微算機系統實習

Final Project

個人報告

組別：19

109590014　　沈煒翔

109590015　　楊挺煜

109590023　　廖堃霖

日期：111/06/24

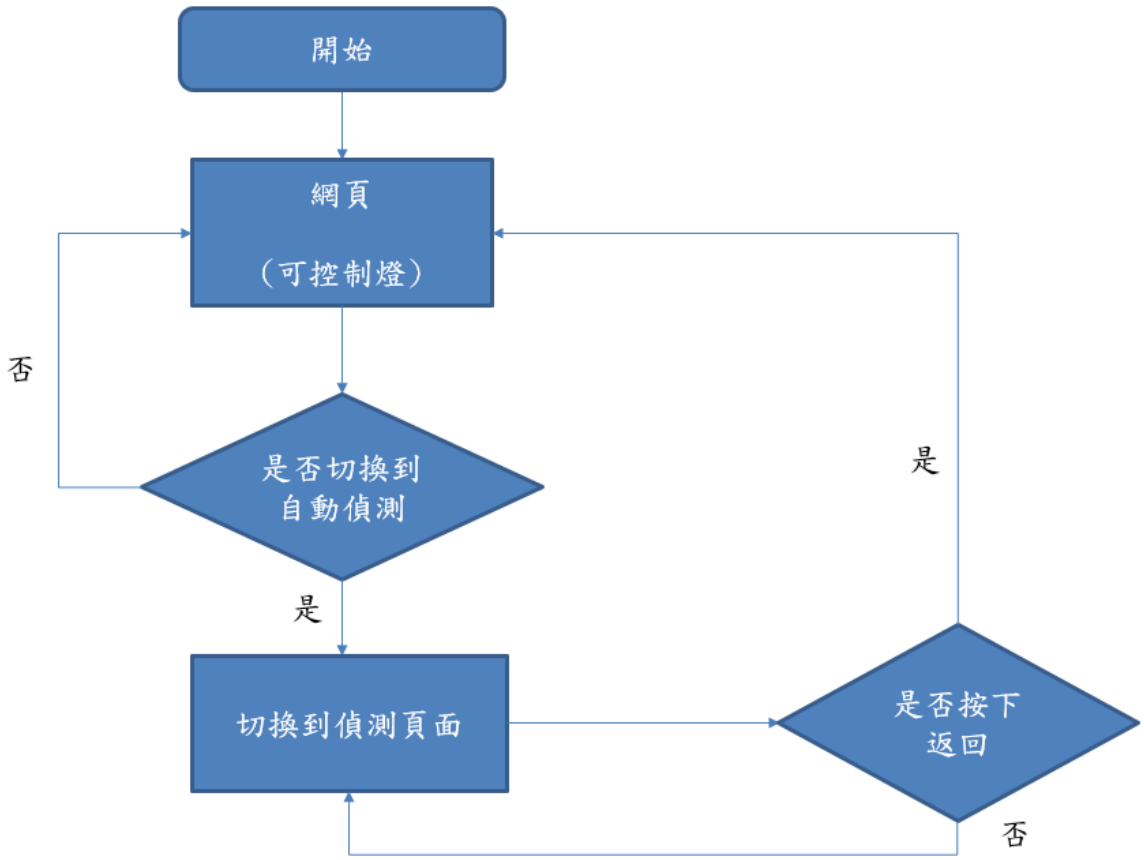
1. 使用情景

(1) 遠端控制電燈

(2) 自動感光進行電燈控制

(3) 遠端警示燈(可控制閃爍次數、頻率)

2. 專案流程圖



3. 專案構思

(1) LED 燈控制：透過字元驅動控制 LED 燈。

(2) 頻率閃爍 LED 燈：透過 Mutex 與 Semaphore 控制每次的 LED 開關。

(3) 光敏電阻值偵測：使用 Python 的 GPIO Module 讀取光敏電阻的類比訊號。
(4) 網頁：使用 NodeJS 與 NPM 架設。
(5) 寄送 Email：使用 NPM 的 NodeMailer 模組與同學的伺服器發送。

4. 實驗步驟

```
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;
            }
        }
        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;
            }
        }
    }
    return count;
}
```

(1) 字元驅動 write function，基本上與 Lab6 相同。

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

#define LED1 466
#define LED2 393

pthread_mutex_t mutex;
sem_t sem;

using namespace std;

// input format: ./final_excute Mode Arguments
// --------------------------------
// Control:
// Mode1: LED Control
// ./final_excute [LED] [ON or OFF]
// --------------------------------
// Control:
// Mode2: LED Frequently Shine
// ./final_excute <頻率> <次數> <mutex or semaphore>
// --------------------------------
```

(2) 執行檔 cpp：Include header、Defines、全域變數、輸入格式註解。

```cpp
int main(int argc, char** argv) {
    // LED Control Mode
    if(argv[2][0] == 'O') { // 'O'N & 'O'FF
        int status = 0;
        if(argv[2][1] == 'N') {
            status = 1;
        }
        else {
            status = 0;
        }

        switch(argv[1][3]) {
            case '1':
                setGPIO(LED1, status);
                break;
            case '2':
                setGPIO(LED2, status);
                break;
        }
        return 0;
    }
    else {
        // LED Frequently Shine Mode
        if(argv[3][0] == 'm') { // 'm'utex
            LED_FS_mutex(argv[1], argv[2]);
        }
        else {
            LED_FS_semaphore(argv[1], argv[2]);
        }
    }
}
```

(3) 執行檔 cpp：main function

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

(4) 執行檔 cpp：控制 GPIO 的 function

```
//mutex子執行緒
void* mutex_child(void* data) {
    cout << "Enter mutex child function" << endl;
    pthread_mutex_lock(&mutex);
    int* input = (int*) data;
    setGPIO(input[0], input[1]);
    setGPIO(input[2], input[3]);
    pthread_mutex_unlock(&mutex);
    pthread_exit(NULL);
}

void LED_FS_mutex(char* freq, char* times) {
    cout << "Enter mutex function" << endl;
    pthread_t t1, t2;
    pthread_mutex_init(&mutex, 0);

    int f = atoi(freq) * 1000;
    int t = atoi(times);

    int data_0[4] = {LED1, 1, LED2, 0};
    int data_1[4] = {LED1, 0, LED2, 1};

    for(int i = 0; i < t; i++) {
        pthread_create(&t1, NULL, mutex_child, data_0);
        usleep(f);
        pthread_create(&t2, NULL, mutex_child, data_1);
        usleep(f);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        setGPIO(LED1, 0);
        setGPIO(LED2, 0);
    }
    return;
}
```

(5) Mutex 模式的 function

```
//semaphore子執行緒
void* sem_child(void* data) {
    cout << "Enter semaphore child function" << endl;
    sem_wait(&sem);
    int* input = (int*) data;
    setGPIO(input[0], input[1]);
    setGPIO(input[2], input[3]);
    pthread_exit(NULL);
}

void LED_FS_semaphore(char* freq, char* times) {
    cout << "Enter semaphore function" << endl;
    sem_init(&sem, 0, 0);
    pthread_t t1, t2;

    int f = atoi(freq) * 1000;
    int t = atoi(times);
    cout << t << endl;

    int data_0[4] = {LED1, 1, LED2, 0};
    int data_1[4] = {LED1, 0, LED2, 1};

    for(int i = 0; i < t; i++) {
        pthread_create(&t1, NULL, sem_child, data_0);
        sem_post(&sem);
        usleep(f);
        pthread_create(&t2, NULL, sem_child, data_1);
        sem_post(&sem);
        usleep(f);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        setGPIO(LED1, 0);
        setGPIO(LED2, 0);
    }
    return;
}
```

(6) Semaphore 模式的 function

```python
def setGPIO(gpio, status):
    subprocess.check_call([r"./Excute/final_excute", gpio, status])
```

(7) final.py 設定 LED function

```python
def main():
    init()
    try:
        adc_value = readadc(photo_ch,SPICLK,SPIMOSI,SPIMISO,SPICS)
        #print("Photoresistor's value:{}".format(adc_value))#光敏電阻數值
        print(adc_value)

        #以下code不確定要上TX2執行後在小小修改用上面那個adc_value判定
        if(adc_value > 800):
            setGPIO("LED1", "OFF")
            setGPIO("LED2", "OFF")
            #print("LED1 is ON \t LED2 is ON")
        elif(adc_value >  200):
            setGPIO("LED1", "ON")
            setGPIO("LED2", "OFF")
            #print("LED1 is ON \t LED2 is OFF")
        else:
            setGPIO("LED1", "ON")
            setGPIO("LED2", "ON")
            #print("LED1 is OFF \t LED2 is OFF")
        #以上code不確定要上TX2執行後在小小修改用上面那個adc_value判定
        time.sleep(1)
    finally:
        GPIO.cleanup()

if __name__ == '__main__':
    main()
```

(8) final.py 透過 adc_value 控制 LED(其餘部份與 Lab5 相同)

```
app.post("/change", async (req,res) => {
    var inp = req.body;
    if(inp["enable"] == 1){
        const transporter = nodemailer.createTransport({
          host: "mail.potatoserver.net",
          port: 465,
          auth: {
          user: "admin",
          pass: "123456789"
        }
        });
        await transporter.sendMail({
          from: 'admin@potatoserver.net',
          to: 'e6031583@gmail.com',
          subject: "智慧家電控制",
          text: inp["value"]
        })
    }else{
        // const transporter = nodemailer.createTransport({
        //     host: "mail.potatoserver.net",
        //     port: 465,
        //     auth: {
        //       user: "admin",
        //       pass: "123456789"
        //     }
        //   });
        //   await transporter.sendMail({
        //     from: 'admin@potatoserver.net',
        //     to: 'e6031583@gmail.com',
        //     subject: "suck",
        //     text: "my dick"
        //   })
    }
    res.end();
})
```

(9) control.js 切換模式寄送 Email 程式(其餘部份與 Lab4 大致相同)

5. 專案心得

這次的專案我們大概是照著老師給的基礎指示去完成，因為有其他的專案還沒有做完，所以時間很趕。不過這次的專案還是有遇到一些小問題。

(1) 閃爍的頻率：以往要讓程式睡眠一段時間是用 Sleep 函式可是這個函式輸入 integer 單位為秒，也就是說最少只能停一秒。最後找到一個 usleep()，他可以在 Arm 上的 Linux 執行而且他的單位是微秒。可以很精細的控制頻率，

(2) 光敏控制 LED：最一開始我是打算直接使用 Lab5 的程式，但是總是在 Initial GPIO 腳位的時候報出了 Permission denied 的錯誤。試了很多次最後發現應該是驅動已經有先 Export 取得權限了，接著就想說透過 Python 去修改”/dev/demo”的內容，可是不管怎麼試寫入”4661”(466 為 gpio，1 為輸出高電位) userChar 永遠只讀的到”466”而已，我找不出問題，不過我在想應該跟 Python 的字串處理方式有關。最後使用 subprocess.check_call()去執行 cpp 編譯出來的執行檔。

我負責的部份比較主要的問題是這兩個，感謝另外兩位組員分擔了不少工作。

6. 小組分工

109590014 沈煒翔 33% 網頁前後端，伺服器，製作報告，DeBug，上機測試

109590015 楊挺煜 33% 執行檔程式製作，驅動製作，製作報告，上機測試

109590023 廖堃霖 33% DeBug，上機測試，程式製作，製作報告