

SHANGHAI JIAOTONG UNIVERSITY



Linux内核课程设计 - Project1

Linux内核模块编程

姓名：薛春宇

学号：518021910698

完成时间：2021/3/18

1 实验目的

编写四个模块，分别实现以下功能：

1. 模块一：加载和卸载模块时在系统日志输出信息
2. 模块二：支持整型、字符串和数组参数在内核加载时读入并打印
3. 模块三：在/proc下创建只读文件，读取该文件并返回部分信息
4. 模块四：在/proc下创建文件夹，并创建一个可读可写的文件

2 实验准备

在正式开始编写内核模块之前，我们需要对一些内核模块编写及proc文件系统操作的基本知识有所了解。

2.1 Linux内核模块相关指令

(1) 插入模块：insmod hello.ko

(2) 删除模块：rmmod hello

(3) 列出已加载模块：lsmod（列出特定名称的已加载模块：lsmod | grep hello）

(4) 查看模块信息：modinfo hello.ko

(5) 插入模块，并自动处理存在依赖关系的模块：modprobe hello.ko

2.2 Linux内核数据结构

这里，我们介绍本次驱动程序中要用到的三个最重要的内核数据结构，分别为inode、file和file\_operations。

Inode是储存文件元信息的索引节点，用于表示文件。每一个文件都有对应的inode，里面包含了与该文件有关的一些信息（除文件名以外的所有文件信息），包括文件字节数、拥有者ID和读写权限等。Linux允许多个文件名指向同一个inode号码，共享一块数据块。Inode的链接有硬链接和软链接两种方式。

File是一种指示已打开文件的文件结构体。系统中的每个打开的文件在内核空间都有一个相应的struct file结构体，它由内核在打开文件时创建，并传递给在文件上进行操作的任何函数，直至文件被关闭。如果文件被关闭，内核就会释放相应的数据结构。可能会存在有多个file结构同时指向单个inode结构。

File\_operations结构体用于说明设备驱动的接口。在系统内部，I/O设备的存取操作通过特定的入口进行，而这组特定的入口由驱动程序来提供，通常这组设备驱动的接口是由file\_operations结构体向系统说明的。常见的成员函数包括open、read、write、llseek和release等。

2.3 /proc文件系统

伪文件系统，追踪、记录系统状态以及进程状态，是Linux内核信息的统一获取接口。

实时变化，存在于虚拟内存中（不存放于任何存储介质），文件夹大小是0（不占据硬盘空间），修改时间是上次启动的时间。每次Linux系统重启时，都会创建新的/proc文件系统。

3 实验内容

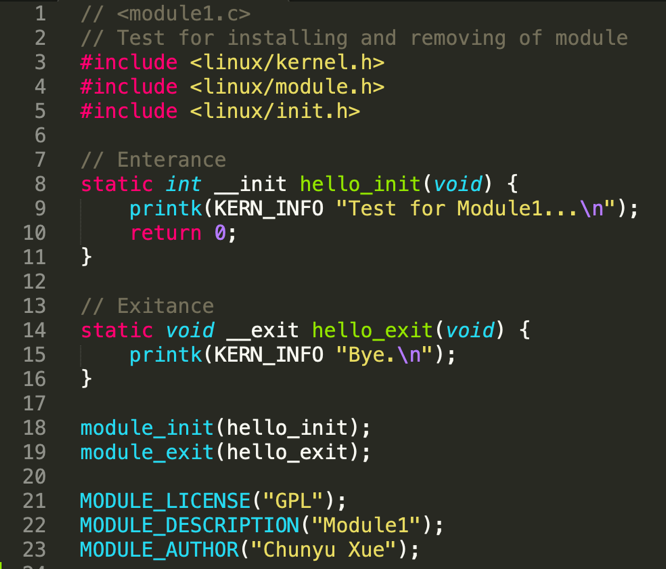
本部分内容由两个部分组成，分别是四个内核模块的实现，以及Makefile的编写。

3.1 模块一：模块的加载/卸载

本模块主要实现Linux内核加载和卸载模块时，系统日志信息的输出，是内核模块化编程最为基础的框架。

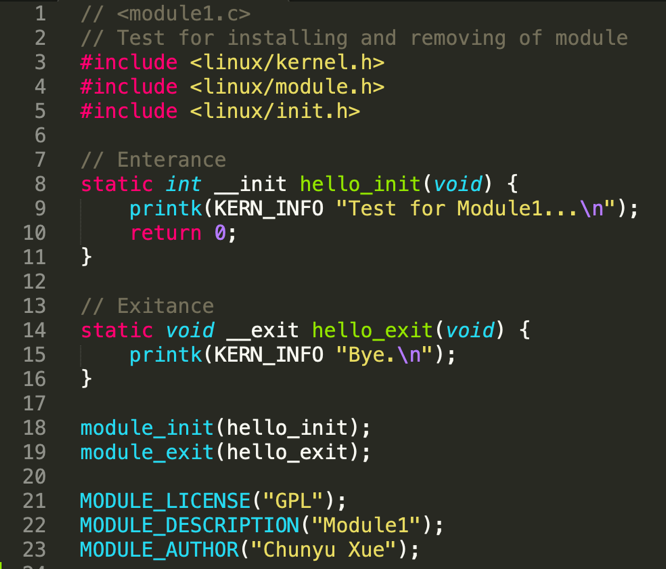
3.1.1 代码结构

首先，在内核模块编程时需要引入的三个基本头文件：

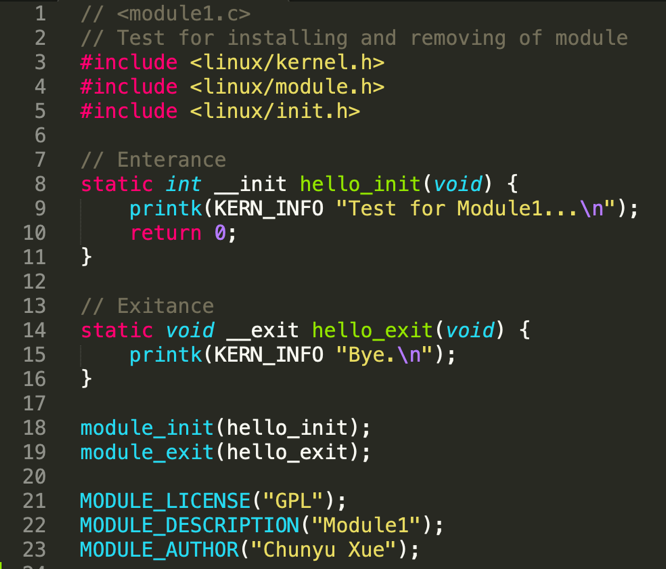


其作用分别是：

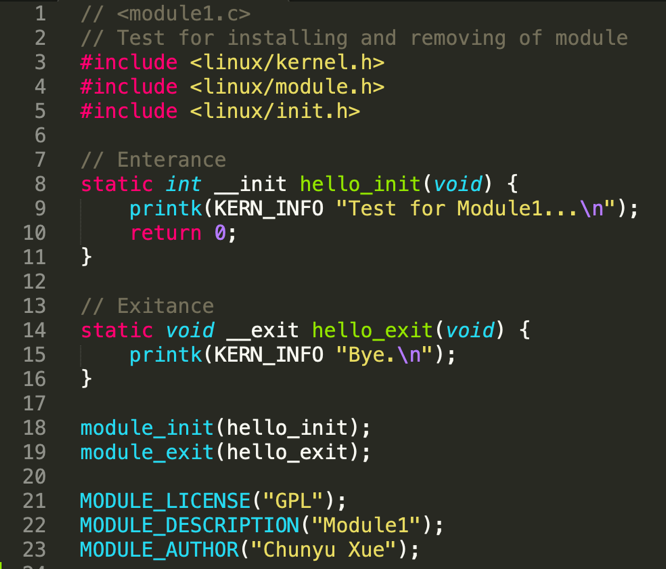
1. <linux/kernel.h>：包含了内核打印函数printk()等基本函数原型
2. <linux/module.h>：作用是动态地将模块加载到内核中（必须加载）
3. <linux/init.h>：包含了模块的初始化的宏定义，以及一些函数的初始化函数



其次，入口函数hello\_init和出口函数hello\_exit是内核模块编程两个最重要的基础，这两个函数的函数名可以进行自定义，但函数原型和参数设置必须符合规定。



在设置好出口和入口函数之后，我们需要将这两个函数分别设置为内核模块的入口和出口，具体的做法是使用在<linux/init.h>头文件中定义的module\_init和module\_exit调用，来对内核模块的出入口进行设置。



最后，我们使用MODULE\_LICENSE、MODULE\_DESCRIPTION和MODULE\_AUTHOR等宏进行模块许可证、描述和作者的声明。

3.1.2 运行效果

运行效果如图1所示。我们首先利用dmesg -C指令清空系统的日志信息，再利用insmod module1.ko指令将已经编译好的.ko文件插入内核。使用dmesg打开系统日志后可以看到，日志输出了“Test for Module1…”的信息，再利用lsmod | grep module1指令查看已加载模块，发现module1模块已成功加载。最后，我们通过rmmod module1指令将module1模块从内核中移除。

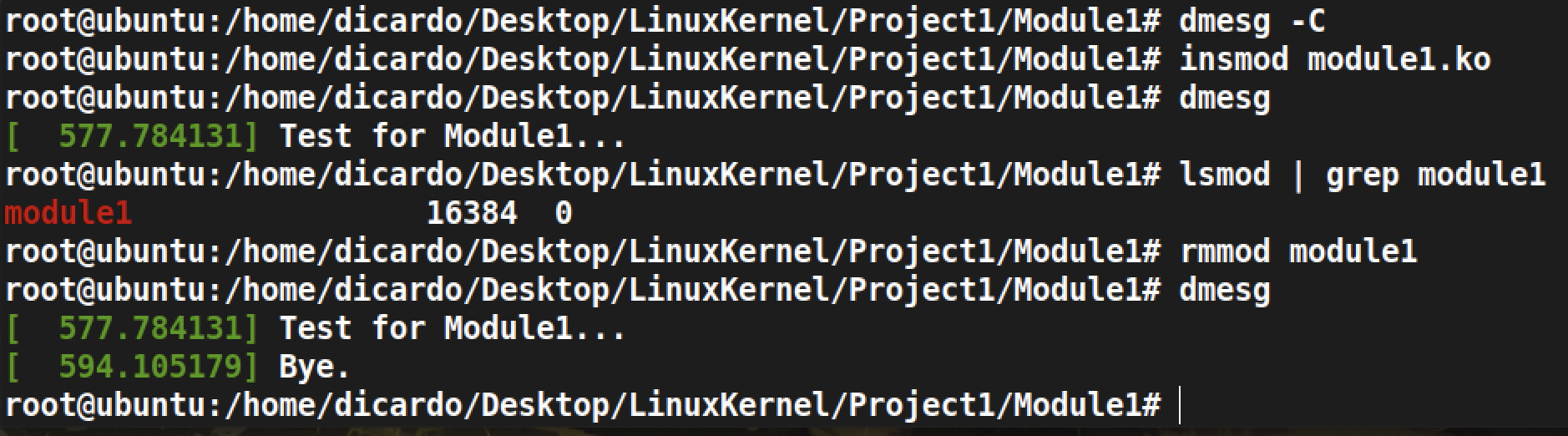


图1 Module1的运行效果

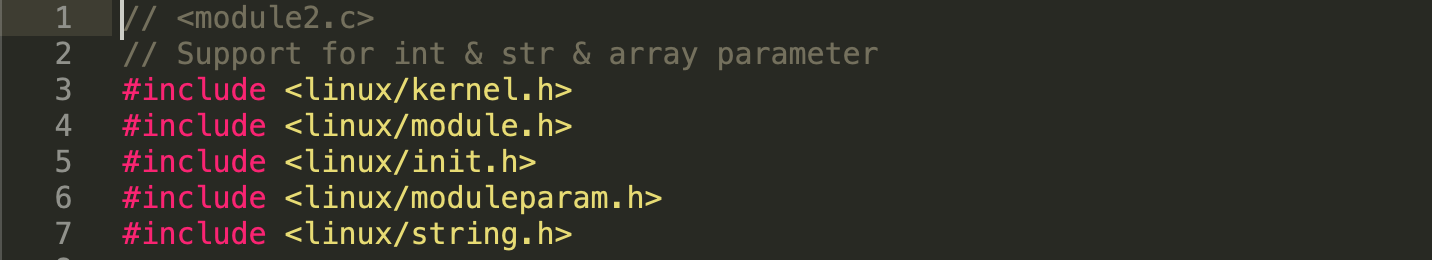
3.2 模块二：模块的参数传递

本模块主要实现Linux内核加载时整型、字符串和数组参数的读入并打印，是模块化编程输入实现的重要组成部分。

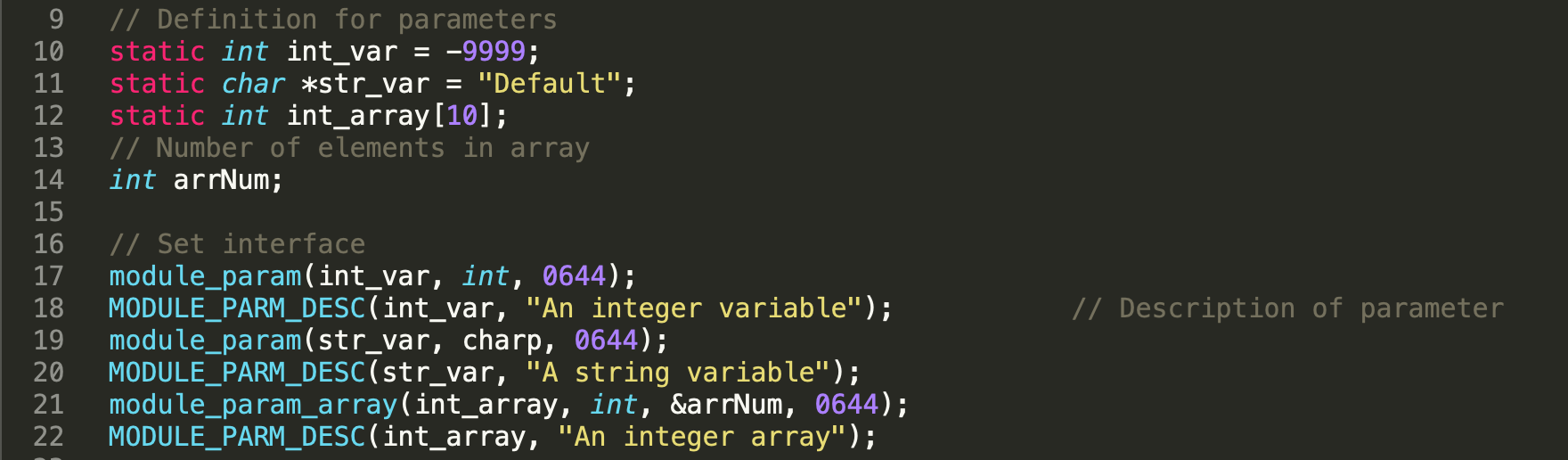
3.2.1 代码结构

首先，在3.1节基本框架的基础上，我们额外引入了两个头文件，名称及作用分别是：

1. <linux/moduleparam.h>：用于模块参数传递宏module\_param的引入
2. <linux/string.h>：用于字符串比对函数strcmp的引入

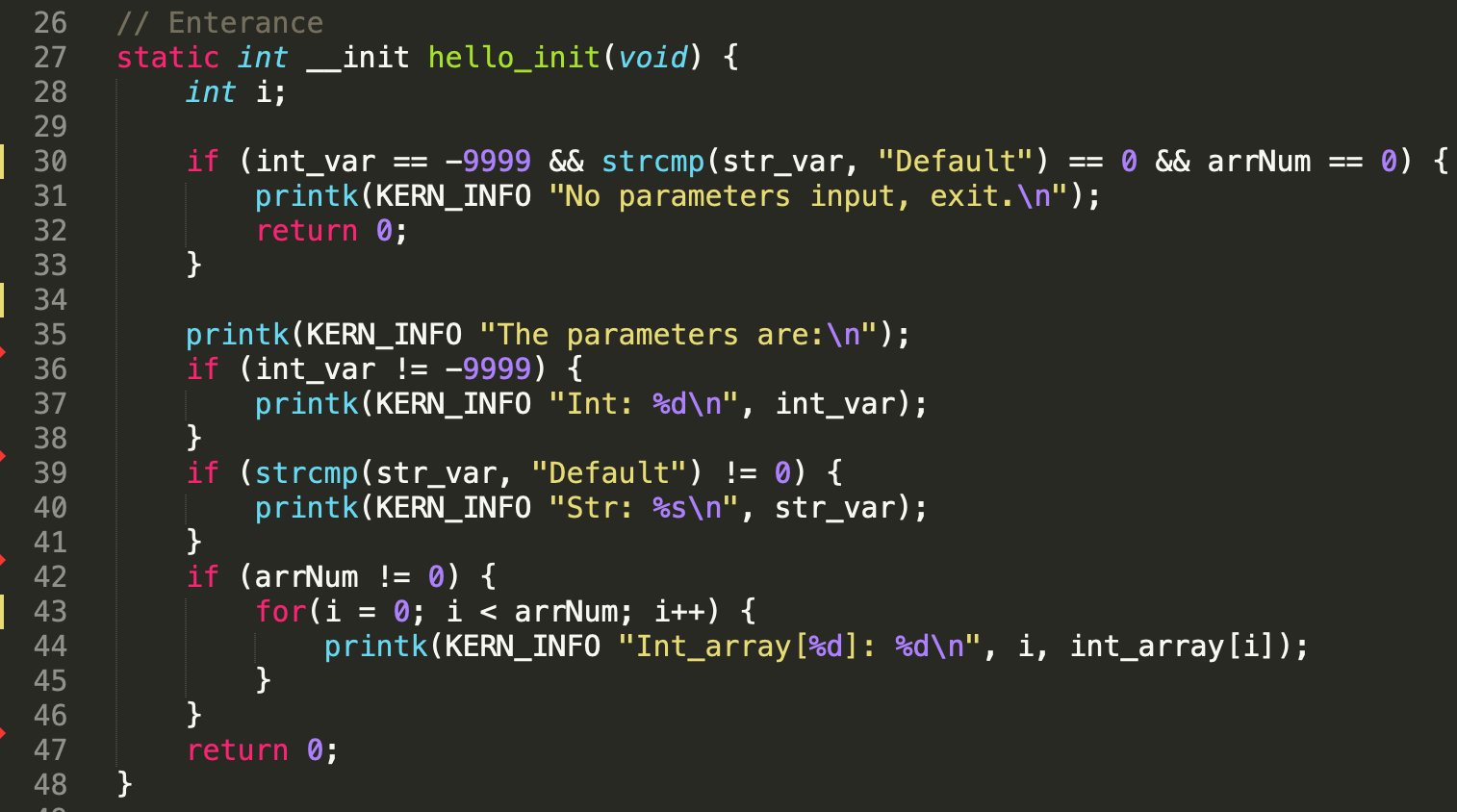


接下来，我们进行一些变量定义和参数传递接口的设置：



可以看到，我们定义了三个变量，分别是整型变量int\_var，字符型指针str\_var和整型数组int\_array，此外我们还设置了一个变量arrNum来指示数组的长度。然后，我们使用module\_param (name, type, perm) 函数来进行变量参数接口的设置，来将这三个变量与命令行参数关联起来。其中，name是变量名，type是变量类别，perm是访问权限。我们使用MODULE\_PRAM\_DESC对参数添加相应的描述。

之后便是入口函数的实现，这是本模块最重要的部分：



下面将对该入口函数的实现进行必要的解释。首先，为了提高函数的泛化性，我们设置了在无参数传递的情况下，会在系统日志中输出“No parameters input, exit.”的信息，并提前返回。若存在参数的输入，函数会分别利用变量的缺省值或数组长度作为判据，来判断应该输出哪类参数。对于数组参数，我们采用每行一个元素的方式进行输出。

最后，同3.1节中的基本框架相同，我们进行模块出入口的设置及版本信息的声明。

3.2.2 运行效果

我们首先演示在输入全部三种类型参数情况下的运行效果，通过在执行insmod内核插入操作的指令时添加int\_var、str\_var及int\_array参数传递的方式，我们能够将相应的参数传入内和空间。需要注意的是，命令行参数需要同在模块代码中定义的变量名称保持一致。运行效果如图2所示。

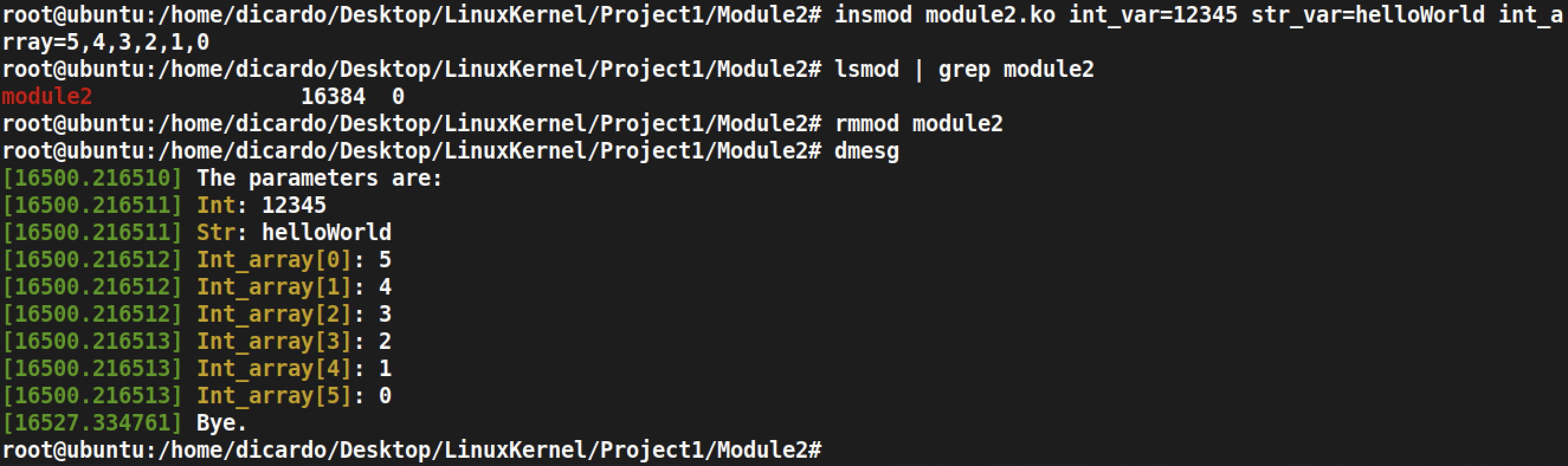


图2(1) Module2的运行效果 (1)

通过lsmod及dmesg相关信息可以看到，我们成功完成了Linux内核加载时整型、字符串和数组参数的读入和打印目标。当我们在内核加载过程中传入的参数不包含甚至没有参数传递时，仍然支持输出的自主选择：

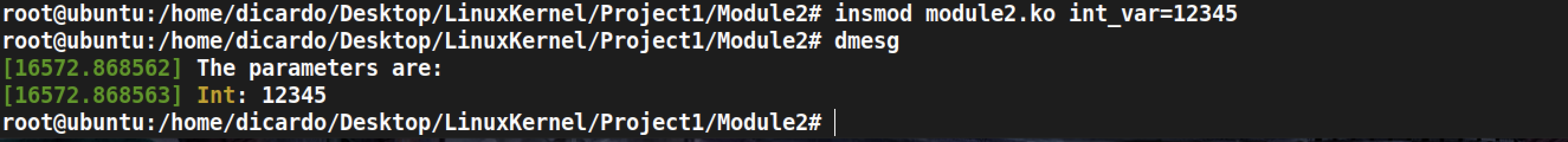


图2(2) Module2的运行效果 (2)

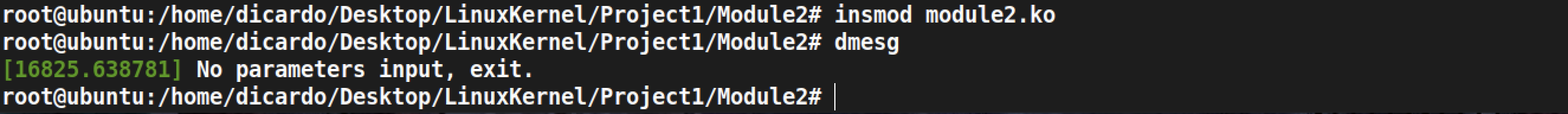


图2(3) Module2的运行效果 (3)

完成实验后，通过rmmod指令将module2模块移出内核即可。

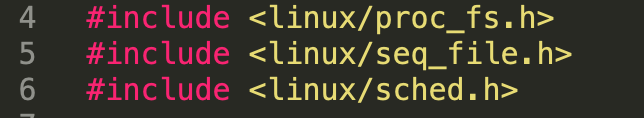
3.3 模块三：模块创建只读文件

本模块主要实现Linux在/proc文件系统下创建只读文件，读取该文件并返回部分信息，是基于proc文件系统编程的基本操作。

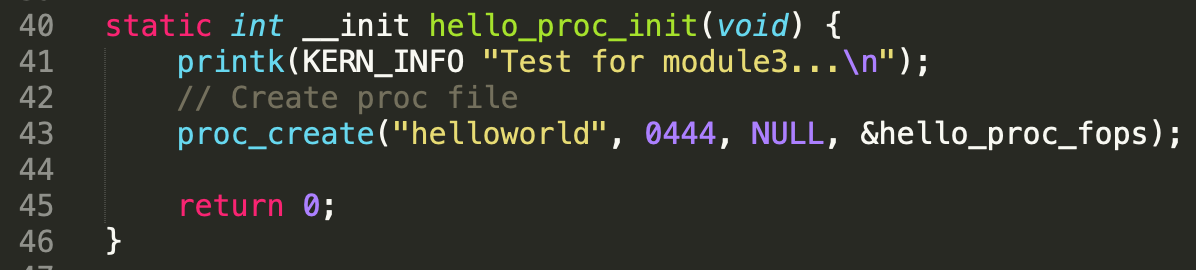
3.3.1 代码结构

在3.1节基本框架的基础上，本模块额外引入了三个头文件，其作用分别是：

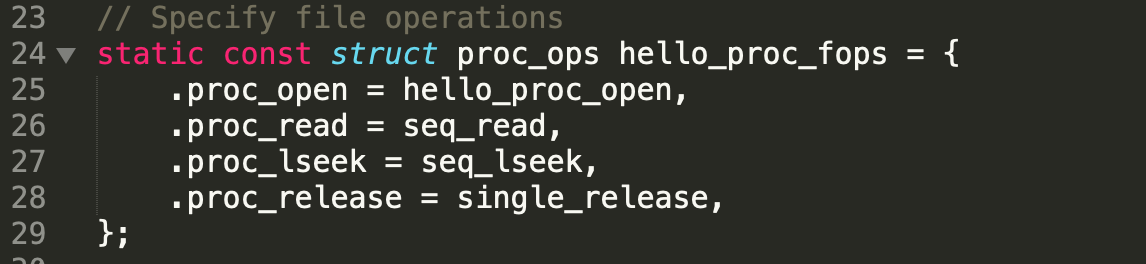
1. <linux/proc\_fs.h>：包含一些proc文件系统读、写和创建操作的基本函数
2. <linux/seq\_file.h>：包含seq-read、seq\_lseek等顺序文件处理的函数
3. <linux/sched.h>：任务调度相关，包含对内核系统时间进行读取的全局变量jiffies



接下来，我们首先介绍模块的入口函数，hello\_proc\_init：

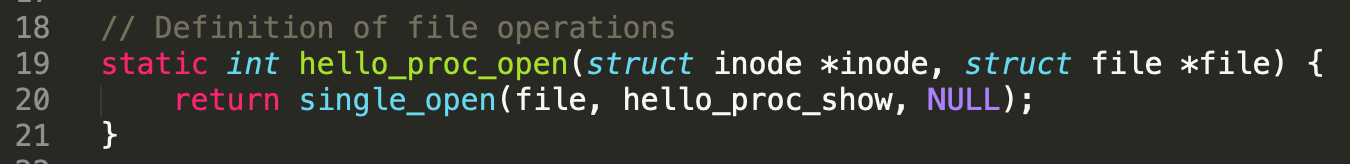


在入口函数中，调用了在<linux/proc\_fs.h>中定义的proc\_create (name, perm, parent, fops)函数，其作用是根据fops文件操作结构体创建一个名称为name，访问权限为perm，父目录为parent的proc文件。上述的fops文件操作结构体实现如下：

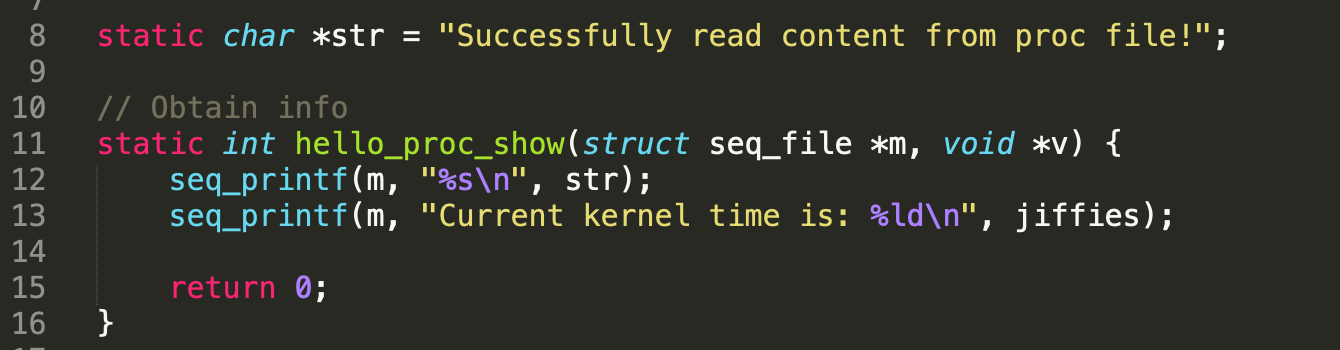


这里，我们定义了打开、读取、定位和释放四个文件操作函数，其中仅.proc\_open被定义为自定义的函数，另外三个全部为封装好的库函数。需要注意的是，我们在这里不使用file\_operations结构体的原因是，在Linux内核5.6或更高版本中，我们需要将file\_operations结构体更换为新的proc\_ops结构体。

当proc文件被成功创建后，会调用proc\_ops结构体中的.proc\_open成员函数，即我们自定义的hello\_proc\_open函数：



注意到，这里的hello\_proc\_open函数只是通过single\_open库函数来对hello\_proc\_show函数进行了封装，后者则是通过seq\_printf函数来进行了内核信息的读取和输出：



我们通过图表的方式来展现代码中几个重要模块的关系：

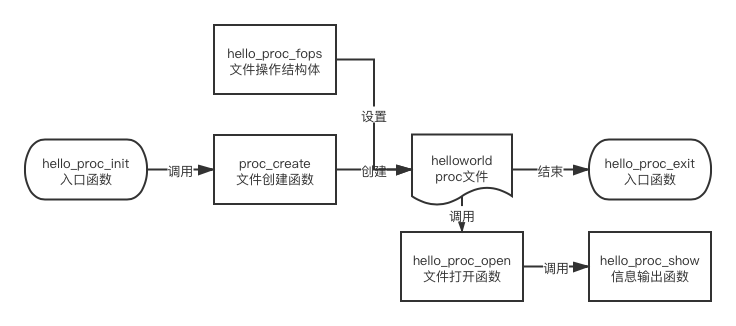


图3 Module3的模块关系

3.3.2 运行效果

运行效果如图4所示。在和之前一样成功插入内核模块module3之后，通过指令cat /proc/helloworld来读取新创建的proc文件的内容。可以看到，该操作成功输出了成功读取的信息字符串，以及当前内核的系统时间。同时，通过运行ls -l /proc/helloworld指令可以看到，该proc文件的读写权限为只读。最后通过rmmod将module3模块卸载。

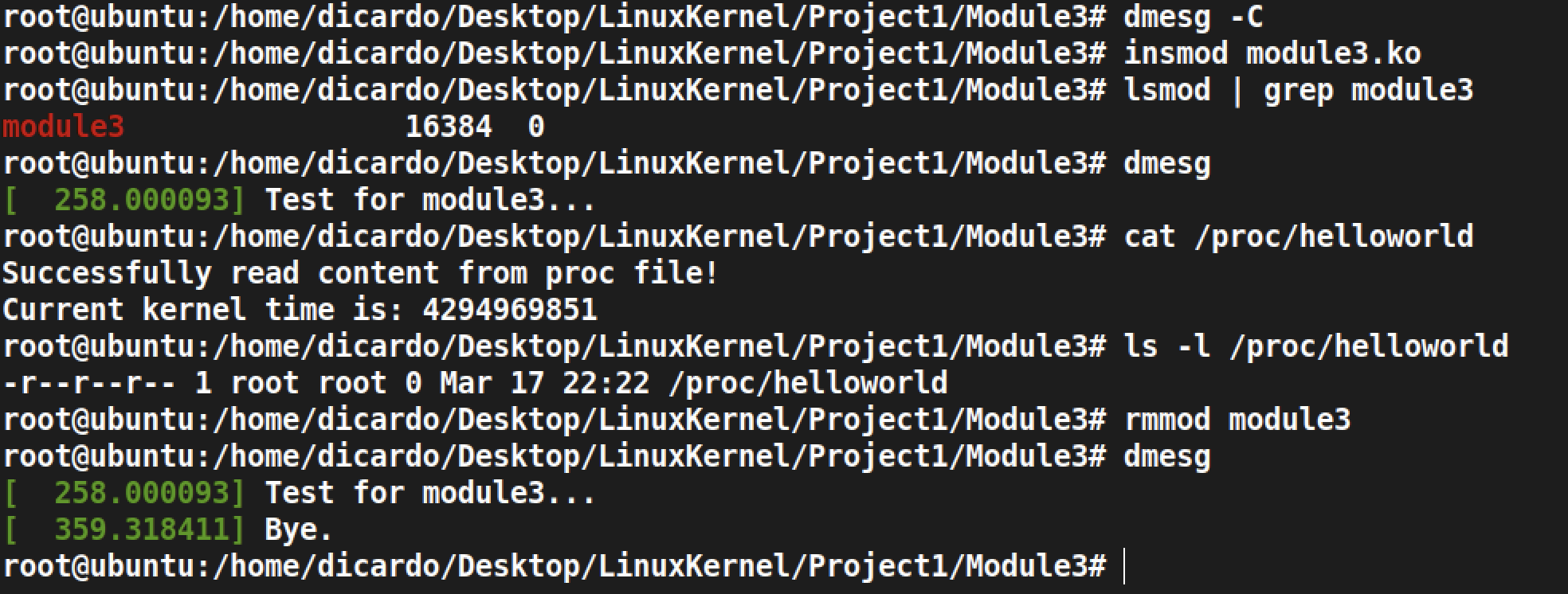


图4 Module3的运行效果

3.4 模块四：模块创建文件夹及读写文件

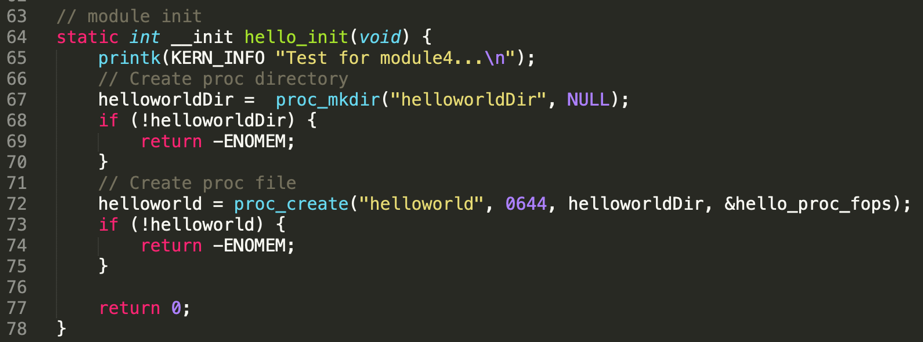
本模块主要实现Linux在/proc文件系统下创建文件夹，并创建一个可读可写的文件，同样是基于proc文件系统编程的基本操作。

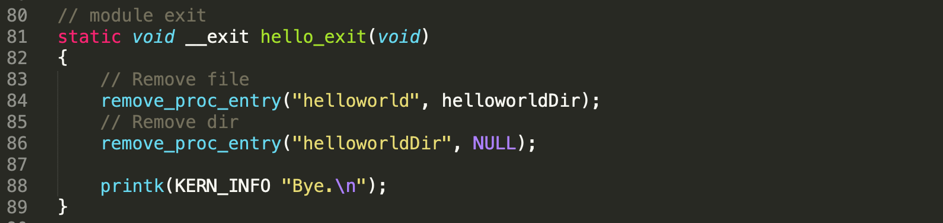
3.4.1 代码结构

在3.3节代码的基础上，本模块额外引入了两个头文件，其作用分别是：

1. <linux/slab.h>：包含了kzalloc、kfree等内存分配函数
2. <linux/uaccess.h>：包含了从用户空间到内核空间的拷贝函数copy\_from\_user

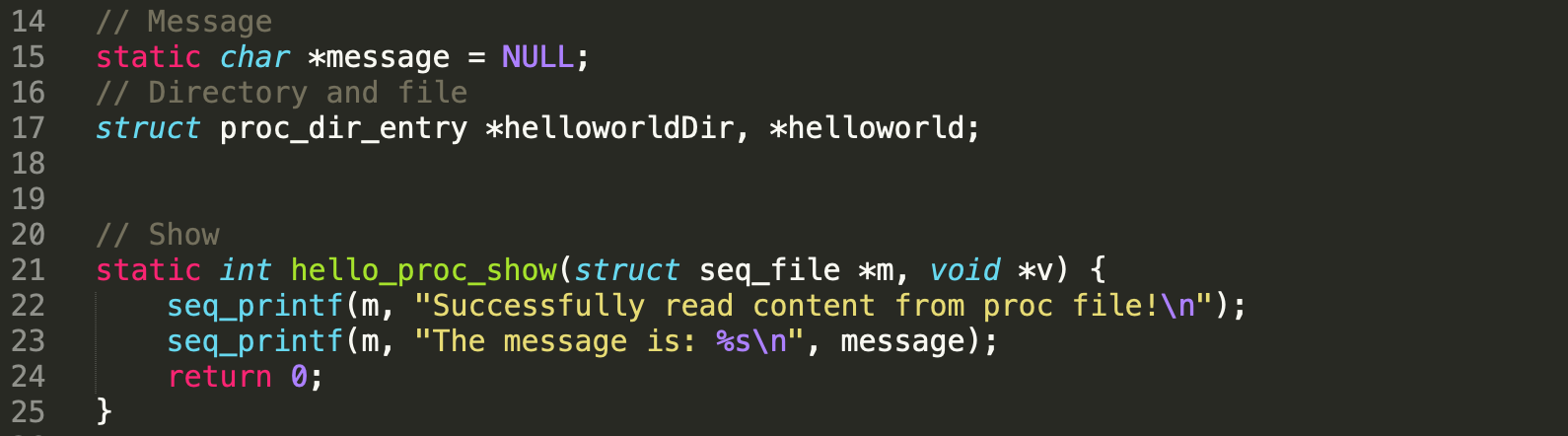
该模块在除了write函数之外的其他部分的相对逻辑关系与3.3节的module3大致相同，不同之处在于在入口和出口函数中添加了proc文件系统下的目录创建和删除：





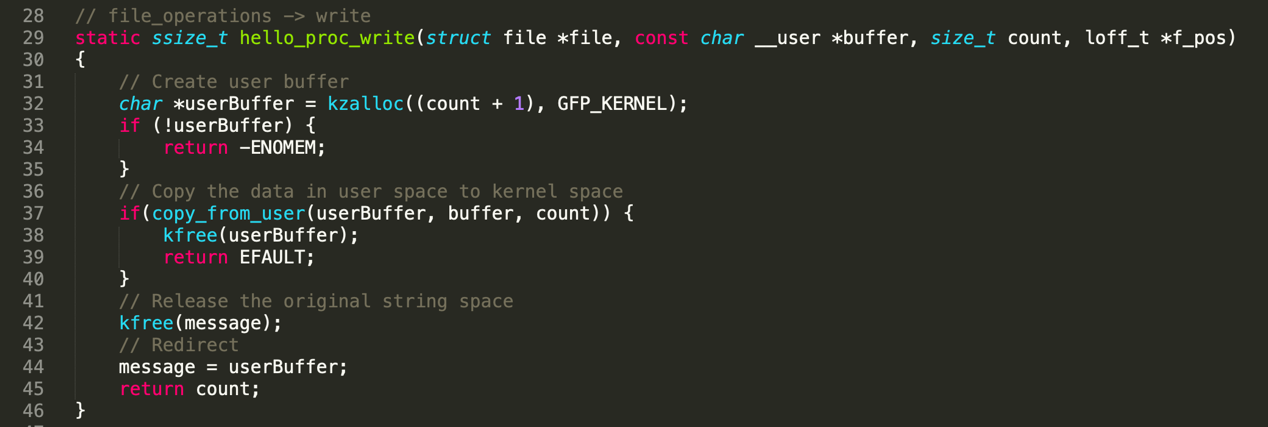
需要注意的是，在创建及删除的过程中要注意目录与文件的相对顺序，在创建时必须先创建目录，再在目录下创建文件；同样的，在删除时必须要先删除文件，再删除目录。且在proc\_create()和remove\_proc\_entry()的过程中，helloworld文件的父目录parent需要设置成创建的目录helloworldDir。

在文件操作结构体的设置中，我们相对于3.3节添加了.proc\_write=hello\_proc\_write的成员函数设置，而hello\_proc\_open的设置与3.3节相同。hello\_proc\_show的输出信息更改为：



其中，message为我们写入helloworld文件的字符串，缺省值为null。

接下来，我们重点看一下hello\_proc\_write函数的代码实现：



可以看到，在hello\_proc\_write函数中，我们首先利用kzalloc函数动态分配了一块空间，再利用copy\_from\_user系统调用将处于buffer中的命令行参数读入内核空间中新开辟的空间里，利用kfree将message指针原本指向的内容所占的空间释放，最后将其指向新开辟的含有命令行参数的空间。

3.4.2 运行效果

运行效果如图5所示。在和之前一样成功插入内核模块module4之后，通过指令echo 1234567 > /proc/helloworldDir/helloworld来向proc文件中写入内容，再通过cat /proc/helloworldDir/helloworld来读出proc文件中刚刚写入的字符串。同时，通过运行ls -l /proc/helloworldDir/helloworld指令可以看到，该proc文件的读写权限为可读可写。最后通过rmmod将module4模块卸载。

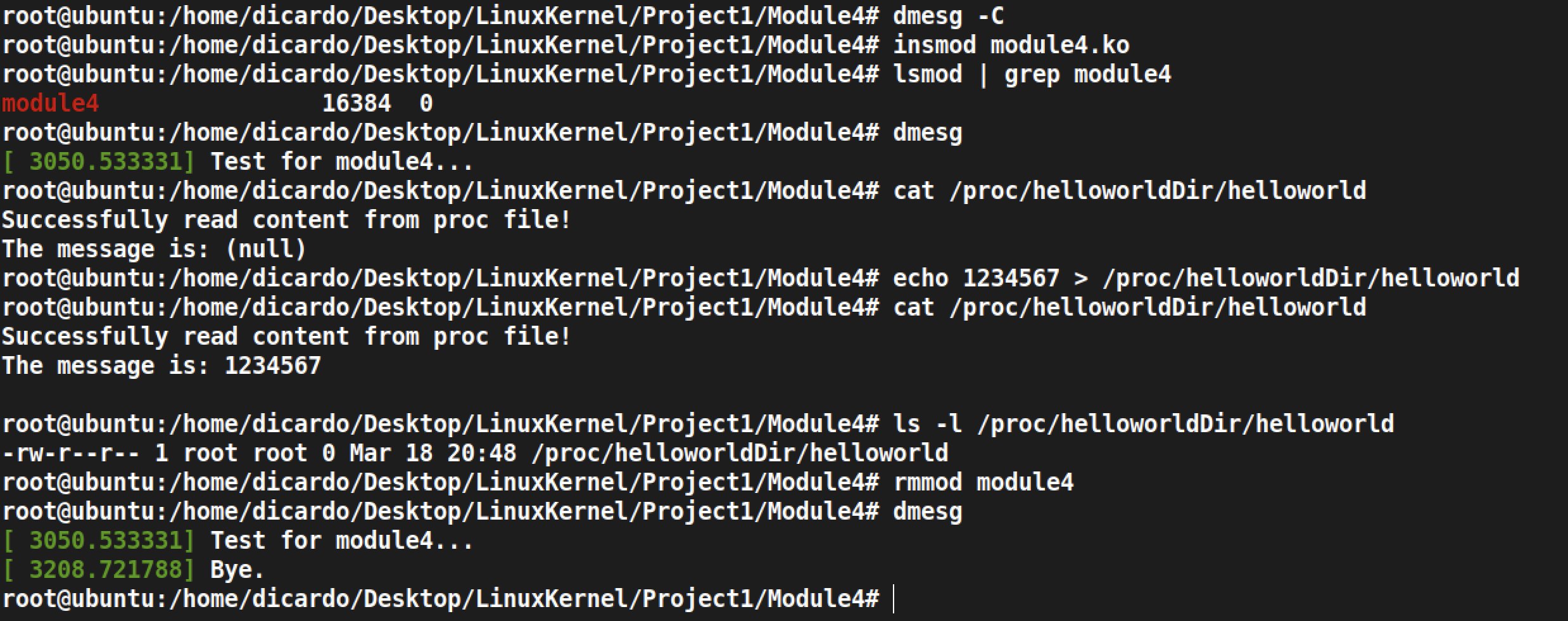


图5 Module4的运行效果

3.4 Makefile文件的编写

为了编译上述四个模块的源码以生成可加载的.ko文件，我们需要编写特定的Makefile文件，并在命令行中运行make指令进行编译。Makefile文件如下所示：

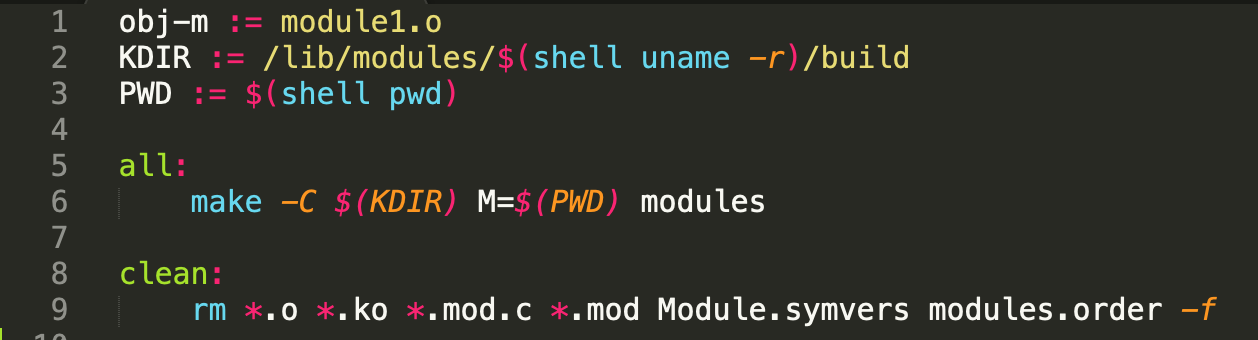


图6 Makefile文件

注意到，在编译不同的模块时，只需要将该Makefile文件同源码放在一个目录下，再修改第一行中的中间文件名，即可完成编译。需要特别注意的是，在进行第四个模块的编译时，在系统日志中输出了如下错误：



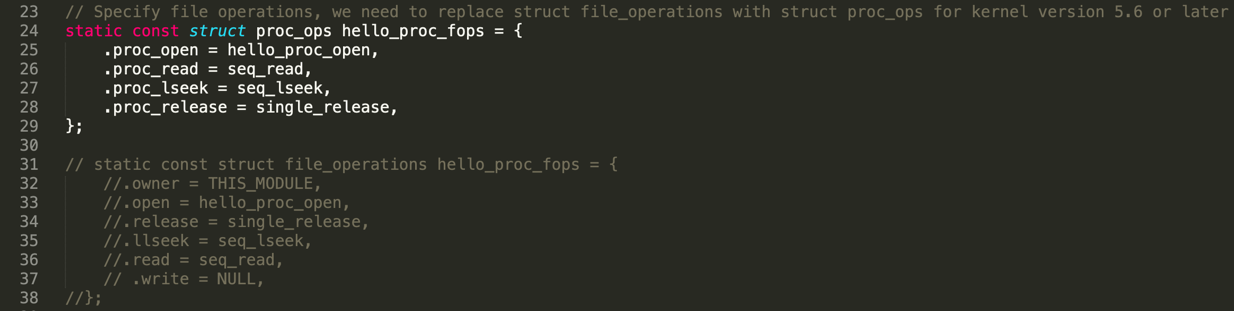
在查阅资料后发现，是内核模块的签名在动态加载时出现了验证的问题，因此在Makefile的开头处加入“CONFIG\_MODULE\_SIG=n”来关闭内核的签名功能，成功解决该问题。

4 实验结果

在解决遇到的一系列问题，经过一天的不断尝试之后，终于顺利在 ubuntu20.04 上实现并验证了几个具有特定功能的内核模块。

5 实验心得

本次实验是Linux内核课程的第一次正式project，旨在掌握面向内核模块化编程及proc文件系统编程的基本知识。由于之前在操作系统的课程中学习过一点关于内核编程的知识，这次实验的整体难度尚在我的能力范围之内。前两个模块的实现较为简单，在第三个模块中实现proc文件系统下的文件创建时，由于内核版本的问题，demo中给出的file\_operations文件操作结构体已经不再适用，且目前网上相关问题的解决方案少之又少。在经过一系列的查阅资料之后，我终于找到了问题的解决办法[[1]](#footnote-1)：将file\_operations结构体更换为proc\_ops结构体，其成员函数也发生相应的改变：



此外，在第四个模块的make过程中，我遇到了内核模块签名的问题，在查阅资料后我发现，只需要在Makefile最开始加上一行“CONFIG\_MODULE\_SIG=n”来关闭内核的签名功能，就能够成功解决该问题。

总的来说，我在本次实验中成功完成了四个内核模块的编写及运行，在增强了内核编程知识的同时，也提高了自身发现问题、解决问题的能力。希望能在下面的实验中收获满满！

1. <https://stackoverflow.com/questions/64931555/how-to-fix-error-passing-argument-4-of-proc-create-from-incompatible-pointer> [↑](#footnote-ref-1)