**2022嵌入式操作系统大作业**



学 院： 国际信息与软件学院

专 业： 计算机系

姓 名： 安睿哲

学 号： 20212251012

姓 名： 安睿哲

学 号： 20212251012

## 1 大作业介绍

本项目是一个字符驱动程序。使用凯撒加密算法，可以实现简单的加密解密功能。

1.1系统功能：

本项目是一个字符驱动程序。该字符设备驱动示例实现了一个简单的字符设备驱动程序，可以在用户空间中通过读写设备节点与内核空间进行数据交互。用户可以使用设备驱动提供的 ioctl 命令来设置加密密钥和加密模式，然后通过设备节点进行数据的读取和写入操作。该驱动示例还实现了基本的加密和解密功能，可以对读取和写入的数据进行简单的加密和解密处理。

1.2系统构成：

该系统由字符设备驱动程序和用户空间应用程序组成。驱动程序负责与内核空间进行交互，提供数据读写和加密解密功能。用户空间应用程序通过设备节点与驱动程序进行通信，实现数据的读写和加密解密设置。

1.3使用的技术要点：

Linux 内核编程：该驱动程序使用了 Linux 内核提供的字符设备接口和相关函数，通过编写相应的回调函数实现了读写和 ioctl 操作。

加密解密算法：驱动程序实现了简单的加密解密算法，用于对数据进行加密和解密操作。

设备节点和设备文件操作：用户空间应用程序通过打开设备节点文件，使用 read、write 和 ioctl 等系统调用与驱动程序进行交互。

1.4程序完成度：

该示例程序是一个基本的字符设备驱动，提供了数据读写和加密解密功能，但并没有涵盖所有可能的功能和异常情况处理。程序的完成度较高，能够正常工作并完成基本的功能。

1.5软件性能：

由于该驱动程序是一个大作业程序，主要用于学习目的，性能并未进行太多优化。驱动程序中的加密解密算法较简单，可能不适合在实际的生产环境中使用。对于性能要求较高的场景，需要根据实际需求进行性能优化和选择更合适的加密算法。

## 软件功能

（1）应用场景

该字符设备驱动示例可以在需要进行数据加密和解密的应用场景中使用。例如，可以在安全性要求较高的系统中使用该驱动程序来对敏感数据进行加密处理，以确保数据在传输和存储过程中的安全性。同时，该驱动程序也可以作为学习和演示用途，在学习 Linux 内核编程和字符设备驱动开发时进行实践和调试。

（2）软件目的

该驱动程序的主要目的是演示字符设备驱动的开发过程和基本功能实现。通过该示例程序，可以了解如何使用 Linux 内核提供的字符设备接口和相关函数，实现设备节点的读写和 ioctl 操作，以及简单的数据加密和解密功能。

（3）功能要求

①数据读写

驱动程序可以接收用户空间应用程序传输的数据，并将其存储在内部缓冲区中，或从内部缓冲区读取数据并传输给用户空间应用程序。

②ioctl 操作

驱动程序支持 ioctl 命令，用于设置加密密钥和加密模式等。

③加密解密功能

驱动程序提供简单的加密和解密功能，能够对读取和写入的数据进行加密和解密处理。

④设备节点

驱动程序创建设备节点文件，用户空间应用程序可以通过打开设备节点文件与驱动程序进行通信。

## 2.1 系统构成

（1）软件环境

①操作系统

该驱动程序是在Linux操作系统下开发和运行的。驱动程序需要与Linux内核进行交互。

②编程语言和工具链

该驱动程序使用C语言编写，利用Linux内核编译工具链进行编译。

（2）硬件环境

①目标设备

该驱动程序是为特定硬件设备（字符设备）开发的，具体设备的类型和规格可以根据具体的应用需求进行选择。

②开发平台

基于Linux的计算机。

## 2.2 功能说明

（1）注册设备：

在内核中注册一个字符设备，使得用户空间可以通过设备文件与该驱动程序进行交互。

（2）实现读取和写入功能：

提供读取设备的接口，用户空间可以通过读取设备文件来获取设备传输的数据。提供写入设备的接口，用户空间可以通过向设备文件写入数据来向设备发送指令或数据。

（3）实现设备控制：

支持使用ioctl系统调用对设备进行控制，通过传递特定的命令和参数来实现设备的配置、状态修改等操作。

（4）数据加密和解密：

驱动程序内部实现了简单的数据加密和解密功能，可以对通过设备传输的数据进行加密和解密处理。加密和解密算法可以根据具体需求进行定制和扩展。

（5）支持用户空间和内核空间的交互：

用户空间可以通过设备文件进行读取和写入操作，与驱动程序进行通信。驱动程序可以通过系统调用和内核函数与内核空间进行交互，如处理设备的读写请求、控制命令等。

## 3．实现方式

（1）设备注册和初始化：

在模块加载时，通过调用`register\_chrdev`函数注册字符设备，并分配设备号。创建设备类别和设备节点，将字符设备与设备节点进行绑定，以在用户空间创建对应的设备文件。

（2）实现设备操作函数：

在驱动程序中定义设备操作函数，如`device\_read`和`device\_write`，用于读取和写入设备数据。

这些函数在用户空间通过设备文件进行调用，并在内核空间处理对应的读写请求。

（3）设备控制函数：

定义设备的控制函数`device\_ioctl`，用于处理`ioctl`系统调用传递的命令和参数。根据命令类型执行相应的操作，如设置密钥、设置加密模式等。

（4）数据加密和解密函数：

实现数据加密和解密的函数，如`encrypt\_data`和`decrypt\_data`。

这些函数在驱动程序中被调用，用于对读取和写入的数据进行加密和解密处理。

（5）内存管理和资源释放：

在驱动程序加载和退出时，进行内存的动态分配和释放，如分配内部缓冲区的内存空间。释放设备节点、设备类别和设备号等资源。

## 3.1 开发环境

表3.1 大作业项目的软件构成

|  |  |
| --- | --- |
|  | 项目说明 |
| 操作系统 | Linux ubuntu 5.15.0-75-generic |
| 内核版本 | 5.15.0-75-generic |
| 虚拟机配置 | 内存4GB硬盘20GB处理器数量2 |
| 编程语言 | C语言 |
| 编程环境 | Linux环境 |

## 3.2算法原理

（1）确定偏移量

选择一个偏移量，通常是一个非负整数，表示字母的移位数。例如，偏移量为3表示字母向右移动3个位置。

（2）加密过程

将明文中的每个字母按照偏移量向右移动相应的位置。如果超出了字母表的边界，可以循环回到字母表的开头继续计数。例如，使用偏移量为3，字母'A'加密后成为字母'D'，字母'Z'加密后成为字母'C'。

（3）解密过程

对密文进行解密时，将每个密文字母向左移动偏移量的位置。同样，如果超出了字母表的边界，可以循环回到字母表的末尾继续计数。

## 3.3模块构成（或者子程序构成）

（1）所需的Linux内核头文件。

（2）定义设备名称、内部缓冲区大小、密钥和加密模式等变量。

（3）用于Base64编码和解码的辅助函数。

（4）加密和解密函数，用于对数据进行加密和解密操作。

（5）ioctl函数，根据接收到的命令执行相应的操作，如设置密钥、设置加密模式、开始读取/写入数据等。

（6）read函数和write函数，用于读取和写入数据到内部缓冲区。

（7设备操作结构体，并将相应的函数指针赋值给对应的操作。

（8）模块的初始化函数和退出函数，其中初始化函数用于注册字符设备、创建设备节点和初始化字符设备结构体，退出函数用于清理资源。

（9）宏定义设置模块的许可证、作者和描述信息。

## 3.4 驱动程序设计方法

（1）头文件

<linux/init.h>：提供了与内核模块初始化和清理相关的宏和函数。主要的数据对象是module\_init和module\_exit，它们分别用于指定模块的初始化和清理函数。

<linux/module.h>：包含了与内核模块相关的宏、结构体和函数。其中的关键数据对象是MODULE\_LICENSE、MODULE\_AUTHOR和MODULE\_DESCRIPTION，它们分别用于指定模块的许可证、作者和描述信息。

<linux/fs.h>：提供了与文件系统相关的宏、数据结构和函数。其中的关键数据对象是struct file\_operations，它定义了字符设备驱动程序的操作函数。

<linux/cdev.h>：包含了与字符设备相关的宏、数据结构和函数。其中的关键数据对象是struct cdev，它表示字符设备对象。

<linux/device.h>：提供了与设备相关的宏、数据结构和函数。其中的关键数据对象是struct class和struct device，它们分别表示设备类和设备对象。

<linux/uaccess.h>：包含了与用户空间访问相关的函数。其中的关键数据对象是copy\_to\_user和copy\_from\_user，它们用于在内核空间和用户空间之间复制数据。

<linux/ioctl.h>：定义了IO控制命令的宏。在这段代码中，使用了一些自定义的设备驱动命令，如\_IOW和\_IO，用于进行设备控制。

<linux/string.h>：提供了与字符串操作相关的函数。在这段代码中，使用了strcpy和strlen函数来进行字符串的复制和长度计算。

<linux/slab.h>：包含了与动态内存分配相关的函数。其中的关键数据对象是kmalloc和kfree，它们用于在内核堆中动态分配和释放内存。

<linux/bio.h>：定义了与块I/O操作相关的数据结构和函数。在这段代码中，使用了struct bio结构体和相关的函数来进行数据的读写操作。

<linux/crypto.h>和<crypto/hash.h>：提供了与加密和哈希算法相关的数据结构和函数。  
（2）宏定义

DEVICE\_NAME：表示设备的名称，用于标识设备驱动程序对应的字符设备节点。

MAX\_BUFFER\_SIZE：定义了设备驱动程序中用于数据缓冲区的最大大小，这里设置为4096字节。

ENCRYPTION\_SET\_KEY：自定义的设备驱动命令之一，用于设置加密密钥。该命令使用\_IOW宏，表示是一个写入命令，并以参数unsigned long表示传递的参数类型。

ENCRYPTION\_SET\_MODE：自定义的设备驱动命令之一，用于设置加密模式。该命令同样使用\_IOW宏，并以参数unsigned long表示传递的参数类型。

ENCRYPTION\_START\_WRITE：自定义的设备驱动命令，用于开始写入操作。该命令使用\_IO宏，表示是一个无参数的控制命令。

ENCRYPTION\_START\_READ：自定义的设备驱动命令，用于开始读取操作。该命令同样使用\_IO宏，表示是一个无参数的控制命令。

ENCRYPTION\_RESET：自定义的设备驱动命令，用于重置设备状态。该命令同样使用\_IO宏，表示是一个无参数的控制命令。

（3）base64编码解码函数

bmem, b64, bio\_out: 这些是struct bio结构的指针，用于进行输入和输出数据的操作。bmem表示输入数据的内存缓冲区，b64表示进行Base64编码的中间步骤缓冲区，bio\_out表示输出数据的内存缓冲区。

output: 这是一个字符指针，指向解码后的二进制数据的起始位置。在解码函数base64\_decode中，解码后的数据将存储在该指针指向的内存区域。

output\_len: 这是一个指向size\_t类型的指针，用于存储解码后的数据的长度。在解码函数中，通过修改output\_len的值来传递解码后数据的长度。

（4）加密解密函数

DEVICE\_NAME：一个宏定义，表示设备的名称。在这个例子中，设备的名称被定义为"encryption\_device"。

MAX\_BUFFER\_SIZE：一个宏定义，表示内部缓冲区的最大大小。在这个例子中，内部缓冲区的最大大小被定义为4096字节。

ENCRYPTION\_SET\_KEY：一个宏定义，表示设备驱动的命令之一，用于设置加密密钥。该宏定义了一个特定的命令标识符，包括了命令的类型、命令的序号和参数的类型信息。

ENCRYPTION\_SET\_MODE：一个宏定义，表示设备驱动的命令之一，用于设置加密模式。该宏定义了一个特定的命令标识符，包括了命令的类型、命令的序号和参数的类型信息。

ENCRYPTION\_START\_WRITE：一个宏定义，表示设备驱动的命令之一，用于开始写入数据。该宏定义了一个特定的命令标识符，包括了命令的类型和命令的序号，但没有参数信息。

ENCRYPTION\_START\_READ：一个宏定义，表示设备驱动的命令之一，用于开始读取数据。该宏定义了一个特定的命令标识符，包括了命令的类型和命令的序号，但没有参数信息。

ENCRYPTION\_RESET：一个宏定义，表示设备驱动的命令之一，用于重置设备状态。该宏定义了一个特定的命令标识符，包括了命令的类型和命令的序号，但没有参数信息。

（5）ioctl函数

file：指向表示打开设备文件的数据结构的指针。它包含了与设备文件相关的信息，如文件状态、文件位置等。

cmd：无符号整数，表示设备驱动的命令标识符。根据该命令标识符，可以确定应该执行哪种操作。

arg：无符号长整型，表示设备驱动的命令参数。根据命令标识符和参数，可以进一步确定设备驱动应该执行的具体操作。

key：一个全局变量，表示加密密钥。通过ENCRYPTION\_SET\_KEY命令可以设置该变量的值。

mode：一个全局变量，表示加密模式。通过ENCRYPTION\_SET\_MODE命令可以设置该变量的值。

buffer：一个指针，指向用于存储数据的内部缓冲区。通过ENCRYPTION\_START\_WRITE命令分配内存，并在ENCRYPTION\_START\_READ命令中对其进行读写操作。

buffer\_size：一个全局变量，表示内部缓冲区的大小。通过ENCRYPTION\_START\_WRITE命令设置该变量的值。

（6）Read和Write函数

file：指向表示打开设备文件的数据结构的指针。它包含了与设备文件相关的信息，如文件状态、文件位置等。

buf：指向用户空间的缓冲区的指针。在device\_read和device\_write函数中，用于读取和写入数据。

count：表示要读取或写入的字节数。在device\_read和device\_write函数中，用于确定要读取或写入的数据大小。

offset：指向文件偏移量的指针。在这段代码中未被使用，可以用于确定读取或写入数据的位置。

buffer：一个指针，指向用于存储数据的内部缓冲区。在device\_read和device\_write函数中，数据将从该缓冲区读取或写入。

buffer\_size：一个全局变量，表示内部缓冲区的大小。用于限制读取和写入的数据大小，以确保不会超出缓冲区的容量。

（7）设备操作结构体

fops：struct file\_operations 结构体类型的变量，用于定义设备驱动的文件操作。它包含了设备驱动所支持的各种操作的函数指针。在这段代码中，定义了以下成员：

.owner：指向拥有该结构体的模块的指针，通常设置为 THIS\_MODULE，表示使用当前模块作为所有操作的拥有者。

.unlocked\_ioctl：指向设备驱动的 ioctl 函数的函数指针，用于处理设备的控制命令。

.read：指向设备驱动的 read 函数的函数指针，用于从设备中读取数据。

.write：指向设备驱动的 write 函数的函数指针，用于向设备中写入数据。

（8）初始化和退出函数

device\_number：dev\_t 类型的变量，用于存储设备的主设备号和次设备号。

device\_class：struct class \* 类型的变量，表示设备的类别。它用于创建设备节点并管理相关的设备文件。

cdev：struct cdev 类型的变量，表示字符设备对象。它用于注册字符设备驱动，并与设备文件进行关联。

fops：在前面已经介绍过了，它是 struct file\_operations 类型的变量，定义了设备驱动的文件操作函数。

buffer：指向缓冲区的指针，用于在设备驱动中存储数据。

buffer\_size：size\_t 类型的变量，表示缓冲区的大小。

（9）模块信息

MODULE\_LICENSE("GPL")：用于指定模块的许可证。在此情况下，许可证为 GPL（GNU General Public License），表示该模块遵循 GPL 许可协议。

MODULE\_AUTHOR("anruizhe")：用于指定模块的作者。在此情况下，作者为 "anruizhe"。

MODULE\_DESCRIPTION("Character device driver for encryption")：用于指定模块的描述。

## 3.5 交互过程描述

（1）模块初始化

在 \_\_init 函数中，首先分配设备号（alloc\_chrdev\_region）。然后创建设备类别（class\_create）。接着创建设备节点（device\_create），将设备添加到设备类别下。初始化字符设备（cdev\_init）并将其添加到内核（cdev\_add）。

（2）用户空间与设备驱动的交互

用户程序可以通过打开设备文件和使用 read、write 等系统调用来与设备驱动进行交互。当用户调用 read 系统调用时，device\_read 函数会被调用，它将设备驱动的数据从内核空间复制到用户空间。当用户调用 write 系统调用时，device\_write 函数会被调用，它将用户空间的数据复制到设备驱动的内核空间。

（3）设备驱动的操作

设备驱动的主要操作在 device\_ioctl 函数中实现，用于处理设备的控制命令。当用户调用 ioctl 系统调用时，device\_ioctl 函数会被调用，根据传入的命令（cmd）执行相应的操作。支持的命令包括设置密钥（ENCRYPTION\_SET\_KEY）、设置模式（ENCRYPTION\_SET\_MODE）、开始写入数据（ENCRYPTION\_START\_WRITE）、开始读取数据（ENCRYPTION\_START\_READ）和重置设备（ENCRYPTION\_RESET）。

（4）模块卸载

在 \_\_exit 函数中，清理资源，包括删除字符设备（cdev\_del）、销毁设备节点（device\_destroy）、销毁设备类别（class\_destroy）以及注销设备号（unregister\_chrdev\_region）。如果内部缓冲区有分配内存，则释放缓冲区的内存（kfree）。

## 3.6 驱动程序编译，安装，运行

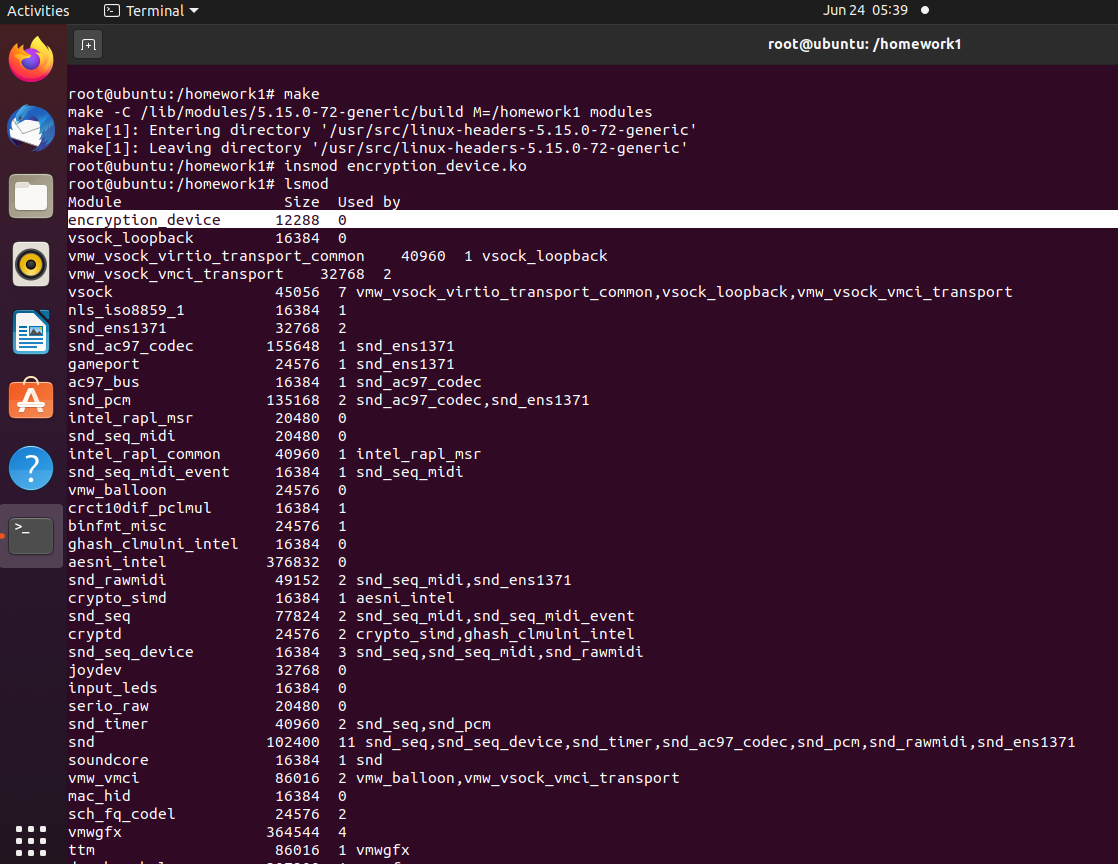


图3.6.1 编译及安装模块

## 3.7 测试截图

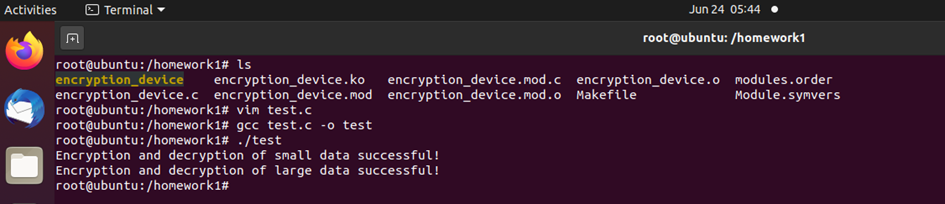


图3.6.2 测试程序运行

## 3.8 开源代码使用

无开源代码使用。

## 4．本课程技术要点

## 4.1 内核线程Kthread

Kthread可以通过kthread\_create()函数来创建。创建Kthread时，需要指定一个处理函数（threadfn）和一个线程名字。然后可以使用wake\_up\_process()函数来唤醒Kthread并开始执行。Kthread的处理函数（threadfn）定义了Kthread要执行的任务逻辑。Kthread主要用于执行与内核相关的异步操作、周期性任务、后台处理和延迟处理等。

## 4.2 SLAB对象

定义slab缓存：在驱动程序初始化阶段，使用kmem\_cache\_create()函数定义一个slab缓存。该函数指定了要分配的内存块的大小、构造和析构函数等。

分配内存：在需要分配内存的地方，可以使用kmem\_cache\_alloc()函数从slab缓存中分配一个内存块。这样可以避免频繁调用内核的内存分配器，提高内存分配的效率。

使用内存：获得分配的内存块后，可以使用它来存储和处理数据。可以像使用普通内存一样访问和操作slab对象。

释放内存：在不再需要使用分配的内存块时，使用kmem\_cache\_free()函数将内存块返回给slab缓存。这样可以避免显式调用内核的内存释放器，提高内存释放的效率。

## 4.3 内核同步机制

选择适当的同步机制：根据应用程序的需求和场景，选择合适的内核同步机制。常见的同步机制包括互斥锁、读写锁、自旋锁、信号量、事件和屏障等。

初始化同步对象：在使用同步机制之前，需要对同步对象进行初始化。这可能涉及分配内存、设置初始状态和参数等操作。

获取同步对象：在访问共享资源之前，获取适当的同步对象，如互斥锁或读写锁。获取同步对象通常使用加锁的操作，以确保只有一个线程或进程可以访问共享资源。

访问共享资源：一旦获得了同步对象的控制权，就可以访问共享资源，进行读取、写入或其他操作。

释放同步对象：在完成对共享资源的访问后，释放同步对象，以允许其他线程或进程获取对共享资源的访问权限。释放同步对象通常使用解锁的操作。

## 5. 程序测试

## 5.1 功能测试

通过比较加密前后的数据是否一致来判断是否加密成功。

## 5.2 性能测试

未进行性能测试。

## 6. 代码目录说明



图6.1 本项目的代码文件结构

表6.1 文件说明

|  |  |
| --- | --- |
| 文件或目录名 | 文件说明 |
| encryption\_device.c | 字符型加密驱动程序 |
| Makefile | 工程文件的编译规则 |
| test.c | 测试程序 |

## 7. 项目总结

在编写的过程中，运用到了实验一和实验二的知识，按照自己的需求定制的内核签名的有关选项，也学会了驱动程序的基本操作，在遇到困难时及时切换内核版本，提升了阅读底层代码的能力。