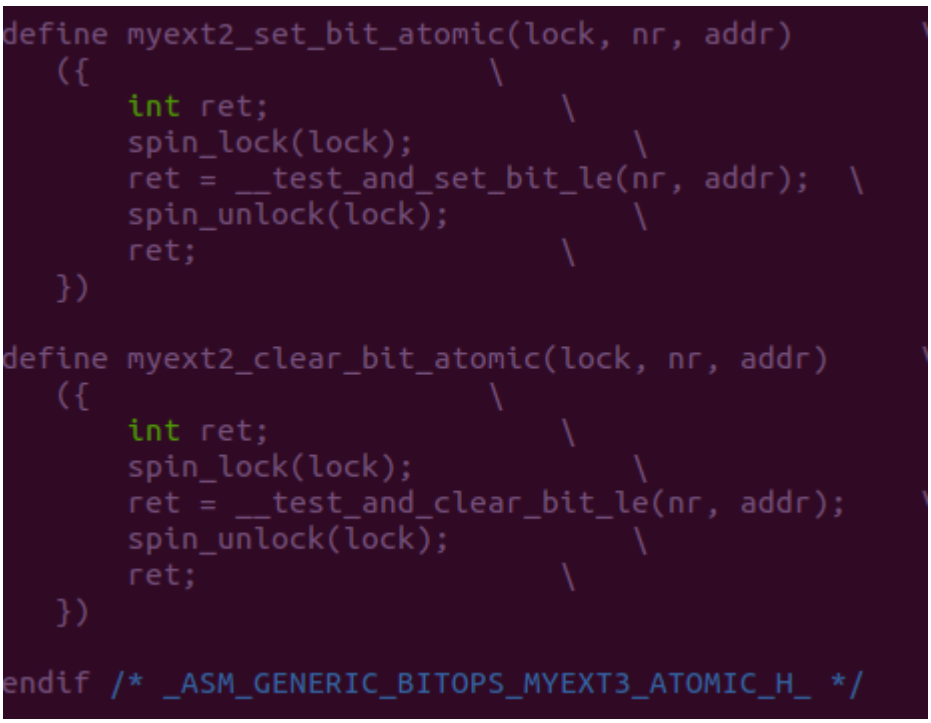
替换 myext2_fs.h 中的内容，以下是替换结果：



下面替换 myext2-atomic.h 中的内容：



以下是替换结果：



最后替换 myext2-atomic-setbit.h 中的内容：

```
~
~
:g/ext2/s//myext2/g
```

```
~
~
:g/EXT2/s//MYEXT2/g
```

以下是替换结果：

```
#ifndef _ASM_GENERIC_BITOPS_MYEXT2_ATOMIC_SETBIT_H_
#define _ASM_GENERIC_BITOPS_MYEXT2_ATOMIC_SETBIT_H_

/*
 * Atomic bitops based version of myext2 atomic bitops
 */

#define myext2_set_bit_atomic(l, nr, addr)  test_and_set_bit_le(nr,
#define myext2_clear_bit_atomic(l, nr, addr)    test_and_clear_bit_

#endif /* _ASM_GENERIC_BITOPS_MYEXT2_ATOMIC_SETBIT_H_ */
~
```

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

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

```
#include <asm-generic/bitops/myext2-atomic.h>
#include <linux/irqflags.h>
#include <linux/compiler.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>

```
#include <asm-generic/bitops/myext2-atomic-setbit.h>
#include <linux/compiler.h>
#include <asm/alternative.h>
```

8. 在/lib/modules/$(uname -r)/build /include/uapi/linux/magic.h 文件中添加：

#define MYEXT2_SUPER_MAGIC 0xEF53

```
#define MYEXT2_SUPER_MAGIC 0xEF53
```

至此，myext2 的源代码修改工作已经全部完成

9. 编译并加载内核模块，并使用命令 **cat /proc/filesystem |grep myext2** 文件系统是否已经成功加载

```
lk@Wholanz:~/linux/fs/myext2$ sudo make
make -C /lib/modules/3.17.6/build M=/home/lk/linux-3.17.6/fs/myext2 modules
make[1]: 正在进入目录 `/home/lk/linux-3.17.6'
  CC [M]  /home/lk/linux-3.17.6/fs/myext2/balloc.o
  CC [M]  /home/lk/linux-3.17.6/fs/myext2/dir.o
  CC [M]  /home/lk/linux-3.17.6/fs/myext2/file.o
  CC [M]  /home/lk/linux-3.17.6/fs/myext2/ialloc.o
  CC [M]  /home/lk/linux-3.17.6/fs/myext2/inode.o
```

```
lk@Wholanz:~/linux/fs/myext2$ sudo insmod myext2.ko
lk@Wholanz:~/linux/fs/myext2$ cat /proc/filesystems |grep myext2
        myext2
```

如图可知，myext2 文件系统已经成功加载，然后就可以对 myext2 文件系统进行测试了，测试步骤和结果见实验结果和分析一栏。

4.2 修改 myext2 的 magic number

1. 将上面实验中定义的 magic number 修改为 0x6666，在 include/uapi/linux/magic.h 中修改，如图：

```
#ifndef __LINUX_MAGIC_H__
#define __LINUX_MAGIC_H__

#define MYEXT2_SUPER_MAGIC 0x6666
```

2. 利用一个 changeMN 来修改创建的 myfs 文件系统中的 magic number，使其与内核中记录 myext2 文件系统的 magic number 匹配，这样 myfs 才能被正确的 mount

```
lk@Wholanz:~$ ./changeMN
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
```

3. 测试内容见实验结果和分析一栏

4.3 修改文件系统操作

这里我们的修改主要是主要是将 myext2 中的 mknod 操作去掉，并输出相应的提示信息。在 fs/myrxt2/namei.c 中，对 myext2_mknod 函数做出修改

```
static int myext2_mknod (struct inode * dir, struct dentry *dentry, umode_t mode
, dev_t rdev)
{
    struct inode * inode;
    int err;


    printk(KERN_ERR "Haha,mknod is not supported by filesystem myext2!You've bee
n cheated!\n");
    return -EPERM;
```

调用该功能时，直接打印功能不可用的信息，并将错误号 EPERM 返回

修改完成后，重新编译并安装内核模块，测试部分见实验结果和分析栏目

4.4 添加文件系统创建工具

编辑一个 shell 程序，能够实现输入一个文件，输出带了文件 myext2 文件系统的文件。我们在主目录下编辑这个程序工具，命名为 mkfs.myext2，格式和其他文件系统创建工具相同，代码部分如下图所示：

```
#!/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
./cfile/changeMN ./tmpfs
dd if=./fs.new of=/dev/loop2
/sbin/losetup -d /dev/loop2
rm -f ./tmpfs
```

程序主要作用就是将输入的文件进行挂载，并利用 changeMN 程序，将读出的文件头 2K 字节的内容中的 Magic number 修改成 0x6666 并写回，得到相应的 myext2 文件系统。
测试内容见实验结果和分析栏目

五、实验结果和分析

4.1 添加一个和 ext2 完全相同的文件系统 myext2

下面对已经安装的文件系统 myext2 进行测试，输入一些测试指令：

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

```
lk@Wholanz:~$ dd if=/dev/zero of=myfs bs=1M count=1
记录了1+0 的读入
记录了1+0 的写出
1048576字节(1.0 MB)已复制，0.0116079 秒，90.3 MB/秒
```

利用 dd 命令创建并初始化文件 myfs，大小 1M

2. /sbin/mkfs.ext2 myfs

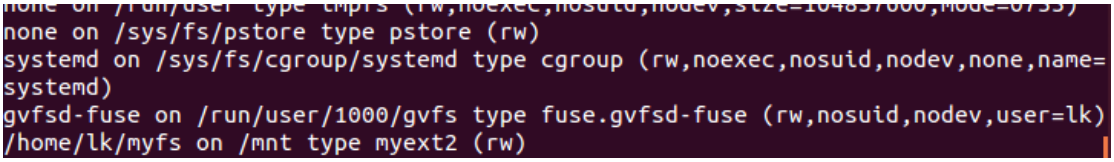
```
lk@Wholanz:~$ sudo  /sbin/mkfs.ext2 myfs
mke2fs 1.42.9 (4-Feb-2014)
myfs is not a block special device.
无论如何也要继续? (y,n) y
Discarding device blocks: 完成
文件系统标签=
OS type: Linux
块大小=1024 (log=0)
分块大小=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
128 inodes, 1024 blocks
51 blocks (4.98%) reserved for the super user
第一个数据块=1
Maximum filesystem blocks=1048576
1 block group
8192 blocks per group, 8192 fragments per group
128 inodes per group

Allocating group tables: 完成
正在写入inode表: 完成
Writing superblocks and filesystem accounting information: 完成
```

用 ext2 文件系统格式建立 myfs 文件

3. mount –t myext2 –o loop ./myfs /mnt

   将 myfs 以 myext2 文件系统的格式挂载到/mnt 目录下，loop 表示将文件当成硬盘分区挂载

4. #mount



   直接用 mount 命令，显示当前所有已经成功挂载的设备。这里可以发现最后一条中，显示 myfs 已经挂载到/mnt 下了，文件系统格式为 myext2，是可读可写的文件

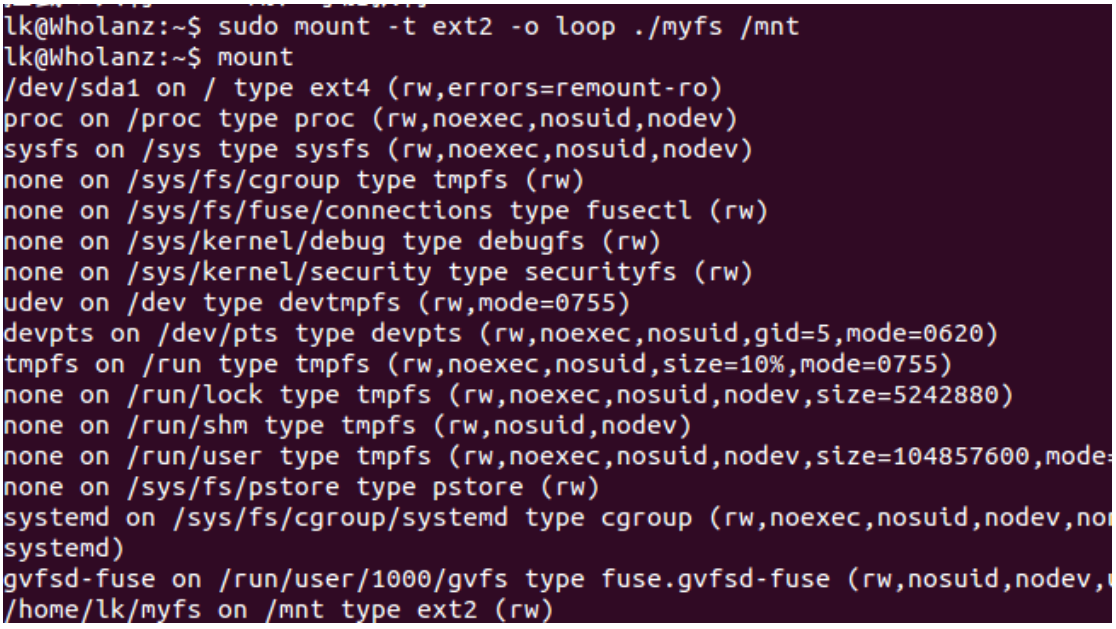
5. umount /mnt

   取消挂载/mnt 下文件的挂载

6. mount –t ext2 –o loop ./myfs /mnt

   将 myfs 文件以 ext2 文件系统的格式挂载到/mnt 目录下

7. mount



   同样可以看到 myfs 已经被成功挂载，与上一次挂载的不同是这次的文件系统格式是 ext2 的，这里说明，myext2 文件系统虽然通过复制 ext2 文件系统的源码已经被成功建立，但是本质上和 ext2 完全相同，也可以用 ext2 文件系统的格式建立并挂载。

8. sudo umount

   取消挂载

9. rmmod myext2

卸载 **myext2** 文件系统模块

4.2 修改 myext2 的 magic number

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

```
lk@Wholanz:~$ dd if=/dev/zero of=myfs bs=1M count=1
记录了1+0 的读入
记录了1+0 的写出
1048576字节(1.0 MB)已复制，0.0123652 秒，84.8 MB/秒
```

和上次以同样的方式创建并初始化文件 **myfs**

2. /sbin/mkfs.ext2 myfs

```
lk@Wholanz:~$ /sbin/mkfs.ext2 myfs
mke2fs 1.42.9 (4-Feb-2014)
myfs is not a block special device.
无论如何也要继续? (y,n) y
Discarding device blocks: 完成
文件系统标签=
OS type: Linux
块大小=1024 (log=0)
分块大小=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
128 inodes, 1024 blocks
51 blocks (4.98%) reserved for the super user
第一个数据块=1
Maximum filesystem blocks=1048576
1 block group
8192 blocks per group, 8192 fragments per group
128 inodes per group

Allocating group tables: 完成
正在写入inode表: 完成
Writing superblocks and filesystem accounting information: 完成
```

将 **myfs** 的文件系统格式设置为 **ext2**

3. ./changeMN myfs

```
lk@Wholanz:~$ ./changeMN myfs
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
```

利用 **changeMN** 程序，将 **myfs** 文件中的文件系统的 **magic number** 修改为 **0x6666**，使之与内核中记录 **myext2** 的 **magic number** 相同，这样才可以以 **myext2** 的文件系统格式挂载。

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

以文件系统 **myext2** 格式挂载新生成的文件 **fs.new**

5. mount

```
lk@Wholanz:~$ sudo mount -t myext2 -o loop ./fs.new /mnt
lk@Wholanz:~$ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs (rw,noexec,nosuid,nodev,size=104857600,
none on /sys/fs/pstore type pstore (rw)
systemd on /sys/fs/cgroup/systemd type cgroup (rw,noexec,nosuid,node
systemd)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nos
fault_permissions,allow_other)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,no
/home/lk/fs.new on /mnt type myext2 (rw)
```

如图，可以发现新生成的文件 fs.new 已经被成功以 myext2 文件系统格式被挂载于/mnt 目录下了

6. sudo umount /mnt

   取消挂载

7. sudo mount –t ext2 –o loop ./fs.new /mnt

```
lk@Wholanz:~$ sudo mount -t -ext2 -o loop ./fs.new /mnt
mount：未知文件系统类型“-ext2”
```

将文件以 ext2 文件系统格式进行挂载，发现文件已经无法被成功挂载了。这说明 magic number 是一个文件系统的 ID，如果不一样，就说明文件系统格式不匹配，无法进行挂载。这里，可以说 myext2 在修改 magic number 后，和 ext2 虽然功能相同，但是已经是两个完全不同的文件系统了。

8. rmmod myext2

4.3 修改文件系统操作

1. mount –t myext2 –o loop ./fs.new /mnt
2. cd /mnt

```
lk@Wholanz:~$ sudo mount -t myext2 -o loop ./fs.new /mnt
lk@Wholanz:~$ cd /mnt
```

成功挂载后进入目录/mnt

3. mknod myfifo p

```
lk@Wholanz:/mnt$ sudo mknod myfifo p
mknod: "myfifo": 不允许的操作
```

对文件进行 **mknod** 操作，发现这个操作不被允许

4. dmesg

```
[ 1491.039149] Haha,mknod is not supported by filesystem myext2!You've been chea
ted!
```

用 **dmesg** 指令查看系统内核日志，发现输出预期的信息，这说明 **mknod** 的功能已经被成功修改

4.4 添加文件系统创建工具

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

```
lk@Wholanz:~$ dd if=/dev/zero of=myfs bs=1M count=1
记录了1+0 的读入
记录了1+0 的写出
1048576字节(1.0 MB)已复制，0.01302 秒，80.5 MB/秒
```

用同样的方式建立并初始化 **myfs** 文件

2. sudo bash mkfs.myext2 myfs

```
Stride=0 blocks, Stripe width=0 blocks
128 inodes, 1024 blocks
51 blocks (4.98%) reserved for the super user
第一个数据块=1
Maximum filesystem blocks=1048576
1 block group
8192 blocks per group, 8192 fragments per group
128 inodes per group

Allocating group tables: 完成
正在写入inode表: 完成
Writing superblocks and filesystem accounting information: 完成

记录了2+0 的读入
记录了2+0 的写出
2048字节(2.0 kB)已复制，0.00188458 秒，1.1 MB/秒
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
记录了2048+0 的读入
记录了2048+0 的写出
1048576字节(1.0 MB)已复制，0.0523437 秒，20.0 MB/秒
```

利用新编辑的文件系统设置工具 **mkfs.myext2**，将 **myfs** 的文件系统格式设置为 **myext2**

3. sudo mount –t myext2 –o loop ./fs.new /mnt

```
lk@Wholanz:~$ sudo mount -t myext2 -o loop ./fs.new /mnt
mount：根据 mtab，/home/lk/fs.new 已作为回环设备挂载于 /mnt
```

成功将新生成的文件 **fs.new** 以文件系统 **myext2** 格式挂载

4. mount

```
lk@Wholanz:~$ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs (rw,noexec,nosuid,nodev,size=104857600,mode=0755)
none on /sys/fs/pstore type pstore (rw)
systemd on /sys/fs/cgroup/systemd type cgroup (rw,noexec,nosuid,nodev,none,name=
systemd)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,nodev,de
fault_permissions,allow_other)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,user=lk)
/home/lk/fs.new on /mnt type myext2 (rw)
```

查看挂载设备，已经可以看到 fs.new 及其文件系统格式,这说明新编辑的文件系统设置工具可以成功使用并设定文件的文件系统格式为 myext2

六、实验中遇到的问题

本次实验中遇到的主要问题是对于操作系统中 shell 程序命令的不熟悉，以及对于 linux 下文件系统机制的不理解，通过学习已经解决上述问题。

七、讨论、心得

1. 在对内核源代码中文件系统的源代码的修改过程中明白了如何添加一个新的文件系统，理解了文件系统是什么，如何工作，已经在内核中的存在形式

2. 了解了如何通过修改 magic number 的方法,并通过修改 magic number 的方法使某种文件系统区别于其他的文件系统

3. 学会了如何修改、添加、删除文件系统的操作函数和命令，并成功通过裁剪文件系统的命令了解了这个过程，为以后修改文件系统打下了坚实的基础。

4. 学会了如何编写工具快速地设定新创建的文件系统格式

5. 大体上对内核中文件系统有了一个较为全面的认识，并且可以简单的创建、修改、定义一个自己的自定义文件系统。