微算機系統實習

LAB 06

組別：19

109590014　　沈煒翔

109590015　　楊挺煜

109590023　　廖堃霖

日期：111/05/28

2. 實驗

(1)makefile(驅動程式)

```
obj-m := demo.o
kernel_DIR = /usr/src/linux-headers-4.9.201-tegra-ubuntu18.04_aarch64/kernel-4.9/ #有可能需要更改位址

PWD := $(shell pwd)
all:
    make -C $(kernel_DIR) SUBDIRS=$(PWD)
clean:
    rm *.o *.ko *mod.c
.PHONY:
    clean

```

(2)makefile(LED)

```
CROSS_COMPILE = aarch64-linux-gnu-
all: Lab6-2.cpp
    g++ -o Lab6-2 Lab6-2.cpp
clean:
    rm Lab6-2

```

(3)LED_gpio 控制

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <iostream>
#include <map>

#define LED1 396
#define LED2 397
#define LED3 429
#define LED4 393

using namespace std;

void setGPIO(unsigned int gpio, string status) {
    int io = open("/dev/demo", O_WRONLY);
    if(io < 0){
        printf("can't open device\n");
        return;
    }
    char buf[10];
    if(status == "on"){
        strcpy(buf, (to_string(gpio) + "1").c_str());
    }
    else {
        strcpy(buf, (to_string(gpio) + "0").c_str());
    }
    cout << "input: " << buf << endl;
    write(io, buf, 5);
    close(io);
    return;
}
```

```cpp
void readGPIO(unsigned int gpio) {
    int io = open("/dev/demo", O_WRONLY);
    if(io < 0){
        printf("can't open device\n");
        return;
    }
    char buf[64];
    strcpy(buf, (to_string(gpio) + "2").c_str());
    write(io, buf, 5);
    read(io, buf, sizeof(buf));
    cout << buf << endl;
    close(io);
    return;
}
```

```c
int main(int argc, char** argv){
    if(argc == 2) {
        switch(argv[1][3]) {
        case '1':
            readGPIO(LED1);
            break;
        case '2':
            readGPIO(LED2);
            break;
        case '3':
            readGPIO(LED3);
            break;
        case '4':
            readGPIO(LED4);
            break;
        }

    }
    else if(argc == 3) {
        switch(argv[1][3]) {
        case '1':
            setGPIO(LED1, argv[2]);
            break;
        case '2':
            setGPIO(LED2, argv[2]);
            break;
        case '3':
            setGPIO(LED3, argv[2]);
            break;
        case '4':
            setGPIO(LED4, argv[2]);
            break;
        }
    }
```

(4)demo.c

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

#define LED1 396
#define LED2 397
#define LED3 429
#define LED4 393

#define MAJOR_NUM 60
#define MODULE_NAME "demo"

static char userChar[100];

int gpioPin[4] = {LED1, LED2, LED3, LED4};
int ledStatus[4] = {0, 0, 0, 0};
static int gpio;
static int status;

static ssize_t drv_read(struct file *filp,char *buf, size_t count, loff_t *ppos){
    printk("Enter Read function\n");
    printk("device read\n");
    return 0;
}

```

```c
static ssize_t drv_write(struct file *filp, const char *buf, size_t count, loff_t *ppos)
{
    printk("Enter Write function\n");
    printk("device write\n");
    printk("W_buf_size: %d\n", (int)count);
    if(copy_from_user(userChar, buf, count) == 0) {
        userChar[count - 1] = '\0';
        printk("userChar: %s\n", userChar);

        //Turn userChar to GPIO input
        gpio = (userChar[0] - '0') * 100 + (userChar[1] - '0') * 10 + (userChar[2] - '0');
        status = (userChar[3] - '0');
        printk("gpio: %d", gpio);

        //excute
        if(status == 2) {
            char r[100] = {'\0'};
            switch(gpio) {
            case LED1:
                snprintf(r, sizeof(r), "LED1(GPIO=%d) Status: %d", gpio, ledStatus[0]);
                copy_to_user(buf, r, strlen(r) + 1);
                printk("%s", r);
                break;
            case LED2:
                snprintf(r, sizeof(r), "LED2(GPIO=%d) Status: %d", gpio, ledStatus[1]);
                                copy_to_user(buf, r, strlen(r) + 1);
                                printk("%s", r);
                break;
            case LED3:
                snprintf(r, sizeof(r), "LED3(GPIO=%d) Status: %d", gpio, ledStatus[2]);
                                copy_to_user(buf, r, strlen(r) + 1);
                                printk("%s", r);
                break;
            case LED4:
                snprintf(r, sizeof(r), "LED4(GPIO=%d) Status: %d", gpio, ledStatus[3]);
                                copy_to_user(buf, r, strlen(r) + 1);
                                printk("%s", r);
                break;
            }
        }
        else {
            switch(gpio) {
            case LED1:
                ledStatus[0] = status;
                gpio_set_value(LED1, status);
                break;
            case LED2:
                ledStatus[1] = status;
                gpio_set_value(LED2, status);
                break;
            case LED3:
                ledStatus[2] = status;
                gpio_set_value(LED3, status);
                break;
            case LED4:
                ledStatus[3] = status;
                gpio_set_value(LED4, status);
                break;
            }
        }
    }
    return count;
}
```

```c
long drv_ioctl(struct file *filp, unsigned int cmd,unsigned long arg)
{
    printk("Enter I/O Control function\n");
    printk("device ioctl\n");
    return 0;
}

static int drv_open(struct inode *inode, struct file *filp)
{
    printk("Enter Open function\n");
    printk("device open\n");
    return 0;
}

static int drv_release(struct inode *inode, struct file *filp)
{
    printk("Enter Release function\n");
    printk("device close\n");
    return 0;
}

struct file_operations drv_fops =
{
        read: drv_read,
        write: drv_write,
        unlocked_ioctl: drv_ioctl,
        open: drv_open,
        release: drv_release,
};

static int demo_init(void)
{
    char* ledName[4] = {"LED1", "LED2", "LED3", "LED4"};
    //Initialize GPIO pins
    int i = 0;
    for(i = 0; i < 4; i++) {
        if(gpio_is_valid(gpioPin[i]) == 0) {
            printk("gpio%d is invaild", gpioPin[i]);
            return (-EBUSY);
        }
        if(gpio_request(gpioPin[i], ledName[i]) < 0) {
            printk("gpio%d request failed", gpioPin[i]);
            return (-EBUSY);
        }

        //Turn off gpio
        gpio_direction_output(gpioPin[i], 0);
        gpio_export(gpioPin[i], false);
    }
    //regist driver
    if(register_chrdev(MAJOR_NUM, "demo", &drv_fops)<0)
    {
        printk("<1>%s: can't get major %d\n", MODULE_NAME, MAJOR_NUM);
        return (-EBUSY);
    }
    printk("<1>%s: started\n", MODULE_NAME);
    return 0;
}
```

```c
static void demo_exit(void)
{
    //Free GPIO pins
    int i = 0;
    for(i = 0; i < 4; i++) {
        gpio_free(gpioPin[i]);
    }
    unregister_chrdev(MAJOR_NUM,"demo");
    printk("<1>%s: removed\n", MODULE_NAME);
}

module_init(demo_init);
module_exit(demo_exit);
MODULE_LICENSE("Dual BSD/GPL");
```

3. 實驗影片

https://youtu.be/80bhZV3nE2o

4. 組員貢獻

沈煒翔：34%

楊挺煜：33%

廖堃霖：33%

5. 心得

沈煒翔：

這次 lab6-2 相較於前幾次的實驗我覺得比較難，雖然是 6-1 的延續做驅動程式，但要結合前面的 LED gpio 的部分，在做實作的時後，想說先把 LED 先處理好，但這部分還是花了蠻長的時間，再來是驅動的部分就不是我去處理的，驅動程式聽隊友說一直遇到問題，所以我也跟他一起找 bug，最後用了差不多 3，4 個小時才處理完，做完整個人都神輕氣爽了。

楊挺煜：

這次的實驗是透過驅動程式控制 GPIO，最一開始我是打算用在 kernel 裡進行檔案處理去修改/sys/class/gpio 檔案，不過系統一直報"accessing userspace outside access.h"，跟同學請教後我是透過 linux/gpio.h 的函式去控制 GPIO。這次的實習讓我對 kernel space 跟 user space 有更深的理解。

廖堃霖：

這次的實驗跟上次差不多但有多加個控制 led 燈還有看 led 燈的狀態。