# CS353 Linux 内核 Project 4

## 实验内容

以Linux内核中的 /fs/romfs 作为文件系统源码基础，修改并编译生成模块 romfs.ko ，实现以下功能：

- romfs.ko 接受三个参数： hided_file_name , encrypted_file_name 和 exec_file_name
  - hided_file_name=xxx : 需要被隐藏的文件路径
  - encrypted_file_name=xxx : 需要被加密的文件路径
  - exec_file_name=xxx ： 需要修改权限为可执行的文件路径

通过 insmod romfs.ko 安装修改后的romfs模块，使用 genromfs 生成格式为romfs的镜像文件，使用 mount 命令挂载镜像文件并验证修改是否成功。

## 实验过程

1、以Linux内核中的 /fs/romfs 作为文件系统源码基础，修改并编译生成模块 romfs.ko。模块需要接受三个参数： hided_file_name , encrypted_file_name 和 exec_file_name，在super.c 文件开头声明以上三个参数：

```
#include <linux/moduleparam.h>

static char *hided_file_name;
static char *encrypted_file_name;
static char *exec_file_name;

module_param(hided_file_name, charp, 0644);
module_param(encrypted_file_name, charp, 0644);
module_param(exec_file_name, charp, 0644);
```

2、文件隐藏的实现：

读取目录到内核空间由super.c中的romfs_readdir实现。课上学到索引节点对象 Inode Object 表示操作文件或目录所需要的所有信息。从Inode 节点中获取节点偏移偏移信息，在romfs_dev_read() 中读入文件名fsname和文件数据，在romfs_dev_strnlen()中读入文件名长度，再调用dir_emit()函数将文件读入内核中，于是即可在目录下显示文件名。

要实现文件路径的隐藏，可以在读取入内核时将当前文件名与要隐藏的文件名相比较，若相同则跳过读取，不读入内核中，否则读入内核，显示文件路径。

```
/*
 * read the entries from a directory
 */
static int romfs_readdir(struct file *file, struct dir_context *ctx)
{
    struct inode *i = file_inode(file);
    struct romfs_inode ri;
    unsigned long offset, maxoff;
    int j, ino, nextfh;
    char fsname[ROMFS_MAXFN];    /* XXX dynamic? */
    int ret;

    maxoff = romfs_maxsize(i->i_sb);
```

```
    offset = ctx->pos;
    if (!offset) {
        // 获取索引节点偏移
        offset = i->i_ino & ROMFH_MASK;
        // 获取索引节点信息
        ret = romfs_dev_read(i->i_sb, offset, &ri, ROMFH_SIZE);
        if (ret < 0)
            goto out;
        offset = be32_to_cpu(ri.spec) & ROMFH_MASK;
    }

    /* Not really failsafe, but we are read-only... */
    for (;;) {
        if (!offset || offset >= maxoff) {
            offset = maxoff;
            ctx->pos = offset;
            goto out;
        }
        ctx->pos = offset;

        /* Fetch inode info */
        ret = romfs_dev_read(i->i_sb, offset, &ri, ROMFH_SIZE);
        if (ret < 0)
            goto out;
        // 获取romfs文件名长度
        j = romfs_dev_strnlen(i->i_sb, offset + ROMFH_SIZE,
                      sizeof(fsname) - 1);
        if (j < 0)
            goto out;
        // 读取文件数据
        ret = romfs_dev_read(i->i_sb, offset + ROMFH_SIZE, fsname, j);
        if (ret < 0)
            goto out;
        fsname[j] = '\0';

        ino = offset;
        nextfh = be32_to_cpu(ri.next);

        // 如果与隐藏文件名相同，则直接跳过，不读入内核
        if (strcmp(fsname, hided_file_name) == 0){
            goto skip;
        }

        if ((nextfh & ROMFH_TYPE) == ROMFH_HRD)
            ino = be32_to_cpu(ri.spec);
        if (!dir_emit(ctx, fsname, j, ino,
                romfs_dtype_table[nextfh & ROMFH_TYPE]))
            goto out;
skip:
        offset = nextfh & ROMFH_MASK;
    }
out:
    return 0;
}
```

3、文件内容加密的实现

文件内容的读取是由`super.c`中的`romfs_readpage()`实现的，通过`page->mapping->host`获取`inode`节点，再由`romfs_dev_read`读取节点的内容。

文件名的检查，可以首先由`file->f_path.dentry->d_iname`获取文件名，与要加密的文件名进行比较，若相同则进行加密，加密方式采用凯撒移位加密，即对缓冲区每一位进行加一操作。

```
static int romfs_readpage(struct file *file, struct page *page)
{
    struct inode *inode = page->mapping->host;
    loff_t offset, size;
    unsigned long fillsize, pos;
    void *buf;
    int ret;
    char fsname[ROMFS_MAXFN];

    buf = kmap(page);
    if (!buf)
        return -ENOMEM;

    /* 32 bit warning -- but not for us :) */
    offset = page_offset(page);
    size = i_size_read(inode);
    fillsize = 0;
    ret = 0;
    // 获取文件名
    fsname = file->f_path.dentry->d_iname;

    if (offset < size) {
        size -= offset;
        fillsize = size > PAGE_SIZE ? PAGE_SIZE : size;

        pos = ROMFS_I(inode)->i_dataoffset + offset;

        ret = romfs_dev_read(inode->i_sb, pos, buf, fillsize);
        if (ret < 0) {
            SetPageError(page);
            fillsize = 0;
            ret = -EIO;
        }
    }

    if (fillsize < PAGE_SIZE)
        memset(buf + fillsize, 0, PAGE_SIZE - fillsize);
    if (ret == 0)
        SetPageUptodate(page);

    // 文件名与要加密的文件名相同则进行加密，即对缓冲区每一位进行加一操作
    if (strcmp(fsname, encrypted_file_name) == 0 && fillsize > 0)
        {
            int i = 0;
            for (i = 0; i < fillsize; i ++)
                buf[i] += 1;
        }

    flush_dcache_page(page);
    kunmap(page);
    unlock_page(page);
```

```
    return ret;
}
```

4、文件权限的修改

在遍历目录时，得到inode节点后，比较文件名并对权限进行修改，在 S_IXUGO 对应位设置为 1（user、group、other 用户权限均设为可执行）。

```
static struct dentry *romfs_lookup(struct inode *dir, struct dentry *dentry,
                   unsigned int flags)
{
    unsigned long offset, maxoff;
    struct inode *inode = NULL;
    struct romfs_inode ri;
    const char *name;       /* got from dentry */
    int len, ret;

    offset = dir->i_ino & ROMFH_MASK;
    ret = romfs_dev_read(dir->i_sb, offset, &ri, ROMFH_SIZE);
    if (ret < 0)
        goto error;

    /* search all the file entries in the list starting from the one
     * pointed to by the directory's special data */
    maxoff = romfs_maxsize(dir->i_sb);
    offset = be32_to_cpu(ri.spec) & ROMFH_MASK;

    name = dentry->d_name.name;
    len = dentry->d_name.len;

    for (;;) {
        if (!offset || offset >= maxoff)
            break;

        ret = romfs_dev_read(dir->i_sb, offset, &ri, sizeof(ri));
        if (ret < 0)
            goto error;

        /* try to match the first 16 bytes of name */
        ret = romfs_dev_strcmp(dir->i_sb, offset + ROMFH_SIZE, name,
                       len);
        if (ret < 0)
            goto error;
        if (ret == 1) {
            /* Hard link handling */
            if ((be32_to_cpu(ri.next) & ROMFH_TYPE) == ROMFH_HRD)
                offset = be32_to_cpu(ri.spec) & ROMFH_MASK;
            inode = romfs_iget(dir->i_sb, offset);

            // 比较文件名并对权限进行修改
            if (exec_file_name && !strcmp(exec_file_name, name))
                inode->i_mode |= S_IXUGO;
            break;
        }

        /* next entry */
        offset = be32_to_cpu(ri.next) & ROMFH_MASK;
```

```
    }

    return d_splice_alias(inode, dentry);
error:
    return ERR_PTR(ret);
}
```

# 实验步骤

1、修改文件后，按照实验手册所示方式重新编译内核，在fs/romfs目录下出现romfs.ko。

2、安装romfs模块

```
insmod romfs.ko hided_file_name=aa encrypted_file_name=bb exec_file_name=cc #安装模块
```

3、使用所提供的镜像文件，挂载镜像到/mnt下 ，注意加上-t romfs指明文件系统。

```
mount -o loop test.img /mnt -t romfs
```

4、查看/mnt目录下的文件 发现aa文件不存在

```
ls -l /mnt
```

5、查看bb内容，应该输出加密后的内容

```
cat /mnt/bb
```

6、输出pass

```
/mnt/cc
```

# 实验结果

```
pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ sudo mount -o loop test.img
/mnt -t romfs
pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ vi /mnt/cc
pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ /mnt/cc
pass
pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ ls -l /mnt/
total 1
-rw-r--r-- 1 root root  8  1月  1  1970 bb
-rwxr-xr-x 1 root root 24  1月  1  1970 cc
pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ cat /mnt/bb
bcdfegh
       pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ cat /mnt/bb
bcdfegh
       pengjq@pengjq-virtual-machine:~/Desktop/lab4/romfs$ /mnt/cc
pass
```

# 实验心得

linux内核代码比较复杂，如果没有实验手册的指导，可能容易迷失方向，不知道操作系统具体的功能实现在哪个函数里定义。经过这次实验更加加深了对虚拟文件系统的理解，调试代码的能力又得到了锻炼TоT，感谢老师和助教的指导。