/\*程序清单2-11\*/

1#include <linux/init.h>

2#include <linux/module.h>

3 #include <linux/fs.h>

4

5 #define DEVICE\_NAME "char\_null"

6 static int major = 232;/\* 保存主设备号的全局变量\*/

7

8 static int \_\_init char\_null\_init(void)

9 {

10 int ret;

11

12 ret = register\_chrdev(major, DEVICE\_NAME, **&major**);/\* 申请设备号和注册\*/

13 if (major > 0) { /\* 静态设备号\*/

14 if (ret < 0) {

15 printk(KERN\_INFO " Can't get major number!\n");

16 return ret;

17 }

18 } else { /\* 动态设备号\*/

19 printk(KERN\_INFO " ret is %d\n", ret);

20 major = ret; /\* 保存动态获取到的主设备号\*/

21 }

22 printk(KERN\_INFO "%s ok!\n", \_\_func\_\_);

23 return ret;

24 }

25

26 static void \_\_exit char\_null\_exit(void)

27 {

28 unregister\_chrdev(major, DEVICE\_NAME);

29 printk(KERN\_INFO "%s\n", \_\_func\_\_);

30 }

31

32 module\_init(char\_null\_init);

33 module\_exit(char\_null\_exit);

34

35 MODULE\_LICENSE("GPL");

36 MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

/\*程序清单2.14支持udev的空壳驱动\*/

#include <linux/init.h>

2 #include <linux/module.h>

3 #include <linux/fs.h>

4 #include <linux/cdev.h>

5 #include <linux/device.h>

6

7 static int major = 232; /\* 静态设备号方式的默认值\*/

8 static int minor = 0; /\* 静态设备号方式的默认值\*/

9 module\_param(major, int, S\_IRUGO);

10 module\_param(minor, int, S\_IRUGO);

11

12 struct cdev \*char\_null\_udev; /\* cdev数据结构\*/

13 static dev\_t devno; /\* 设备编号\*/

14 static struct class \*char\_null\_udev\_class;

15

16 #define DEVICE\_NAME "char\_null\_udev"

17

18 static int \_\_init char\_null\_udev\_init(void)

19 {

20 int ret;

21

22 if (major > 0) { /\* 静态设备号\*/

23 devno = MKDEV(major, minor);

24 ret = register\_chrdev\_region(devno, 1, "char\_null\_udev");

25 } else { /\* 动态设备号\*/

26 ret = alloc\_chrdev\_region(&devno, minor, 1, "char\_null\_udev"); /\* 从系统获取主设备号\*/

27 major = MAJOR(devno);

28 }

29 if (ret < 0) {

30 printk(KERN\_ERR "cannot get major %d \n", major);

31 return -1;

32 }

33

34 char\_null\_udev = cdev\_alloc(); /\* 分配char\_null\_udev结构\*/

35 if (char\_null\_udev != NULL) {

36 cdev\_init(char\_null\_udev, &major); /\* 初始化char\_null\_udev结构\*/

37 char\_null\_udev->owner = THIS\_MODULE;

38 if (cdev\_add(char\_null\_udev, devno, 1) != 0) { /\* 增加char\_null\_udev到系统中\*/

39 printk(KERN\_ERR "add cdev error!\n");

40 goto error;

41 }

42} else {

43printk(KERN\_ERR "cdev\_alloc error!\n");

44return -1;

45}

46

47//在/sys/class/下创建char\_null\_udev\_class目录

48char\_null\_udev\_class = class\_create(THIS\_MODULE, "char\_null\_udev\_class");

49if (IS\_ERR(char\_null\_udev\_class)) {

50printk(KERN\_INFO "create class error\n");

51return -1;

52}

53/\* 将创建/dev/char\_null\_udev0文件\*/

54//device\_create(char\_null\_udev\_class, NULL, devno, NULL, "char\_null\_udev" "%d", MINOR(devno));

55/\* 将创建/dev/char\_null\_udev文件\*/

56device\_create(char\_null\_udev\_class, NULL, devno, NULL, "char\_null\_udev");

57

58return 0;

59

60error:

61unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/

62return ret;

63}

64

65static void \_\_exit char\_null\_udev\_exit(void)

66{

67device\_destroy(char\_null\_udev\_class, devno);

68class\_destroy(char\_null\_udev\_class);

69cdev\_del(char\_null\_udev); /\* 移除字符设备\*/

70unregister\_chrdev\_region(devno, 1); /\* 释放设备号\*/

71}

72

73module\_init(char\_null\_udev\_init);

74module\_exit(char\_null\_udev\_exit);

75

76MODULE\_LICENSE("GPL");

77MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

/\*程序清单2-17\*/

1 #include <linux/init.h>

2 #include <linux/module.h>

3 #include <linux/fs.h>

4 #include <linux/cdev.h>

5 #include <linux/device.h>

6

7 static int major = 232; /\* 静态设备号方式的默认值\*/

8 static int minor = 0; /\* 静态设备号方式的默认值\*/

9 module\_param(major, int, S\_IRUGO);

10 module\_param(minor, int, S\_IRUGO);

11

12 struct cdev \*char\_cdev; /\* cdev数据结构\*/

13 static dev\_t devno; /\* 设备编号\*/

14 static struct class \*char\_cdev\_class;

15

16 #define DEVICE\_NAME "char\_cdev"

17

18 static int char\_cdev\_open(struct inode \*inode, struct file \*file )

19 {

20 try\_module\_get(THIS\_MODULE);

21 printk(KERN\_INFO DEVICE\_NAME " opened!\n");

22 return 0;

23 }

24

25 static int char\_cdev\_release(struct inode \*inode, struct file \*file )

26 {

27 printk(KERN\_INFO DEVICE\_NAME " closed!\n");

28 module\_put(THIS\_MODULE);

29 return 0;

30 }

31

32 static ssize\_t char\_cdev\_read(struct file \*file, char \*buf,size\_t count, loff\_t \*f\_pos)

33 {

34 printk(KERN\_INFO DEVICE\_NAME " read method!

35 return count;

36 }

37

38 static ssize\_t char\_cdev\_write(struct file \*file, const char \*buf, size\_t count, loff\_t \*f\_pos)

39 {

40 printk(KERN\_INFO DEVICE\_NAME " write method!\n");

41 return count;

42 }

43

44 static int char\_cdev\_ioctl(struct inode \*inode, struct file \*file, unsigned int cmd, unsigned long arg)

45 {

46 printk(KERN\_INFO DEVICE\_NAME " ioctl method!\n");

47 return 0;

48 }

49

50 struct file\_operations char\_cdev\_fops = {

51 .owner = THIS\_MODULE,

52 .read = char\_cdev\_read,

53 .write = char\_cdev\_write,

54 .open = char\_cdev\_open,

55 .release = char\_cdev\_release,

56 .ioctl = char\_cdev\_ioctl

57 };

58

59 static int \_\_init char\_cdev\_init(void)

60 {

61 int ret;

62

63 if (major > 0) { /\* 静态设备号\*/

64 devno = MKDEV(major, minor);

65 ret = register\_chrdev\_region(devno, 1, "char\_cdev");

66 } else { /\* 动态设备号\*/

67 ret = alloc\_chrdev\_region(&devno, minor, 1, "char\_cdev"); /\* 从系统获取主设备号\*/

68 major = MAJOR(devno);

69 }

70 if (ret < 0) {

71 printk(KERN\_ERR "cannot get major %d \n", major);

72 return -1;

73 }

74

75 char\_cdev = cdev\_alloc(); /\* 分配char\_cdev结构\*/

76 if (char\_cdev != NULL) {

77 cdev\_init(char\_cdev, &char\_cdev\_fops); /\* 初始化char\_cdev结构\*/

78 char\_cdev->owner = THIS\_MODULE;

79 if (cdev\_add(char\_cdev, devno, 1) != 0) { /\* 增加char\_cdev到系统中\*/

80 printk(KERN\_ERR "add cdev error!\n");

81 goto error;

82 }

83 } else {

84 printk(KERN\_ERR "cdev\_alloc error!\n");

85 return -1;

86 }

87

88 char\_cdev\_class = class\_create(THIS\_MODULE, "char\_cdev\_class");

89 if (IS\_ERR(char\_cdev\_class)) {

90 printk(KERN\_INFO "create class error\n");

91 return –

92 }

93

94 //device\_create(char\_cdev\_class, NULL, devno, NULL, "char\_cdev" "%d", MINOR(devno));

95 device\_create(char\_cdev\_class, NULL, devno, NULL, "char\_cdev", NULL);

96 return 0;

97

98 error:

99 unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/

100 return ret;

101 }

102

103 static void \_\_exit char\_cdev\_exit(void)

104 {

105 cdev\_del(char\_cdev); /\* 移除字符设备\*/

106 unregister\_chrdev\_region(devno, 1); /\* 释放设备号\*/

107 device\_destroy(char\_cdev\_class, devno);

108 class\_destroy(char\_cdev\_class);

109 }

110

111 module\_init(char\_cdev\_init);

112 module\_exit(char\_cdev\_exit);

113

114 MODULE\_LICENSE("GPL");

115 MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

程序清单2-18 测试范例程序

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <unistd.h>

4 #include <sys/ioctl.h>

5 #include <errno.h>

6 #include <fcntl.h>

7

8 #define DEV\_NAME "/dev/char\_cdev"

9

10 int main(int argc, char \*argv[])

11 {

12 int i;

13 int fd = 0;

14 int dat = 0;

15

16 fd = open (DEV\_NAME, O\_RDWR);

17 if (fd < 0) {

18 perror("Open "DEV\_NAME" Failed!\n");

19 exit(1);

20 }

21

22 i = read(fd, &dat, 1);

23 if (!i) {

24 perror("read "DEV\_NAME" Failed!);

25 exit(1);

26 }

27

28 dat = 0;

29 i = write(fd, &dat, 1);

30 if (!i) {

31 perror("write "DEV\_NAME" Failed!\n");

32 exit(1);

33 }

34

35 i = ioctl(fd, NULL, NULL);

36 if (!!i) {

37 perror("ioctl "DEV\_NAME" Failed!\n");

38 exit(1);

39 }

40

41 close(fd);

42 return 0;

43 }

/\*程序清单2.19LED驱动头文件\*/

1 #ifndef \_LED\_DRV\_H

2 #define \_LED\_DRV\_H

3

4 #define LED\_IOC\_MAGIC 'L'

5 #define LED\_ON \_IO(LED\_IOC\_MAGIC, 0)

/\*程序清单2.20LED驱动文件\*/

1 #include <linux/init.h>

2 #include <linux/module.h>

3 #include <linux/fs.h>

4 #include <linux/cdev.h>

5 #include <linux/device.h>

6 #include <linux/version.h>

7

8 #include <asm/mach/arch.h>

9 #include <mach/hardware.h>

10 #include <mach/gpio.h>

11 #include <asm/gpio.h>

12

13 #include "led\_drv.h"

14

15 static int major;

16 static int minor;

17 struct cdev \*led; /\* cdev数据结构\*/

18 static dev\_t devno; /\* 设备编号\*/

19 static struct class \*led\_class;

20

21 #define DEVICE\_NAME "led"

22

23 #define GPIO\_LED\_PIN\_NUM 55 /\* gpio 1\_23 \*/

24

25 static int led\_open(struct inode \*inode, struct file \*file )

26 {

27 try\_module\_get(THIS\_MODULE);

28 gpio\_direction\_output(GPIO\_LED\_PIN\_NUM, 1);

29 return 0;

30 }

31

32 static int led\_release(struct inode \*inode, struct file \*file )

33 {

34 module\_put(THIS\_MODULE);

35 gpio\_direction\_output(GPIO\_LED\_PIN\_NUM, 1);

36 return 0;

37 }

38

39 #if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,6,36)

40 int led\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)

41 #else

42 static int led\_ioctl(struct inode \*inode, struct file \*file, unsigned int cmd, unsigned long arg)

43 #endif

44 {

45 if (\_IOC\_TYPE(cmd) != LED\_IOC\_MAGIC) {

46 return -ENOTTY;

47 }

48

49 if (\_IOC\_NR(cmd) > LED\_IOCTL\_MAXNR) {

50 return -ENOTTY;

51 }

52

53 switch(cmd) {

54 case LED\_ON:

55 gpio\_set\_value(GPIO\_LED\_PIN\_NUM, 0);

56 break;

57

58 case LED\_OFF:

59 gpio\_set\_value(GPIO\_LED\_PIN\_NUM, 1);

60 break;

61

62 default:

63 gpio\_set\_value(27, 0);

64 break;

65 }

66

67 return 0;

68 }

69

70 struct file\_operations led\_fops = {

71 .owner = THIS\_MODULE,

72 .open = led\_open,

73 .release = led\_release,

74 #if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,6,36)

75 .unlocked\_ioctl = led\_ioctl

76 #else

77 .ioctl = led\_ioctl

78 #endif

79 };

80

81 static int \_\_init led\_init(void)

82 {

83 int ret;

84

85 gpio\_free(GPIO\_LED\_PIN\_NUM);

86 if (gpio\_request(GPIO\_LED\_PIN\_NUM, "led\_run")) {

87 printk("request %s gpio faile \n", "led\_run");

88 return -1;

89 }

90

91 ret = alloc\_chrdev\_region(&devno, minor, 1, "led"); /\*从系统获取主设备号\*/

92 major = MAJOR(devno);

93 if (ret < 0) {

94 printk(KERN\_ERR "cannot get major %d \n", major);

95 return -1;

96 }

97

98 led = cdev\_alloc(); /\* 分配led结构\*/

99if (led != NULL) {

100 cdev\_init(led, &led\_fops); /\* 初始化led结构\*/

101 led->owner = THIS\_MODULE;

102 if (cdev\_add(led, devno, 1) != 0) { /\* 增加led到系统中\*/

103 printk(KERN\_ERR "add cdev error!\n");

104 goto error;

105 }

106 } else {

107 printk(KERN\_ERR "cdev\_alloc error!\n");

108 return -1;

109 }

110

111 led\_class = class\_create(THIS\_MODULE, "led\_class");

112 if (IS\_ERR(led\_class)) {

113 printk(KERN\_INFO "create class error\n");

114 return -1;

115 }

116

117 device\_create(led\_class, NULL, devno, NULL, "led");

118 return 0;

119

120 error:

121 unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/

122 return ret;

123 }

124

125 static void \_\_exit led\_exit(void)

126 {

127 cdev\_del(led); /\* 移除字符设备\*/

128 unregister\_chrdev\_region(devno, 1); /\*释放设备号\*/

129 device\_destroy(led\_class, devno);

130 class\_destroy(led\_class);

131 }

132

133 module\_init(led\_init);

134 module\_exit(led\_exit);

135

136 MODULE\_LICENSE("GPL");

137 MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

/\*Led测试程序\*/

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <unistd.h>

4 #include <sys/ioctl.h>

5 #include <errno.h>

6 #include <fcntl.h>

7 #include "../led\_drv.h"

8

9 #define DEV\_NAME "/dev/led"10

11 int main(int argc, char \*argv[])

12 {

13 int i;

14 int fd = 0;

15

16 fd = open (DEV\_NAME, O\_RDONLY);

17 if (fd < 0) {

18 perror("Open "DEV\_NAME" Failed!\n");

19 exit(1);

20 }

21

22 for (i=0; i<3; i++) {

23 ioctl(fd, LED\_ON);

24 sleep(1);

25 ioctl(fd, LED\_OFF);

26 sleep(1);

27 }

28

29 close(fd);

30 return 0;

31 }

/\*put\_user()例子\*/

static int char\_cdev\_ioctl (struct inode \*inode, struct file \*filp, unsigned int cmd, unsigned long arg)

{

int ret;

u32 dat,switch(cmd){

case CHAR\_CDEV\_READ:...其它操作dat = 数据；

if (put\_user(dat,(u32 \*)arg)) {

printk("put\_user err\n");

return -EFAULT;

}

}

...其它操作

return ret;

}

/\*程序清单2.22设备读写方法实现范例程序\*/

1 #include <linux/init.h>

2 #include <linux/module.h>

3 #include <linux/fs.h>

4 #include <linux/cdev.h>

5 #include <linux/device.h>

6#include <asm/uaccess.h> /\* copy\_to\_user ... \*/

7

8static int major;

9static int minor;

10struct cdev \*char\_cdev\_rw; /\* cdev数据结构\*/

11static dev\_t devno; /\* 设备编号\*/

12static struct class \*char\_cdev\_rw\_class;

13static char dev\_rw\_buff[64]; /\* 设备内部读写缓冲区\*/

14

15#define DEVICE\_NAME "char\_cdev\_rw"

16

17static int char\_cdev\_rw\_open(struct inode \*inode, struct file \*file )

18{

19try\_module\_get(THIS\_MODULE);

20return 0;

21}

21

23static int char\_cdev\_rw\_release(struct inode \*inode, struct file \*file )

24{

25module\_put(THIS\_MODULE);

26return 0;

27}

28

29 static ssize\_t char\_cdev\_rw\_read(struct file \*file, char \*buf,size\_t count, loff\_t \*f\_pos)

30{

31if (count > 64) {

32printk("Max length is 64\n");

33count = 64;

34}

35

36if (copy\_to\_user((void \*)buf, dev\_rw\_buff, count)) {

37printk("copy\_to\_user err\n");

38return -EFAULT;

39 }

40

41return count;

42}

43

44static ssize\_t char\_cdev\_rw\_write(struct file \*file, const char \*buf, size\_t count, loff\_t \*f\_pos)

45{

46if (count > 64) {

47printk("Max length is 64\n");

48count = 64;

49}

50

51if (copy\_from\_user(&dev\_rw\_buff, buf, count) ) {

52printk("copy\_from\_user err\n");

53return -EFAULT;

54}

55

56return count;

57}

58

59struct file\_operations char\_cdev\_rw\_fops = {

60.owner = THIS\_MODULE,

61.read = char\_cdev\_rw\_read,

62.write = char\_cdev\_rw\_write,

63.open = char\_cdev\_rw\_open,

64.release = char\_cdev\_rw\_release,

65};

66

67static int \_\_init char\_cdev\_rw\_init(void)

68{

69 int ret;

70int i;

71

72 ret = alloc\_chrdev\_region(&devno, minor, 1, "char\_cdev\_rw");/\* 从系统获取主设备号\*/

73major = MAJOR(devno);

74if (ret < 0) {

75printk(KERN\_ERR "cannot get major %d \n", major);

76return -1;

77}

78

79 char\_cdev\_rw = cdev\_alloc(); /\* 分配char\_cdev\_rw结构\*/

80if (char\_cdev\_rw != NULL) {

81 cdev\_init(char\_cdev\_rw, &char\_cdev\_rw\_fops); /\* 初始化char\_cdev\_rw结构\*/

82char\_cdev\_rw->owner = THIS\_MODULE;

83 if(cdev\_add(char\_cdev\_rw, devno, 1) != 0) {/\* 增加char\_cdev\_rw到系统中\*/

84printk(KERN\_ERR "add cdev error!\n");

85goto error;

86}

87} else {

88printk(KERN\_ERR "cdev\_alloc error!\n");

89return -1;

90}

91

92char\_cdev\_rw\_class = class\_create(THIS\_MODULE, "char\_cdev\_rw\_class");

93if (IS\_ERR(char\_cdev\_rw\_class)) {

94printk(KERN\_INFO "create class error\n");

95return -1;

96}

97

98device\_create(char\_cdev\_rw\_class, NULL, devno, NULL, "char\_cdev\_rw");

99

100for (i=0; i<64; i++) {

101 dev\_rw\_buff[i] = i; /\* 初始化设备内部读写缓冲区\*/

102}

103

104return 0;

105

106error:

107 unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/

108return ret;

109}

110

111static void \_\_exit char\_cdev\_rw\_exit(void)

112{

113 cdev\_del(char\_cdev\_rw); /\* 移除字符设备\*/

114 unregister\_chrdev\_region(devno, 1); /\* 释放设备号\*/

115device\_destroy(char\_cdev\_rw\_class, devno);

116class\_destroy(char\_cdev\_rw\_class);

117}

118

119module\_init(char\_cdev\_rw\_init);

120module\_exit(char\_cdev\_rw\_exit);

121

122MODULE\_LICENSE("GPL");

123MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

/\*程序清单2.23设备读写应用范例\*/

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <unistd.h>

4 #include <sys/ioctl.h>

5 #include <errno.h>

6 #include <fcntl.h>

7

8 #define DEV\_NAME "/dev/char\_cdev\_rw"

9

10 int main(int argc, char \*argv[])

11 {

12 int i;

13 int fd = 0;

14 char buff[64];

15

16 fd = open (DEV\_NAME, O\_RDWR);

17 if (fd < 0) {

18 perror("Open "DEV\_NAME" Failed!\n");

19 exit(1);

20 }

21

22 printf("read orig data from device\n");

23 i = read(fd, &buff, 64);

24 if (!i) {

25 perror("read "DEV\_NAME" Failed!\n");

26 exit(1);

27 }

28 for (i=0; i<64; i++ ) {

29 printf("0x%02x ", buff[i]);

30 }

31 printf("\n");

32

33 printf("write data into device\n");

34 for (i=0; i<64; i++ ) {

35 buff[i] = 63-i;

36 }

37 i = write(fd, &buff, 64);

38 if (!i) {

39 perror("write "DEV\_NAME" Failed!\n");

40 exit(1);

41 }

42

43 printf("read new data from device\n");

44 i = read(fd, &buff, 64);

45 if (!i) {

46 perror("read "DEV\_NAME" Failed!\n");

47 exit(1);

48 }

49 for (i=0; i<64; i++ ) {

50 printf("0x%02x ", buff[i]);

51 }

52 printf("\n");

53

54 close(fd);

55 return 0;

56 }

/\*程序清单2.25按键驱动范例\*/

1 #include <linux/init.h>

2 #include <linux/module.h>

3 #include <linux/fs.h>4 #include <linux/cdev.h>5 #include <linux/device.h>6 #include <linux/interrupt.h>7 #include <linux/irq.h>8 #include <linux/gpio.h>9 10 #define KEY\_GPIO 70 /\* GPIO2\_6 \*/11 #define KEY\_GPIO\_IRQ gpio\_to\_irq(KEY\_GPIO) /\* 中断号\*/12 #define DEVICE\_NAME "key\_irq"13 14 static int major;15 static int minor;16 struct cdev \*key\_irq; /\* cdev数据结构\*/17 static dev\_t devno; /\* 设备编号\*/18 static struct class \*key\_irq\_class;19 20 char const irq\_types[5] = {21 IRQ\_TYPE\_EDGE\_RISING,22 IRQ\_TYPE\_EDGE\_FALLING,23 IRQ\_TYPE\_EDGE\_BOTH,24 IRQ\_TYPE\_LEVEL\_HIGH,25 IRQ\_TYPE\_LEVEL\_LOW26 };27 28 static int key\_irq\_open(struct inode \*inode, struct file \*file )29 {30 try\_module\_get(THIS\_MODULE);31 printk(KERN\_INFO DEVICE\_NAME " opened!\n"); 32 return 0;33 }34 35 static int key\_irq\_release(struct inode \*inode, struct file \*file )36 {37 printk(KERN\_INFO DEVICE\_NAME " closed!\n");38 module\_put(THIS\_MODULE);39 return 0;40 }41 42 static irqreturn\_t key\_irq\_irq\_handler (unsigned int irq, void \*dev\_id)43 {44 printk("KEY IRQ HAPPENED!\n");45 return IRQ\_HANDLED;46 }

47 48 struct file\_operations key\_irq\_fops = {49 .owner = THIS\_MODULE,50 .open = key\_irq\_open,51 .release = key\_irq\_release,52 };53 54 static int \_\_init key\_irq\_init(void)55 {56 int ret;57 58 gpio\_free(KEY\_GPIO);59 ret = gpio\_request\_one(KEY\_GPIO, GPIOF\_IN, "KEY IRQ"); /\* 申请IO \*/60 if (ret < 0) {61 printk(KERN\_ERR "Failed to request GPIO for KEY\n");62 }63 64 gpio\_direction\_input(KEY\_GPIO); /\* 设置GPIO为输入\*/65 if (request\_irq(KEY\_GPIO\_IRQ, key\_irq\_irq\_handler, IRQF\_DISABLED, "key\_irq irq", NULL) ){ /\* 申请中断\*/66 printk(KERN\_WARNING DEVICE\_NAME": Can't get IRQ: %d!\n", KEY\_GPIO\_IRQ);67 }68 set\_irq\_type(KEY\_GPIO\_IRQ, irq\_types[1]);69 disable\_irq(KEY\_GPIO\_IRQ);70 enable\_irq(KEY\_GPIO\_IRQ);71 72 ret = alloc\_chrdev\_region(&devno, minor, 1, DEVICE\_NAME); /\* 从系统获取主设备号\*/73 major = MAJOR(devno);74 if (ret < 0) {75 printk(KERN\_ERR "cannot get major %d \n", major);76 return-1;77 }78 79 key\_irq = cdev\_alloc(); /\* 分配key\_irq结构\*/80 if (key\_irq != NULL) {81 cdev\_init(key\_irq, &key\_irq\_fops); /\* 初始化key\_irq结构\*/82 key\_irq->owner = THIS\_MODULE;83 if (cdev\_add(key\_irq, devno, 1) != 0) { /\* 增加key\_irq到系统中\*/84 printk(KERN\_ERR "add cdev error!\n");85 goto error;86 }87 } else {88 printk(KERN\_ERR "cdev\_alloc error!\n");

89 return ；

90 }91 92 key\_irq\_class = class\_create(THIS\_MODULE, "key\_irq\_class");93 if (IS\_ERR(key\_irq\_class)) {94 printk(KERN\_INFO "create class error\n");95 return -1;96 }97 98 device\_create(key\_irq\_class, NULL, devno, NULL, DEVICE\_NAME);99 return 0;100 101 error:102 unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/103 return ret;104 }105 106 static void \_\_exit key\_irq\_exit(void)107 {108 gpio\_free(KEY\_GPIO);109 disable\_irq(KEY\_GPIO\_IRQ);110 free\_irq(KEY\_GPIO\_IRQ, NULL);111 cdev\_del(key\_irq); /\* 移除字符设备\*/112 unregister\_chrdev\_region(devno, 1); /\* 释放设备号\*/113 device\_destroy(key\_irq\_class, devno);114 class\_destroy(key\_irq\_class);115 }116 117 module\_init(key\_irq\_init);118 module\_exit(key\_irq\_exit);119 120 MODULE\_LICENSE("GPL");121 MODULE\_AUTHOR("Chenxibing, linux@zlgmcu.com");

1 #include <linux/init.h>2 #include <linux/module.h>3 #include <linux/device.h>4 #include <linux/platform\_device.h>56 #define GPIO\_LED\_PIN\_NUM 55 /\* gpio 1\_23 \*/78 /\* 定义LED资源\*/9 static struct resource led\_resources[] = {10 [0] = {11 .start = GPIO\_LED\_PIN\_NUM,12 .end = GPIO\_LED\_PIN\_NUM,13 .flags = IORESOURCE\_IO,14 },15 };1617 static void led\_platform\_release(struct device \*dev)18 {19 return;20 }2122 /\* 定义平台设备\*/23 static struct platform\_device led\_platform\_device = {24 .name = "led", /\* platform\_driver中，.name必须与该名字相同\*/

25 .id = -1,26 .num\_resources = ARRAY\_SIZE(led\_resources),27 .resource = led\_resources,28 .dev = {29 /\* Device 'led' does not have a release() function, it is broken and must be fixed. \*/30 .release = led\_platform\_release,31 .platform\_data = NULL,32 },33 };3435 static int \_\_init led\_platform\_init(void)36 {37 int ret;3839 ret = platform\_device\_register(&led\_platform\_device);40 if (ret < 0) {41 platform\_device\_put(&led\_platform\_device);42 return ret;43 }4445 return 0;46 }4748 static void \_\_exit led\_platform\_exit(void)49 {50 platform\_device\_unregister(&led\_platform\_device);51 }5253 module\_init(led\_platform\_init);54 module\_exit(led\_platform\_exit);5556 MODULE\_LICENSE("GPL");57 MODULE\_AUTHOR("Chenxibing, [linux@zlgmcu.com](mailto:linux@zlgmcu.com)");

led\_platform.c

/\*程序清单2.41led\_drv.c参考代码\*/

1 #include <linux/init.h>2 #include <linux/module.h>3 #include <linux/version.h>4 #include <linux/fs.h>5 #include <linux/cdev.h>6 #include <linux/device.h>7 #include <linux/platform\_device.h>8 #include <asm/gpio.h>9 10 #include "led\_drv.h"11 12 static int major;13 static int minor;14 struct cdev \*led; /\* cdev数据结构\*/15 static dev\_t devno; /\* 设备编号\*/16 static struct class \*led\_class;17 static int led\_io;/\* 用于保存GPIO编号\*/18 19 #define DEVICE\_NAME "led"20 21 static int led\_open(struct inode \*inode, struct file \*file )22 {23 try\_module\_get(THIS\_MODULE);24 gpio\_direction\_output(led\_io, 1);25 return 0;26 }27 28static int led\_release(struct inode \*inode, struct file \*file )29 {30 module\_put(THIS\_MODULE);31 gpio\_direction\_output(led\_io, 1);32 return 0;33 }34 35 #if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,6,36)36 int led\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)37 #else

38 static int led\_ioctl(struct inode \*inode, struct file \*file, unsigned int cmd, unsigned long arg)39 #endif40 {41 if (\_IOC\_TYPE(cmd) != LED\_IOC\_MAGIC) {42 return -ENOTTY;43 }44 45 if (\_IOC\_NR(cmd) > LED\_IOCTL\_MAXNR) {46 return -ENOTTY;47 }48 49 switch(cmd) {50 case LED\_ON:51 gpio\_set\_value(led\_io, 0);52 break;53 54 case LED\_OFF:55 gpio\_set\_value(led\_io, 1);56 break;57 58 default:59 gpio\_set\_value(led\_io, 0);60 break;61 }62 63 return 0;64 }65 66 struct file\_operations led\_fops = {67 .owner = THIS\_MODULE,68 .open = led\_open,69 .release = led\_release,70 #if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,6,36)71 .unlocked\_ioctl = led\_ioctl72 #else73 .ioctl = led\_ioctl74 #endif75 };76 77 static int \_\_devinitled\_probe(struct platform\_device \*pdev)78 {79 int ret;80 struct resource \*res\_io;81

82 res\_io = platform\_get\_resource(pdev, IORESOURCE\_IO, 0);/\*从设备资源获取IO引脚\*/83 led\_io = res\_io->start;84 85 ret = alloc\_chrdev\_region(&devno, minor, 1, DEVICE\_NAME); /\* 从系统获取主设备号\*/86 major = MAJOR(devno);87 if (ret < 0) {88 printk(KERN\_ERR "cannot get major %d \n", major);89 return -1;90 }91 92 led = cdev\_alloc(); /\* 分配led结构\*/93if (led != NULL) {94 cdev\_init(led, &led\_fops); /\* 初始化led结构\*/95 led->owner = THIS\_MODULE;96 if (cdev\_add(led, devno, 1) != 0) { /\* 增加led到系统中\*/97 printk(KERN\_ERR "add cdev error!\n");98 goto error;99 }100 } else {101 printk(KERN\_ERR "cdev\_alloc error!\n");102 return -1;103 }104 105 led\_class = class\_create(THIS\_MODULE, DEVICE\_NAME"\_class");106 if (IS\_ERR(led\_class)) {107 printk(KERN\_INFO "create class error\n");108 return -1;109 }110 111 device\_create(led\_class, NULL, devno, NULL, DEVICE\_NAME);112 return 0;113 114 error:115 unregister\_chrdev\_region(devno, 1); /\* 释放已经获得的设备号\*/116 return ret;117 }118 119 static int \_\_devexit led\_remove(struct platform\_device \*dev)120 {121 cdev\_del(led); /\* 移除字符设备\*/122 unregister\_chrdev\_region(devno, 1); /\* 释放设备号\*/123 device\_destroy(led\_class, devno);124 class\_destroy(led\_class);125 return 0;

126 }127 128 /\* 定义和初始化平台驱动\*/129 static struct platform\_driver led\_platform\_driver = {130 .probe = led\_probe,131 .remove = \_\_devexit\_p(led\_remove),132 .driver = {133 .name = "led",/\* 该名称必须与platform\_device的.name相同\*/134 .owner = THIS\_MODULE,135 },136 };137 138static int \_\_init led\_init(void)139{140return(platform\_driver\_register(&led\_platform\_driver));141}142143static void \_\_exit led\_exit(void)144{145 platform\_driver\_unregister(&led\_platform\_driver);146}147148module\_init(led\_init);149 module\_exit(led\_exit);150151MODULE\_LICENSE("GPL");152MODULE\_AUTHOR("Chenxibing, linux@zlgmcu.com");