Linux 内核 --- Project 4

1. Romfs

这一次的实验主要是修改一个简单的 romfs 文件系统，从而完成一些简单的功能。Romfs 是一种简单的只读的文件系统，由于其不能修改，所有的文件都是储存在固定的位置，而支持的 vfs 的接口也就只是有限的关于文件读取的部分。

实验有三个内容，分别是添加隐藏文件，加密文件以及添加执行权限，主要的修改都在 fs/romfs/super.c 文件之中。Romfs 可以作为莫读了加载，Makefile 的写法与之前类似，只不过这里需要添加 romfs-y，因为还要用到其它两个源文件，而最后的编译与加载与之前一样，通过 make 以及 insmod 命令即可。

```
 5 obj-m += romfs.o
 6 romfs-y := storage.o super.o
 7
 8 KDIR := /lib/modules/$(shell uname -r)/build
 9
10 all:
11     make -C $(KDIR) M=$(PWD) modules
```

添加的功能的实现是通过 insmod 时的模块参数来传递参数的，这里与之前的实验也是类似的，在 super.c 文件中声明 module 的参数即可，这里简单起见，每个功能都只允许一个文件名，即传递的参数是三个字符串（作为文件名）。

```
77 #include <linux/moduleparam.h>
78 #include <linux/string.h>
79 /* parameters for Project4 */
80 static char *hided_file_name=0;
81 static char *encrypted_file_name=0;
82 static char *exec_file_name=0;
83 module_param(hided_file_name,charp,0664);
84 module_param(encrypted_file_name,charp,0664);
85 module_param(exec_file_name,charp,0664);
```

其中第一个实验是实现文件的隐藏，隐藏文件名由 hided_file_name 参数决定。这里主要是修改对于文件查找的操作以及文件夹的读取操作，即修改 romfs_readdir 以及 romfs_lookup。

对于前者，是执行读取文件夹内容时所对应的操作，利用 ls 命令时，可能会引发这一操作，而对于隐藏文件，只需要在最后检查所找到的目录项的文件名，如果与隐藏文件名一致，则不输出即可。

```
/* check for hided file --- Project 4-1 */
if(hided_file_name && strcmp(hided_file_name,fsname)==0){
}else{
    if (!dir_emit(ctx, fsname, j, ino,
        romfs_dtype_table[nextfh & ROMFH_TYPE]))
        goto out;
}
```

对于后者，是进行文件名的解析或是文件的查找是所调用的函数，这里的处理更加简单，只需要在一开始检查所要找的文件名是否与隐藏文件名相同，如果符合，则不需任何

查找，直接返回没有找到的标志即可。

```
    /* hided-file --- Project 4-1 */
    if(hided_file_name)
        if(strcmp(hided_file_name,name)==0)
            /* if name match, not found */
            goto out0;
```

第二个实验是实现文件加密，这里的“加密”仅是表现对于文件内容的修改，因此具体实现也就是对于每个 ascii 码加一。这个实验是三个实验中最为麻烦的，因为需要修改文件的读取操作，而 romfs 的读取操作使用的是内核中通用的 generic_ro_fops，而其中的读操作也是通用的 do_sync_read 函数，因此，这里采用一种简单的方法，即先调用 do_sync_read 函数将文件内容复制到用户的 buffer，然后再用 copy_from_user 从用户空间取回文件内容，之后执行加密操作，最后用 copy_to_user 在返还给用户加密后的文件内容。虽然，这样的方式是多此一举，但是这样的却是一种修改代码较少的方式。

```
87 /* encrypt_file --- Project 4-2 */
88 ssize_t encrypt_read(struct file* f,char __user *buf,size_t len,loff_t *ppos
   )
89 {
90     char *kbuf;
91     int i;
92     /* use simple method --- use do_sync_read first and then read back */
93     ssize_t s = do_sync_read(f,buf,len,ppos);
94     if(s > 0){
95         kbuf = kmalloc(s,GFP_KERNEL);
96         if(!kbuf)
97             return -ENOMEM;
98         copy_from_user(kbuf,buf,s);
99         for(i=0;i<s;i++){
100            /* simple encrypt */
101            kbuf[i] = kbuf[i] + 1;
102        }
103        copy_to_user(buf,kbuf,s);
104    }
105    return s;
106 }
```

另外，这里还需要修改 romfs_iget 函数的接口以及 romfs_iget 函数的一些部分，实际上按道理来说应该避免修改接口，但 romfs_iget 函数是 romfs 内部的 static 函数并且只有两次用到，因此修改起来还算是简单，添加的参数是两个布尔值，表示是否加密以及添加执行权限（执行权限用于第三个实验）。如果文件需要加密，则返回的 inode 的文件操作则赋值为 romfs_encrypt_fops，否则则是默认的 romfs_ro_fops。

```
413     case ROMFH_REG:
414         /* encrypt_file --- Project 4-2 */
415         if(encrypt)
416             i->i_fop = &romfs_encrypt_fops;
417         else
418             i->i_fop = &romfs_ro_fops;
419
```

这两者的差别只在于 romfs_encrypt_fops 的读操作换成了之前定义的 encrypt_read。

```
108 static const struct file_operations romfs_encrypt_fops = {
109     /* only change the read option */
110     .llseek     = generic_file_llseek,
111     .read       = encrypt_read,
112     .aio_read   = generic_file_aio_read,
113     .mmap       = generic_file_readonly_mmap,
114     .splice_read    = generic_file_splice_read,
115 };
```

最后的一个实验是为文件添加执行权限，这应该是最简单的一个，像之前所说的，为 romfs_iget 函数添加了关于执行权限的参数，如果需要执行权限，只需要在返回的 inode 之中加上执行模式即可。

```
401     /* exec_file --- Project 4-3 */
402     if(exec)
403         mode |= S_IXUGO;
404
```

2. Testing

测试所用的文件是由 genromfs 命令所产生的 test.img，其中的文件系统内容为 aa bb ft fo fo/aa 这几个文件以及文件夹，之后用 mount 命令挂载到 test 目录。

测试结果如下（第一个为没有添加参数的情形，第二个添加了相关的参数使用的命令为 sudo insmod ./romfs.ko hided_file_name=aa encrypted_file_name=bb exec_file_name=ft）：

```
[zzs@localhost testing_romfs]$ ls -l test
total 0
-rw-r--r-- 1 root root 25 Jan  1  1970 aa
-rw-r--r-- 1 root root 42 Jan  1  1970 bb
drwxr-xr-x 1 root root 32 Jan  1  1970 fo
-rw-r--r-- 1 root root 23 Jan  1  1970 ft
[zzs@localhost testing_romfs]$ ls -l test/fo
total 0
-rw-r--r-- 1 root root 49 Jan  1  1970 aa
[zzs@localhost testing_romfs]$ find test -name aa
test/aa
test/fo/aa
[zzs@localhost testing_romfs]$ cat test/bb
This is file of bb
 some letters of bbbbb
[zzs@localhost testing_romfs]$ ls -l test/ft
-rw-r--r-- 1 root root 23 Jan  1  1970 test/ft
```

```
[zzs@localhost testing_romfs]$ ls -l test
total 0
-rw-r--r-- 1 root root 42 Jan  1  1970 bb
drwxr-xr-x 1 root root 32 Jan  1  1970 fo
-rwxr-xr-x 1 root root 23 Jan  1  1970 ft
[zzs@localhost testing_romfs]$ ls -l test/fo
total 0
[zzs@localhost testing_romfs]$ find test -name aa
[zzs@localhost testing_romfs]$ cat test/bb
Uijt!jt!gjmf!pg!cc
                !tpnf!mfuufst!pg!ccccc
                                    [zzs@localhost testing_romfs]$
[zzs@localhost testing_romfs]$ ls -l test/ft
-rwxr-xr-x 1 root root 23 Jan  1  1970 test/ft
```

从中可以看到所添加的参数的效果，对于隐藏文件实验来说，之前使用 ls 以及 find 命令都能找到 aa 文件，但添加隐藏参数之后则无法找到；对与加密文件实验来说，之前使用 cat 命令可以看到文件 bb 的内容，但是之后则看到的是“加密”之后的内容；对于执行权限实验来说，之前文件 ft 没有执行权限，而添加执行权限参数之后，可以看到 ft 有了 x 权限。