华 中 科 技 大 学

网络空间安全学院

本科：《操作系统原理实验》

实验报告

题目：设备管理实验

姓　　名　柳骁恩

班　　级　网安 2304 班

学　　号　U202314510

联系方式　13635763585

分　　数

评 分 人

2026 年 1 月 12 日

报告要求

1. 报告不可以抄袭，发现雷同者记为 **0** 分。

2. 报告中不可以只粘贴大段代码，应是文字与图、表结合的，需要说明流程的时候，也应该用流程图或者伪代码来说明；如果发现有大段代码粘贴者，报告需重写。

3. 报告格式严格按照要求规范，并作为评分标准。

课程目标评价标准

课程目标	评价环节	评价标准		
		高于预期（优良）	达到预期（及格）	低于预期(不及格)
目标 1.能够依据实际工程问题中软硬件的约束，应用操作系统的基础理论、抽象和分层设计思想，对特定操作系统进行评价、分析，并能进行合理的优化和裁减。具备设计、实现、开发小型或简化的操作系统的能力，并能体现一定程度的创新性。能熟练应用操作系统的开发和调试技术和工具。	当堂验收	能很合理地设计程序结构、算法和主要数据结构；完成 80%以上的预定功能。	能比较合理地设计程序结构、算法和关键的数据结构；完成 80%以上的预定功能。	程序结构、算法和主要数据结构设计不很合理；仅完成不到50%的预定功能。
	实验报告	预定功能全部完成；截止日期前提交；源代码完整，且可编译；源代码注释清晰可读。	预定功能 80%以上完成；截止日期前提交；源代码完整，且可编译；源代码注释比较清晰可读。	预定功能仅完成不到50%以上完成；截止日期以后提交；内容单薄；图文排版杂乱无章；源代码缺失或无法编译；源代码无注释或注释不清晰。
目标 2.具备操作系统国产化和自主创新意识，掌握国产操作系统，例如麒麟操作系统，鸿蒙操作系统的应用和开发环境，支持和推广国产操作系统，积极建设国产操作系统的技术生态。	当堂验收	完全使用国产操作系统；代码规范；能正确回答老师的绝大部分提问。	完全使用国产操作系统；代码比较规范；能正确回答老师的半数以上提问。	仅部分功能使用国产操作系统；代码不够规范，注释缺乏；仅能正确回答老师的极少数提问。
	实验报告	技术框架非常合理高效；符合用户硬件环境和约束参数。报告内容充实；图文排版规范；使用国产操作系统完成全部实验。	技术框架基本合理高效；符合用户硬件环境和约束参数。报告内容基本充实；图文排版基本规范；大部分实验过程使用国产操作系统。	技术框架不合理，代码臃肿；不完全符合用户硬件环境和约束参数。报告内容单薄；图文排版杂乱无章；仅小部分实验过程使用国产操作系统，或完全没用。

报告评分表

评 分 项 目		满分	得分	评分标准
设备管理	总体设计（目标 1）	15		15-11：能够给出明确需求，系统功能完整、正确和适当。 10-6：能够给出需求，但不够完整，能够阐述系统的设计，但不够完整、恰当和准确。 5-0：需求不够明确，系统设计不够完整、正确和恰当。
	详细设计（目标 1）	15		15-11：函数和数据结构描述完整，关系清晰，流程设计正确规范。 10-6：函数和数据结构描述基本完整，流程设计基本正确。 5-0：函数和数据结构描述不完整，流程设计有错误。
	代码实现（目标 1，目标 2）	10		10-8：代码能够实现设计的功能要求，考虑错误处理和边界条件。有充分的注释，代码格式规范。 7-5：代码能够实现基本的功能要求，但可能缺少错误处理和边界条件的考虑。关键部分有简单注释，代码格式较为规范。 4-0：代码未能完全实现功能要求，缺少错误处理和边界条件的考虑。注释不足，代码格式不够规范。
	测试及结果分析（目标 1，目标 2）	20		20-14：测试方法科学、完整，结果分析准确完备。 13-8：测试方法描述基本正确、完整，结果分析准确完备。 7-0：测试仅针对数据集的通过性进行描述。
	问题描述及解决方案（目标 1）	10		10-8：遇到的问题及解决方案真实具体 7-5：遇到描述不够详细，解决方案不够具体 4-0：没有写什么内容。
感想（含思政）（目标 2）		10		10-8：感想真实具体。 7-5：感想比较空洞。 4-0：没有写什么感想。
遇到的问题和解决方案，及意见和建议（目标 1）		10		10-8：意见和建议有的放矢。 7-5：意见和建议不够明确。 4-0：没有写什么内容。
文档格式（段落、行间距、缩进、图表、编号等）（目标 1）		10		基本要求：目录、标题、行间距、缩进、正文字体字号按照模板要求执行，图、表清晰且有标号。 10-8：格式规范美观，满足要求。 7-5：基本满足要求。 4-0：格式较为混乱。
总分		**100**		
教师签名			日期	

目　　录

1 实验概述

1.1 设备管理实验

本实验实现了 Linux 字符设备驱动（支持读写整数并返回最近两数之和）、配套的内核模块管理及应用测试程序，并通过 proc 接口记录和回显驱动与应用的交互信息。

1.2 实验目的

（1）理解设备是文件的概念。

（2）掌握 Linux 模块的概念和编程流程。

（3）掌握 Linux 驱动程序的编写流程和基本编程技巧。

1.3 实验环境

虚拟机 Ubuntu 6.8.12

1.4 实验内容

（1）编写一个 Linux 内核模块，并完成模块的安装/卸载等操作。

（2）在 Linux 平台编写字符设备驱动程序和相应的 2 个应用测试程序(1 个用 write 输入，一个用 read 输出)。驱动程序功能：能接收应用程序用 write 输入的整数；能响应应用 read 调用输出最近接收的两个整数的和(若当前累计只输入了一个整数，则返回该整数)。

（3）接上一题：实现 proc 接口，将应用程序和驱动程序的部分交互信息写入 proc 文件系统，并在应用程序快结束时读出相关文件信息并显示出来。

1.5 实验要求

（1）（2）必做。

2 实验过程

2.1 总体设计

2.1.1 编写内核模块任务

2.1.1.1 模块框图

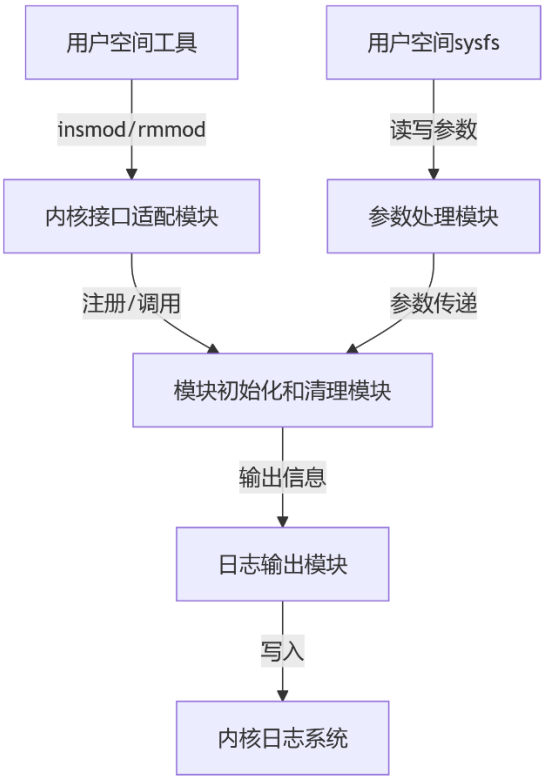

图 2-1 编写内核模块框图

2.1.1.2 模块功能说明

1.模块初始化和清理模块

hello_init(): 模块加载时的初始化函数

hello_exit(): 模块卸载时的清理函数

2.参数处理模块

module_param(): 处理模块参数 mytest

提供 sysfs 接口供用户空间读写参数

3.内核接口适配模块

module_init()/module_exit(): 向内核注册模块入口/出口

符合 Linux 内核模块框架要求

4.日志输出模块

printk(): 内核日志输出功能

2.1.1.3 模块之间的接口说明

1.用户空间 ↔ 内核空间接口

insmod/rmmod 命令

/sys/module/<module_name>/parameters/mytest (sysfs)

2.模块 ↔ 内核框架接口

module_init(): 向内核注册初始化函数

module_exit(): 向内核注册清理函数

MODULE_*宏: 模块元信息

3.模块内部接口

mytest 参数 → hello_init()函数

printk() → 内核日志系统

2.1.1.4 数据处理流程

1.模块加载流程：

（1）用户执行 insmod 命令

（2）内核加载.ko 文件到内存

（3）执行 module_init 注册的 hello_init()

（4）从模块参数获取 mytest 值(默认 100 或用户指定值)

（5）printk 输出日志到内核缓冲区

（6）返回 0 表示成功，模块加载完成

（7）通过 sysfs 可动态修改 mytest 参数

2.模块卸载流程：

（1）用户执行 rmmod 命令

（2）内核检查模块引用计数

（3）执行 module_exit 注册的 hello_exit()

（4）printk 输出退出信息

（5）清理模块占用的资源

（6）从内核模块列表移除

3.运行时数据流：

（1）用户空间写入/sys/module/hello/parameters/mytest

（2）内核参数处理层

（3）mytest 变量更新

2.1.2 编写驱动程序任务

2.1.2.1 模块框图

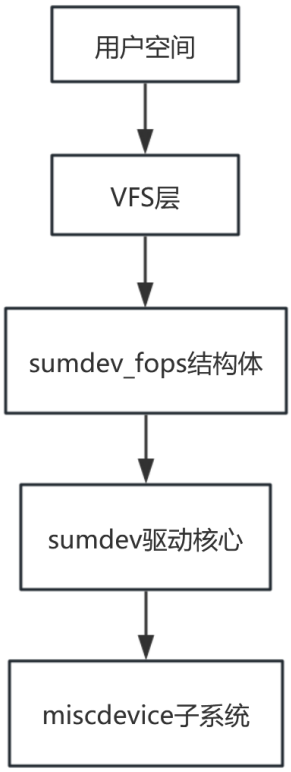

图 2-2 驱动程序模块框图

2.1.2.2 模块功能说明

1.核心功能

（1）数据存储功能

提供两个整型变量 a 和 b，用于存储用户输入的两个整数

使用标志位 flag 控制当前写入操作的目标变量（0 写入 a，1 写入 b）

（2）计算功能

实时计算并返回 a 和 b 的和

根据输入状态提供不同的反馈信息

（3）设备管理功能

通过 Linux miscdevice 子系统注册为杂项设备

自动在 /dev 目录下创建设备节点 /dev/sumdev

支持标准的文件操作接口

2.状态管理

模块维护三种工作状态：

初始状态：a=0, b=0, flag=0

单数状态：仅输入了第一个数

完成状态：已输入两个数，可计算和

2.1.2.3 模块之间的接口说明

1.用户空间接口

```
// 用户空间应用程序通过以下系统调用访问设备：
int fd = open("/dev/sumdev", O_RDWR);       // 打开设备
read(fd, buffer, size);                     // 读取计算结果
write(fd, data, size);                      // 写入输入数据
close(fd);                                  // 关闭设备
```

2.内核空间接口

（1）miscdevice 子系统接口

```
// 设备注册接口
int misc_register(struct miscdevice *misc);
void misc_deregister(struct miscdevice *misc);


// 设备结构体定义
static struct miscdevice sumdev_misc_device = {
```

```
    .minor = MISC_DYNAMIC_MINOR,   // 动态分配次设备号
    .name = "sumdev",              // 设备名称
    .fops = &sumdev_fops,          // 文件操作函数集
    .mode = 0666,                  // 设备文件权限
};
```

（2）VFS 接口

```
// 文件操作函数集结构体
static struct file_operations sumdev_fops = {
    .owner = THIS_MODULE,          // 模块所有者
    .open = sumdev_open,           // 打开设备回调
    .release = sumdev_release,     // 关闭设备回调
    .read = sumdev_read,           // 读设备回调
    .write = sumdev_write,         // 写设备回调
};
```

（3）内核内存管理接口

```
// 用户空间-内核空间数据拷贝
copy_from_user(kbuf, buf, count);   // 从用户空间拷贝数据到内核空间
copy_to_user(buf, msg, len);        // 从内核空间拷贝数据到用户空间
```

（4）内核日志接口

```
// 内核消息打印
printk(KERN_INFO "格式化消息");      // 信息级日志
printk(KERN_ERR "错误消息");         // 错误级日志
```

2.1.2.4 数据处理流程

1.模块初始化与注册流程

用户执行 insmod sumdev.ko 命令加载内核模块，内核调用 module_init()宏指定的 sumdev_init()函数。该函数调用 misc_register()向 miscdevice 子系统注册设备，分配动态次设备号，在/dev 目录下创建设备节点/dev/sumdev，并设置设备文件权限为 0666。同时初始化三个

静态全局变量：a = 0 存储第一个整数，b = 0 存储第二个整数，flag = 0 作为写入状态标志。

2.设备打开流程（open 操作）

应用程序调用 open("/dev/sumdev", O_RDWR)系统调用，VFS 层查找对应的文件操作函数集并调用 sumdev_open()回调函数。该函数仅记录日志信息 printk(KERN_INFO "sumdev_open called\n")后返回 0，表示设备打开成功。

3.数据写入流程（write 操作）

应用程序调用 write(fd, data, size)系统调用，VFS 层调用 sumdev_write()函数。该函数首先检查用户缓冲区长度，防止内核缓冲区溢出。接着通过 copy_from_user()将用户空间数据安全拷贝到内核缓冲区，并为字符串添加结束符。然后使用 kstrtoint()将字符串转换为整数，若转换失败则返回-EINVAL 错误。

4.状态机控制与数据存储

根据 flag 标志位的值决定数据存储位置：当 flag == 0 时，将转换后的整数存入变量 a；当 flag == 1 时，存入变量 b。每次写入操作后，执行 flag = !flag 翻转标志位，实现交替写入 a 和 b 的效果。最后记录操作日志并返回实际写入的字节数。

5.数据读取流程（read 操作）

应用程序调用 read(fd, buffer, size)系统调用，VFS 层调用 sumdev_read()函数。该函数首先检查文件偏移位置，若*ppos > 0 则返回 0 表示文件结束。然后根据 flag、a、b 的值判断当前状态：未输入任何数据时返回提示信息；仅输入一个数时显示该数值；输入两个数时计算并显示和值。

6.结果格式化与输出

使用 snprintf()将计算结果格式化为可读字符串，状态包括“No data yet”、“Only one number: X”或“Sum = X (a=Y, b=Z)”。然后通过 copy_to_user()将格式化后的字符串安全拷贝到用户空间缓冲区。更新文件偏移位置*ppos = len，并返回实际读取的字节数。

7.错误处理机制

在整个数据处理过程中，对关键操作进行错误检查：copy_from_user()和 copy_to_user()失败时返回-EFAULT；输入数据长度超过内核缓冲区时返回-EFAULT；字符串到整数转换失败时返回-EINVAL。所有错误情况都通过 printk(KERN_ERR)记录详细日志信息。

8.设备关闭流程（release 操作）

应用程序调用 close(fd)系统调用，VFS 层调用 sumdev_release()回调函数。该函数仅记录日志信息 printk(KERN_INFO "sumdev_release called\n")，不对存储的数据进行重置，保持 a、b、flag 的当前状态。

9.模块卸载流程

用户执行 rmmod sumdev 命令卸载内核模块，内核调用 module_exit()宏指定的 sumdev_exit()函数。该函数调用 misc_deregister()从 miscdevice 子系统注销设备，删除/dev/sumdev 设备节点，并记录模块卸载日志。所有静态全局变量随模块卸载而自动释放。

2.2 详细设计

2.2.1 编写内核模块任务

2.2.1.1 核心函数流程

1.模块初始化流程

```
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello, Kernel! mytest = %d\n", mytest);
    return 0;
}
module_init(hello_init);
```

当使用 insmod 或 modprobe 加载模块时，打印内核日志，显示模块参数 mytest 的值。返回 0 表示初始化成功。

2.模块清理流程

```
static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye, Kernel!\n");
}
module_exit(hello_exit);
```

当使用 rmmod 卸载模块时，打印退出信息。

2.2.1.2 核心数据结构

1.模块参数

```
static int mytest = 100;
module_param(mytest, int, 0644);
```

数据结构：普通的静态整型变量

module_param 宏：

参数 1：变量名 (mytest)

参数 2：数据类型 (int)

参数 3：访问权限 (0644 = 用户可读，root 可写)

功能：允许在加载模块时通过命令行传递参数。

2.模块信息结构

```
MODULE_LICENSE("GPL");              // 许可证声明
MODULE_AUTHOR("YourName");      // 作者信息
MODULE_DESCRIPTION("A simple hello kernel module"); // 描述信息
```

module_init(x)：将函数 x 注册为模块初始化函数

module_exit(x)：将函数 x 注册为模块退出函数

MODULE_LICENSE()：指定许可证（必须，否则会有警告）

printk()：内核空间打印函数，不同于用户空间的 printf

2.2.2 编写驱动程序任务

2.2.2.1 核心函数流程

1.初始化流程

注册 misc 设备，创建设备节点/dev/sumdev

2.用户空间操作流程

（1）打开设备：open("/dev/sumdev", ...)、sumdev_open()

（2）写入数据：write(fd, "123", ...)、sumdev_write()

（3）读取数据：read(fd, buffer, ...)、sumdev_read()

（4）关闭设备：close(fd)、sumdev_release()

2.2.2.2 核心数据结构

1. struct file_operations

```
static struct file_operations sumdev_fops = {
    .owner = THIS_MODULE,
    .open = sumdev_open,
    .release = sumdev_release,
    .read = sumdev_read,
    .write = sumdev_write,
};
```

定义了设备文件操作接口，是驱动与用户空间交互的核心桥梁。

2. struct miscdevice

```
static struct miscdevice sumdev_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "sumdev",
    .fops = &sumdev_fops,
    .mode = 0666,
};
```

使用 miscdevice 框架简化字符设备注册，自动分配次设备号，创建/dev/sumdev 设备节点。

2.3 代码实现

2.3.1 编写内核模块任务

1.模块参数声明

```
static int mytest = 100;
module_param(mytest, int, 0644);
```

static int mytest = 100; 定义了一个静态整数变量，初始值为 100

module_param(mytest, int, 0644); 将该变量声明为模块参数：

mytest：参数名

int：参数类型

0644：文件系统权限（用户可读写，组和其他用户只读）

这样在加载模块时可以通过 insmod module.ko mytest=200 来改变参数值

2.初始化函数

```
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello, Kernel! mytest = %d\n", mytest);
    return 0;
}
```

static：限定函数作用域只在当前文件

__init：这是一个宏，表示这个函数只在初始化时使用，初始化完成后可以释放其内存

功能：在模块加载时执行，打印欢迎信息和当前参数值

返回 0 表示成功，负值表示失败

3.清理函数

```
static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye, Kernel!\n");
}
```

__exit：宏，表示这个函数只在模块卸载时使用

功能：在模块卸载时执行，打印退出信息

4.模块声明宏

```
module_init(hello_init);
module_exit(hello_exit);
```

module_init()：告诉内核哪个函数是初始化函数

module_exit()：告诉内核哪个函数是清理函数

这些宏定义了模块的入口点和出口点

5.模块信息

```
MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("YourName");
MODULE_DESCRIPTION("A simple hello kernel module");
```

MODULE_LICENSE()：指定模块许可证（必需，否则内核会抱怨）

MODULE_AUTHOR()：指定作者信息

MODULE_DESCRIPTION()：模块描述

2.3.2 编写驱动程序任务

1. 数据存储和状态管理

```
static int a = 0, b = 0, flag = 0;
```

a, b: 存储用户输入的两个整数，初始为 0

flag: 状态标志（0/1），控制当前应该写入 a 还是 b

使用 static 修饰使其在模块生命周期内保持状态

2.写操作实现

```
static ssize_t sumdev_write(struct file *file, const char __user *buf,
                            size_t count, loff_t *f_pos)
{
    char kbuf[32];              // 内核缓冲区，用于接收用户数据
    int num;                    // 转换后的整数

    // 1. 安全检查：防止缓冲区溢出
    if (count >= sizeof(kbuf)) {
        return -EFAULT;
    }

    // 2. 安全复制：用户空间→内核空间
    if (copy_from_user(kbuf, buf, count)) {
        return -EFAULT;
    }
    kbuf[count] = '\0';       // 添加字符串结束符
```

```
    // 3. 字符串转整数（内核安全版本）
    if (kstrtoint(kbuf, 10, &num)) {
        printk(KERN_ERR "sumdev_write: invalid integer input: %s\n", kbuf);
        return -EINVAL;
    }

    // 4. 核心逻辑：根据 flag 决定写入 a 或 b
    if (flag == 0) {
        a = num;                    // flag=0 时写入 a
        printk(KERN_INFO "sumdev_write: wrote %d to a\n", num);
    } else {
        b = num;                    // flag=1 时写入 b
        printk(KERN_INFO "sumdev_write: wrote %d to b\n", num);
    }

    // 5. 切换状态：0↔1 翻转，实现交替写入
    flag = !flag;

    // 6. 返回成功写入的字节数
    return count;
}
```

copy_from_user()：安全地从用户空间复制数据

kstrtoint()：内核安全字符串转整数函数

交替逻辑：flag = !flag 实现简单状态机

3.读操作实现

```
static ssize_t sumdev_read(struct file *file, char __user *buf,
                           size_t lbuf, loff_t *ppos)
```

```
{
    int sum = a + b;              // 计算和
    char msg[32];                 // 格式化消息缓冲区
    int len;                      // 消息长度
    int ret;                      // 返回值

    printk(KERN_INFO "sumdev_read called, a=%d, b=%d, flag=%d\n", a, b, flag);

    // EOF 检查：防止重复读取
    if (*ppos > 0) {
        return 0;                 // 已读取完毕，返回 0 表示 EOF
    }

    // 智能状态判断：根据当前数据情况输出不同消息
    if (flag == 0 && a == 0 && b == 0) {
        // 情况 1：完全没有数据（初始状态）
        len = snprintf(msg, sizeof(msg), "No data yet\n");
    } else if (flag == 0 && b == 0) {
        // 情况 2：只输入了一个数（flag=0 表示刚写完 b，但 b=0 说明只有 a）
        len = snprintf(msg, sizeof(msg), "Only one number: %d\n", a);
    } else {
        // 情况 3：正常情况，输出和及详细数据
        len = snprintf(msg, sizeof(msg), "Sum = %d (a=%d, b=%d)\n", sum, a, b);
    }

    // 安全复制：内核空间→用户空间
    if (copy_to_user(buf, msg, len)) {
        return -EFAULT;           // 复制失败
```

```
    }

    // 更新读取位置，标记已读取的数据量
    *ppos = len;

    // 返回实际读取的字节数
    return len;
}
```

flag == 0 && a == 0 && b == 0: 初始状态，未写入任何数据

flag == 0 && b == 0: 只输入了 a，此时 flag=0 表示刚写完 b，但 b 还是 0 说明从未写入过 b

其他情况：正常状态，输出完整信息

4.设备注册

```
static int __init sumdev_init(void)
{
    int ret;

    // 单次调用注册所有设备属性
    ret = misc_register(&sumdev_misc_device);
    if (ret) {
        printk(KERN_ERR "sumdev: failed to register misc device\n");
        return ret;
    }

    printk(KERN_INFO "sumdev driver loaded, device: /dev/sumdev\n");
    return 0;
}
```

5.文件操作结构体

```
static struct file_operations sumdev_fops = {
    .owner = THIS_MODULE,     // 模块所有者，防止模块卸载时设备被使用
    .open = sumdev_open,      // 打开设备时调用
    .release = sumdev_release, // 关闭设备时调用
    .read = sumdev_read,      // 用户 read()时调用
    .write = sumdev_write,    // 用户 write()时调用
};
```

6.模块信息声明

```
MODULE_LICENSE("GPL");                  // 必须声明 GPL 许可证
MODULE_AUTHOR("YourName");              // 作者信息
MODULE_DESCRIPTION("Simple sum device driver"); // 模块描述
MODULE_VERSION("1.0");                  // 版本信息
```

3 测试与分析

3.1 系统测试及结果说明

3.1.1 编写内核模块任务

1.编译、加载模块，查看日志如图 3-1 所示。

```
liu@liu-virtual-machine:~/Desktop/experiment4/module$ make
make -C /lib/modules/6.8.12/build M=/home/liu/Desktop/experiment4/module modules
make[1]: Entering directory '/home/liu/Desktop/linux-6.8.12'
make[1]: Leaving directory '/home/liu/Desktop/linux-6.8.12'
liu@liu-virtual-machine:~/Desktop/experiment4/module$ sudo insmod hello.ko mytes
t=200
[sudo] password for liu:
liu@liu-virtual-machine:~/Desktop/experiment4/module$ sudo dmesg | tail -3
[32375.223345] audit: type=1400 audit(1768223623.003:69): apparmor="DENIED" oper
ation="capable" class="cap" profile="/usr/sbin/cupsd" pid=26205 comm="cupsd" cap
ability=12  capname="net_admin"
[32378.610243] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[32538.047615] Hello, Kernel! mytest = 200
```

图 3-1 加载模块日志

2.卸载模块，查看日志如图 3-2 所示。

```
liu@liu-virtual-machine:~/Desktop/experiment4/module$ sudo rmmod hello
sudo dmesg | tail -3
[32378.610243] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[32538.047615] Hello, Kernel! mytest = 200
[32578.896765] Goodbye, Kernel!
```

图 3-2 卸载模块日志

3.1.2 编写驱动程序任务

1.正常情况（两数求和）

```
liu@liu-virtual-machine:~/Desktop/experiment4/test$ sudo ./test_write 10
sudo ./test_write 20
sudo ./test_read
Wrote: 10
Wrote: 20
Driver says: Sum = 30 (a=10, b=20)
```

图 3-3 两数求和

2.边界情况（只有一个数）

```
liu@liu-virtual-machine:~/Desktop/experiment4/test$ sudo rmmod sumdev
sudo insmod ../driver/sumdev.ko
sudo ./test_write 5
sudo ./test_read
Wrote: 5
Driver says: Sum = 5 (a=5, b=0)
```

图 3-4 只有一个数

3.没有数据时读取

```
liu@liu-virtual-machine:~/Desktop/experiment4/test$ sudo rmmod sumdev
sudo insmod ../driver/sumdev.ko
sudo ./test_read
Driver says: No data yet
```

图 3-5 没有数据时

3.2 遇到的问题及解决方法

3.2.1 编写内核模块任务

3.2.2 编写驱动程序任务

1.问题一：E: Unable to locate package linux-headers-6.8.12

解决方案：使用现有的内核源码目录 /home/liu/Desktop/linux-6.8.12

2.问题二：dmesg: read kernel buffer failed: Operation not permitted

解决方案：使用 sudo dmesg

3.3 设计方案存在的不足

3.3.1 编写内核模块任务

1.功能过于简单

```
// 不足：仅打印信息，无实际功能
printk(KERN_INFO "Hello, Kernel! mytest = %d\n", mytest);
```

2.错误处理不足

3.3.2 编写驱动程序任务

1.并发安全问题

```
static int a = 0, b = 0, flag = 0;
// 问题：多个进程同时访问时会出现竞争条件
```

2.缓冲区溢出风险

```
char kbuf[32];
if (count >= sizeof(kbuf)) {
    return -EFAULT; // 简单拒绝
}
```

4 实验总结

4.1 实验感想

4.1.1 编写内核模块任务感想

完成简单内核模块的编写与测试，我深刻体会到操作系统安全基石的脆弱性与重要性。这个看似仅打印欢迎信息的模块，实则揭示了内核空间与用户空间之间那道微妙而关键的安全边界。当我在终端输入 sudo insmod hello.ko 时，这简单的命令背后是一次特权的跨越，是普通用户权限向内核级权限的跃迁，这种跃迁正是操作系统安全机制需要严格管控的关键节点。内核给出的“taints kernel”警告如同一面镜子，映照出非标准模块引入可能带来的系统性风险，这种风险不仅仅是功能上的不稳定，更是安全链条上的潜在断点。模块参数的传递机制让我反思输入验证的重要性，即使是教学实验中的简单整数传递，在真实场景中若无严格校验，便可能成为缓冲区溢出或整数溢出的突破口，进而演变为整个系统的安全漏洞。作为网络安全专业的学生，我意识到内核模块的开发不仅是功能的实现，更是安全责任的确立——每一行运行在内核空间的代码都承载着维护系统完整性的重担。这次实验让我明白，操作系统的安全始于最基础的组件，而内核模块作为这些组件中的活跃元素，其安全性需要开发者从第一个字符输入时就始终保持警惕。

4.1.2 编写驱动程序任务感想

编写字符设备驱动程序的过程，让我亲身体验了操作系统安全防线的复杂性与纵深性。驱动作为硬件与操作系统间的翻译官，其安全与否直接决定了外部输入能否安全转化为内部指令。copy_from_user 和 copy_to_user 这两个看似普通的函数调用，实则是用户空间与内核空间之间的安全闸门，每一次数据跨越这条边界都需要严格的审查与保护。实验中未加锁的全局变量如同敞开的门扉，在多进程环境下可能引发竞态条件，这种并发安全问题在真实攻击中常被利用来实施时间差攻击，篡改关键数据或提升权限。设备节点的权限设置让我深思最小权限原则在实际开发中的落实困境——为了方便测试而设置的宽松权限，在生产环境中可能成为攻击者长驱直入的通道。网络安全视角下的设备驱动开发，要求我们不仅要实现功能，更要构建防御：输入验证需要层层设防，资源访问需要细粒度控制，错误处理需要安全降级，日志输出需要避免信息泄露。这次实验让我认识到，设备驱动的安全性不是单一层面的问题，而是涉及访问控

制、数据完整性、审计监控、故障隔离等多维度的综合体系。作为未来的网络安全从业者，我意识到安全驱动的编写需要建立纵深防御的思想，从代码的第一行开始就贯穿安全思维，因为这不仅关乎单个设备的功能正常，更关乎整个操作系统的安全根基。

4.2 意见和建议

可以有一些和网络安全有关的拓展实验。

原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

已阅读并同意以下内容。

判定为不合格的一些情形：

（1）请人代做或冒名顶替者；

（2）替人做且不听劝告者；

（3）实验报告内容抄袭或雷同者；

（4）实验报告内容与实际实验内容不一致者；

（5）实验代码抄袭者。

作者签名：