A collection of codes for smart homes

mainPro.c（主函数）

```
#include <stdio.h>
#include <string.h>
#include "contrlEquipments.h"
#include "inputCommand.h"
#include <pthread.h>
#include <unistd.h>

struct Equipment *findEquipByName(char *name,struct Equipment *phead);       //一些函数声明
struct Command *findCommandByName(char *name,struct Command *phead);
void *voiceControlThread(void *data);
void *socketControlThread(void *data);
void *socketReadThread(void *data);
void *fireAlarmThread(void *data);
void *airAlarmThread(void *data);

struct Equipment *equiphead = NULL;          //设备工厂链表头节点
struct Command *cmdhead = NULL;              //指令控制工厂链表节点头
struct Command *socketHandler = NULL;        //“网络控制线程”执行的函数使用到的全局变量


int main()
{
    if(wiringPiSetup() == -1){                  //使用wiringPi库需要初始化
        printf("wiringPiSetup failed!\n");
        return -1;
    }

    pthread_t voiceControl_thread;
    pthread_t socketControl_thread;
    pthread_t fireAlarm_thread;
    pthread_t airAlarm_thread;

    //1、设备工厂初始化
    equiphead = addBathroomLightToEquipmentLink(equiphead);          //各设备加入设备工厂
    equiphead = addSecondfloorLightToEquipmentLink(equiphead);
    equiphead = addLivingroomLightToEquipmentLink(equiphead);
    equiphead = addRestaurantLightToEquipmentLink(equiphead);
    equiphead = addFireDetectionToEquipmentLink(equiphead);
    equiphead = addBuzzerToEquipmentLink(equiphead);
    equiphead = addlockToEquipmentLink(equiphead);
    equiphead = addAirDetectionToEquipmentLink(equiphead);       //煤气传感器（对象）加入设备链表函数

    struct Equipment *tmpequiphead = equiphead;
    while(tmpequiphead != NULL){                          //设备工厂所有设备初始化
        tmpequiphead->Init(tmpequiphead->pinNum);
        tmpequiphead = tmpequiphead->next;
```

```
    }


    //2、指令工厂初始化
    cmdhead = addVoiceControlToCommandLink(cmdhead);                    //各指令控制加入指令控制工厂
    cmdhead = addSocketControlToCommandLink(cmdhead);



    //3、线程池建立
    //3.1 语音线程
    //int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void *), void *restrict arg);
    pthread_create(&voiceControl_thread,NULL,voiceControlThread,NULL);       //创建线程：语音控制
    //3.2 网络线程
    pthread_create(&socketControl_thread,NULL,socketControlThread,NULL);     //创建线程：网络控制
    //3.3 火灾线程
    pthread_create(&fireAlarm_thread,NULL,fireAlarmThread,NULL);             //创建线程：火灾报警系统
    //3.4 摄像头线程
    //3.5煤气线程
    pthread_create(&airAlarm_thread,NULL,airAlarmThread,NULL);           //创建线程：火灾报警系统

    pthread_join(voiceControl_thread, NULL);          //主函数等待线程退出
    pthread_join(socketControl_thread, NULL);         //主函数等待线程退出
    pthread_join(fireAlarm_thread, NULL);             //主函数等待线程退出
    pthread_join(airAlarm_thread, NULL);              //主函数等待线程退出
    return 0;
}



void *voiceControlThread(void *data)              //“语音控制线程”执行的函数
{
    int nread;
    char *temName = NULL;
    struct Command *voiceHandler = NULL;
    struct Equipment *linkHandler;


    voiceHandler = findCommandByName("voiceControl",cmdhead);        //寻找“语音控制”所在节点，返回给voiceHandler
    if(voiceHandler == NULL){
        printf("find voiceHandler error\n");
        pthread_exit(NULL);
    }
    if(voiceHandler->Init(voiceHandler) < 0){                 //“语音控制”功能初始化
        printf("voiceControl init error\n");
        pthread_exit(NULL);
    }


    while(1){
        nread = voiceHandler->getCommand(voiceHandler);          //获取指令
```

```
        if(nread == 0){                                         //没有获取到指令
            printf("No voiceCommand received\n");
        }else{                                                  //获取到指令
            printf("Get voice command:%s\n",voiceHandler->command);

            //以下为根据不用指令执行相应操作

            //语音模块串口传出来的后面带\r\n，不加对比不出来
            if(strcmp("ycdo\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);//改为泳
池灯
                linkHandler->open(linkHandler->pinNum);
                //printf("已打开浴室灯\n");
                printf("已打开泳池灯\n");
            }

            if(strcmp("ycdc\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
                //printf("已关闭浴室灯\n");
                printf("已关闭泳池灯\n");
            }

            if(strcmp("eldo\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("secondfloorLight",equiphead);
                linkHandler->open(linkHandler->pinNum);
            }

            if(strcmp("eldc\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("secondfloorLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
            }

            if(strcmp("ktdo\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("livingroomLight",equiphead);
                linkHandler->open(linkHandler->pinNum);
            }

            if(strcmp("ktdc\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("livingroomLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
            }

            if(strcmp("wsdo\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("restaurantLight",equiphead);//改为
卧室灯
                linkHandler->open(linkHandler->pinNum);
            }

            if(strcmp("wsdc\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("restaurantLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
            }

            if(strcmp("allo\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);//灯全部
打开
                linkHandler->open(linkHandler->pinNum);
```

```
                linkHandler = findEquipByName("secondfloorLight",equiphead);
                linkHandler->open(linkHandler->pinNum);

                linkHandler = findEquipByName("livingroomLight",equiphead);
                linkHandler->open(linkHandler->pinNum);

                linkHandler = findEquipByName("restaurantLight",equiphead);
                linkHandler->open(linkHandler->pinNum);
            }

            if(strcmp("allc\r\n",voiceHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);//灯全部
关闭
                linkHandler->close(linkHandler->pinNum);

                linkHandler = findEquipByName("secondfloorLight",equiphead);
                linkHandler->close(linkHandler->pinNum);

                linkHandler = findEquipByName("livingroomLight",equiphead);
                linkHandler->close(linkHandler->pinNum);

                linkHandler = findEquipByName("restaurantLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
            }
            if(strcmp("dooro\r\n",voiceHandler->command) == 0)
            {
                system("./smartHomeFaceRec");
            }
            memset(voiceHandler->command,'\0',sizeof(voiceHandler->command));//
不添加这个，不然只能识别一次

        }
    }
}


void *socketControlThread(void *data)               //“网络控制线程”执行的函数
{
    int c_fd;
    struct sockaddr_in c_addr;
    memset(&c_addr,0,sizeof(struct sockaddr_in));
    socklen_t clen = sizeof(struct sockaddr_in);
    pthread_t socketRead_thread; //线程里面套线程，网络连接后信息通信


    socketHandler = findCommandByName("socketControl",cmdhead);     //寻找“网络控
制”所在节点，返回给socketHandler
    if(socketHandler == NULL){
        printf("find socketHandler error\n");
        pthread_exit(NULL);
    }
    if(socketHandler->Init(socketHandler) < 0){             //“网络控制”功能初始化
        printf("socketControl init error\n");
        pthread_exit(NULL);
    }
```

```
    while(1){
        c_fd = accept(socketHandler->s_fd,(struct sockaddr*)&c_addr,&clen);
//接收连接请求，阻塞至有客户端完成三次握手
        socketHandler->fd = c_fd;                    //将套接字描述符返回给“网络控
制”链表节点

        pthread_create(&socketRead_thread,NULL,socketReadThread,NULL);
//创建新线程：用于读取TCP端口指令
//只要有连接，就创建线程去对接。线程共用内存资源，同一时刻，所有设备只有一种状态。也可PV操作
//所有线程 只操控一个结构体 再新来一个线程(新手机客户端接入) 前一个客户端失效 因为c_fd被改了。
fork()可实现多个客户端同时控制
//不过好像寄存器和内存不是完全同步的 可能缓存没改？还可以多个客户端同时控制？
//如果直接把socketReadThread()拿过来循环的话，则同时刻不能接受新的客户端接入了，因为循环卡在
了socketReadThread()函数里面了
    }
}


void *socketReadThread(void *data)              //“读取tcp端口指令线程”执行的函数
{

    int nread;
    struct Equipment *linkHandler;
    //这里没加while循环，客户端只能发送一次
    while(1)
    {
    printf("socketConnect...");
        memset(socketHandler->command,'\0',sizeof(socketHandler->command));
//将指令存放的空间置空

        nread = read(socketHandler->fd,socketHandler-
>command,sizeof(socketHandler->command));      //读取指令

        if(nread == 0){
            printf("No socketCommand received\n");          //没有读取到指令
        }else{
            printf("Get socketCommand:%s\n",socketHandler->command);        //读
取到指令

            //以下为根据不用指令执行相应操作

            if(strcmp("ycdo",socketHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);//改为泳
池灯
                linkHandler->open(linkHandler->pinNum);
                //printf("已打开浴室灯\n");
                printf("已打开泳池灯\n");
            }

            if(strcmp("ycdc",socketHandler->command) == 0){
                linkHandler = findEquipByName("bathroomLight",equiphead);
                linkHandler->close(linkHandler->pinNum);
                //printf("已关闭浴室灯\n");
                printf("已关闭泳池灯\n");
            }

            if(strcmp("eldo",socketHandler->command) == 0){
                linkHandler = findEquipByName("secondfloorLight",equiphead);
```

```
            linkHandler->open(linkHandler->pinNum);
        }

        if(strcmp("eldc",socketHandler->command) == 0){
            linkHandler = findEquipByName("secondfloorLight",equiphead);
            linkHandler->close(linkHandler->pinNum);
        }

        if(strcmp("ktdo",socketHandler->command) == 0){
            linkHandler = findEquipByName("livingroomLight",equiphead);
            linkHandler->open(linkHandler->pinNum);
        }

        if(strcmp("ktdc",socketHandler->command) == 0){
            linkHandler = findEquipByName("livingroomLight",equiphead);
            linkHandler->close(linkHandler->pinNum);
        }

        if(strcmp("wsdo",socketHandler->command) == 0){
            linkHandler = findEquipByName("restaurantLight",equiphead);//改为
卧室灯
            linkHandler->open(linkHandler->pinNum);
        }

        if(strcmp("wsdc",socketHandler->command) == 0){
            linkHandler = findEquipByName("restaurantLight",equiphead);
            linkHandler->close(linkHandler->pinNum);
        }

        if(strcmp("allo",socketHandler->command) == 0){
            linkHandler = findEquipByName("bathroomLight",equiphead);//灯全部
打开
            linkHandler->open(linkHandler->pinNum);

            linkHandler = findEquipByName("secondfloorLight",equiphead);
            linkHandler->open(linkHandler->pinNum);

            linkHandler = findEquipByName("livingroomLight",equiphead);
            linkHandler->open(linkHandler->pinNum);

            linkHandler = findEquipByName("restaurantLight",equiphead);
            linkHandler->open(linkHandler->pinNum);
        }

        if(strcmp("allc",socketHandler->command) == 0){
            linkHandler = findEquipByName("bathroomLight",equiphead);//灯全部
关闭
            linkHandler->close(linkHandler->pinNum);

            linkHandler = findEquipByName("secondfloorLight",equiphead);
            linkHandler->close(linkHandler->pinNum);

            linkHandler = findEquipByName("livingroomLight",equiphead);
            linkHandler->close(linkHandler->pinNum);

            linkHandler = findEquipByName("restaurantLight",equiphead);
            linkHandler->close(linkHandler->pinNum);
        }
```

```
            if(strcmp("dooro",socketHandler->command) == 0)
            {
                linkHandler = findEquipByName("lock",equiphead);
                linkHandler->open(linkHandler->pinNum);
                delay(2000);
                linkHandler->close(linkHandler->pinNum);

            }

        }
    }
}


void *fireAlarmThread(void *data)//有火-返回高电平                //“火灾报警器线程”执行的函数
{
    int status;
    struct Equipment *firetmp = NULL;
    struct Equipment *buztmp = NULL;

    firetmp = findEquipByName("fireDetection",equiphead);        //寻找“火焰传感器”链表节点，返回给firetmp
    buztmp = findEquipByName("buzzer",equiphead);                //寻找“蜂鸣器”链表节点，返回给buztmp

    while(1){
        status = firetmp->readStatus(firetmp->pinNum);           //读取“火焰传感器”状态

        if(status == 1){                            //检测到火焰或强光源
            buztmp->open(buztmp->pinNum);           //打开蜂鸣器
            perror("fire-why");
            delay(1000);                            //延时1000毫秒=1秒
        }

        if(status == 0){                            //未检测到火焰、强光源或解除警报
            buztmp->close(buztmp->pinNum);          //关闭蜂鸣器
        }
    }
}

void *airAlarmThread(void *data)//煤气泄漏-返回低电平                //“煤气泄漏报警器线程”执行的函数
{
    int status;
    struct Equipment *airtmp = NULL;
    struct Equipment *buztmp = NULL;

    airtmp = findEquipByName("airDetection",equiphead);      //寻找“煤气传感器”链表节点，返回给airtmp
    buztmp = findEquipByName("buzzer",equiphead);                //寻找“蜂鸣器”链表节点，返回给buztmp

    while(1){
        status = airtmp->readStatus(airtmp->pinNum);             //读取“煤气传感器”状态
/*  //玄学，加了煤气检测代码，要煤气检测，和火灾同时有，蜂鸣器才能响
```

```
        if(status == 0){                        //检测到煤气
            buztmp->open(buztmp->pinNum);       //打开蜂鸣器
            perror("air-why");
            delay(1500);                        //延时1000毫秒=1秒
        }
        if(status == 1){                        //未检测到煤气解除警报
            buztmp->close(buztmp->pinNum);      //关闭蜂鸣器
        }
*/
        }
}

struct Equipment *findEquipByName(char *name,struct Equipment *phead)        //根据名字寻找设备工厂链表链节函数，并返回链节
{
    struct Equipment *tmp = phead;

    if(phead == NULL){
        return NULL;
    }

    while(tmp != NULL){
        if(strcmp(name,tmp->equipName) == 0){
            return tmp;
        }
        tmp = tmp->next;
    }
    return NULL;
}


struct Command *findCommandByName(char *name,struct Command *phead)          //根据名字寻找指令控制工厂链表链节函数，并返回链节
{
    struct Command *tmp = phead;

    if(phead == NULL){
        return NULL;
    }

    while(tmp != NULL){
        if(strcmp(name,tmp->commandName) == 0){
            return tmp;
        }
        tmp = tmp->next;
    }
    return NULL;
}
```

指令工厂

inputCommand.h

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

struct Command                                  //指令控制工厂链表节点定义
{
    char commandName[128];                      //“控制方式”名字
    char deviceFilesName[128];                  //存放初始化功能需要打开的文件的路径
    char command[32];                           //存放指令
    int fd;                                     //存放文件描述符 用于串口/客户端fd
    int (*Init)(struct Command *file);          //“初始化”函数指针
    int s_fd;                                   //存放套接字描述符
    char ipAdress[32];                          //存放IP地址
    char port[12];                              //存放端口号
    int (*getCommand)(struct Command *cmd); //“获取指令”函数指针
    char log[1024];                             //日志（暂未使用）

    struct Command *next;
};

struct Command *addVoiceControlToCommandLink(struct Command *phead);
//“语音控制”加入指令控制工厂链表函数声明
struct Command *addSocketControlToCommandLink(struct Command *phead);
//“网络控制”加入指令控制工厂链表函数声明
```

voiceControl.c（语音控制）

```c
#include "inputCommand.h"
#include <unistd.h>

int voiceControlInit(struct Command *file);                             //“语音控制”功能初始化函数声明
int voiceControlGetCommand(struct Command *cmd);                        //“获取指令”函数声明
//struct Command *addVoiceControlToLink(struct Command *phead);      //“语音控制”加入指令控制工厂链表函数声明


struct Command voiceControl = {                 //“语音控制”链表节点
    .commandName = "voiceControl",
    .deviceFilesName = "/dev/ttyAMA0",
    .command = {'\0'},
    .Init = voiceControlInit,                   //这里只是定义，还未调用改函数
    .getCommand = voiceControlGetCommand,
    .log = {'\0'},
};


int voiceControlInit(struct Command *file)
{
    int fd;
```

```
    if((fd = serialOpen(file->deviceFilesName,9600)) == -1){        //打开树莓派串口，波特率为9600
        exit(-1);
    }
    file->fd = fd;              //打开串口文件成功，返回“文件描述符”到“语音控制”链表节点中
}


int voiceControlGetCommand(struct Command *cmd)                 //“获取指令”函数
{
    int nread = 0;
    memset(cmd->command,'\0',sizeof(cmd->command));                 //防止老的消息影响新的消息
    nread = read(cmd->fd,cmd->command,sizeof(cmd->command));        //返回读取到数据的字节数
    return nread;
}


struct Command *addVoiceControlToCommandLink(struct Command *phead)     //头插法将“语音控制”链表节点加入指令控制工厂链表函数
{
    if(phead == NULL){
        return &voiceControl;
    }else{
        voiceControl.next = phead;
        phead = &voiceControl;
        return phead;
    }
}
```

socketControl.c（网络线程）

```
#include "inputCommand.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>


int socketControlInit(struct Command *file);                    //“网络控制”功能初始化函数声明
//struct Command *addSocketControlToLink(struct Command *phead);    //“网络控制”加入指令控制工厂链表函数声明


struct Command socketControl = {        //“网络控制”链表节点
    .commandName = "socketControl",
    .command = {'\0'},
    .Init = socketControlInit,
    .ipAdress = "192.168.0.19",     //树莓派连接网络时的IP地址
    .port = "8088",                     //树莓派打开待外界连接的端口号
    .log = {'\0'},
```

```
};


int socketControlInit(struct Command *file)
{
    int s_fd;                                                   //套接字描述符
    struct sockaddr_in s_addr;
    memset(&s_addr,0,sizeof(struct sockaddr_in));

    s_fd = socket(AF_INET,SOCK_STREAM,0);                       //创建套接字
    if(s_fd == -1){                                             //创建套接字失败时
            perror("socketControl error");
            exit(-1);
    }

    s_addr.sin_family = AF_INET;
    s_addr.sin_port = htons(atoi(file->port));
    inet_aton(file->ipAdress,&s_addr.sin_addr);
    if(bind(s_fd,(struct sockaddr*)&s_addr,sizeof(struct sockaddr_in)) == -1){
//套接字与端口号绑定
        perror("bind error");
        exit(-1);
    }

    if(listen(s_fd,10) == -1){      //打开监听 accept放到主函数线程里
        perror("listen error");
        exit(-1);
    }

    file->s_fd = s_fd;                          //套接字描述符返回到“网络控制”链表节点
}


struct Command *addSocketControlToCommandLink(struct Command *phead)
//头插法将设备节点加入设备工厂链表函数
{
    if(phead == NULL){
        return &socketControl;
    }else{
        socketControl.next = phead;
        phead = &socketControl;
        return phead;
    }
}
```

控制工厂

contrlEquipments.h

```
#include <wiringPi.h>                   //wiringPi库
#include <stdio.h>
#include <stdlib.h>
struct Equipment                        //设备类
{
    char equipName[128];                //设备名
```

```
    int pinNum;                         //引脚号
    int (*Init)(int pinNum);            //“初始化设备”函数指针
    int (*open)(int pinNum);            //“打开设备”函数指针
    int (*close)(int pinNum);           //“关闭设备”函数指针

    int (*readStatus)(int pinNum);      //“读取设备状态”函数指针   为火灾报警器准备
    int (*changeStatus)(int status);    //“改变设备状态”函数指针

    struct Equipment *next;
};

struct Equipment *addBathroomLightToEquipmentLink(struct Equipment *phead);
//“浴室灯”加入设备链表函数声明
struct Equipment *addSecondfloorLightToEquipmentLink(struct Equipment *phead);
//“二楼灯”加入设备链表函数声明
struct Equipment *addLivingroomLightToEquipmentLink(struct Equipment *phead);
//“客厅灯”加入设备链表函数声明
struct Equipment *addRestaurantLightToEquipmentLink(struct Equipment *phead);
//“餐厅灯”加入设备链表函数声明
struct Equipment *addFireDetectionToEquipmentLink(struct Equipment *phead);
//“火焰传感器”加入设备链表函数声明
struct Equipment *addBuzzerToEquipmentLink(struct Equipment *phead);
//“蜂鸣器”加入设备链表函数声明
struct Equipment *addlockToEquipmentLink(struct Equipment *phead);      //“电磁锁
加入设备链表函数声明”
```

bathroomLight.c（浴室灯）

```
#include "contrlEquipments.h"           //自定义设备类的文件

int bathroomLightInit();        //初始化继电器函数声明
int bathroomLightOpen();        //“打开灯”函数声明
int bathroomLightClose();       //“关闭灯”函数声明
//struct Equipment *addBathroomLightToEquipmentLink(struct Equipment *phead);
浴室灯（对象）加入设备链表函数声明

struct Equipment bathroomLight = {          //定义浴室灯（对象）
    .equipName = "bathroomLight",           //名字
    .pinNum = 2,                            //树莓派 2号(wPi)引脚
    .Init = bathroomLightInit,              //指定初始化函数
    .open = bathroomLightOpen,              //指定“打开灯”函数
    .close = bathroomLightClose             //指定“关闭灯”函数
};


int bathroomLightInit(int pinNum)           //C语言必须要传参，JAVA不用，可直接访问变
量的值
{
    pinMode(pinNum,OUTPUT);                 //配置引脚为输出模式
    digitalWrite(pinNum,HIGH);              //引脚置高电平，断开继电器
}

int bathroomLightOpen(int pinNum)
{
    digitalWrite(pinNum,LOW);               //引脚置低电平，闭合继电器
}
```

```
int bathroomLightClose(int pinNum)
{
    digitalWrite(pinNum,HIGH);              //引脚置高电平，断开继电器
}

struct Equipment *addBathroomLightToEquipmentLink(struct Equipment *phead)
//浴室灯（对象）加入设备链表函数
{
    if(phead == NULL){
        return &bathroomLight;
    }else{
        bathroomLight.next = phead;  //以前的头变成.next
        phead = &bathroomLight;      //更新头
        return phead;
    }
}
```

secondfloorLight.c（二楼灯）

```
#include "contrlEquipments.h"               //自定义设备类的文件

int secondfloorLightInit();          //初始化继电器函数声明
int secondfloorLightOpen();          //“打开灯”函数声明
int secondfloorLightClose();         //“关闭灯”函数声明

//struct Equipment *addSecondfloorLightToLink(struct Equipment *phead);
二楼灯（对象）加入设备链表函数声明

struct Equipment secondfloorLight = {           //定义二楼灯（对象）
    .equipName = "secondfloorLight",            //名字
    .pinNum = 4,                                //树莓派 4号(wPi)引脚
    .Init = secondfloorLightInit,               //指定初始化函数
    .open = secondfloorLightOpen,               //指定“打开灯”函数
    .close = secondfloorLightClose,             //指定“关闭灯”函数
};


int secondfloorLightInit(int pinNum)
{
    pinMode(pinNum,OUTPUT);                     //配置引脚为输出模式
    digitalWrite(pinNum,HIGH);                  //引脚置高电平，断开继电器
}

int secondfloorLightOpen(int pinNum)
{
    digitalWrite(pinNum,LOW);                   //引脚置低电平，闭合继电器
}

int secondfloorLightClose(int pinNum)
{
    digitalWrite(pinNum,HIGH);                  //引脚置高电平，断开继电器
}

struct Equipment *addSecondfloorLightToEquipmentLink(struct Equipment *phead)
//二楼灯（对象）加入设备链表函数
```

```
{
    if(phead == NULL){
        return &secondfloorLight;
    }else{
        secondfloorLight.next = phead;
        phead = &secondfloorLight;
        return phead;
    }
}
```

livingroomLight.c（客厅灯）

```
#include "contrlEquipments.h"                //自定义设备类的文件

int livingroomLightInit();           //初始化继电器函数声明
int livingroomLightOpen();           //“打开灯”函数声明
int livingroomLightClose();          //“关闭灯”函数声明
//struct Equipment *addLivingroomLightToLink(struct Equipment *phead);
//客厅灯（对象）加入设备链表函数声明
/*
struct Equipment                          //设备类
{
    char equipName[128];                  //设备名
    int pinNum;                           //引脚号
    int (*Init)(int pinNum);              //“初始化设备”函数指针
    int (*open)(int pinNum);              //“打开设备”函数指针
    int (*close)(int pinNum);             //“关闭设备”函数指针
    int (*readStatus)(int pinNum);        //“读取设备状态”函数指针   为火灾报警器准备
    int (*changeStatus)(int status);      //“改变设备状态”函数指针
    struct Equipment *next;
};
*/

struct Equipment livingroomLight = {             //定义客厅灯（对象）
    .equipName = "livingroomLight",              //名字
    .pinNum = 1,                          //树莓派 1号(wPi)引脚
    .Init = livingroomLightInit,                 //指定初始化函数
    .open = livingroomLightOpen,                 //指定“打开灯”函数
    .close = livingroomLightClose,               //指定“关闭灯”函数
};


int livingroomLightInit(int pinNum)
{
    pinMode(pinNum,OUTPUT);                      //配置引脚为输出模式
    digitalWrite(pinNum,HIGH);                   //引脚置高电平，断开继电器
}

int livingroomLightOpen(int pinNum)
{
    digitalWrite(pinNum,LOW);                    //引脚置低电平，闭合继电器
}

int livingroomLightClose(int pinNum)
```

```
{
    digitalWrite(pinNum,HIGH);                  //引脚置高电平，断开继电器
}

struct Equipment *addLivingroomLightToEquipmentLink(struct Equipment *phead)
//客厅灯（对象）加入设备链表函数
{
    if(phead == NULL){
        return &livingroomLight;
    }else{
        livingroomLight.next = phead;
        phead = &livingroomLight;
        return phead;
    }
}
```

restaurantLight.c（餐厅灯）

```
#include "contrlEquipments.h"               //自定义设备类的文件

int restaurantLightInit();          //初始化继电器函数声明
int restaurantLightOpen();          //“打开灯”函数声明
int restaurantLightClose();         //“关闭灯”函数声明
struct Equipment *addRestaurantLightToLink(struct Equipment *phead);
//餐厅灯（对象）加入设备链表函数声明


struct Equipment restaurantLight = {            //定义餐厅灯（对象）
    .equipName = "restaurantLight",             //名字
    .pinNum = 3,                                //树莓派 3号(wPi)引脚
    .Init = restaurantLightInit,                //指定初始化函数
    .open = restaurantLightOpen,                //指定“打开灯”函数
    .close = restaurantLightClose,              //指定“关闭灯”函数
};


int restaurantLightInit(int pinNum)
{
    pinMode(pinNum,OUTPUT);                     //配置引脚为输出模式
    digitalWrite(pinNum,HIGH);                  //引脚置高电平，断开继电器
}

int restaurantLightOpen(int pinNum)
{
    digitalWrite(pinNum,LOW);                   //引脚置低电平，闭合继电器
}

int restaurantLightClose(int pinNum)
{
    digitalWrite(pinNum,HIGH);                  //引脚置高电平，断开继电器
}

struct Equipment *addRestaurantLightToEquipmentLink(struct Equipment *phead)
//餐厅灯（对象）加入设备链表函数
{
    if(phead == NULL){
```

```
        return &restaurantLight;
    }else{
        restaurantLight.next = phead;
        phead = &restaurantLight;
        return phead;
    }
}
```

fireDetection.c（火焰传感器）

```
#include "contrlEquipments.h"

int fireDetectionInit(int pinNum);                    //一些函数声明
int readFireDetectionStatus(int pinNum);
//struct Equipment *addFireDetectionToLink(struct Equipment *phead);


struct Equipment fireDetection = {          //“火焰传感器”设备链表节点
    .equipName = "fireDetection",
    .pinNum = 21,                           //树莓派gpio引脚21
    .Init = fireDetectionInit,
    .readStatus = readFireDetectionStatus,
};


int fireDetectionInit(int pinNum)           //初始化函数
{
    pinMode(pinNum,INPUT);                  //配置引脚为输入引脚
    digitalWrite(pinNum,HIGH);              //引脚输出高电平，即默认为关闭状态
}

int readFireDetectionStatus(int pinNum)     //读取“火焰传感器”状态函数
{
    return digitalRead(pinNum);
}


struct Equipment *addFireDetectionToEquipmentLink(struct Equipment *phead)
{
    if(phead == NULL){
        return &fireDetection;
    }else{
        fireDetection.next = phead;
        phead = &fireDetection;
        return phead;
    }
}
```

buzzer.c （蜂鸣器）

```
#include "contrlEquipments.h"

int buzzerInit(int pinNum);                 //一些函数声明
int buzzerOpen(int pinNum);
```

```
int buzzerClose(int pinNum);
struct Equipment *addBuzzerToEquipmentLink(struct Equipment *phead);


struct Equipment buzzer = {          //“蜂鸣器”设备链表节点
    .equipName = "buzzer",
    .pinNum = 22,                    //树莓派gpio引脚22
    .Init = buzzerInit,
    .open = buzzerOpen,
    .close = buzzerClose,
};


int buzzerInit(int pinNum)                    //初始化函数
{
    pinMode(pinNum,OUTPUT);                   //配置引脚为输出引脚
    digitalWrite(pinNum,HIGH);                //引脚输出高电平，即默认为关闭状态
}

int buzzerOpen(int pinNum)                    //打开函数
{
    digitalWrite(pinNum,LOW);
}

int buzzerClose(int pinNum)                   //关闭函数
{
    digitalWrite(pinNum,HIGH);
}


struct Equipment *addBuzzerToEquipmentLink(struct Equipment *phead)          //头
插法将设备节点加入设备工厂链表函数
{
    if(phead == NULL){
        return &buzzer;
    }else{
        buzzer.next = phead;
        phead = &buzzer;
        return phead;
    }
}
```

lock.c

```
#include "contrlEquipments.h"                 //自定义设备类的文件

int lockInit();          //初始化继电器函数声明
int lockOpen();          //“打开lock”函数声明
int lockClose();             //“关闭lock”函数声明

struct Equipment lock = {            //定义客厅lock（对象）
    .equipName = "lock",                 //名字
    .pinNum = 5,                         //树莓派 5号(wPi)引脚
    .Init = lockInit,                //指定初始化函数
    .open = lockOpen,                //指定“打开lock”函数
    .close = lockClose,              //指定“关闭lock”函数
```

```
};


int lockInit(int pinNum)
{
    pinMode(pinNum,OUTPUT);                         //配置引脚为输出模式
    digitalWrite(pinNum,HIGH);                      //引脚置高电平，断开继电器
}

int lockOpen(int pinNum)
{
    digitalWrite(pinNum,LOW);                       //引脚置低电平，闭合继电器
}

int lockClose(int pinNum)
{
    digitalWrite(pinNum,HIGH);                      //引脚置高电平，断开继电器
}

struct Equipment *addlockToEquipmentLink(struct Equipment *phead)
//lock（对象）加入设备链表函数
{
    if(phead == NULL){
        return &lock;
    }else{
        lock.next = phead;
        phead = &lock;
        return phead;
    }
}
```

camera.c(人脸识别)

```
#include <stdio.h>
#include <curl/curl.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include<wiringPi.h>

#define true 1
#define false 0
typedef unsigned int bool;


int gpio_init(void)
{
    int err_num = wiringPiSetup();
    pinMode(5, OUTPUT);
    digitalWrite(5,HIGH);
    pinMode(29, OUTPUT);
    digitalWrite(29,HIGH);
    return err_num;
```

```
}

void lock_open(void)
{
    digitalWrite(5,LOW);
    digitalWrite(29,LOW);
    delay(20);
    digitalWrite(29,HIGH);
    delay(2000);
    digitalWrite(5,HIGH);
}

size_t read_func( void *ptr, size_t size, size_t nmemb, void *stream)
{
        char buf[1024] = {0};
        strncpy(buf, ptr, 1024);
        printf("======================get data======================\n");
        printf("%s\n",buf);

        printf("======================result======================\n");
        if(strstr(buf,"是") != NULL)
        {
        lock_open();
                printf("the same person\n");
        }
        else
        {
                printf("different person\n");
        }

}
char* getPicBase64FromFile(char *filePath)
{

        char *bufPic;
        char cmd[128]={'\0'};

        sprintf(cmd,"base64 %s >tmpFile",filePath);

        system(cmd);

        int fd = open("./tmpFile",O_RDWR);
        int fd_len = lseek(fd,0,SEEK_END);
        lseek(fd,0,SEEK_SET);

        bufPic = (char*)malloc(fd_len+2);

        memset(bufPic,'\0',fd_len+2);

        read(fd,bufPic,fd_len);

        close(fd);

        system("rm -rf tmpFile");

        return bufPic;

}
```

```
bool postUrl()
{
        CURL *curl;
        CURLcode res;

        char *postString;

        char *img1;
        char *img2;
        char *key = "7qUgFcjemNwL5m5dCFH7gk";
        char *secret = "c8a238f46dae4ea3a98eb8baaa2ab2d0";
        int typeId = 21;
        char *formate = "xml";

    //拍摄图片
    system("raspistill -o after.jpg -t 1000");

        //图片转字符流

        img1 = getPicBase64FromFile("./before.jpg");

        img2 = getPicBase64FromFile("./after.jpg");

    //删除拍摄的图片
    system("rm -rf after.jpg");

        postString =
(char*)malloc(strlen(key)+strlen(secret)+strlen(img1)+strlen(img2)+1024);
        memset(postString,'\0',strlen(postString));

        //字符串拼接

 sprintf(postString,"&img1=%s&img2=%s&key=%s&secret=%s&typeId=%d&formate=%s",
                        img1,     img2,    key,    secret,   typeId,
 formate);

        curl = curl_easy_init();
        if (curl)
        {
                curl_easy_setopt(curl, CURLOPT_COOKIEFILE, "/tmp/cookie.txt");
// 指定cookie文件
                curl_easy_setopt(curl, CURLOPT_POSTFIELDS, postString);    // 指
定post内容
                curl_easy_setopt(curl,
CURLOPT_URL,"https://netocr.com/api/faceliu.do");   // 指定url
                curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, read_func); //通过
回调函数存储数据
                res = curl_easy_perform(curl);

                printf("OK:%d\n",res);


                curl_easy_cleanup(curl);
        }
        free(img1);
        free(img2);
        free(postString);
```

```
        return true;
}
int main(void)
{
    int err_num = gpio_init();
    if(err_num == -1)
    {
        return -1;
    }

        postUrl();
}
```

语音模块部分：

```
#include "asr.h"
#include "setup.h"
#include "HardwareSerial.h"
#include "myLib/luxiaoban.h"
#include "myLib/asr_event.h"

uint32_t snid;
void app();

//{ID:250,keyword:"命令词",ASR:"最大音量",ASRTO:"音量调整到最大"}
//{ID:251,keyword:"命令词",ASR:"中等音量",ASRTO:"音量调整到中等"}
//{ID:252,keyword:"命令词",ASR:"最小音量",ASRTO:"音量调整到最小"}
void app(){
  while (1) {
    if(luxiaoban_digital_read(7)==0){
      //{ID:500,keyword:"命令词",ASR:"要接官",ASRTO:"门已打开，欢迎主人"}
      play_audio(500);
    }
    luxiaoban_digital_write(7,1);
    delay(1);
  }
  vTaskDelete(NULL);
}

void ASR_CODE()
{
  //{ID:501,keyword:"唤醒词",ASR:"小明同学",ASRTO:"我在"}
  if(snid == 501){
    Serial.println("hello");
    Serial.flush();
  }
  //{ID:502,keyword:"命令词",ASR:"打开泳池灯",ASRTO:"泳池灯已打开"}
  if(snid == 502){
    Serial.println("ycdo");
    Serial.flush();
  }
  //{ID:503,keyword:"命令词",ASR:"关闭泳池灯",ASRTO:"泳池灯已关闭"}
  if(snid == 503){
    Serial.println("ycdc");
```

```
    Serial.flush();
  }
  //{ID:504,keyword:"命令词",ASR:"打开二楼灯",ASRTO:"二楼灯已打开"}
  if(snid == 504){
    Serial.println("eldo");
    Serial.flush();
  }
  //{ID:505,keyword:"命令词",ASR:"关闭二楼灯",ASRTO:"二楼灯已关闭"}
  if(snid == 505){
    Serial.println("eldc");
    Serial.flush();
  }
  //{ID:506,keyword:"命令词",ASR:"打开客厅灯",ASRTO:"客厅灯已打开"}
  if(snid == 506){
    Serial.println("ktdo");
    Serial.flush();
  }
  //{ID:507,keyword:"命令词",ASR:"关闭客厅灯",ASRTO:"客厅灯已关闭"}
  if(snid == 507){
    Serial.println("ktdc");
    Serial.flush();
  }
  //{ID:508,keyword:"命令词",ASR:"打开卧室灯",ASRTO:"卧室灯已打开"}
  if(snid == 508){
    Serial.println("wsdo");
    Serial.flush();
  }
  //{ID:509,keyword:"命令词",ASR:"关闭卧室灯",ASRTO:"卧室灯已关闭"}
  if(snid == 509){
    Serial.println("wsdc");
    Serial.flush();
  }
  //{ID:510,keyword:"命令词",ASR:"开门",ASRTO:""}
  if(snid == 510){
    Serial.println("dooro");
    Serial.flush();
  }
  //{ID:511,keyword:"命令词",ASR:"打开所有灯光",ASRTO:"灯光已全部打开"}
  if(snid == 511){
    Serial.println("allo");
    Serial.flush();
  }
  //{ID:512,keyword:"命令词",ASR:"关闭所有灯光",ASRTO:"灯光已全部关闭"}
  if(snid == 512){
    Serial.println("allc");
    Serial.flush();
  }
}

void setup()
{
  Serial.begin(115200);
  luxiaoban_digital_write_all(1);
  //{speak:小蝶-清新女声,vol:10,speed:10,platform:haohaodada}
  //{playid:10001,voice:欢迎使用小明同学，用小明同学唤醒我。}
  //{playid:10002,voice:我退下了，用小明同学唤醒我}
  set_wakeup_forever();
  xTaskCreate(app,"app",128,NULL,1,NULL);
```

```
}
```