实验12 字符设备驱动程序

实验目的

1. 学习嵌入式Linux的内核GPIO库函数的使用方式；
2. 学习Linux设备驱动程序的开发过程。

实验器材

硬件

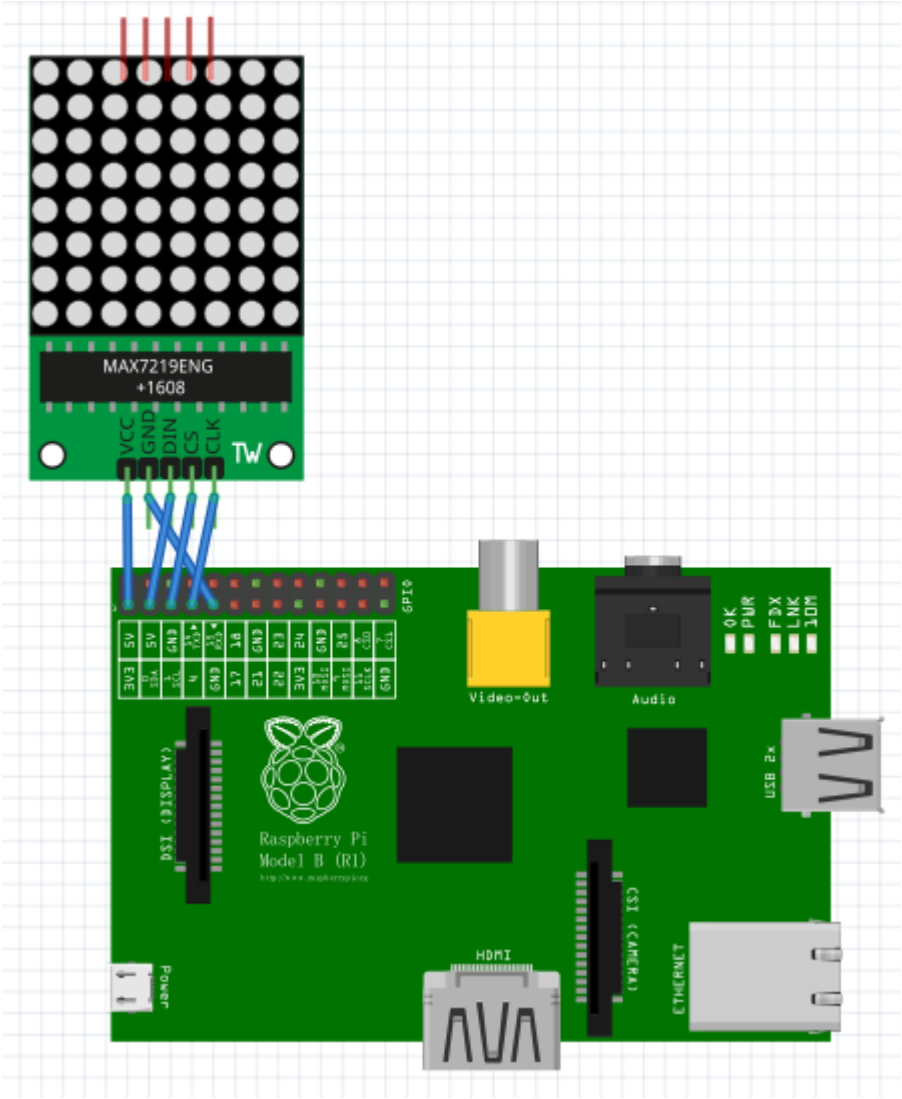
- Linux实验板卡1块；
- 5V/1A电源1个；
- microUSB线1根；
- 面包板1块；
- 串行接口8x8 LED矩阵1个；
- 面包线若干。

软件

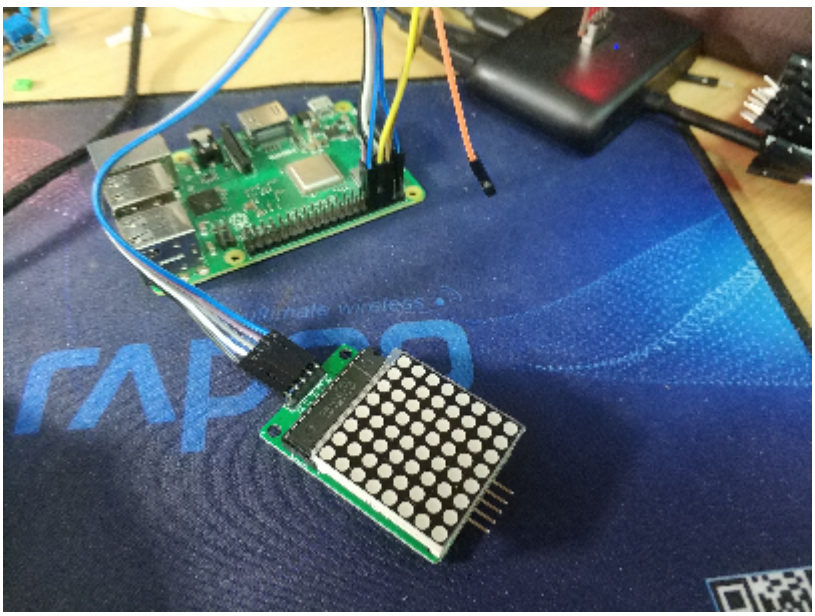
- 编译软件；
- Fritzing。

实验接线

示意图


MAX7219ENG
+1608
VCC
GND
DIN
CS
CLK
GPIO
Video-Out
Audio
OK
PWR
FDX
LNK
10M
USB 2x
DSI (DISPLAY)
Raspberry Pi
Model B (R1)
CSI (CAMERA)
ETHERNET
HDMI
Power

实际图



实验步骤

配置步骤

编写c程序显示字符

1. 下载wiringPi库，链接为https://github.com/WiringPi/WiringPi，并在树莓派上安装

```
pi@raspberrypi:~/Desktop $ cd \WiringPi-master
pi@raspberrypi:~/Desktop/WiringPi-master $ ./build
wiringPi Build script
======================


WiringPi Library
[UnInstall]
[Compile] wiringPi.c
[Compile] wiringSerial.c
```

2. 需要在编译.c时加上`-lwiringPi`：

```
NOTE: To compile programs with wiringPi, you need to add:
    -lwiringPi
  to your compile line(s) To use the Gertboard, MaxDetect, etc.
  code (the devLib), you need to also add:
    -lwiringPiDev
  to your compile line(s).

pi@raspberrypi:~/Desktop/WiringPi-master $
```

3. 编写c程序并编译运行

```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ gcc test.c -lwiringPi
pi@raspberrypi:~/Desktop $ ./a.out
^C
pi@raspberrypi:~/Desktop $
```

编写驱动程序

1. `sudo apt install raspberrypi-kernel-headers`下载内核头文件

```
pi@raspberrypi:~/Desktop $ sudo apt install raspberrypi-kernel-headers
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列【新】软件包将被安装：
  raspberrypi-kernel-headers
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 197 个软件包未被升
级。
需要下载 24.9 MB 的归档。
解压缩后会消耗 163 MB 的额外空间。
获取:1 http://archive.raspberrypi.org/debian buster/main armhf raspberrypi-kerne
l-headers armhf 1.20200601-1 [24.9 MB]
已下载 24.9 MB，耗时 17秒 (1,442 kB/s)
正在选中未选择的软件包 raspberrypi-kernel-headers。
(正在读取数据库 ... 系统当前共安装有 155373 个文件和目录。)
准备解压 .../raspberrypi-kernel-headers_1.20200601-1_armhf.deb ...
正在解压 raspberrypi-kernel-headers (1.20200601-1) ...

正在设置 raspberrypi-kernel-headers (1.20200601-1) ...
```

2. `ls /usr/src` `uname -r` 分别查看所安装的内核头文件版本和系统的内核版本，发现内核头文件的版本和内核版本不匹配，因此需要通过 `sudo apt-get install --reinstall raspberrypi-bootloader raspberrypi-kernel` 升级系统内核

```
pi@raspberrypi:~/Desktop $ ls /usr/src
linux-headers-4.19.118+     linux-headers-4.19.118-v7l+
linux-headers-4.19.118-v7+  sense-hat
pi@raspberrypi:~/Desktop $ uname -r
4.19.97-v7+
pi@raspberrypi:~/Desktop $ uname -a
Linux raspberrypi 4.19.97-v7+ #1294 SMP Thu Jan 30 13:15:58 GMT 2020 armv7l GNU/
Linux
pi@raspberrypi:~/Desktop $ sudo apt-get install --reinstall raspberrypi-bootload
er raspberrypi-kernel
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
```

3. 升级完成后重启发现内核与内核头文件版本号一致

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.118-v7+ #1311 SMP Mon Apr 27 14:21:24 BST 2020 armv7l GNU
/Linux
pi@raspberrypi:~ $ ls /usr/src
linux-headers-4.19.118+     linux-headers-4.19.118-v7l+
linux-headers-4.19.118-v7+  sense-hat
pi@raspberrypi:~ $
```

4. 编写代码与Makefile

   代码包含以下内容：

   - 以下三个函数的实现

     ```
     static int char_driver_open(struct inode *node, struct file *file);
     static int char_driver_close(struct inode *node, struct file *file);
     static ssize_t char_driver_write(struct file *file, const char __user *buf,
     size_t size, loff_t *offset);
     ```

   - 将文件操作结构体与上面三个函数相关联
   - 点阵交互的接口程序实现

   Makefile中要注意使用的内核头文件与内核版本相对应

5. 编译

```
pi@raspberrypi:~/Desktop/LAB12/2 $ make all
make -C /usr/src/linux-headers-4.19.118-v7+/ M=/home/pi/Desktop/LAB12/2 modules
make[1]: 进入目录“/usr/src/linux-headers-4.19.118-v7+”
  CC [M]  /home/pi/Desktop/LAB12/2/char_driver.o
/home/pi/Desktop/LAB12/2/char_driver.c: In function ‘char_driver_write’:
/home/pi/Desktop/LAB12/2/char_driver.c:127:1: warning: no return statement in fu
nction returning non-void [-Wreturn-type]
 }
 ^
/home/pi/Desktop/LAB12/2/char_driver.c:120:9: warning: ignoring return value of
‘copy_from_user’, declared with attribute warn_unused_result [-Wunused-result]
         copy_from_user(&ch, &buf[i], 1);
         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/pi/Desktop/LAB12/2/char_driver.mod.o
  LD [M]  /home/pi/Desktop/LAB12/2/char_driver.ko
make[1]: 离开目录“/usr/src/linux-headers-4.19.118-v7+”
```

6. `sudo insmod char_driver.ko` 加载驱动模块

```
pi@raspberrypi:~/Desktop/LAB12/2 $ sudo insmod char_driver.ko
pi@raspberrypi:~/Desktop/LAB12/2 $ lsmod
Module                  Size  Used by
char_driver            16384  0
fuse                  110592  3
rfcomm                 49152  4
bnep                   20480  2
hci_uart               40960  1
btbcm                  16384  1 hci_uart
serdev                 20480  1 hci_uart
bluetooth             389120  29 hci_uart,bnep,btbcm,rfcomm
```

7. `cat /proc/devices` 查看设备编号

```
pi@raspberrypi:~/Desktop/LAB12/2 $ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
 89 i2c
116 alsa
128 ptm
136 pts
162 raw
180 usb
189 usb_device
204 ttyAMA
216 rfcomm
240 char_driver
241 media
```

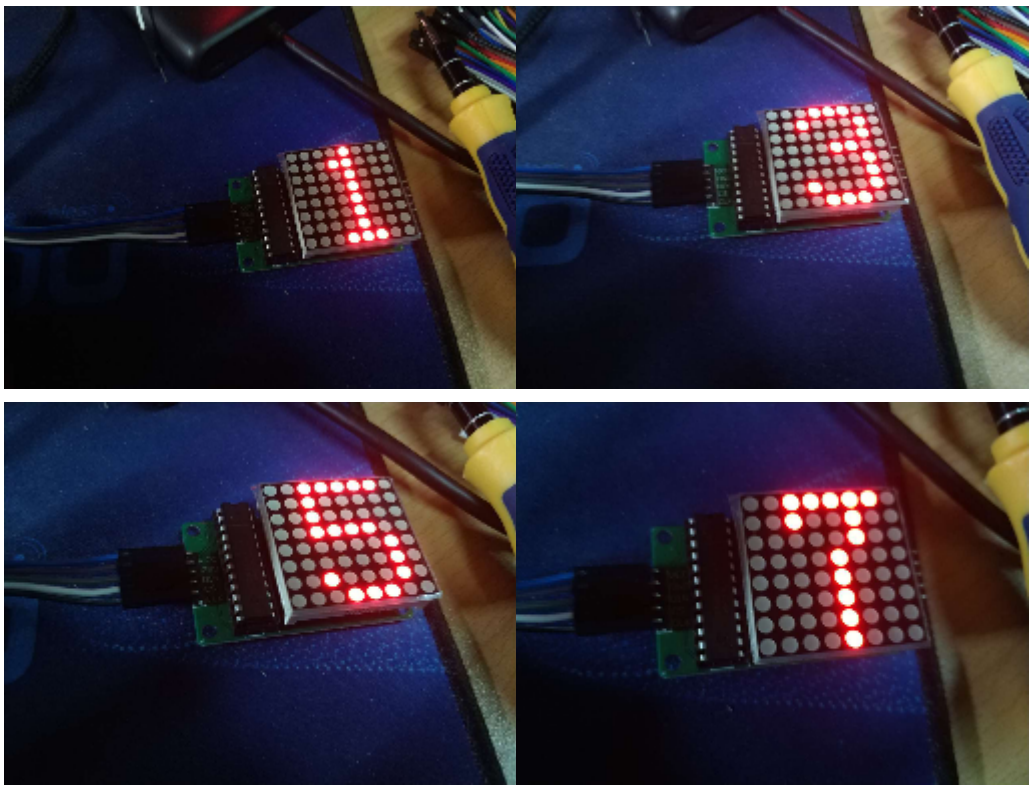
8. 用 `sudo mknod /dev/char_driver` 创建设备文件

```
pi@raspberrypi:~/Desktop/LAB12/2 $ sudo mknod /dev/char_driver c 240 0
```

9. `sudo passwd --unlock root` `su root`获取root权限，并写入字符

```
root@raspberrypi:/home/pi/Desktop/LAB12/2# echo "1" > /dev/mydriver
root@raspberrypi:/home/pi/Desktop/LAB12/2# echo "1" > /dev/char_driver
root@raspberrypi:/home/pi/Desktop/LAB12/2# echo "3" > /dev/char_driver
root@raspberrypi:/home/pi/Desktop/LAB12/2# echo "5" > /dev/char_driver
root@raspberrypi:/home/pi/Desktop/LAB12/2# echo "7" > /dev/char_driver
```

10. 实验结果



源代码

编写c程序显示字符

```c
#include <wiringPi.h>

#define DIN 8
#define CS 9
#define CLK 7

#define uchar unsigned char
#define uint  unsigned int

// 译码
uchar disp[36][8]={
{0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C},//0
{0x10,0x18,0x14,0x10,0x10,0x10,0x10,0x10},//1
{0x7E,0x2,0x2,0x7E,0x40,0x40,0x40,0x7E},//2
{0x3E,0x2,0x2,0x3E,0x2,0x2,0x3E,0x0},//3
{0x8,0x18,0x28,0x48,0xFE,0x8,0x8,0x8},//4
{0x3C,0x20,0x20,0x3C,0x4,0x4,0x3C,0x0},//5
```

```
{0x3C,0x20,0x20,0x3C,0x24,0x24,0x3C,0x0},//6
{0x3E,0x22,0x4,0x8,0x8,0x8,0x8,0x8},//7
{0x0,0x3E,0x22,0x22,0x3E,0x22,0x22,0x3E},//8
{0x3E,0x22,0x22,0x3E,0x2,0x2,0x2,0x3E},//9
{0x8,0x14,0x22,0x3E,0x22,0x22,0x22,0x22},//A
{0x3C,0x22,0x22,0x3E,0x22,0x22,0x3C,0x0},//B
{0x3C,0x40,0x40,0x40,0x40,0x40,0x3C,0x0},//C
{0x7C,0x42,0x42,0x42,0x42,0x42,0x7C,0x0},//D
{0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x7C},//E
{0x7C,0x40,0x40,0x7C,0x40,0x40,0x40,0x40},//F
{0x3C,0x40,0x40,0x40,0x40,0x44,0x44,0x3C},//G
{0x44,0x44,0x44,0x7C,0x44,0x44,0x44,0x44},//H
{0x7C,0x10,0x10,0x10,0x10,0x10,0x10,0x7C},//I
{0x3C,0x8,0x8,0x8,0x8,0x8,0x48,0x30},//J
{0x0,0x24,0x28,0x30,0x20,0x30,0x28,0x24},//K
{0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x7C},//L
{0x81,0xC3,0xA5,0x99,0x81,0x81,0x81,0x81},//M
{0x0,0x42,0x62,0x52,0x4A,0x46,0x42,0x0},//N
{0x3C,0x42,0x42,0x42,0x42,0x42,0x42,0x3C},//O
{0x3C,0x22,0x22,0x22,0x3C,0x20,0x20,0x20},//P
{0x1C,0x22,0x22,0x22,0x22,0x26,0x22,0x1D},//Q
{0x3C,0x22,0x22,0x22,0x3C,0x24,0x22,0x21},//R
{0x0,0x1E,0x20,0x20,0x3E,0x2,0x2,0x3C},//S
{0x0,0x3E,0x8,0x8,0x8,0x8,0x8,0x8},//T
{0x42,0x42,0x42,0x42,0x42,0x42,0x22,0x1C},//U
{0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18},//V
{0x0,0x49,0x49,0x49,0x49,0x2A,0x1C,0x0},//W
{0x0,0x41,0x22,0x14,0x8,0x14,0x22,0x41},//X
{0x41,0x22,0x14,0x8,0x8,0x8,0x8,0x8},//Y
{0x0,0x7F,0x2,0x4,0x8,0x10,0x20,0x7F},//Z
};

// 向MAX7219(U3)写入字节
void Write_Max7219_byte(uchar b)
{
  uchar i;
  digitalWrite(CS,LOW);
  for(i = 0; i < 8; ++i)
  {
    if(b & 0x80)
      digitalWrite(DIN, HIGH);
    else
      digitalWrite(DIN, LOW);
    b = b << 1;
    digitalWrite(CLK, HIGH);
    digitalWrite(CLK, LOW);
  }
}

// 向MAX7219指定位置写入数据
void Write_Max7219(uchar addr,uchar data)
{
  digitalWrite(CS, HIGH);
```

```
  digitalWrite(CS, LOW);
  digitalWrite(CS, LOW);
  Write_Max7219_byte(addr);           //写入地址
  Write_Max7219_byte(data);           //写入数据
  digitalWrite(CS, HIGH);

}


void Init_MAX7219(void)
{
  Write_Max7219(0x09, 0x00);       //设定译码方式：BCD码
  Write_Max7219(0x0a, 0x03);       //设定亮度
  Write_Max7219(0x0b, 0x07);       //扫描界限；8个数码管显示
  Write_Max7219(0x0c, 0x01);       //掉电模式：0，普通模式：1
}

int main()
{
    wiringPiSetup();  // wiringPi初始化
    pinMode(CLK, OUTPUT);
    pinMode(CS, OUTPUT);
    pinMode(DIN, OUTPUT);
    uchar i, j;
    delay(50);
    Init_MAX7219();
    while(1)
    {
      for(j=0; j<36; ++j)
      {
        for(i=1; i<9; ++i)
          Write_Max7219(i, disp[j][i-1]);
        delay(1000);
      }
    }
}

```

编写驱动程序

`makefile`

```
obj-m += char_driver.o
KERNEL := /usr/src/linux-headers-4.19.118-v7+/
all:
    make -C $(KERNEL) M=$(shell pwd) modules
clean:
    make -C $(KERNEL) M=$(shell pwd) clean
```

`char_drive.c`

```
#include <linux/delay.h>
```

```c
#include <linux/fs.h>
#include <linux/gpio.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/miscdevice.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/string.h>
#include <linux/uaccess.h>

#define DIN 2
#define CS 3
#define CLK 4
#define uchar unsigned char

static int device_num;
static int char_driver_open(struct inode *node, struct file *file);
static int char_driver_close(struct inode *node, struct file *file);
static ssize_t char_driver_write(struct file *file, const char __user *buf, size_t size, loff_t *offset);


// 文件操作结构体
static struct file_operations char_driver_fops =
{
    .owner   = THIS_MODULE,
    .open    = char_driver_open,
    .write   = char_driver_write,
    .close   = char_driver_close,
};

// 向MAX7219写一个byte
static void write_byte(uchar b)
{
    uchar i;
    for(i = 0; i < 8; ++i)
    {
        if((b & 0x80) > 0)
            gpio_set_value(DIN, 1);
        else
            gpio_set_value(DIN, 0);
        gpio_set_value(CLK, 1);
        gpio_set_value(CLK, 0);
        b <<= 1;
    }
}

// 向MAX7219指定位置写一个字节
static void write_word(uchar addr, uchar data)
{
    gpio_set_value(CS, 1);
    gpio_set_value(CLK, 0);
    gpio_set_value(CS, 0); // 允许写入
```

```
    write_byte(addr); // 写地址
    write_byte(data); // 写数据
    gpio_set_value(CS, 1); // 结束写入
}

// 按照对应的数据，点亮整个矩阵中的对应点
static void matrix_render(uchar *data)
{
    uchar i;
    for(i = 0; i < 8; ++i)
    {
        write_word(i + 1, data[i]);
    }
}

// 字符译码
uchar digits[][8]={
    {0x1c, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x1c}, // 0
    {0x08, 0x18, 0x08, 0x08, 0x08, 0x08, 0x08, 0x1c}, // 1
    {0x1c, 0x22, 0x22, 0x04, 0x08, 0x10, 0x20, 0x3e}, // 2
    {0x1c, 0x22, 0x02, 0x0c, 0x02, 0x02, 0x22, 0x1c}, // 3
    {0x04, 0x0c, 0x14, 0x14, 0x24, 0x1e, 0x04, 0x04}, // 4
    {0x3e, 0x20, 0x20, 0x3c, 0x02, 0x02, 0x22, 0x1c}, // 5
    {0x1c, 0x22, 0x20, 0x3c, 0x22, 0x22, 0x22, 0x1c}, // 6
    {0x3e, 0x24, 0x04, 0x08, 0x08, 0x08, 0x08, 0x08}, // 7
    {0x1c, 0x22, 0x22, 0x1c, 0x22, 0x22, 0x22, 0x1c}, // 8
    {0x1c, 0x22, 0x22, 0x22, 0x1e, 0x02, 0x22, 0x1c}, // 9
};

// 启动ui
uchar start_ui[] = {0xff, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0xff};

// open函数，由应用程序调用
static int char_driver_open(struct inode *node, struct file *file)
{
    // 申请gpio口 // TODO:可能有问题
    gpio_request(DIN, "sysfs");
    gpio_direction_output(DIN, 0);
    gpio_request(CS, "sysfs");
    gpio_direction_output(CS, 1);
    gpio_request(CLK, "sysfs");
    gpio_direction_output(CLK, 0);
    // 初始化设备
    write_word(0x09, 0x00);
    write_word(0x0a, 0x03);
    write_word(0x0b, 0x07);
    write_word(0x0c, 0x01);
    // 开启界面
    matrix_render(start_ui);
    // 输出提示
    printk(KERN_INFO "device opened successfully!\n");
    return 0;
}
```

```
107
108 // close函数，由应用程序调用
109 static int char_driver_close(struct inode *node, struct file *file)
110 {
111     printk(KERN_INFO "device opened successfully!\n");
112     return 0;
113 }
114
115 // write函数，由应用程序调用
116 static ssize_t char_driver_write(struct file *file, const char __user *buf, size_t
    size, loff_t *offset)
117 {
118     int i;
119     char ch;
120     for(i = 0; i < size; ++i)
121     {
122         copy_from_user(&ch, &buf[i], 1);
123         if(ch >= '0' && ch <= '9')
124         {
125             matrix_render(digits[ch - '0']);
126             mdelay(1000);
127         }
128     }
129     return size;
130 }
131
132 // 驱动的入口函数，安装时调用
133 static int __init char_driver_init(void)
134 {
135     // 注册字符设备
136     device_num = register_chrdev(0, "char_driver", &char_driver_fops);
137     if(device_num < 0)
138         return -1;
139     else
140         return 0;
141     printk(KERN_INFO "device initialized successfully!\n");
142 }
143
144 //驱动的出口函数，卸载时调用
145 static void __exit char_driver_exit(void)
146 {
147     unregister_chrdev(device_num, "char_driver");
148     printk(KERN_INFO "device exited successfully!\n");
149 }
150
151 // 内核哦通过这个宏来到这个驱动的入口和出口函数
152 module_init(char_driver_init);
153 module_exit(char_driver_exit);
154
155 MODULE_AUTHOR("Rookie");
156 MODULE_LICENSE("GPL");  // 指定协议
```

扩展内容

需要将字符串间隔一段时间显示，需要根据`size`的值判断字符串长度，逐一显示并调用`mdelay()`即可

```
// write函数，由应用程序调用
static ssize_t char_driver_write(struct file *file, const char __user *buf, size_t
size, loff_t *offset)
{
    int i;
    char ch;
    for(i = 0; i < size; ++i)
    {
        copy_from_user(&ch, &buf[i], 1);
        if(ch >= '0' && ch <= '9')
        {
            matrix_render(digits[ch - '0']);
            // 延时1s
            mdelay(1000);
        }
    }
    return size;
}
```

总结

最开始一直没能懂这个驱动和以前在STM32裸机上写驱动有什么区别。后来才搞明白这个驱动是基于linux内核态的编程，对于几个需要完成的函数都弄明白之后就没有很大的问题了。