**操作系统实验班大作业**

myfs的钩子扩展

**蒋捷 / 1200012708 & 兰兆千 / 1100012458 & 邢曜鹏 / 1200012835 &**

**赵万荣 / 1200012808 & 周昊宇 / 1200012823 （音序）**

# 概述

作为本小组的完成项目之一，该项目实现了一个钩子，在vfs调用的ramfs文件系统实现的函数中加入了钩子，并提供方便的可注册钩子函数的代码段。方便在某些特定时刻如创建inode、删除inode时，调用一些用户自定义的函数，实现一些功能。本实验实现了一个最简单的利用钩子来统计inode数目和占用空间的功能。

# 使用方法

1. 进入目录，修改hooks.c文件，调整自己想要的钩子处理函数；
2. sudo make
3. 使用sudo ./install.sh安装文件系统；使用sudo ./loadramdisk.sh挂载
4. 现在hooks.c中的函数已经会被执行，如果其中有内核态输出，请使用./viewlog.sh查看输出

# 具体实现

1. myfs.h中加入钩子的定义，定义了钩子函数的声明、结构体，注册函数的声明。

typedef int (\*hook\_func)(struct inode\*);

struct myfs\_hook\_operations {

hook\_func create\_inode;

hook\_func delete\_inode;

};

extern struct myfs\_hook\_operations myfs\_hook\_ops;

extern int myfs\_hook\_reg\_create(hook\_func);

extern int myfs\_hook\_reg\_delete(hook\_func);

extern int myfs\_hook\_reg\_entry(void);

1. hooks.c上半部分加入了钩子函数的结构体的实现、初始化，注册函数的实现；下半部分则实现了一个简单地统计inode数目的工具，利用了之前定义的一些工具函数。

#include "myfs.h"

int void\_inode(struct inode \* inode) {

printk("void\_hook\_activated.\n");

return -ENOSPC;

}

struct myfs\_hook\_operations myfs\_hook\_ops = {

.create\_inode = void\_inode,

.delete\_inode = void\_inode

};

int myfs\_hook\_reg\_create(hook\_func fun){

myfs\_hook\_ops.create\_inode = fun;

printk("myfs\_hook\_reg\_create called.\n");

return -ENOSPC;

}

EXPORT\_SYMBOL(myfs\_hook\_reg\_create);

int myfs\_hook\_reg\_delete(hook\_func fun) {

myfs\_hook\_ops.delete\_inode = fun;

printk("myfs\_hook\_reg\_delete called.\n");

return -ENOSPC;

}

EXPORT\_SYMBOL(myfs\_hook\_reg\_delete);

/\*

\* Modyfy below

\*/

static int inode\_count;

static int inode\_size;

static void printk\_ana(void){

printk("myfs inodes count is %d and size is %d .\n",inode\_count,inode\_size);

}

//自定义在创建inode时的钩子

static int myfs\_custom\_inode\_create\_hook(struct inode \* inode) {

inode\_count++;

inode\_size+=sizeof(struct inode);

printk\_ana();

return -ENOSPC;

}

//自定义在删除inode时的钩子

static int myfs\_custom\_inode\_delete\_hook(struct inode \* inode) {

inode\_count--;

inode\_size-=sizeof(struct inode);

printk\_ana();

return -ENOSPC;

}

//注册

int myfs\_hook\_reg\_entry(void) {

printk("hook\_reg\_entry\n");

inode\_count = 0;

inode\_size = 0;

myfs\_hook\_reg\_create(myfs\_custom\_inode\_create\_hook);

myfs\_hook\_reg\_delete(myfs\_custom\_inode\_delete\_hook);

return -ENOSPC;

}

1. operations.c中加入了钩子入口。目前只加入了两种（创建inode和删除inode）。

static int myfs\_delete\_inode(struct inode \* inode)

{

myfs\_hook\_ops.delete\_inode(inode);

generic\_delete\_inode(inode);

return -ENOSPC;

};

struct inode \*myfs\_get\_inode(struct super\_block \*sb,

const struct inode \*dir, umode\_t mode, dev\_t dev)

{

……

printk("myfs\_get\_inode called;\n");

myfs\_hook\_ops.create\_inode(inode);

return inode;

}

# 感想与展望

## 内核实现理解加深，内核调试的不易

本实验要求有各类文件内函数指针的保存，跨文件调用等等，加深了我们对于C语言与内核对于注册函数实现的理解。内核态最方便的还是通过输出调试，很难能够精确定位问题所在，也算是有了一种体会吧。

## Linux内核模块的小缺陷

本实验本意是实现一个能够让其他模块来注册钩子函数的模块。但是实践中发现，两个模块间不能很好地访问各自符号，即使利用了EXPORT\_SYMBOL与头文件，必须共享Module.symvers才可以。这导致了我们只能退而求其次，在一个模块中实现钩子函数与文件系统。

## 能实现更多功能

理论上我们可以在每个操作处实现一个钩子入口，借此实现更多更复杂的功能，例如通过阻止inode创建达到隐藏文件的目的。但是由于时间不足，未能实现，也算是一大遗憾。