**操作系统课程设计实验报告**

实验题目： 实验一添加内核模块文件拷贝

姓 名： 王一凡

学 号： 19061536

专 业： 软件工程

班 级： 19052713

老师姓名： 刘真

日 期： 2021 年 4 月 21 日

目 录

[一 题目介绍 1](#_Toc515881940)

[二 实验思路 1](#_Toc515881941)

[三 遇到问题及解决方法 1](#_Toc515881942)

[四 核心代码及实验结果展示 1](#_Toc515881943)

[五 个人实验改进与总结 1](#_Toc515881944)

[5.1 个人实验改进 1](#_Toc515881945)

[5.2 个人实验总结 1](#_Toc515881946)

[六 参考文献 2](#_Toc515881947)

八 源代码详单 2

# 一 题目介绍

设计一个带参数的模块，其参数为源文件和目标文件的文件名（可能带路径），模块功能是实现文件拷贝功能。

考察的知识点为内核模块的编写，需要熟练的掌握C语言的编写，在Linux内核态完成文件拷贝的功能。

问题的关键点首先在于模块的书写格式，模块的安装与卸载。关键点二在于代码的编写，要掌握Linux内核的编程方式，有些C语言的函数、库到Linux内核可能换了个名字，需要找到对应的函数。

# 二 实验思路

1. 编写C语言文件，实现一个带参数的内核模块。调用内核态的函数，实现文件拷贝。首先打开源文件，然后获取其大小，然后通过kernel\_file\_read读取文件的内容。最后打开目标文件，通过kernel\_file\_write写进去。
2. 编写makefile文件，执行make命令，生成.ko文件。
3. 安装模块sudo insmod copyy.ko file1=das file2=sdsa。
4. 查看目标文件夹中有没有生成拷贝的文件。
5. 通过dmesg查看输出的信息。

# 三 遇到问题及解决方法

# 参数传递中：传入的为字符串的类型，两个字符串以空格间隔，空格不是字符吗？

Linux文件名不能有空格，传入的字符串严格讲带空格的要加引号，否则会造成空指针异常。

# 2、文件复制：传入的参数必须为文件，即使目标目录没有该文件，必须传入一个文件名，没有的话会创建，不可以是目录，否则会空指针异常，导致卸载不掉此模块。解决方法：安装一个卸载模块的模块，传入卸载不掉的模块名，强制解除占用。

# 四 核心代码及实验结果展示

static int kernel\_file\_open( char \*file\_path)

{

file = filp\_open(file\_path,O\_RDWR|O\_CREAT,0777);

if (IS\_ERR(file)){

printk( "Open file %s failed..\n" ,file\_path);

return 0;

}

return 0;

}

static loff\_t kernel\_file\_size( struct file \*file)

{

inode = file->f\_dentry->d\_inode;

//file->f\_dentry->d\_inode;废除掉了改为file\_inode(file)

fsize = inode->i\_size;

printk(KERN\_ALERT "size = %d\n" ,( int )fsize);

return fsize;

}

void \*kernel\_file\_read( struct file \*file,loff\_t fsize)

{

int \*buff;

loff\_t \*pos = &(file->f\_pos);//文件的读写位置

buff = ( int \*)kmalloc(fsize+100,GFP\_KERNEL);

vfs\_read(file,buff,fsize,pos);

return buff;

}

static int kernel\_file\_write( struct file \*file, int \*buf,loff\_t fsize)

{

loff\_t \*pos = &(file->f\_pos);

vfs\_write(file,buf,fsize,pos);

return 0;

}

static int hello\_init( void )

{

printk(KERN\_ALERT "hello wangyifan's file-test in kernel module.\n" );

kernel\_file\_open(FILE\_PATH\_READ);

kernel\_file\_size(file);

old\_fs = get\_fs();

set\_fs(KERNEL\_DS); //扩展内核空间到用户空间

file\_buf = kernel\_file\_read(file,fsize);

filp\_close(file,NULL);

set\_fs(old\_fs); //恢复内核空间

kernel\_file\_open(FILE\_PATH\_WRITE);

old\_fs = get\_fs();

set\_fs(KERNEL\_DS);

kernel\_file\_write(file,file\_buf,fsize);

filp\_close(file,NULL);

set\_fs(old\_fs);

return 0;

}

static void hello\_exit( void )

{

printk(KERN\_ALERT "bye bye wangyifan's kernel spac..\n" );

}

module\_init(hello\_init);

module\_exit(hello\_exit);

MODULE\_LICENSE( "GPL" );

Makefile文件：

obj-m:=copyy.o

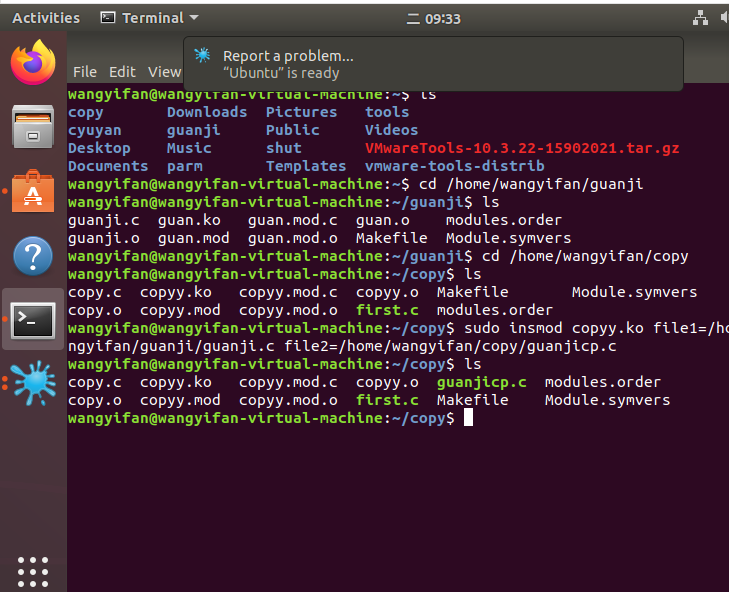
KDIR?= /lib/modules/ $(shell uname -r) /build

all:

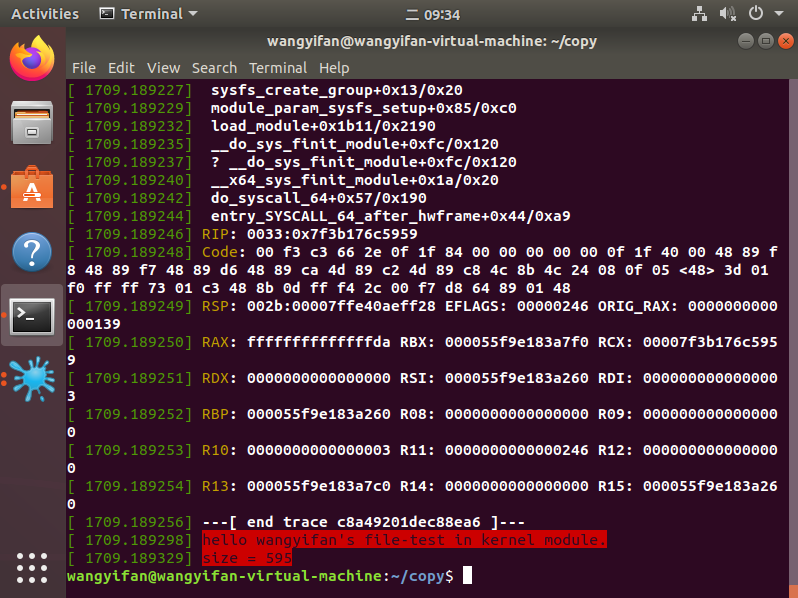
make -C $(KDIR) M=$(PWD) modules

clean:

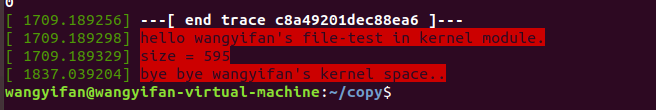
make -C $(KDIR) M=$(PWD) clean



图表 1加载模块



图表 2效果展示



图表 3卸载模块

# 五 个人实验改进与总结

## 5.1 个人实验改进

在vfs\_read和vfs\_write函数中，其参数buf指向的用户空间的内存地址，如果直接使用内核空间的指针，则会返回-EFALUT。所以需要使用set\_fs()和get\_fs()宏来改变内核对内存地址检查的处理方式，所以在内核空间对文件的读写流程为：

mm\_segment\_t fs = get\_fs();

set\_fs(KERNEL\_FS);//扩展内核空间

vfs\_write(); 或者 vfs\_read();

set\_fs(fs);

提供卸载模块，当用户不小心传入不合法的复制路径时，会造成空指针的异常。此时rmmod命令会提示模块被占用，通过新增的模块强制解除占用，卸载掉空指针的模块

文件的d\_inode存储着关于文件的基本信息，比如大小，创建时间等，旧版的Linux是在file->f\_dentry->d\_inode，而新版的直接调用函数file\_inode(file)。

## 5.2 个人实验总结

通过本实验的学习，我初步了解了Linux内核模块的编程方法，掌握了安装模块、卸载模块的步骤。对Linux用户态和内核态有了更加深入的了解。掌握了Linux带参数的内核模块传参的方法。

此外还学会了如下技巧。装入带参数的内核模块后，由于传入的地址错误导致空指针异常，但是删除卸载模块时显示被占用，通过程序解除占用。学习到了这种思维方式。通过程序去解决问题。

对makefile文件有了一定的了解。Makefile可以完成自动化编译，提高了软件开发的效率。

在编写模块时，一定要写exit退出函数。否则在安装模块时可以安装，但是在卸载时会出现模块被占用的情况，这体现了编程规范的重要性。按照规范的步骤、格式编程可以减少很多不必要的错误。

# 六 参考文献

1. <https://blog.csdn.net/gatieme/article/details/75108154>
2. [www.kernel.org](http://www.kernel.org)
3. 操作系统课本

# 七 源代码详单

文件拷贝内核模块代码：

#include <linux/kernel.h>  
#include <linux/init.h>  
#include <linux/module.h>  
#include <linux/fs.h>  
#include <linux/string.h>  
#include <linux/mm.h>  
#include <linux/syscalls.h>  
#include <asm/unistd.h>  
#include <asm/uaccess.h>  
#include <linux/moduleparam.h>  
   
static char \*file1;  
static char \*file2;  
module\_param(file1,charp,0645);  
module\_param(file2,charp,0645);  
  
struct file \*file = NULL;  
struct inode \*inode = NULL;  
   
char \*file\_buf;  
loff\_t fsize;  
mm\_segment\_t old\_fs;  
   
  
static int kernel\_file\_open( char \*file\_path)  
{  
     file = filp\_open(file\_path,O\_RDWR|O\_CREAT,0777);  
     if (IS\_ERR(file)){  
         printk( "Open file %s failed..\n" ,file\_path);  
         return 0;  
     }  
   
     return 0;  
}  
   
  
static loff\_t kernel\_file\_size( struct file \*file)  
{  
     inode = file\_inode(file);  
     fsize = inode->i\_size;  
     printk(KERN\_ALERT "size = %d\n" ,( int )fsize);  
     return fsize;  
}  
   
  
static char\*  kernel\_file\_read( struct file \*file,loff\_t fsize)  
{  
     char \*buff;  
     loff\_t \*pos = &(file->f\_pos);  
     buff = ( char \*)kmalloc(fsize+100,GFP\_KERNEL);  
     vfs\_read(file,buff,fsize,pos);  
     return buff;  
}  
   
  
static int kernel\_file\_write( struct file \*file, char \*buf,loff\_t fsize)  
{  
     loff\_t \*pos = &(file->f\_pos);  
     vfs\_write(file,buf,fsize,pos);  
     return 0;  
}  
   
static int hello\_init( void )  
{  
     printk(KERN\_ALERT "hello wangyifan&apos;s file-test in kernel module.\n" );  
     kernel\_file\_open(file1);  
     kernel\_file\_size(file);  
     old\_fs = get\_fs();  
     set\_fs(KERNEL\_DS);  
     file\_buf = kernel\_file\_read(file,fsize);  
     filp\_close(file,NULL);  
     set\_fs(old\_fs);   
   
     kernel\_file\_open(file2);  
     old\_fs = get\_fs();  
     set\_fs(KERNEL\_DS);  
     kernel\_file\_write(file,file\_buf,fsize);  
     filp\_close(file,NULL);  
     set\_fs(old\_fs);  
   
     return 0;  
}  
   
static void hello\_exit( void )  
{  
     printk(KERN\_ALERT "bye bye wangyifan&apos;s kernel space..\n" );  
}  
   
module\_init(hello\_init);  
module\_exit(hello\_exit);  
MODULE\_LICENSE( "GPL" );

obj-m:=copyy.o

KDIR?= /lib/modules/ $(shell uname -r) /build

all:

make -C $(KDIR) M=$(PWD) modules

clean:

make -C $(KDIR) M=$(PWD) clean

所用的强制卸载的工具类代码：

#include <linux/module.h>

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/cpumask.h>

#include <linux/list.h>

#include <asm-generic/local.h>

#include <linux/platform\_device.h>

#include <linux/kallsyms.h>

#include <linux/sched.h>

/\*

\* 加载模块的时候, 传递字符串到模块的一个全局字符数组里面

\*

\* module\_param\_string(name, string, len, perm)

\*

\* @name 在加载模块时，参数的名字

\* @string 模块内部的字符数组的名字

\* @len 模块内部的字符数组的大小

\* #perm 访问权限

\*

\* \*/

static char \*modname = NULL;

module\_param(modname, charp, 0644);

MODULE\_PARM\_DESC(modname, "The name of module you want do clean or delete...\n");

//#define CONFIG\_REPLACE\_EXIT\_FUNCTION

#ifdef CONFIG\_REPLACE\_EXIT\_FUNCTION

// 此处为外部注册的待卸载模块的exit函数

// 用于替代模块原来的exit函数

// 注意--此函数由于需要被待删除模块引用, 因此不能声明为static

/\* static \*/ void force\_replace\_exit\_module\_function(void)

{

/

// 此处完善待卸载驱动的 exit/cleanup 函数

/

printk("module %s exit SUCCESS...\n", modname);

// platform\_device\_unregister((struct platform\_device\*)(\*(int\*)symbol\_addr));

}

#endif // CONFIG\_REPLACE\_EXIT\_FUNCTION

static int force\_cleanup\_module(char \*del\_mod\_name)

{

struct module \*mod = NULL, \*relate = NULL;

int cpu;

#ifdef CONFIG\_REPLACE\_EXIT\_FUNCTION

void \*origin\_exit\_addr = NULL;

#endif

/

// 找到待删除模块的内核module信息

/

#if 0

// 方法一, 遍历内核模块树list\_mod查询

struct module \*list\_mod = NULL;

/\* 遍历模块列表, 查找 del\_mod\_name 模块 \*/

list\_for\_each\_entry(list\_mod, THIS\_MODULE->list.prev, list)

{

if (strcmp(list\_mod->name, del\_mod\_name) == 0)

{

mod = list\_mod;

}

}

/\* 如果未找到 del\_mod\_name 则直接退出 \*/

if(mod == NULL)

{

printk("[%s] module %s not found\n", THIS\_MODULE->name, modname);

return -1;

}

#endif

// 方法二, 通过find\_mod函数查找

if((mod = find\_module(del\_mod\_name)) == NULL)

{

printk("[%s] module %s not found\n", THIS\_MODULE->name, del\_mod\_name);

return -1;

}

else

{

printk("[before] name:%s, state:%d, refcnt:%u\n",

mod->name ,mod->state, module\_refcount(mod));

}

/

// 如果有其他驱动依赖于当前驱动, 则不能强制卸载, 立刻退出

/

/\* 如果有其他模块依赖于 del\_mod \*/

if (!list\_empty(&mod->source\_list))

{

/\* 打印出所有依赖target的模块名 \*/

list\_for\_each\_entry(relate, &mod->source\_list, source\_list)

{

printk("[relate]:%s\n", relate->name);

}

}

else

{

printk("No modules depond on %s...\n", del\_mod\_name);

}

/

// 清除驱动的状态和引用计数

/

// 修正驱动的状态为LIVE

mod->state = MODULE\_STATE\_LIVE;

// 清除驱动的引用计数

for\_each\_possible\_cpu(cpu)

{

local\_set((local\_t\*)per\_cpu\_ptr(&(mod->refcnt), cpu), 0);

//local\_set(\_\_module\_ref\_addr(mod, cpu), 0);

//per\_cpu\_ptr(mod->refptr, cpu)->decs;

//module\_put(mod);

}

atomic\_set(&mod->refcnt, 1);

#ifdef CONFIG\_REPLACE\_EXIT\_FUNCTION

/

// 重新注册驱动的exit函数

/

origin\_exit\_addr = mod->exit;

if (origin\_exit\_addr == NULL)

{

printk("module %s don't have exit function...\n", mod->name);

}

else

{

printk("module %s exit function address %p\n", mod->name, origin\_exit\_addr);

}

mod->exit = force\_replace\_exit\_module\_function;

printk("replace module %s exit function address (%p -=> %p)\n", mod->name, origin\_exit\_addr, mod->exit);

#endif

printk("[after] name:%s, state:%d, refcnt:%u\n",

mod->name, mod->state, module\_refcount(mod));

return 0;

}

static int \_\_init force\_rmmod\_init(void)

{

return force\_cleanup\_module(modname);

}

static void \_\_exit force\_rmmod\_exit(void)

{

printk("=======name : %s, state : %d EXIT=======\n", THIS\_MODULE->name, THIS\_MODULE->state);

}

module\_init(force\_rmmod\_init);

module\_exit(force\_rmmod\_exit);

MODULE\_LICENSE("GPL");

MODULE\_NAME := force\_rmmod

#MODCFLAGS:=-O2 -Wall -DMODULE -D\_\_KERNEL\_\_ -DLINUX -std=c99

#EXTRA\_CFLAGS += $(MODULE\_FLAGS) $(CFG\_INC) $(CFG\_INC)

EXTRA\_CFLAGS += -g -std=gnu99 -Wfatal-errors

ifneq ($(KERNELRELEASE),) # kernelspace

obj-m := $(MODULE\_NAME).o

else # userspace

CURRENT\_PATH ?= $(shell pwd)

LINUX\_KERNEL ?= $(shell uname -r)

LINUX\_KERNEL\_PATH ?= /lib/modules/$(LINUX\_KERNEL)/build

CURRENT\_PATH := $(shell pwd)

modules:

make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) modules

modules\_install:

make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) modules\_install

clean:

make -C $(LINUX\_KERNEL\_PATH) M=$(CURRENT\_PATH) clean

rm -f modules.order Module.symvers Module.markers

.PHNOY:

modules modules\_install clean

endif