

龙芯应用开发实战

提纲 CONTENTS

- 1/ 驱动模块开发 (I2C、PWM)
- 2/ 应用程序开发
- 3/ WEB服务开发
- 4/ 实战案例：城市环境检测系统设计

一、PWM 驱动应用开发

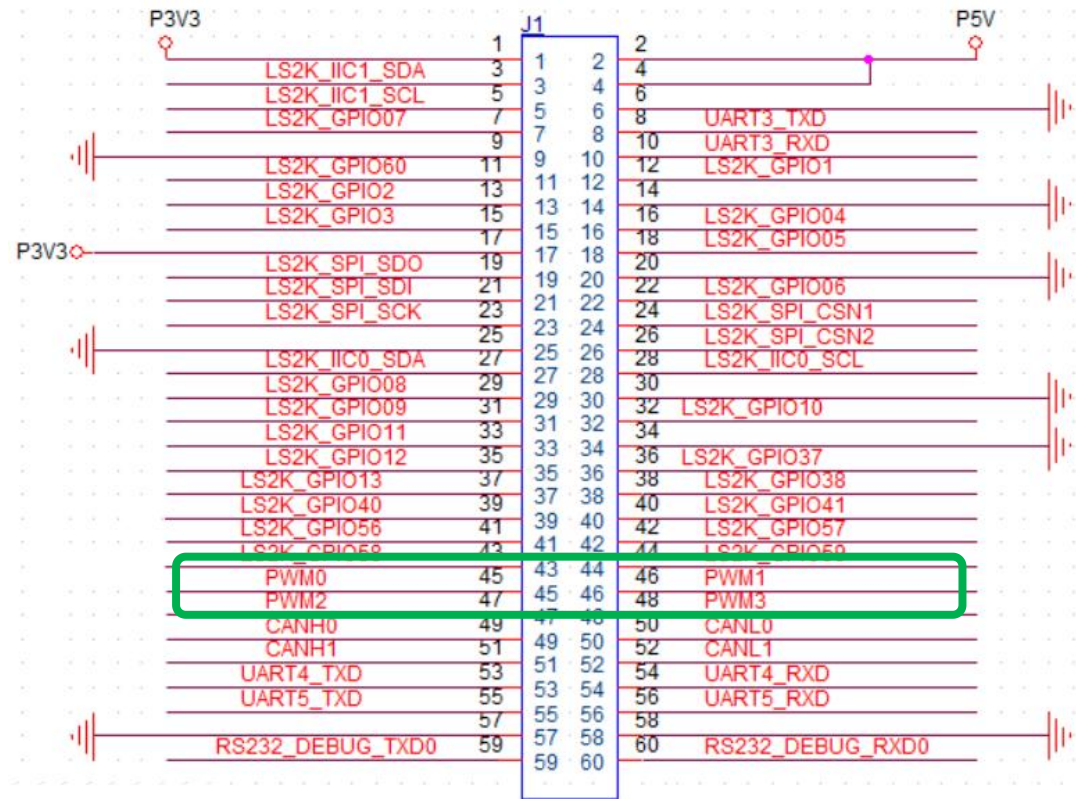
➤ 2K1000的PWM

2K1000 芯片里实现了四路脉冲宽度调节/计数控制器，以下简称 PWM。每一路 PWM 工作和控制方式完全相同。每路 PWM 有一路脉冲宽度输出信号和一路待测脉冲输入信号。系统时钟高达 125MHz，计数寄存器和参考寄存器均 32 位数据宽度。

一、PWM驱动

龙芯2K1000的PWM

2K1000芯片里实现了四路脉冲宽度调节/计数控制器，以下简称 PWM。每一路 PWM 工作和控制方式完全相同。每路 PWM 有一路脉冲宽度输出信号和一路待测脉冲输入信号。系统时钟高达 125MHz，计数寄存器和参考寄存器均 32 位数据宽度。



接口定义

➤ PWM - LED 闪光灯

PWM，是一种对模拟信号电平进行数字编码的方法，就是通过调整高低电平在一个周期信号里的比例时间。在嵌入式设备中，PWM 多用于控制马达、LED、振动器等模拟器件。

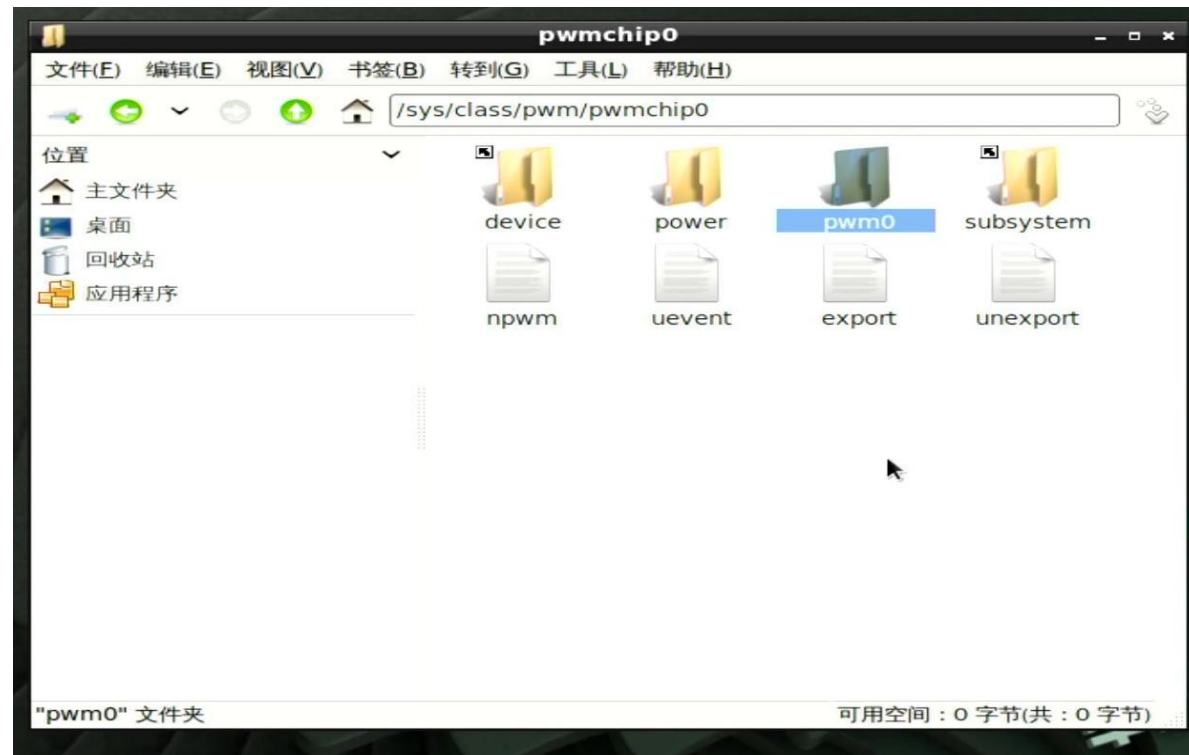
龙芯派 2 代集成了四路 PWM 控制器，每一路 PWM 工作和控制方式完全相同。每路 PWM 有一路脉冲宽度输出信号和一路待测脉冲输入信号。系统时钟高达 125MHz，计数寄存器和参考寄存器均 32 位数据宽度。

一、PWM驱动

➤ 龙芯2K1000的PWM

龙芯2K1000的PWM设备驱动文件位于
/sys/class/pwm目录下，可以看到4个PWM接口文件，分别对应了PWM0~PWM3。

其中export功能与GPIO中的export功能一致，都是用来将接口暴露到用户空间中，写0表示导出到用户空间，对unexport写0则表示关闭。在命令行输入命令：`echo 0 > export`。可以看到接口文件就暴露了出来。



PWM驱动文件

一、PWM驱动

龙芯2K1000的PWM

PWM的参数设置文件就位于该文件接口内，其中period用于设置PWM的周期，单位是ns纳秒。duty_cycle用于设置PWM的高占空比（占空比是指在一个脉冲循环内器件通电时间相对于总时间所占的比例， $\text{占空比} = \text{duty_cycle} / \text{period}$ ）。enable使能PWM开始工作，写入1表示使能工作，写入0表示禁止。



PWM驱动文件

➤ PWM的驱动代码

PWM的驱动编写十分简单，只需要修改这几个文件的值就能输出相应宽度频率的脉冲，从而达到控制马达、LED、振动器等模拟器件的目的。

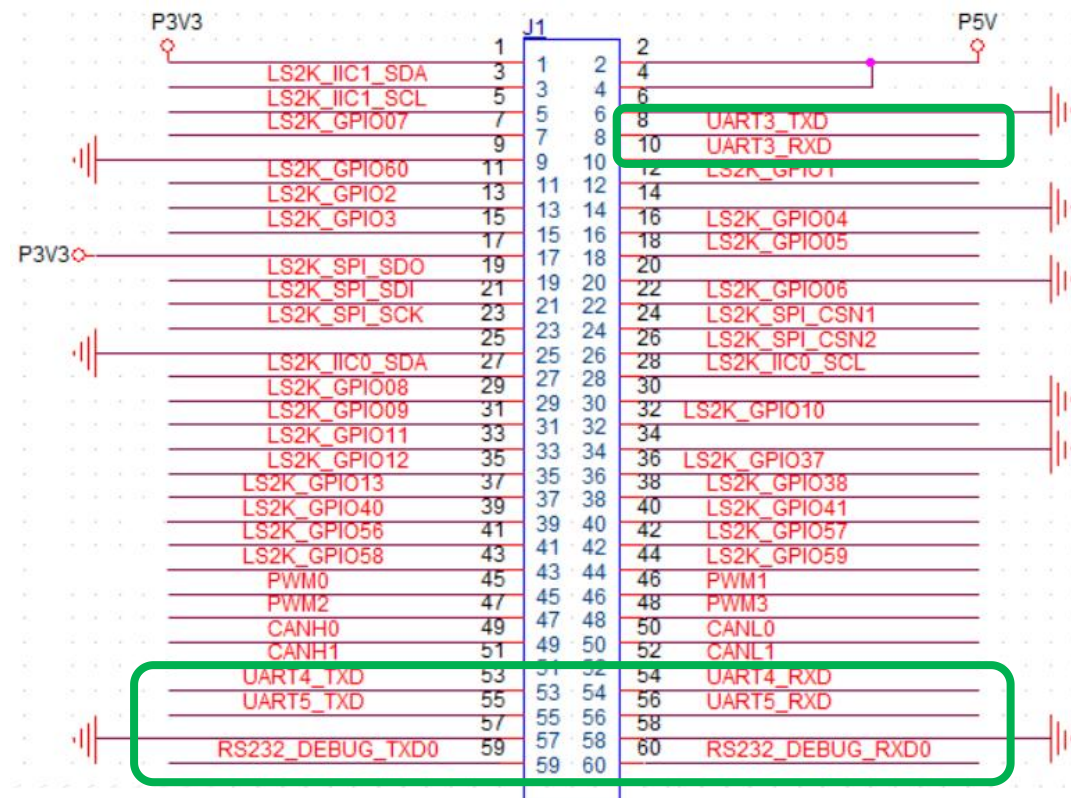
其核心代码就是使用open()函数打开对应的文件，然后用write()函数修改文件的值。如图所示。

```
//使能pwm
int pwm_enable(unsigned int pwm)
{
    int fd;
    if(pwm == 0)
    {fd = open("/sys/class/pwm/pwmchip0/pwm0/enable",O_WRONLY);}
    else if(pwm == 1)
    {fd = open("/sys/class/pwm/pwmchip1/pwm0/enable",O_WRONLY);}
    else if(pwm == 2)
    {fd = open("/sys/class/pwm/pwmchip2/pwm0/enable",O_WRONLY);}
    else if(pwm == 3)
    {fd = open("/sys/class/pwm/pwmchip3/pwm0/enable",O_WRONLY);}
    if(fd<0)
    {
        printf("\nFailed enable PWM%d\n",pwm);
        return -1;
    }
    write(fd,"1",2);
    close(fd);
    return 0;
}
```


一、UART串口驱动

龙芯2K1000的UART串口

龙芯2K1000开发版预留了4路UART串口通信接口，其中UART3~5是TTL，还有一路是RS232协议的串口，调试时要分清楚使用的串口协议类型和相应的串口线。

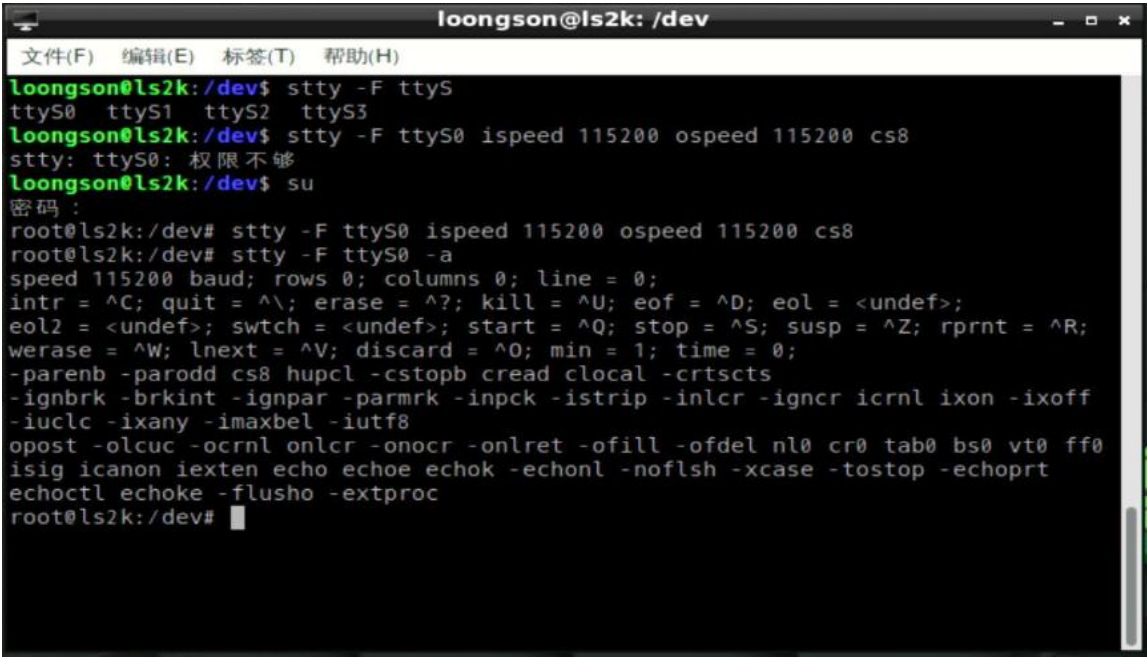


接口定义

一、UART串口驱动

➤ 串口驱动文件

龙芯2K1000的UART串口驱动文件位于/dev下的ttyS0~ttyS3，分别对应UART_RS232、UART3~UART5。可以通过stty命令设置和测试串口通信。



```
loongson@ls2k: /dev
文件(F) 编辑(E) 标签(T) 帮助(H)
loongson@ls2k:/dev$ stty -F ttyS
ttyS0  ttyS1  ttyS2  ttyS3
loongson@ls2k:/dev$ stty -F ttyS0 ispeed 115200 ospeed 115200 cs8
stty: ttyS0: 权限不够
loongson@ls2k:/dev$ su
密码:
root@ls2k:/dev# stty -F ttyS0 ispeed 115200 ospeed 115200 cs8
root@ls2k:/dev# stty -F ttyS0 -a
speed 115200 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R;
werase = ^W; lnext = ^V; discard = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread clocal -crtcts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echopr
echoctl echoke -flusho -extproc
root@ls2k:/dev#
```

通过stty命令设置ttyS0

一、UART串口驱动

➤ 龙芯2K1000串口驱动编写

龙芯2K1000的UART串口驱动与PWM驱动类似，也是通过操控这几个设备文件来驱动串口传输数据。

首先打开设备文件并初始化波特率、数据位数、校验位、停止位等。传入的路径就是/dev/ttyS*。其次就是读写的操作，通过打开的设备文件来读取、写入串口缓存的数据。

```
//打开串口初始化
int open_and_init(const char *path) {
    struct termios cfg;
    speed_t speed;
    int fd;
    memset(&cfg, 0, sizeof(struct termios));
    fd = open(path, O_RDWR);
    if (fd < 0) return -1;
    speed = B115200;
    cfmakeraw(&cfg);
    cfsetispeed(&cfg, speed);
    cfsetospeed(&cfg, speed);

    if (tcsetattr(fd, TCSANOW, &cfg)) {
        printf("tcsetattr fail\n");
        close(fd);
    }
    return fd;
}
```

串口驱动的C代码

一、UART串口驱动

➤ 龙芯2K1000串口驱动编写

龙芯2K1000的UART串口驱动与PWM驱动类似，也是通过操控这几个设备文件来驱动串口传输数据。

首先打开设备文件并初始化波特率、数据位数、校验位、停止位等。传入的路径就是/dev/ttyS*。其次就是读写的操作，通过打开的设备文件来读取、写入串口缓存的数据。

```
//串口读写
void *read_and_echo_port(void *port) {
    int fd = (int) port;
    char buf[1] = {0};
    while (1) {
        ssize_t ret = read(fd, buf, 1);
        if (ret < 0) {
            break;
        } else {
            write(STDOUT_FILENO, buf, 1);
        }
    }
    return NULL;
}
```

串口驱动的C代码

一、UART串口驱动

➤ 龙芯2K串口库pyserial

pyserial模块封装了python对串口的访问，
为多平台的使用提供了统一的接口。

安装：

pip3 install pyserial

```
import serial #导入模块
try:
    #端口, Linux上的/dev/ttyS0 等 或 Windows上的 COM3 等
    portx="/dev/ttyS0"
    #波特率
    bps=115200
    #超时设置,None: 永远等待操作, 0为立即返回请求结果, 其他值为等待超时时间
    timex=5
    # 打开串口, 并得到串口对象
    ser=serial.Serial(portx,bps,timeout=timex)
    # 写数据
    result=ser.write("你好".encode("gbk"))
    print("写总字节数:",result)
    ser.close()#关闭串口
except Exception as e:
    print("---异常---:",e)
```

pyserial实现串口通信

一、IIC 总线与设备驱动

➤ I2C总线

I2C 总线是一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。

主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。

如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件。然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下，主机负责产生定时时钟和终止数据传送。

一、IIC 总线与设备驱动

➤ I2C 协议

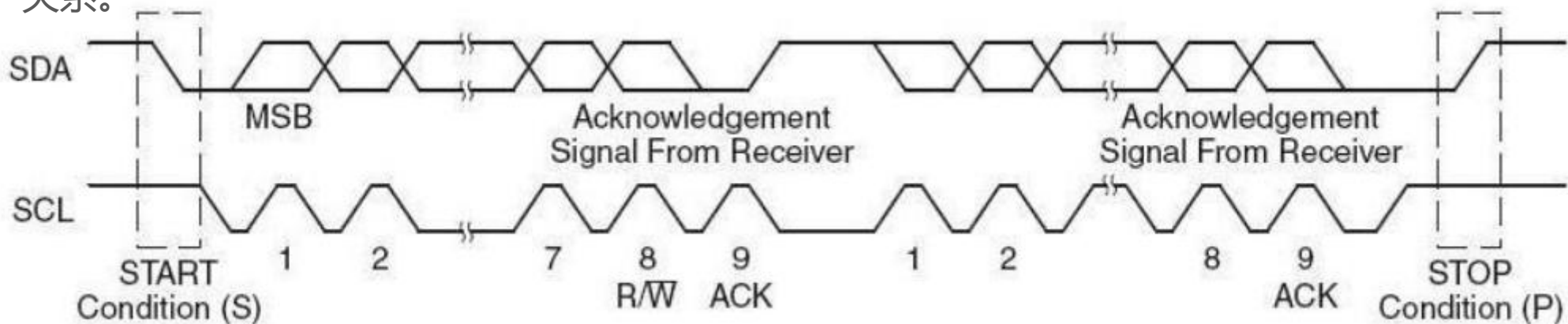
I2C 总线只有两根双向信号线。一根是数据线 SDA，另一根是时钟线 SCL。

SCL：上升沿将数据输入到每个 EEPROM 器件中；下降沿驱动 EEPROM 器件输出数据。（边沿触发）

SDA：双向数据线，为 OD 门，与其它任意数量的 OD 与 OC 门成线与关系。

I2C 总线通过上拉电阻接正电源。当总线空闲时，两根线均为高电平（SDL=1;SCL=1）。

连到总线上的任一器件输出的低电平，都将使总线的信号变低，即各器件的 SDA 及 SCL 都是线“与”关系。



二、I2C驱动

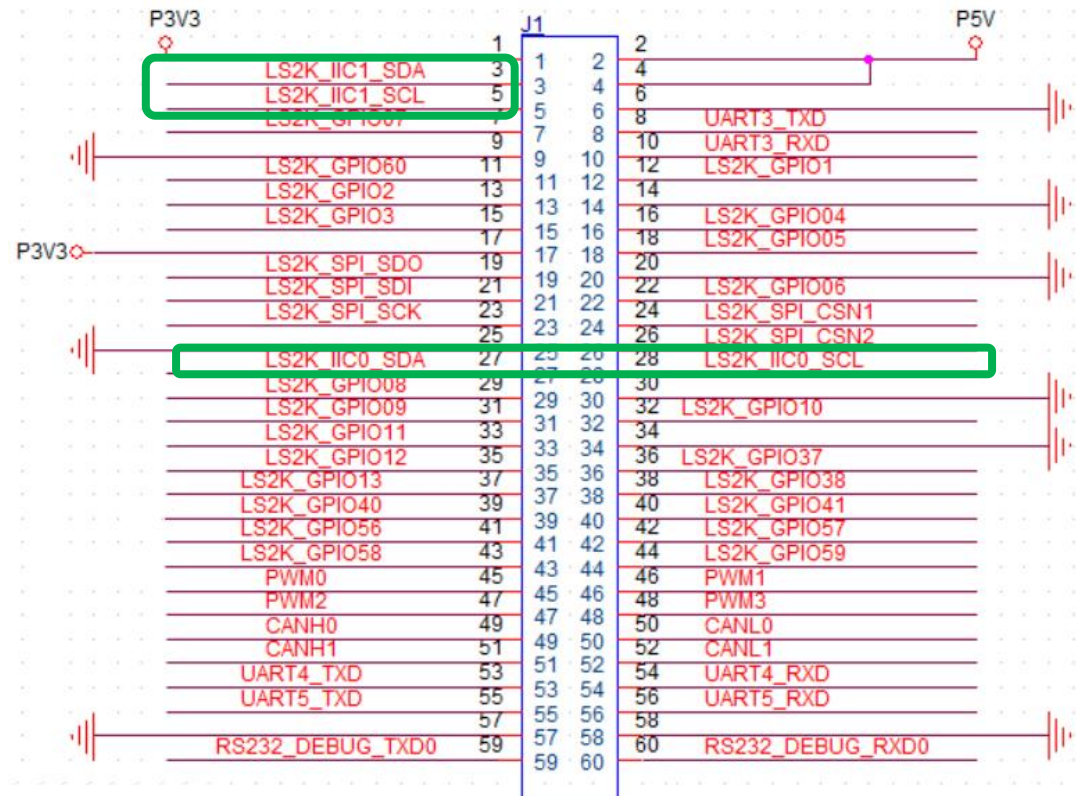
龙芯2K1000的I2C

龙芯2K1000拥有两路I2C接口，用于与连接在总线上的器件之间传输数据。

I2C总线只有两根双向信号线。一根是数据线SDA，另一根是时钟线SCL。

SCL：上升沿将数据输入到每个EEPROM器件中；下降沿驱动EEPROM器件输出数据。（边沿触发）

SDA：双向数据线，为OD门，与其它任意数量的OD与OC门成线与关系。



接口定义

一、I2C驱动

➤ I2C调试工具i2c-tools

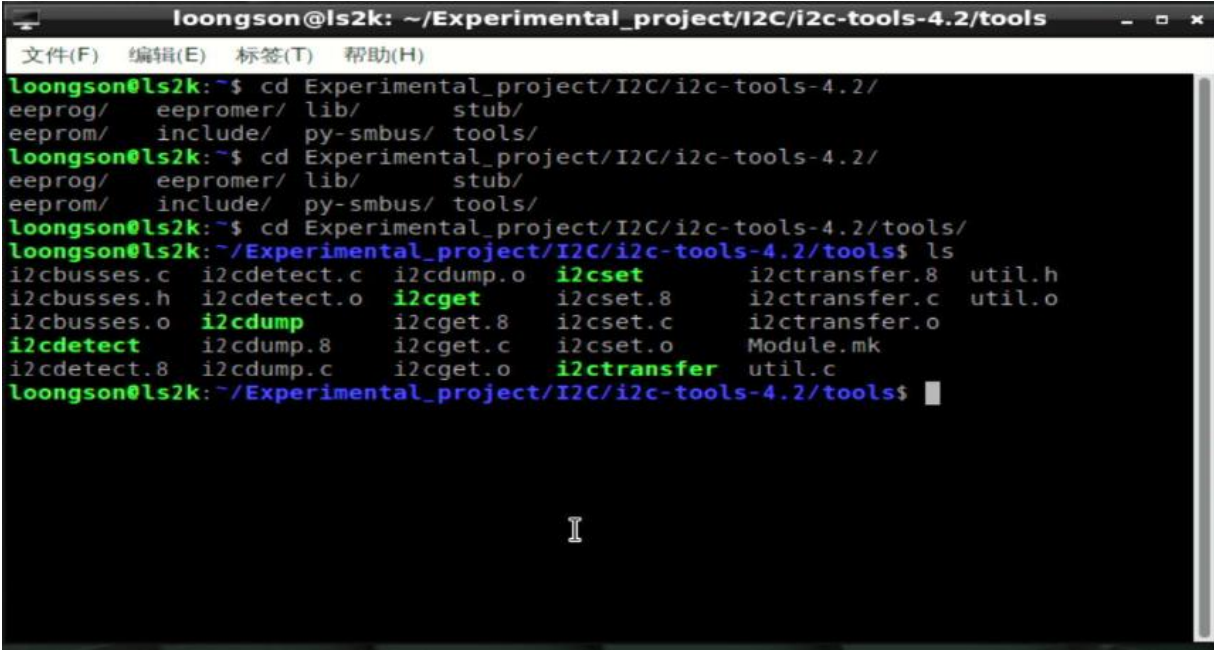
在调试I2C驱动时，可以使用i2c-tools-4.2测试工具来进行测试。下载地址：

<http://sources.buildroot.net/i2c-tools/>

编译：make EXTRA=tools BUILD_STATIC_LIB=1

BUILD_DYNAMIC_LIB=0

编译完成后在tools下生成i2cdetect、i2cdetect、i2cdump、i2cset、i2cget可执行文件。



```
loongson@ls2k: ~/Experimental_project/I2C/i2c-tools-4.2/tools
文件(F) 编辑(E) 标签(T) 帮助(H)
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/
eeprog/ eepromer/ lib/ stub/
eeprom/ include/ py-smbus/ tools/
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/
eeprog/ eepromer/ lib/ stub/
eeprom/ include/ py-smbus/ tools/
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/tools/
loongson@ls2k:~/Experimental_project/I2C/i2c-tools-4.2/tools$ ls
i2cbusses.c i2cdetect.c i2cdump.o i2cset i2ctransfer.8 util.h
i2cbusses.h i2cdetect.o i2cget i2cset.8 i2ctransfer.c util.o
i2cbusses.o i2cdump i2cget.8 i2cset.c i2ctransfer.o
i2cdetect i2cdump.8 i2cget.c i2cset.o Module.mk
i2cdetect.8 i2cdump.c i2cget.o i2ctransfer util.c
loongson@ls2k:~/Experimental_project/I2C/i2c-tools-4.2/tools$
```

i2c-tools-4.2测试工具

一、I2C驱动

➤ I2C调试工具i2c-tools

//检测有几组i2c总线

i2cdetect -l

//检测挂载在i2c-1总线上器件

i2cdetect-r -y 1

//查看i2c-1总线上0x50设备的所有寄存器值

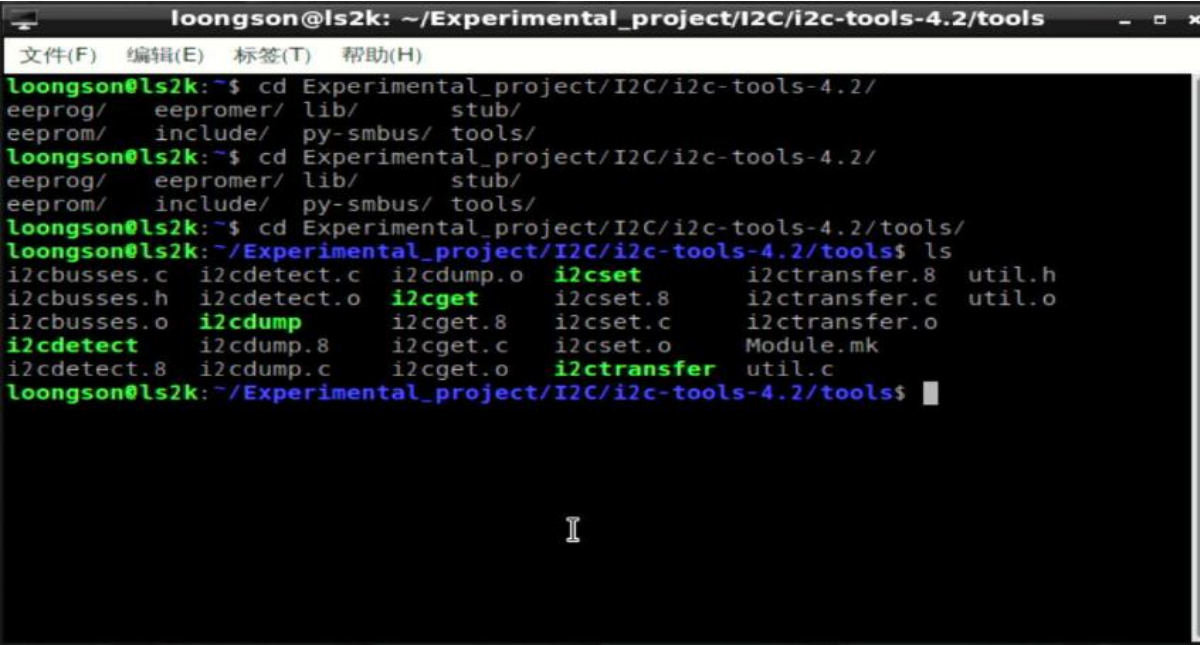
i2cdump -f -y 1 0x50

//往i2c-1总线上0x50设备0x01寄存器写0xaa

i2cset -f -y 0 0x50 0x01 0xaa

//读取i2c-1总线上0x50设备0x01寄存器的值

i2cget -f -y 1 0x50 0x01



```
loongson@ls2k: ~/Experimental_project/I2C/i2c-tools-4.2/tools
文件(F) 编辑(E) 标签(T) 帮助(H)
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/
eeprog/ eepromer/ lib/ stub/
eeprom/ include/ py-smbus/ tools/
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/
eeprog/ eepromer/ lib/ stub/
eeprom/ include/ py-smbus/ tools/
loongson@ls2k:~$ cd Experimental_project/I2C/i2c-tools-4.2/tools/
loongson@ls2k:~/Experimental_project/I2C/i2c-tools-4.2/tools$ ls
i2cbusses.c i2cdetect.c i2cdump.o i2cset i2ctransfer.8 util.h
i2cbusses.h i2cdetect.o i2cget i2cset.8 i2ctransfer.c util.o
i2cbusses.o i2cdump i2cget.8 i2cset.c i2ctransfer.o
i2cdetect i2cdump.8 i2cget.c i2cset.o Module.mk
i2cdetect.8 i2cdump.c i2cget.o i2ctransfer util.c
loongson@ls2k:~/Experimental_project/I2C/i2c-tools-4.2/tools$
```

i2c-tools-4.2测试工具

➤ I2C驱动程序

通过分析i2c-tools里面的源码，可以发现其调用系统I2C接口的文件有：smbus、i2cbusses。

i2cbusses负责初始化I2C，smbus里面则是一些读取写入的函数，看函数名都可以看出来是什么功能。

在自己编写I2C的驱动函数时，只需要先调用i2cbusses文件的open_i2c_dev函数打开I2C接口，然后就可以调用smbus里面的读写函数进行I2C的读写操作了。

```
#ifndef LIB_I2C_SMBUS_H
#define LIB_I2C_SMBUS_H

#define I2C_API_VERSION      0x100

#include <linux/types.h>
#include <linux/i2c.h>

extern __s32 i2c_smbus_access(int file, char read_write, __u8 command,
                             int size, union i2c_smbus_data *data);

extern __s32 i2c_smbus_write_quick(int file, __u8 value);
extern __s32 i2c_smbus_read_byte(int file);
extern __s32 i2c_smbus_write_byte(int file, __u8 value);
extern __s32 i2c_smbus_read_byte_data(int file, __u8 command);
extern __s32 i2c_smbus_write_byte_data(int file, __u8 command, __u8 value);
extern __s32 i2c_smbus_read_word_data(int file, __u8 command);
extern __s32 i2c_smbus_write_word_data(int file, __u8 command, __u16 value);
extern __s32 i2c_smbus_process_call(int file, __u8 command, __u16 value);

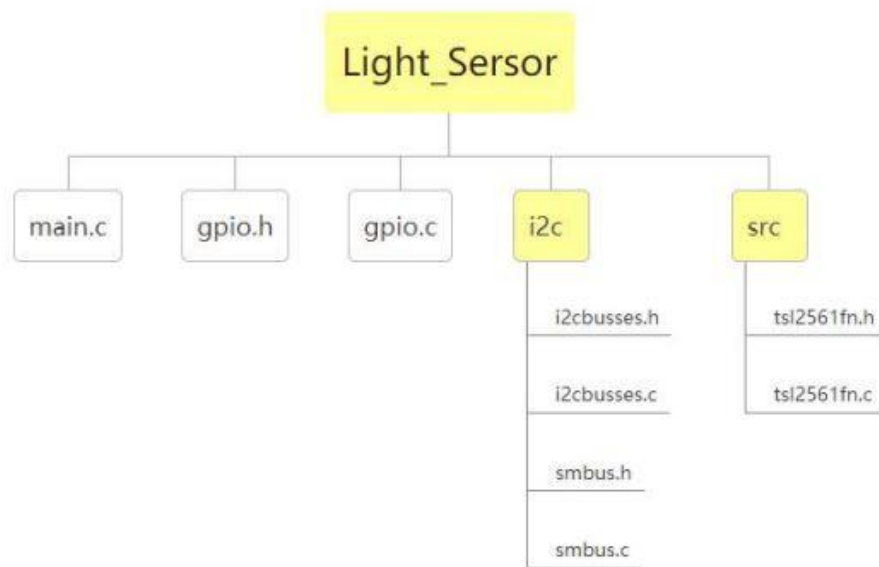
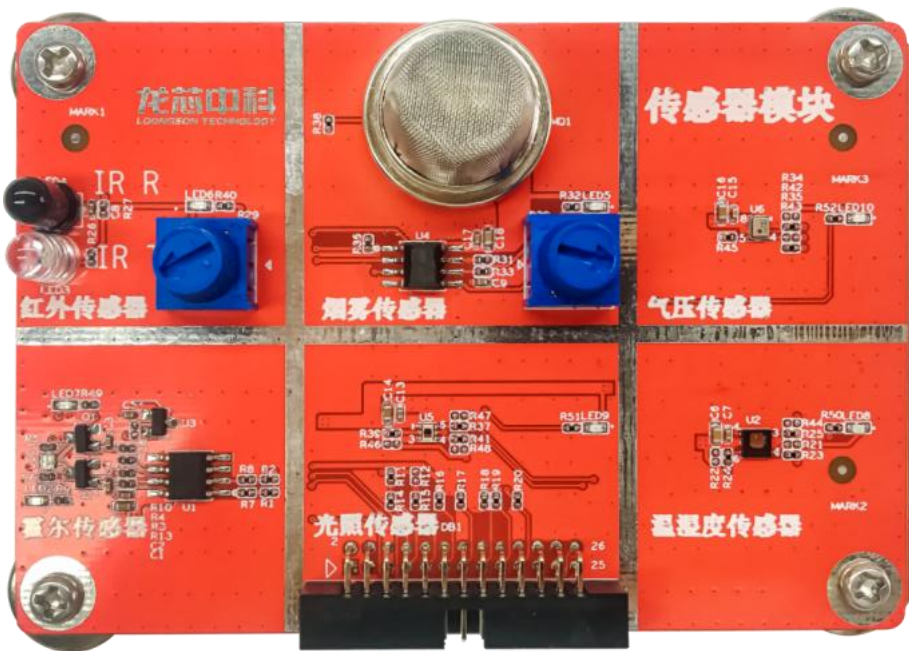
/* Returns the number of read bytes */
extern __s32 i2c_smbus_read_block_data(int file, __u8 command, __u8 *values);
extern __s32 i2c_smbus_write_block_data(int file, __u8 command, __u8 length,
                                         const __u8 *values);
```

smbus里面的读写函数

一、I2C驱动

➤ I2C驱动程序

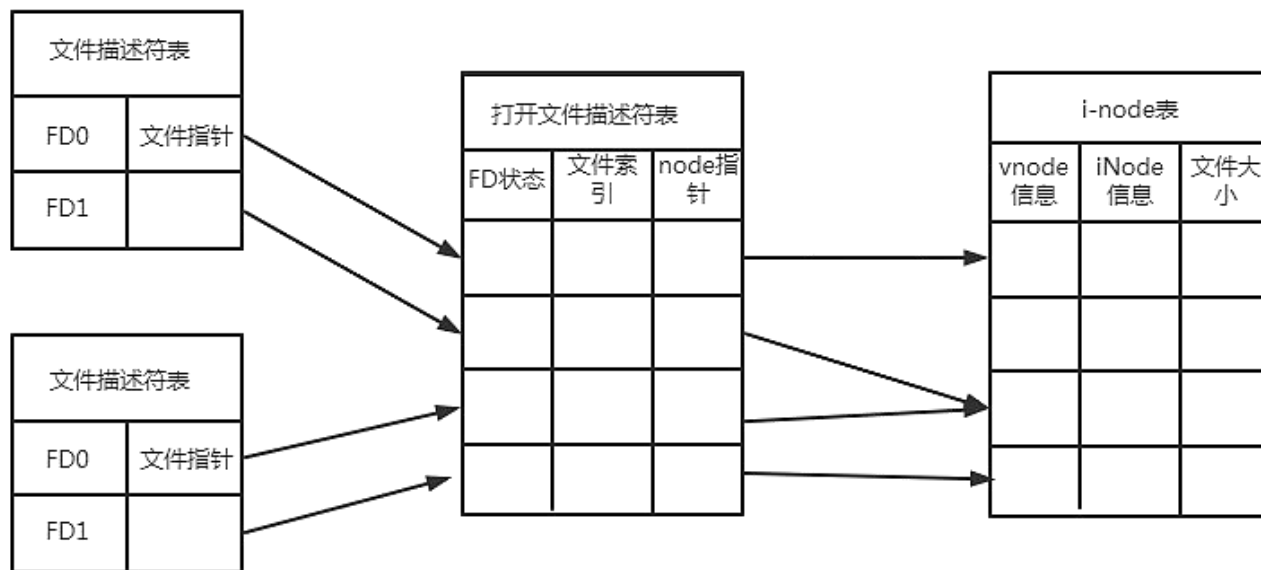
使用 I2C 接口读取传感器模块上的光照传感器的数值。



文件描述符

FD(file descriptor): 文件描述符。一般指的是Linux中一个进程访问文件的唯一标识。当我们在Linux中打开一个文件的时候，都会创建一个文件描述符。每一个文件描述符的出现必然有一个对应的“文件”。不同的文件描述符可能对应同一个文件。即文件描述符和文件的关系是n:1。

系统会为每一个进程维护一个文件描述符表，所有的文件描述符都会在这个表里面创建一个索引。



创建文件描述符

如何创建一个文件描述符，或者说如何打开一个设备文件？这里就需要使用到open()函数，调用后会返回一个文件描述符。如：

```
/*创建并打开文件 */
fd = open("hello.txt", O_CREAT | O_RDWR);
if (fd){
    write(fd, "Hello World", strlen("Hello World")); /*写入字符串 */
    close(fd);
}
```

标志	含义
O_RDONLY	以只读的方式打开文件
O_WRONLY	以只写的方式打开文件
O_RDWR	以读写的方式打开文件
O_APPEND	以追加的方式打开文件
O_CREAT	创建一个文件
O_EXEC	如果使用了O_CREAT而且文件已经存在，就会发生一个错误
O_NONBLOCK	以非阻塞的方式打开一个文件
O_TRUNC	如果文件已经存在，则删除文件的内容

上面的代码创建并打开了一个“hello.txt”文件，将其设置为读写模式，向其写入字符串后，关闭这个字符描述符，也就是关闭了文件。

➤ 文件操作函数

除了上面的open()函数外，我们看到还使用其他函数对文件进行了操作，比较常见的操作函数有：

```
/*文件打开函数*/  
int open(const char *pathname, int flags);  
int open(const char *pathname, int flags, mode_t mode);  
/*文件读写函数*/  
int read(int fd, const void *buf, size_t length);  
int write(int fd, const void *buf, size_t length);  
/*文件指针定位函数*/  
int lseek(int fd, off_t offset, int whence);  
/*文件关闭函数*/  
int close(int fd);
```

➤ C库文件函数

同时，C库提供了另一套操作独立于具体操作系统平台的文件函数。

```
/*文件创建和打开函数*/  
FILE *fopen(const char *path, const char *mode);  
/*文件读写函数*/  
int fgetc(FILE *stream);  
char *fgets(char *s, int size, FILE *stream);  
int fputc(int c, FILE *stream);  
int fputs(const char *s, FILE *stream);  
int fprintf(FILE *stream, const char *format, ...);  
int fscanf(FILE *stream, const char *format, ...);  
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);  
/*文件指针定位函数*/  
int fseek(FILE *stream, long offset, int whence);  
/*文件关闭函数*/  
int fclose(FILE *stream);
```


编写程序

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define LENGTH 100
void main(void){
    int fd, len;
    char str[LENGTH];
    /*创建并打开文件 */
    fd = open("hello.txt", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (fd){
        write(fd, "Hello World", strlen("Hello World")); /*写入字符串 */
        close(fd);
    }
    fd = open("hello.txt", O_RDWR);
    len = read(fd, str, LENGTH); /* 读取文件内容 */
    str[len] = '\0';
    printf("%s\n", str);
    close(fd);
}
```



程序运行结果



文成的可执行文件fd和txt文本



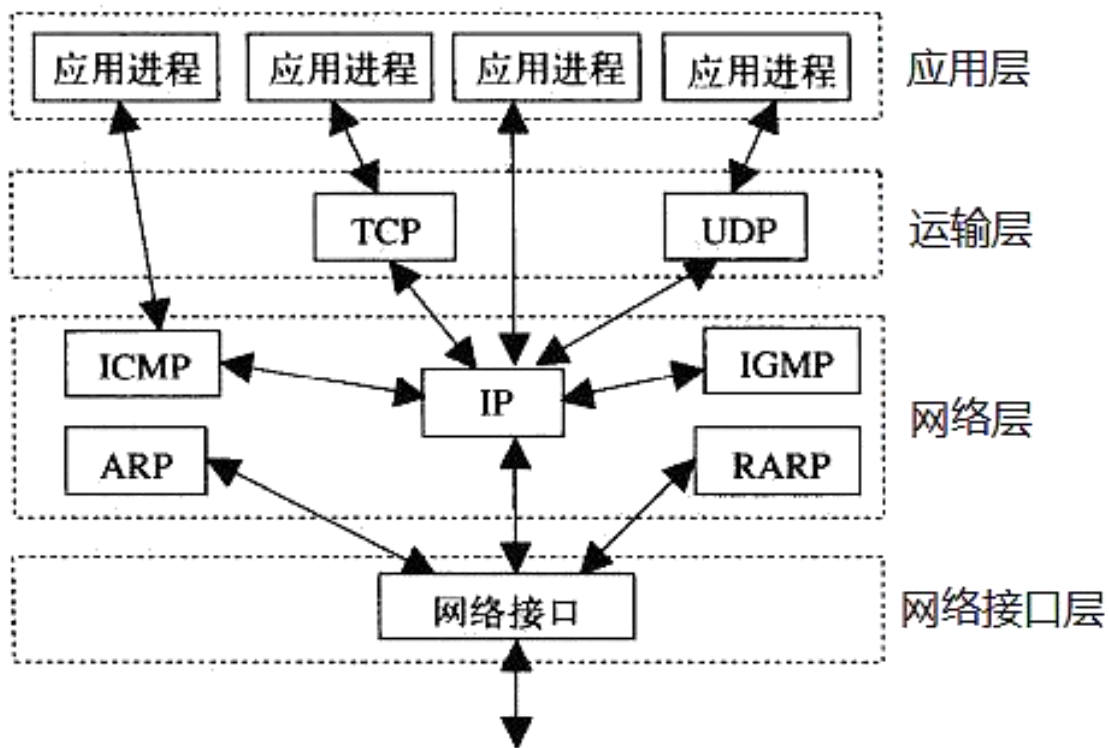
网络编程的基础概念

计算机网络学习的核心内容就是网络协议的学习。网络协议是为计算机网络进行数据交换而建立的规则、标准或者说是约定的集合。因为不同用户的数据终端可能采取的字符集是不同的，两者需要进行通信，就必须要在一定的标准上进行。

计算机网络协议同我们的语言一样，多种多样。而ARPA公司于1977年到1979年推出了一种名为ARPANET的网络协议受到了广泛的热捧，其中最主要的原因就是它推出了人尽皆知的TCP/IP标准网络协议。目前TCP/IP协议已经成为Internet事实上的国际标准。

➤ TCP/IP协议

TCP/IP (Transmission Control Protocol/Internet Protocol, 传输控制协议/网际协议) 是指能够在多个不同网络间实现信息传输的协议簇。TCP/IP协议不仅仅指的是TCP 和IP两个协议, 而是指一个由FTP、SMTP、TCP、UDP、IP等协议构成的协议簇, 只是因为TCP/IP协议中TCP协议和IP协议最具代表性, 所以被称为TCP/IP协议。



➤ TCP/IP协议

应用层：应用层是TCP/IP协议的第一层，是直接为应用进程提供服务的。

(1) 对不同种类的应用程序它们会根据自己的需要来使用应用层的不同协议，邮件传输应用使用了SMTP协议、万维网应用使用了HTTP协议、远程登录服务应用使用了有TELNET协议。

(2) 应用层还能加密、解密、格式化数据。

(3) 应用层可以建立或解除与其他节点的联系，这样可以充分节省网络资源。

运输层：作为TCP/IP协议的第二层，运输层在整个TCP/IP协议中起到了中流砥柱的作用。且在运输层中，TCP和UDP也同样起到了中流砥柱的作用。

网络层：网络层在TCP/IP协议中的位于第三层。在TCP/IP协议中网络层可以进行网络连接的建立和终止以及IP地址的寻找等功能。

网络接口层：在TCP/IP协议中，网络接口层位于第四层。由于网络接口层兼并了物理层和数据链路层所以，网络接口层既是传输数据的物理媒介，也可以为网络层提供一条准确无误的线路。

➤ IP地址、端口与域名

IP地址的作用是标识计算机的网卡地址，每一台计算机都有唯一的IP地址。在程序中是通过IP地址来访问一台计算机的。IP地址具有统一的格式，其长度是32位的二进制数值，4个字节。为了便于记忆，通常化为十进制的整数来表示，如192.168.1.100。在终端输入命令ifconfig，可查看本机的IP地址和MAC（介质访问控制）地址，如图所示。

```
loongson@ls2k:~$ ifconfig
enp0s3f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.124.37  netmask 255.255.255.0  broadcast 192.168.124.255
    inet6 fe80::65d4:c188:dd8f:d6f4  prefixlen 64  scopeid 0x20<link>
    ether 00:01:20:21:22:38  txqueuelen 1000  (Ethernet)
    RX packets 8303  bytes 862170 (841.9 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 348  bytes 41788 (40.8 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
    device interrupt 12

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 94  bytes 7521 (7.3 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 94  bytes 7521 (7.3 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

loongson@ls2k:~$
```

➤ IP地址、端口与域名

端口是指为了标识同一计算机中，不同程序访问网络而设置的编号。每个程序在访问网络时都会分配一个标识符，程序在访问网络或接受访问时，会用这个标识符表示这一网络数据属于这个程序。端口号其实是一个16位的无符号整数(unsigned short)，也就是范围为0~65535。不同编号范围的端口号有不同的作用。低于256的端口是系统保留端口号，主要用于系统进程通信，如 WWW服务使用的是80 号端口，FTP服务使用的是21号端口。不在这一范围内的端口号是自由端口号，在编程时可以调用这些端口号。

域名是用来代替IP地址来标识计算机的一种直观名称，如百度IP地址是180.97.33.108，没有任何逻辑含义，不便于记忆。一般选择 www.baidu.com这个域名来代替IP地址。可以使用命令 ping www.baidu.com来查看该域名对应的IP地址。

➤ 套接字 (socket)

套接字(socket)也叫套接口,在网络中用来描述计算机中不同程序与其他计算机程序的通信方式。人们常说的套接字其实是一种特殊的IO接口,也是一种文件描述符。套接字分为以下3种类型。

(1) 流式套接字 (SOCK_STREAM)

流式套接字提供可靠的、面向连接的通信流。它使用TCP,从而保证了数据传输的正确性和顺序性。

(2) 数据报套接字 (SOCK_DGRAM)

数据报套接字定义了一种无连接的服务,数据通过相互独立的报文进行传输,是无序的,并且不保证是可靠、无差错的。它使用UDP协议。

(3) 原始套接字

原始套接字允许对底层协议(如IP或ICMP)进行直接访问,其功能强大但使用较为不便,主要用于一些协议的开发。

➤ 创建套接字 (socket)

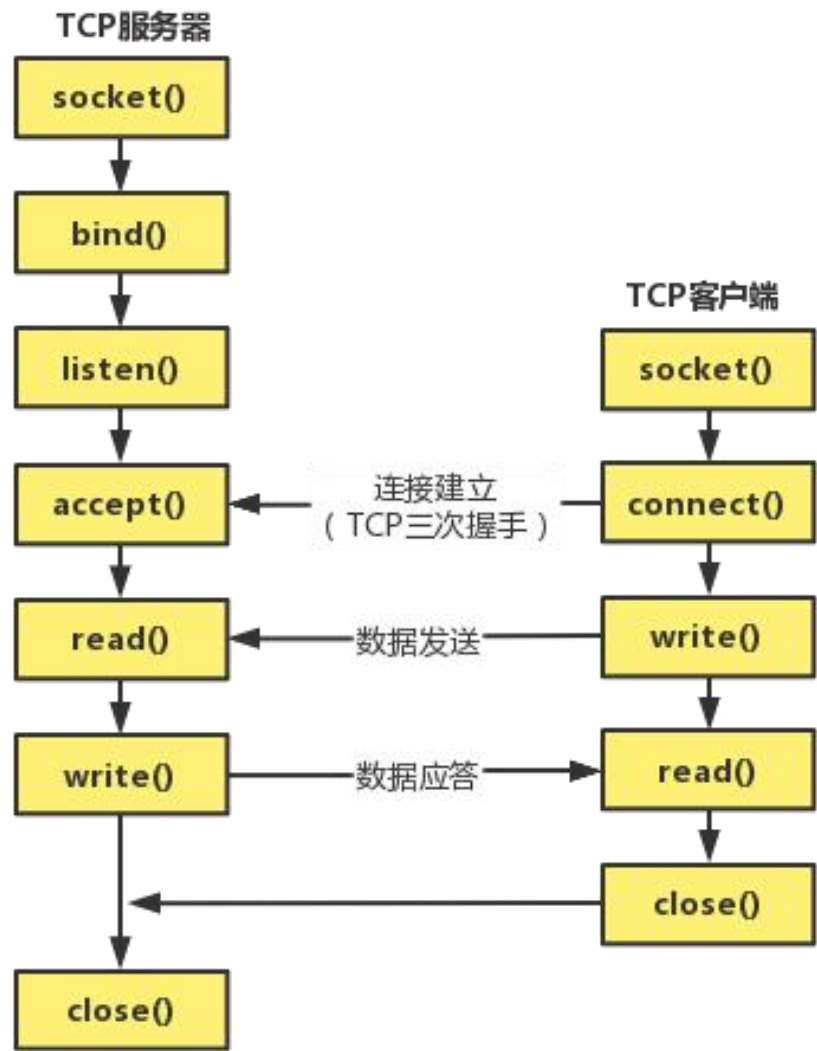
由3个参数构成:IP地址、端口号和传输层协议,以区分不同应用程序进程间的网络通信与连接。套接字也有一个类似于打开文件的函数调用,该函数返回一个整型的套接字描述符,随后建立的连接、数据传输等操作都是通过描述符来实现的。

```
/*创建套接字 socket()*/  
#include <sys/types.h>  
#include <sys/socket.h>  
int socket(int domain, int type, int protocol);
```

```
/* 套接字地址,每个套接字域都有其自己的地址格式 sockaddr */  
/* AF_UNIX 域地址格式 */  
struct sockaddr_un {  
    sa_family_t sun_family;  
    char        sun_path[];  
};  
/* AF_INET 域地址格式 */  
struct sockaddr_in {  
    short int sin_family;  
    unsigned short int sin_port;  
    struct in_addr sin_addr;  
};  
/* in_addr 结构体 */  
struct in_addr {  
    unsigned long int s_addr;  
};
```


➤ TCP/IP通信简单实现

TCP 编程实例分为服务器端(server) 和客户端(client), 其中服务器端首先建立起 socket, 接着绑定本地端口, 建立与客户端的联系, 并接收客户端发送的消息。而客户端则在建立socket 之后, 调用connect函数来与服务器端建立连接, 连接后, 调用send函数发送数据到服务器端。





TCP服务端代码

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<string.h>
int main(){
    //创建套接字
    int serv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    //初始化socket元素
    struct sockaddr_in serv_addr;
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    //serv_addr.sin_addr.s_addr = inet_addr("192.168.124.37");
    serv_addr.sin_port = htons(1234);
    //绑定文件描述符和服务器的ip和端口号
    bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    printf("serve start:127.0.0.1\n");
    //进入监听状态,等待用户发起请求
    listen(serv_sock, 20);
```

```
//接受客户端请求
//定义客户端的套接字,这里返回一个新的套接字,后面通信时,就用这个clnt_sock进行通信
struct sockaddr_in clnt_addr;
socklen_t clnt_addr_size = sizeof(clnt_addr);
int clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
//接收客户端数据,并相应
char str[256];
read(clnt_sock, str, sizeof(str));
printf("client send: %s\n",str);
//向客户端返回信息
char ACK[] = "hello+ACK";
write(clnt_sock, ACK, sizeof(ACK));
//关闭套接字
close(clnt_sock);
close(serv_sock);
return 0;
```

```
}
```

➤ TCP客户端代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>
int main(){
    //创建套接字
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    //服务器的ip为本地，端口号1234
    struct sockaddr_in serv_addr;
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(1234);
    //向服务器发送连接请求
    connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    //发送并接收数据
    char buffer[40];
    printf("Please write:");
```

```
    scanf("%s", buffer);
    write(sock, buffer, sizeof(buffer));
    read(sock, buffer, sizeof(buffer) - 1);
    printf("Serve send: %s\n", buffer);
    //断开连接
    close(sock);
    return 0;
```

```
}
```

四、实战案例：城市环境检测系统设计

社会服务	卓越工科工程师联合培养		嵌入式边缘计算职业认证		国产化信创专业人才输出	
技术支持	智慧城市运维	城市数据可视化	无人计费系统	智能监控系统	无人驾驶系统	语音问答系统
	智能充电桩系统	智慧灯杆系统	智慧停车系统	智慧管网系统	智慧设施系统	共享单车系统
	智慧城市产学研创基地（智慧路灯杆在智慧城市网络节点中的产业实践）					
	嵌入式	物联网	移动互联网	云计算	大数据	人工智能
应用设施	智慧灯杆		智慧设施		智慧交通	
	智慧路灯	充电桩	水电管网	消防栓	智能车	红绿灯
	气象站	广告屏	井盖	垃圾桶	交通标志	ETC
	监控	共享单车	路边停车
基础设施	IaaS管理平台		AI算力平台		IoT云平台	
	服务器、网络	数据大屏	Wi-Fi基站		IoT基站	边缘网关

四、实战案例：城市环境检测系统设计

- 智慧基站**
异构网关基站支持ZigBee、LoRa、Wi-Fi、NB-10T等传感网设备接入。
- 智慧无线**
利用路灯灯杆快速建设城市免费的Wi-Fi系统，实现城市路网的全网覆盖。
- 智慧广告**
建设公共信息发布平台，提供广告推送服务和公共区域的消毒防疫服务。
- 城市巡检**
利用路灯灯杆作为路网定位节点，实现巡检、求助、报警等治安保障服务。



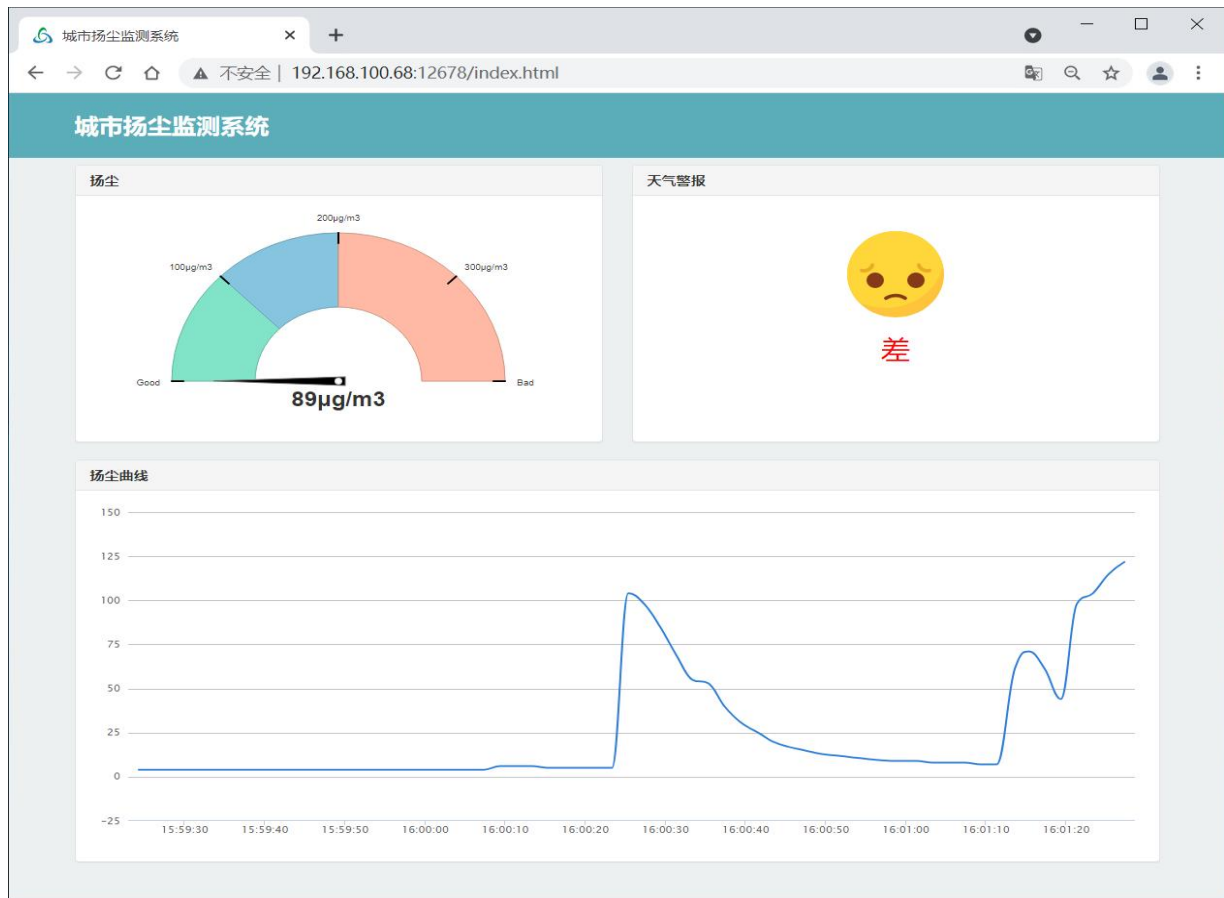
- 智慧照明**
对路灯进行统一智能调光、智能管控、功耗监测，实现供电系统的节能。
- 智慧气象**
建设城市传感网系统对温湿度、PM2.5、城市大气环境等进行无缝监测。
- 安防监控**
加快建设和完善公共安全监控，为城市每个角落提供更加安全的居住环境。
- 智慧充电**
高效利用路灯电网资源，提供电动汽车/城市路网的无人值守充电计费。



四、实战案例：城市环境检测系统设计



四、实战案例：扬尘监测系统实验设计



实验目标

- 掌握Boa WebServer的移植
- 掌握简单的CGI代码设计
- 基于龙芯派设计扬尘监测系统

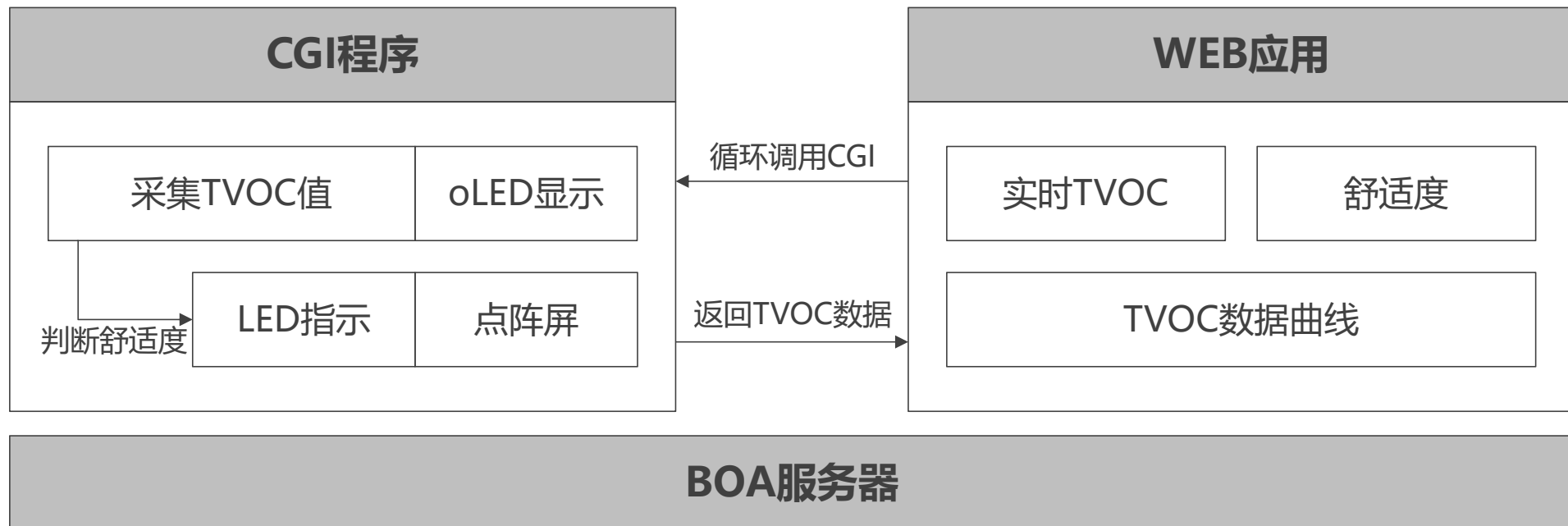
实验设备

- 龙芯教育派
- 龙芯教育派扩展板 (TVOC、数码管、oLED)

实验内容

- 实时显示当前TVOC数据
- 曲线显示TVOC历史数据
- 根据TVOC值进行环境舒适度报警
- oLED实时显示当前TVOC数据
- LED、数码管指示环境舒适度指数

四、实战案例：扬尘监测系统设计方案



➤ BOA服务器

- BOA服务器是一个小巧高效的web服务器，是一个运行于unix或linux下的，支持CGI的、适合于嵌入式系统的单任务的http服务器
- BOA支持CGI，能够为CGI程序创建一个进程来执行相应的客户请求。
- Common Gateway Interface，简称CGI，在物理上是一段程序，运行在服务器上，提供与客户端HTML页面的交互接口。CGI是外部应用程序（CGI程序）与WEB服务器之间的接口标准，是在CGI程序和Web服务器之间传递信息的过程，CGI程序使网页具有交互功能。
- CGI应用程序可以由大多数的编程语言编写，比如： C、C++、Python、Linux Shell等。

四、实战案例：BOA 服务移植

- BOA服务有开源网站进行维护：www.boa.org，可以下载进行嵌入式移植
- 下载源码包，进行编译：

```
$ cd ~/exp/tvoc_test/boa/src  
$ make
```

- CGI程序代码：

```
#!/bin/bash  
cd /home/loonson/www/cgi-bin  
echo "Content-Type:text/html;charset=utf-8 "  
echo  
lsmod | grep pcf8951 > /dev/null 2>&1 || sudo insmod pcf8951.ko  
val=`cat /sys/bus/i2c/devices/0-0048/channel01`  
sudo ./oledTest TVOC:$val  
if [ $val -lt 30 ]  
then  
    echo $val 1  
    sudo ./ledsTest 7  
    sudo ./led8x8Test 7  
fi
```

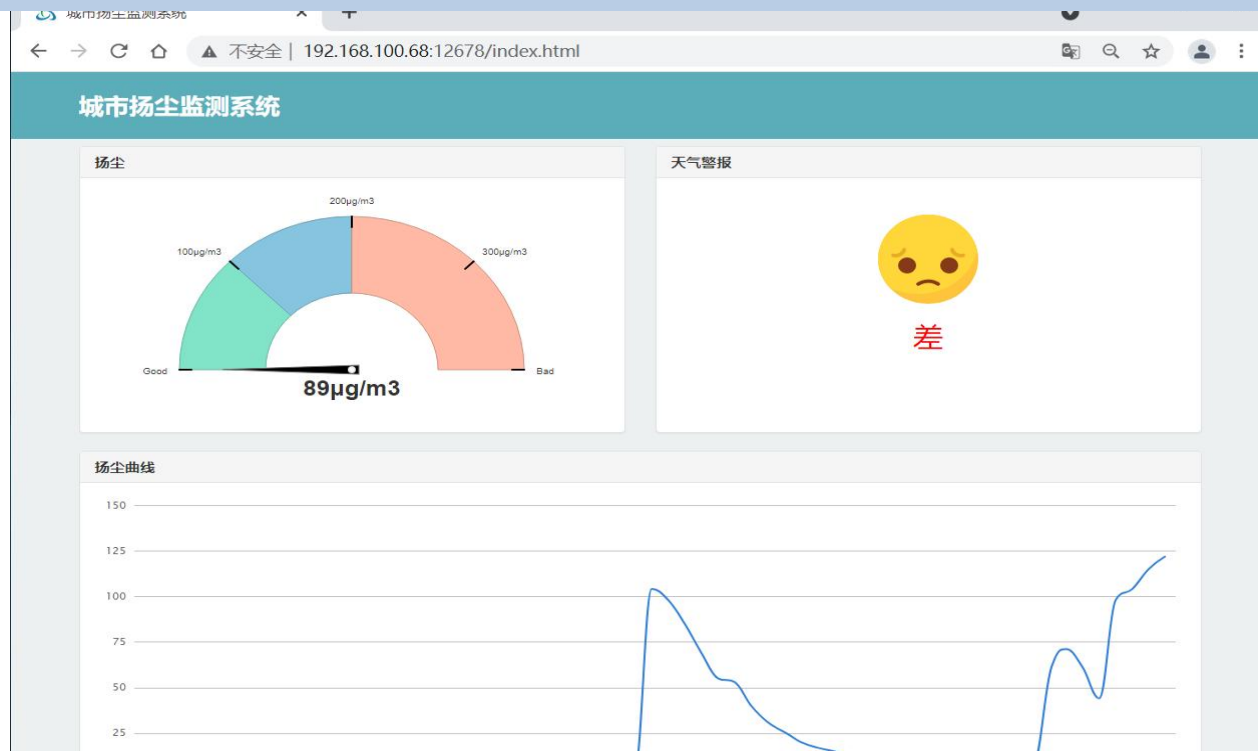
四、实战案例：BOA 服务部署

- 将tvoc_test文件夹复制到龙芯教育派~/exp/目录下。
- 将编译好的boa、boa.conf文件拷贝到龙芯教育派~/exp/tvoc_test目录下。
- 执行命令部署BOA服务并启动：

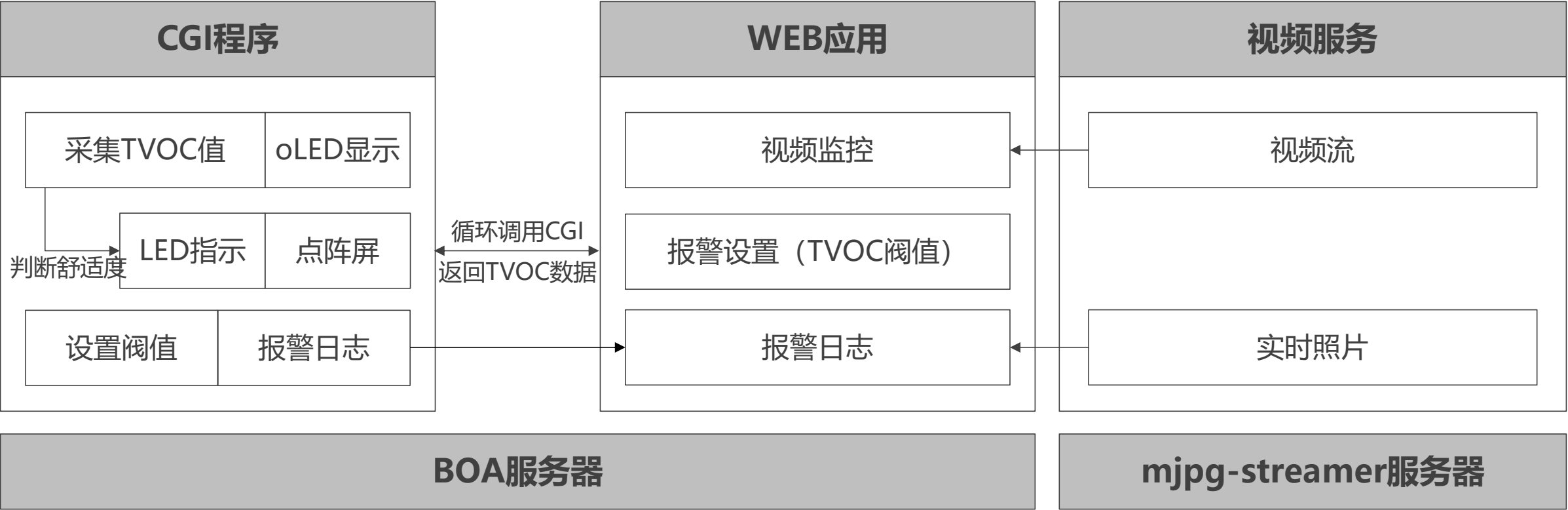
```
$ cd ~/exp/tvoc_test/
$ sudo mkdir /etc/boa/
$ sudo cp -r ./boa.conf /etc/boa/
$ sudo chmod 777 /etc/boa/boa.conf
$ sudo mkdir /var/log/boa
$ sudo chmod 777 /var/log/boa
$ sudo cp -r ./www ~/
$ chmod 777 ~/www/cgi-bin/*
$ sudo ./boa
[12/Jul/2021:07:56:57 +0000] loongson@ls2k:~/exp/tvoc_test$ boa: server version Boa/0.94.13
[12/Jul/2021:07:56:57 +0000] boa: server built Jul 9 2021 at 17:26:14.
[12/Jul/2021:07:56:57 +0000] boa: starting server pid=7549, port 12678
```

四、实战案例：扬尘监测系统应用测试

通过浏览器打开boa应用页面：<http://192.168.100.68:12678/index.html>
(地址为龙芯教育派的IP地址)



四、实战案例：安防监控系统设计思路



四、实战案例：mjpg-streamer

- mjpg-streamer是一款免费基于IP地址的视频流服务器，通过摄像头读取视频数据，并以视频流的方式传输到浏览器访问。<https://github.com/jacksonliam/mjpg-streamer.git>
- mjpg-streamer主要的插件模块如下：
 - input_uvc.so：此文件调用USB摄像头驱动程序V4L2，从摄像头读取视频数据。
 - input_control.so：这个文件实现对摄像头转动的控制接口。
 - output_http.so：这是一个功能齐全的网站服务器，它不仅可以从单一文件夹中处理文件，还可以执行一定的命令，它可以从输入插件中处理一幅图像，也可以将输入插件的视频文件根据现有M-JPEG标准以HTTP视频数据服务流形式输出。
 - output_file.so：这个插件的功能是将输入插件的JPEG图像存储到特定的文件夹下，它可以用来抓取图像。

四、实战案例：mjpg-streamer移植

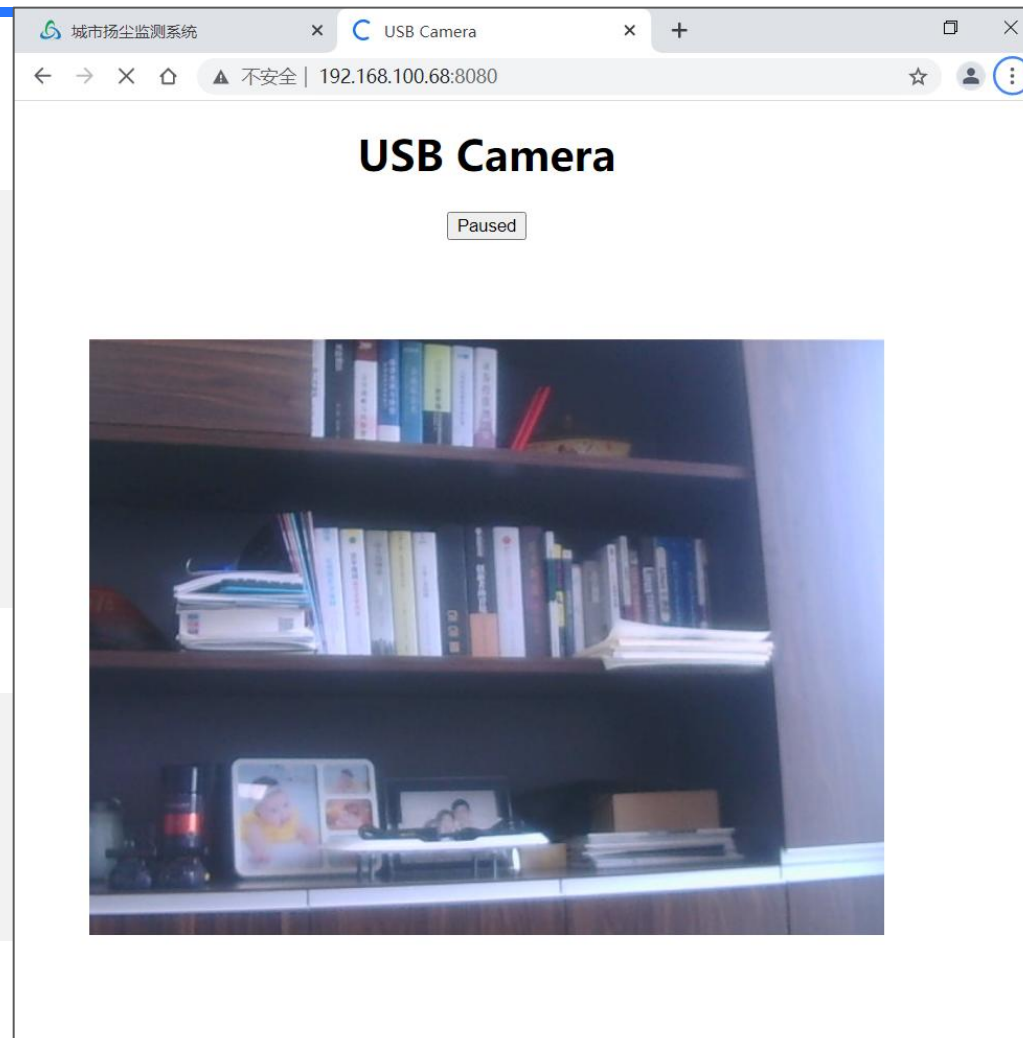
- mjpg-streamer依赖于libjpeg，需要先编译
- 编译libjpeg库：

```
$ export PATH="/opt/gcc-4.9.3-64-gnu/bin:$PATH"
$ cd ~/gw2k1000/exp/04-monitor_test/mjpg-streamer
$ tar xvf libjpeg-master.tgz
$ cd libjpeg-master
$ mkdir -p _build && cd _build
$ cmake -DCMAKE_INSTALL_PREFIX=./install ..
$ make & make install
```

- 编译mjpg-streamer：

```
$ cd ~/gw2k1000/exp/04-monitor_test/mjpg-streamer
$ tar xvf mjpg-streamer-master.tgz
$ cd mjpg-streamer-master
$ make
```

- 测试mjpg-streamer：



实验操作：编译mjpg-streamer服务并测试

四、实战案例：安防监控系统应用设计

- 编译和部署BOA服务器。
- 编写安防监控程序monitor。
- 编写CGI程序实时获取TVOC数据。
- 编写CGI程序设置TVOC舒适度阈值。
- 编写CGI程序记录超标的TVOC数据及现场图片，并保存报警日志。
- 部署安防监控系统Web应用。
- 启动mjpg-streamer服务，启动BOA服务，运行monitor程序。

```
$ cd ~/exp/04-monitor_test
$ sudo cp -r ./web/www ~/
$ chmod 777 ~/www/cgi-bin/*
$ sudo ./boa &
$ cd ~/exp/04-monitor_test/mjpg-streamer/output
$ sudo ./mjpg_streamer -i "./input_uvc.so" -o "./output_http.so -w ." &
$ sudo ~/www/cgi-bin/monitor.sh
```

四、实战案例：安防监控系统应用测试

通过浏览器打开boa应用页面：<http://192.168.100.68:12678/index.html>
(地址为龙芯教育派的IP地址)



实验操作：部署安防监控系统应用并测试

- 1、接口开发的示例（传感器，重要）
- 2、应用开发示例(文件、线程等)
- 3、网络编程（客户端和服务端）
- 4、BOA和mjpg-streamer服务器移植部署

感谢观看！