

嵌入式软件开发技术与工具实验报告六

黎炜彬

一、实验目的

能够综合应用课程所学的技术与工具，包括：

- Socket通信
- 多进程、多线程编程
- 交叉调试目标端程序
- 磁盘分区与文件系统创建
- 模块与驱动编程

二、实验内容

1. 将树莓派设为智能家居Linux服务器，可用来采集并维护环境数据，如PM2.5、温度、湿度、气味、电器状态数据等。在实际环境中数据来自相应的传感器，本次试验中用scull设备模拟。有条件的小组鼓励使用真实设备采集数据。
2. 要求创建2个以上的scull设备，设备驱动可选择从内核源码树外(Kbuild)编译安装，或加入到内核源码树内。驱动函数要求包括：open, release, read, write, llseek, ioctl。
3. 实验中的环境数据存储在特定文件系统中。该文件系统要求具备属性：在线写入、持久性、断电可靠性。
4. PC机、移动设备或另外一个树莓派用作远程客户端，随时请求获取环境数据，客户端和服务端之间采用Socket通信。
5. APP编译采用交叉编译，用gdb-gdbserver交叉调试APP。

三、实验过程与结果

1. scull设备驱动

- scull设备创建

本实验创建两个scull设备，函数包括：open, release, read, write, llseek, ioctl，实现对数据的操作。

源代码分析：

- scull_open

```
int scull_open(struct inode *inode, struct file *filp)
{
    struct scull_dev *dev;

    dev = container_of(inode->i_cdev, struct scull_dev, cdev);
    filp->private_data = dev;

    if ((filp->f_flags & O_ACCMODE) == O_WRONLY)
```

```

{
    if (down_interruptible(&dev->sem))
    {
        return -ERESTARTSYS;
    }
    scull_trim(dev);
    up(&dev->sem);
}

```

scull_open函数实现打开一个scull设备。应具有以下功能：1、检查设备特定的错误；2、如果设备是首次打开，则对其进行初始化；3、如有必要，更新f_op指针；4、分配并填写置于filp->private_data里的数据结构。container_of是在/linux/kernel.h中定义的宏，根据一个结构体变量中的一个域成员变量的指针来获取指向整个结构体变量的指针，此处获得我们自定义的scull结构体指针，并存入filp->private_data中。再调用scull_trim函数实现对所有内存空间的释放。

- scull_release

```

int scull_release(struct inode *inode, struct file *filp)
{
    return 0;
}

```

scull_release函数实现关闭设备，return 0即可。

- scull_read

```

ssize_t scull_read(struct file *filp, char __user *buf, size_t count, loff_t
*f_pos)
{
    struct scull_dev *dev = filp->private_data;
    ssize_t retval = 0;

    if (down_interruptible(&dev->sem))
    {
        return -ERESTARTSYS;
    }
    if (*f_pos >= dev->size)
    {
        goto out;
    }
    if (*f_pos + count > dev->size)
    {
        count = dev->size - *f_pos;
    }

    if (!dev->data)
    {
        goto out;
    }
}

```

```

    if (raw_copy_to_user(buf, dev->data + *f_pos, count))
    {
        retval = -EFAULT;
        goto out;
    }

    *f_pos += count;
    retval = count;

out:
    up(&dev->sem);
    return retval;
}

```

scull_read实现从设备中读取数据，调用`raw_copy_to_user`函数实现从内核空间到用户空间的拷贝，根据返回值判断读取情况。如果返回值等于传递给read系统调用的count参数，则说明所请求的字节数传输成功完成了。如果返回值是正的，但是比count小，则说明只有部分数据成功传送。如果返回值为0，则表示已经到达了文件尾。负值意味着发生了错误，该值指明了发生什么错误，错误码在<linux/errno.h>中定义。

- scull_write

```

ssize_t scull_write(struct file *filp, const char __user *buf, size_t count,
loff_t *f_pos)
{
    struct scull_dev *dev = filp->private_data;
    ssize_t retval = -ENOMEM;

    if (down_interruptible(&dev->sem))
    {
        return -ERESTARTSYS;
    }

    if (!dev->data)
    {
        dev->data = kmalloc(SCULL_BUFFER_SIZE, GFP_KERNEL);
        if (!dev->data)
        {
            goto out;
        }
        memset(dev->data, 0, SCULL_BUFFER_SIZE);
    }

    if (count > SCULL_BUFFER_SIZE - dev->size)
    {
        count = SCULL_BUFFER_SIZE - dev->size;
    }

    if (raw_copy_from_user(dev->data + dev->size, buf, count))
    {

```

```

        retval = -EFAULT;
        goto out;
    }

    dev->size += count;
    retval = count;

out:
    up(&dev->sem);
    return retval;
}

```

scull_write实现向设备写入数据，调用raw_copy_from_user函数将数据从用户空间拷贝到内核空间，根据返回值判断写入的情况。如果返回值等于count，则完成了所请求数目的字节传送。如果返回值是正的，但小于count，则只传送了部分数据。程序很可能再次试图写入余下的数据。如果值为0，意味着什么也没有写入。负值意味着发生了错误。

- scull_ioctl

```

long scull_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    struct scull_dev *dev = filp->private_data;
    int err = 0, retval = 0;

    if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC)
        return -ENOTTY;

    if (_IOC_NR(cmd) > SCULL_IOC_MAXNR)
        return -ENOTTY;

    if (_IOC_DIR(cmd) & _IOC_READ)
        err = !access_ok(VERIFY_WRITE, (void *)arg, _IOC_SIZE(cmd));
    else if (_IOC_DIRscull_(cmd) & _IOC_WRITE)
        err = !access_ok(VERIFY_READ, (void *)arg, _IOC_SIZE(cmd));
    if (err)
        return -EFAULT;

    switch (cmd) {
        case SCULL_IOC_CLEAR:
            dev->data_i = 0;
            printk(KERN_EMERG"SCULL_IOC_CLEAR data: 0\n");
            break;
        case SCULL_IOC_GET:
            retval = __put_user(dev->data_i, (int __user *)arg);
            printk(KERN_EMERG"SCULL_IOC_GET data: %d\n", dev->data_i);
            break;
        case SCULL_IOC_QUERY:
            printk(KERN_EMERG"SCULL_IOC_QUERY data: %d\n", dev->data_i);
            retval = dev->data_i;
            break;
        case SCULL_IOC_SET:

```

```

        retval = __get_user(dev->data_i, (int __user *)arg);
        printk(KERN_EMERG"SCULL_IOC_SET data: %d\n", dev->data_i);
        break;
    case SCULL_IOC_TELL:
        dev->data_i = arg;
        printk(KERN_EMERG"SCULL_IOC_TELL data: %d\n", arg);
        break;
    default:
        retval = -EINVAL;
        break;
}

return retval;
}

```

在驱动程序中，IO指令由4个部分组成：幻数magic、序数number、位置direction、大小size，四部分组合可唯一确定一条指令，在头文件scull.h中定义。

scull_ioctl中定义了五条指令，分别是清除数据、通过指针从设备获得数据、通过返回值从设备获得数据、通过指针向设备写数据、通过参数向设备写数据，使用printk输出调试信息。

- scull_llseek

```

loff_t scull_llseek(struct file *filp, loff_t off, int whence)
{
    struct scull_dev *dev = filp->private_data;
    loff_t newpos;

    switch(whence)
    {
        case 0:
            newpos = off;
            break;
        case 1:
            newpos = filp->f_pos + off;
            break;
        case 2:
            newpos = dev->size + off;
            break;
        default:
            return -EINVAL;
    }
    if (newpos < 0)
    {
        return -EINVAL;
    }
    filp->f_pos = newpos;
    return newpos;
}

```

lseek的作用是重新定位读/写文件偏移量，whence共有三种取值与代码中一一对应：1.SEEK_SET, 偏移量设置为offset字节；2.SEEK_CUR, 偏移量设置为当前位置加上offset字节；3.SEEK_END, 偏移量设置为文件大小加上偏移字节大小。成功完成后，lseek () 返回的结果是从文件开头的字节偏移位置。否则，返回-1并设置errno以指示错误。

- scull设备安装

本实验共创建两个scull设备，分别为scull0，scull1。采用在内核源码树内编译的方式。

将scull0.c scull0.h scull1.c scull1.h复制到内核源码树/linux/drivers/char/目录下，修改该目录下的kconfig，加入两个设备

```
config MYSCULL0
    tristate "sculldriver0 by 0B703"
    default m
    help
    This is a virtual char device driver. Choose M to install it as a module.

config MYSCULL1
    tristate "sculldriver1 by 0B703"
    default m
    help
    This is a virtual char device driver. Choose M to install it as a module.
```

在kconfig选项中加入default m可以默认将两个scull设备作为模块编译，而不用在menuconfig中进行选择

再在该目录的makefile中加入编译选项

```
obj-$(CONFIG_MYSCULL0)      += scull0.o
obj-$(CONFIG_MYSCULL1)      += scull1.o
```

在linux源码根目录打开终端，使用make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules进行编译，即可在/linux/drivers/char/目录下得到scull0.ko scull1.ko两个驱动文件。

```

liwieheng@liwieheng-ThinkPad-X1-Carbon-5th: ~/linux
    from ./include/linux/kernel.h:14,
    from ./include/linux/list.h:9,
    from ./include/linux/module.h:9,
    from drivers/char/scull1.c:3:
drivers/char/scull1.c: In function 'scull_ioctl':
./include/linux/kern_levels.h:5:18: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat=]
#define KERN_SOH "\001" /* ASCII Start Of Header */
    ^
./include/linux/kern_levels.h:8:20: note: in expansion of macro 'KERN_SOH'
#define KERN_EMERG KERN_SOH "0" /* system is unusable */
    ^
drivers/char/scull1.c:174:20: note: in expansion of macro 'KERN_EMERG'
    printk(KERN_EMERG"SCULL_IOC_TELL data: %d\n", arg);
    ^
Kernel: arch/arm/boot/Image is ready
Kernel: arch/arm/boot/zImage is ready
Building modules, stage 2.
MODPOST 1172 modules
CC      drivers/char/scull0.mod.o
LD [M]  drivers/char/scull0.ko
CC      drivers/char/scull1.mod.o
LD [M]  drivers/char/scull1.ko
liwieheng@liwieheng-ThinkPad-X1-Carbon-5th:~/linux$

```

将文件传输到树莓派端

```

liwieheng@liwieheng-ThinkPad-X1-Carbon-5th:~/linux/drivers/char$ sudo scp scull1
.ko pi@192.168.0.109:/home/pi
[sudo] liwieheng 的密码:
pi@192.168.0.109's password:
scull1.ko                                100% 10KB  9.8KB/s  00:00
liwieheng@liwieheng-ThinkPad-X1-Carbon-5th:~/linux/drivers/char$ sudo scp scull0
.ko pi@192.168.0.109:/home/pi
pi@192.168.0.109's password:
scull0.ko                                100% 10KB  9.8KB/s  00:00
liwieheng@liwieheng-ThinkPad-X1-Carbon-5th:~/linux/drivers/char$

```

在树莓派端使用 `sudo insmod` 命令加载驱动，`lsmod` 可观察到两个设备加载成功。

```

pi@raspberrypi: ~
文件(F) 编辑(E) 标签(T) 帮助(H)
pi@raspberrypi:~ $ insmod scull0.ko
insmod: ERROR: could not insert module scull0.ko: Operation not permitted
pi@raspberrypi:~ $ sudo insmod scull0.ko
pi@raspberrypi:~ $ sudo insmod scull1.ko
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
scull1                  16384  0
scull0                  16384  0
rfcomm                  49152  4
bnep                    20480  2
hci_uart                40960  1
btbcm                   16384  1 hci_uart
serdev                  20480  1 hci_uart
bluetooth               385024 29 hci_uart,bnep,btbcm,rfcomm
ecdh_generic            28672  1 bluetooth
fuse                    110592  3
brcmfmac                315392  0
brcmutil                16384  1 brcmfmac
raspberrypi_hwmon       16384  0
hwmon                   16384  1 raspberrypi_hwmon
cfg80211                647168  1 brcmfmac
rfkill                  28672  6 bluetooth,cfg80211
snd_bcm2835             28672  2
bcm2835_codec           36864  0

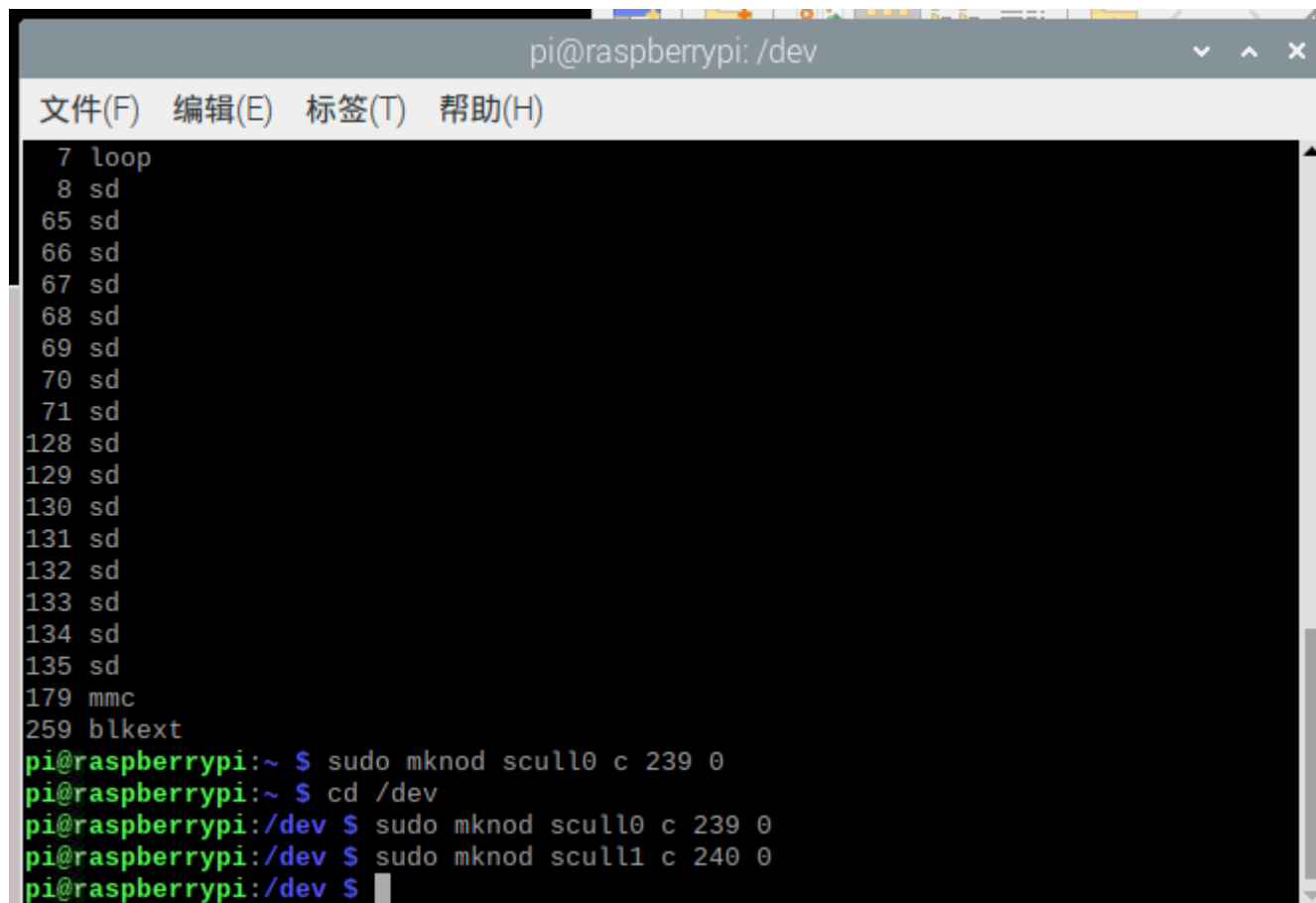
```

输入`cat /proc/devices`获得驱动的主设备号，并根据设备号使用`sudo mknod`创建驱动文件

```

pi@raspberrypi: ~
文件(F) 编辑(E) 标签(T) 帮助(H)
89 i2c
116 alsa
128 ptm
136 pts
162 raw
180 usb
189 usb_device
204 ttyAMA
216 rfcomm
239 scull0
240 scull1
241 media
242 uio
243 vchiq
244 vcsn
245 hidraw
246 rpmb
247 bcm2835-gpiomem
248 vcio
249 vc-mem
250 bsg
251 watchdog
252 BaseRemoteCtl
253 rtc

```

```
pi@raspberrypi: /dev
文件(F) 编辑(E) 标签(T) 帮助(H)
7 loop
8 sd
65 sd
66 sd
67 sd
68 sd
69 sd
70 sd
71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
259 blkext
pi@raspberrypi:~ $ sudo mknod scull0 c 239 0
pi@raspberrypi:~ $ cd /dev
pi@raspberrypi:/dev $ sudo mknod scull0 c 239 0
pi@raspberrypi:/dev $ sudo mknod scull1 c 240 0
pi@raspberrypi:/dev $
```

scull设备在树莓派上加载成功。

- scull设备测试

编写test.c文件对scull设备进行测试，主要测试其read、write、ioctl功能，test.c源码如下

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <linux/i2c.h>
#include <linux/fcntl.h>
#include <linux/fs.h>
#include <error.h>
#include <sys/ioctl.h>

#define SCULL_IOC_MAGIC '$'

//定义命令->
//数据清零
#define SCULL_IOC_CLEAR _IO(SCULL_IOC_MAGIC, 0)
//获取数据(通过指针)
#define SCULL_IOC_GET _IOR(SCULL_IOC_MAGIC, 1, int)
//获取数据(通过返回值)
#define SCULL_IOC_QUERY _IO(SCULL_IOC_MAGIC, 2)
//设置数据(通过指针)
#define SCULL_IOC_SET _IOW(SCULL_IOC_MAGIC, 3, int)
```

```
//设置数据(通过直接引用参数值)
#define SCULL_IOC_TELL _IO(SCULL_IOC_MAGIC, 4)

int main()
{
    int fd;
    int data;
    char buf[]="scull character device test by liweiheng";//写入 scull设备的内容
    char buf_read[4096]; //scull设备的内容读入到该buf中

    if((fd=open("/dev/scull1",O_RDWR))<0){
        //打开scull设备
        perror("open");
        printf("open scull WRONG! \n");
        exit(1);
    }
    else
        printf("open scull SUCCESS!\n");

    printf("buf is :%s\n",buf);
    write(fd,buf,sizeof(buf)); //把buf中的内容写入scull设备

    lseek(fd,0,SEEK_SET); //把文件指针重新定位到文件开始的位置

    read(fd,buf_read,sizeof(buf)); //把scull设备中的内容读入到 buf_read中

    printf("buf_read is :%s\n",buf_read);

    //数据清零
    ioctl(fd, SCULL_IOC_CLEAR);

    //直接传值测试
    data = ioctl(fd, SCULL_IOC_QUERY);
    data = 100;
    ioctl(fd, SCULL_IOC_TELL, data);

    //指针传值测试
    ioctl(fd, SCULL_IOC_GET, &data);
    data = 122;
    ioctl(fd, SCULL_IOC_SET, &data);

    return 0;
}
```

编译后运行./test，成功输出结果，左侧终端使用`sudo cat /proc/kmsg`用于观察printk输出的调试信息，右侧终端观察正常调试信息，可见read、write、ioctl功能正常，设备测试成功。

pi@raspberrypi: ~

文件(F) 编辑(E) 标签(T) 帮助(H)

pi@raspberrypi:~ \$ sudo cat /proc/kmsg
<0>[306.743058] SCULL_IOC_CLEAR data: 0
<0>[306.743091] SCULL_IOC_QUERY data: 0
<0>[306.743103] SCULL_IOC_TELL data: 100
<0>[306.743114] SCULL_IOC_GET data: 100
<0>[306.743125] SCULL_IOC_SET data: 122
█

pi@raspberrypi: ~

文件(F) 编辑(E) 标签(T) 帮助(H)

videobuf2_common 45056 4 bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
snd_timer 32768 1 snd_pcm
videodev 200704 6 bcm2835_codec,v4l2_common,videobuf2_common,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
snd 73728 7 snd_timer,snd_bcm2835,snd_pcm
media 36864 3 bcm2835_codec,videodev,v4l2_mem2mem
vc_sm_cma 36864 1 bcm2835_mmal_vchiq
fixed 16384 0
uio_pdrv_genirq 16384 0
uio 20480 1 uio_pdrv_genirq
i2c_dev 20480 0
ip_tables 24576 0
x_tables 32768 1 ip_tables
ipv6 454656 30
pi@raspberrypi:~ \$ sudo ./test1
open scull SUCCESS!
buf is :scull character device test by liweiheng
buf_read is :scull character device test by liweiheng
pi@raspberrypi:~ \$ sudo ./test1
open scull SUCCESS!
buf is :scull character device test by liweiheng
buf_read is :scull character device test by liweiheng
pi@raspberrypi:~ \$ █

- 总结

scull设备是一种学习linux驱动编写的工具，通过编写scull驱动，加深了对linux运行机制的认识，在驱动编写的过程中也遇到过一些问题，例如：

- 1. `copy_to_user`，`copy_from_user`在新版linux内核中已不再使用，需要改为`raw_copy_from_user`，`raw_copy_to_user`才能成功编译。
- 2. 在linux内核源码树外编译scull遇到了无法找到头文件的问题，多次尝试后不能解决，尝试在内核源码树内编译成功，在kconfig选项中加入default m可以默认将两个scull设备作为模块编译，而不用在menuconfig中进行选择，相对更加方便。
- 3. printk无法在终端显示，需要打开一个新的终端，使用`sudo cat /proc/kmsg`才能观察printk的调试输出。

四、实验总结