**《操作系统课程设计》**

**实验报告**

徽标, 公司名称

描述已自动生成

**题目: 设备管理试验**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212171**

**姓 名 杨晨**

**2024年 4月**

## 目 录

第一章 实验概述 1

1.1 实验目的 1

1.2 实验内容 1

1.2.1 XXX 1

1.2.2 XXXX 1

1.3 XXXX 1

第二章 实验步骤 2

2.1 XXXX 2

2.1.1 XXXX 2

2.1.2 XXX 2

2.2 XXXX 2

2.2.1 XXX 2

2.2.2 XXX 2

2.3 XXXXX 10

2.3.1 XXXX 10

2.4 XXXXX 12

2.4.1 XXXX 12

2.4.2 XXXX 16

第三章 实验结果分析 22

3.1 XXX 22

3.1.1 XXXX 22

3.1.2 XXXX 22

3.2 XXXXX 22

3.2.1 XXX 22

3.2.2 XXX 24

3.3 XXXX 24

3.3.1 XXXX 24

3.3.2 XXX 26

3.4 XXX 26

3.4.1 XXXXX 26

3.4.2 XXXX 29

3.4.3 XXXX 29

第四章 总结 36

4.1XXXXX 36

# **第一章 实验概述**

##### 1.1 实验目的

# **1.1.1 关于目录的制作和修改**

1. 了解Linux操作系统中的设备驱动程序的组成
2. 编写简单的字符设备(Simple Character Utility for Loading Localities,SCULL)驱动程序并进行测试。
3. 编写简单的块设备(Simple Block Utility for Loading Localities,SBULL)的驱动程序并进行测试。
4. 理解Linux操作系统的设备管理机制。

尽管计算机系统的I/O设备种类繁多，但操作系统在设备管理中为了提高通用性，往往采用层次化、模块化的方法实现I/O功能。这样用户和操作系统内核可以使用通用的接口来完成I/O操作，而屏蔽具体设备的操作细节。设备驱动程序一方面控制设备的操作，另一方面为内核提供统一的操作接口。字符设备和块设备是inux系统两大基本的设备类型，因此在设备管理实验中，通过编写字符设备和块设备的驱动程序，便可深入理解Linux操作系统的设备管理机制。

##### 1.2 实验内容

# **1.2.1 字符类型设备的驱动程序**

编写一个简单的字符设备驱动程序scull。要求该字符设备包括scull\_open()、scullwrite()、scull read()、scull ioctl()和 scull release()五个基本操作，还应再编写一个测试程序来测试这个字符设备驱动程序。

# **1.2.2 块类型设备的驱动程序**

编写一个简单的块设备驱动程序sbull，实现一套内存中的虚拟磁盘驱动器。要求该块设备包括 sbull\_open()、sbull\_ioctl()和sbull\_release()等基本操作。对每个驱动器，sbull分配一个内存数组，然后使这个数组可通过块操作来访问。sbull驱动可通过在该驱动器上进行分区、建立文件系统以及加载到系统层级中来测试。

此外，sbull设备被定义为一个可移出的设备。当最后一个用户关闭设备时，一个30s的定时器将被设置；如果设备在这个时间内不再次被打开，设备的内容将被清除，并且内核被告知介质已被改变。30s延迟给了用户一定时间做必要的操作，例如，在创建一个文件系统之后来卸载一个sbull设备

# **第二章 实验步骤**

##### 2.1 字符类型设备的驱动程序

先给出字符设备驱动程序要用到的数据结构定义：

// 定义设备结构体

struct scull\_dev

{

void \*\*data; // 数据指针数组

int quantum; // 当前量子大小

int qset; // 当前数组大小

unsigned long size; // 设备大小

unsigned int access\_key; // 用于sculluid和scullpriv

unsigned int usage; // 使用设备时锁定

unsigned int new\_msg; // 新消息标志

struct scull\_dev \*next; // 指向下一个设备的指针

};

struct scull\_dev scull;

# **2.1.1 字符设备的结构**

字符设备的结构device\_struct即字符设备的开关表。当字符设备注册到内核后，字符设备的名字和相关操作被添加到device\_struct结构类型的chrdevs全局数组中，称chrdevs为字符设备的开关表。

下面以一个简单的例子说明字符设备驱动程序中字符设备结构的定义(假设设备名为scul)。

/\*打开字符设备\*/

int scull\_open(struct inode \* inode,struct file \* filp);

/\*释放字符设备\*/

int scull\_release(struct inode \* inode, struct file \* filp);

/\*将数据送往字符设备\*/

ssize\_t scull\_write(struct file \* filp, const char \* buffer, size\_t count, loff\_t \* off);

/\*从字符设备读出数据,写人用户空间\*/

ssize\_t scull\_read(struct file \* filp, char \* buffer, size\_t count, loff\_t w off);

/\*字符设备的控制操作\*/

int seull\_ioctl(struct inode \* inode, struct file \* filp, unsigned int cmd, unsigned long arg);

struct file\_operations scull\_chops={

read:scull\_read,

write:scull\_write,

ioctl:scull\_ioctl,

open:scull\_open,

release:scull\_release

};

# **2.1.2 字符设备驱动程序入口点**

字符设备驱动程序入口点主要包括初始化字符设备、字符设备的I/O调用和中断。在引导系统时，每个设备驱动程序通过其内部的初始化函数对其控制的设备及其自身初始化字符设备初始化函数为 chr\_dev\_init()，包含在/linux/drivers/char/mem.c中,它的主要功能之一是在内核中登记设备驱动程序。具体调用是通过register\_chrdev()函数。registerchrdev()函数定义如下：

#include <linux/errno.h>

#include <linux/fs.h>

int register\_chrdev(unsigned int major, const char \*name, struct file\_operations \*chops)

其中major是为设备驱动程序向系统申请的主设备号。如果为0，则系统为该驱动程序动态地分配一个主设备号；name是设备名；fops是前面定义的file\_operations结构的指针。在登记成功的情况下，如果指定了major，则register\_chrdev()函数返回值为0；如果major值为0，则返回内核分配的主设备号。并且当register\_chrdev()函数操作成功时，设备名就会出现在/proc/devices文件里；在登记失败的情况下，register\_chrdev()函数返回值为负。

初始化部分一般还负责给设备驱动程序申请系统资源，包括内存、中断、时钟、I/O端口等，这些资源也可以在open()子程序或别的地方申请。当不用这些资源的时候，应该释放它们，以利于资源的共享。

用于字符设备的I/〇调用主要有 open()、release()、read()、write()和 ioctl()。open()函数的使用比较简单，当一个设备被进程打开时，open()函数被唤醒：

int scull\_open(struct inode \*inode,struct file \*filp){

...

try\_module\_get(THIS\_MODULE);

return 0;

}

为了避免模块在使用过程中被卸载，Linux内核通过“使用计数”来描述模块是否在被使用。在2.4版本内核中，模块使用MODINCUSECOUNT和MODDEC USECOUNT两个宏来管理自己被使用的计数。2.6版本的Linux内核提供了更健壮、灵活的模块计数管理接口：try\_module\_get(&module)和 module\_put(&module)，用来替换 2.4版本内核中的模块使用计数管理宏。

release()函数的使用和open()数相似：

int scull\_release(struct inode \* inode, struct file \* filp){

...

module\_put(THIS\_MODULE);

return 0;

}

当设备文件执行read()调用时，看起来是从设备中读取数据，实际上是从内核数据队列中读取，并传送给用户空间。设备驱动程序的write()函数的使用和read()函数相似,只不过是数据传送的方向发生了变化，即按要求的字节数count从用户空间的缓冲区buf复制到硬件或内核的缓冲区中。

ioctl是设备驱动程序中对设备的I/O通道进行管理的函数。所谓对I/0通道进行管理，就是对设备的一些特性进行控制，例如串口的传输波特率、马达的转速等。有时我们需要获取或改变正在运行的设备的参数，这时就要用到ioctl()函数：

int scull\_ioctl(struct inode \*inode, struct file \*filp, unsigned int cmd, unsigned long arg)

其中参数cmd是驱动程序要执行的命令的特殊代码；参数arg是任何类型的4字节数，它为特定的cmd提供参数。Linux中定义了4种ioctl()函数调用：

IO(type, nr) //无参数的 ioctl

IOR( type, nr,size) //有读参数的 ioctl(copy\_to\_user)

IOW(type,nr,size) //有写参数的ioctl(copy\_from\_user)

IOWR(type,nr,size) //有读写参数的ioct1

上述的“读”和“写”是从用户视角，就像系统调用read和write，用户“读”实际上需要内核从内核空间或设备上复制数据到用户空间(copy\_to\_user)，反之亦然。以上4个宏定义中，第一个参数type是标识符，有时也通过设备的主设备号来标识。由于有大量的设备，因此有很多设备会共享同一个type标识符。第二个参数nr是一个用来区分不同ioctl的序列号。第三个参数size是需要传递给内核或从内核中传回数据的数据类型。

为字符设备定义如下3个没有参数传递的I/O操作，分别是SCULL\_RESET、SCULLQUERY\_NEW\_MSG 和SCULL QUERY MSG\_LENGTH.

#include<linux/ioctl.h>

#define SCULL\_MAJOR 253

#define SCULL\_NAME "test\_device"

#define DEVICE\_FIE "/dev/scull"

#define SCULL\_MAGIC SCULL\_MAJOR

#define SCULL\_RESET\_IO(SCULL\_MAGIC，0) //重置数据

#define SCULL\_QUERY\_NEW\_MSGIO(SCULL\_MAGIC，1) //检查新消息

#define SCULL\_QUERY\_MSG\_LENGTH\_IO(SCULL\_MAGIC，2) //获取消息长度

#define IOC\_NEW\_MSG 1

下面给出几个人口函数流程图的参考设计。

1. 函数scull\_open()，如图所示

图示

描述已自动生成

1. 函数 scull\_write()，如图所示

图示

描述已自动生成

1. 函数scull\_read()，如图所示

图示

描述已自动生成

1. 函数scull\_ioctl()，如图所示

图示

描述已自动生成

1. 函数scull\_release()，如图所示

图示

描述已自动生成

# **2.1.3 字符设备驱动程序的安装**

编写完设备驱动程序后，下一项任务是对它进行编译并装入内核。若要编译字符设备驱动程序，可以使用如下Makefile文件：

obj-m+=scull.o

KDIR=/usr/src/kernels/$(shell uname -r)

PWD:=$(shell pwd)

scull.o: scull.c scull.h

@$(MAKE) -C $(KDIR) M=${PWD} modules

@echo ""

@echo "Run \"insmod scull.ko\" to install scull device."

@echo "Run \"mknod DEVICE\_FILE c MAJOR MINOR\" to make a device file."

@echo ""

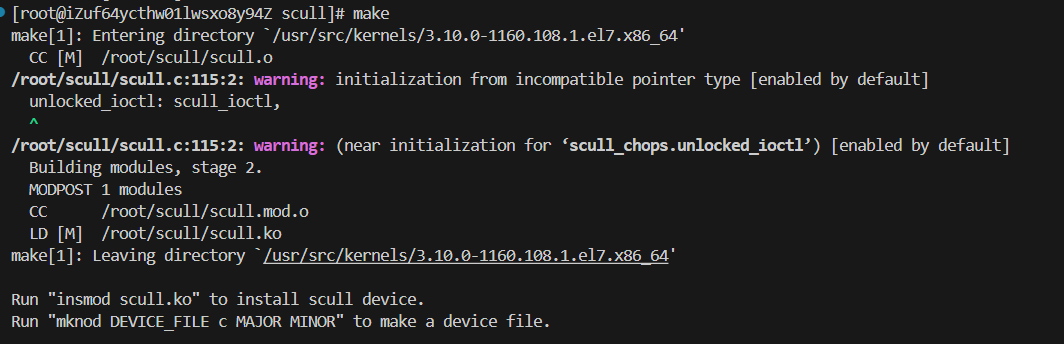
clean:

@rm -f \*.ko \*.o \*.mod.c \*.symvers \*.order \*.unsigned scull\_test

scull\_test: scull\_test.c scull.h

@gcc -o scull\_test scull\_test.c

.PHONY: clean



然后，使用insmod命令装人编译好的内核模块



===内容替换成自己的论文文字===

# **2.1.4 测试程序**

在该字符设备驱动程序编译加载后，再在/dev目录下创建字符设备文件scull，使用如下命令：



其中c表示scull是字符设备，“major"是scull的主设备号。该字符设备驱动程序编译加载后，可在/proc/devices文件中获得主设备号。

文本

描述已自动生成

测试程序由write\_proc和read\_proc两个函数组成。其中，write\_proc负责向字符设备/dev/scull中写人消息，read\_proc则从该设备中读取刚刚写人的消息。为了测试字符设备，需要通过两个终端接入系统，每个终端分别运行一个scull\_test程序。其中，一个传人read参数，另一个传人write参数。当write进程还没有向字符设备写人数据时，read 进程将挂起，等待字符设备可以读出数据；同样，当read进程还没有读取字符设备上的消息时write进程将挂起，不接收用户输人。程序运行的效果是，用户从write进程输人文本并按回车后，read进程上将立即显示刚刚输人的消息。当用户从write进程输人exit后，write进程将退出；read进程从字符设备上读取到exit后，read进程将退出。

##### 参考文献

[1] 附件1-1-1操作系统实验指导——基于Linux内核“第8章设备管理实验”