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);
```

2） 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);
```

```c
/*
* 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;
}
```

3） 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。