基本使用

2021年3月22日 21:41

1.

cd	路径跳转 (cd .. : 返回上一级目录 , cd / : 返回根目录 , "." : 当前目录 , ".." : 上一级目录)
ls / ll	显示当前目录所有文件 (ll : 显示所有文件并显示详细信息)
sudo su root	切换最高权限
pwd	显示当前目录
./xxx	运行可执行文件
mkdir	在当前目录创建一个目录
touch	创建文件 , 可更改文件的时间戳
xxx | grep xxx	重定向管道查询关键字 (例如: ps -ef | grep docker : 查询包含docker关键字的进程)
cp 需复制的文件名 复制的文件名	复制文件
mv	移动文件到指定路径 , 重命名
rm	删除文件 (rm -r : 删除文件夹及其包含的所有文件)

2. 编辑器的使用:
   a. vim
      i. 使用:
         1) dd : 剪切光标所在行
         2) n + dd : 剪切光标所在行往下n行
         3) p : 粘贴
         4) n + yy : 复制n行
         5) V : 进入可视模式
         6) 自动对齐的方法:
            a) 按V进入可视模式,选中需对齐的行按=
            b) gg=G : 全文自动缩进
         7) 插入:
            a) o : 光标下一行开始插入编辑
            b) i : 光标开始前位置插入编辑
            c) a : 光标开始后位置插入
            d) esc : 退出插入模式
         8) 底行模式:
            shift+:
            a) w : 保存
            b) q : 退出
            c) q! : 强制退出
            d) wq : 保存并推出
            e) x : 保存并推出
            f) set mouse = a : 开启鼠标操作
            g) set mouse -= a : 不使用鼠标
            h) vsp 文件路径 : 横向展开多窗口文件
            i) ctrl + ww : 使光标在多个窗口间跳转
         9) gcc 文件名 : 生成可执行文件 a.out , ./a.out : 运行生成的可执行文件就

3. C语言基础
    a. 屏蔽代码段
        i. #if boolean 代码段 #endif (若boolean=0,则代码段被屏蔽,否则不屏蔽)

```
#if 0
int main(){...}
#endif
```

main函数不会执行

C语言基础

2021年7月20日 13:57

一. 控制语句

1. 选择语句
   a. `if ( expression ) { codes1 } else { codes2 }`
      如果expression == true , 则执行codes1 , codes2不执行;
      如果expression == true , 则执行codes2 , codes1不执行.

   b. `switch ( expresssion ) { case condition1 : codes1 ; break ; case condition2 : codes2 ; break ; … }`
      如果expression == condition1 , 则执行codes1 , 若无break,则出现语句穿透现象,后续语句也会执行 , 有break后续语句不会执行 ;
      如果expression == condition2 , 则执行codes2 ….

2. 循环语句
   a. `while ( expression ) { codes }`
      如果expresssion条件成立 , 则执行codes , 一般codes中包含使循环终止的条件.

   b. `for ( expression1 ; condition ; expresssion2 ) { codes }`
      先执行expression1进行条件初始化 , 然后执行condition进行判断 , 若成立则执行codes 后执行expression2 (expression2一般包含循环退出条件) , 若condition不成立直接退出循环.

   c. `do { codes } while ( expression );`
      先执行一次codes内容 , 其他同while.

3. 代码示例:

```
#include "static/controlStatement.h"
#include <iostream>
using namespace std;

void controlStatement(){
    int num;
    cout<<"Please input a num..."<<endl;
    cin>>num;
    for(int i = 0 ; i < num ; i++){
        for (int j = 0; j < i+1; ++j) {
            printf("* ");
        }
        cout<<endl;
    }
}
```

二. 数组

1. 数组的定义
   a. 数据类型 变量名[] = { ele1 , ele2 , ele3… };
   b. 数据类型 变量名[num]; //数组内元素个数就是数组大小=num

2. 数组的访问
   a. 通过下标访问 :
      例: int arr[] = { 1 , 2 , 3 , 4 , 5 };
      int num = arr[0];
      则num等于1.

   b. 通过指针访问
      例: int* p = arr;
      int num = *(p+1);
      则num等于2.

3. 数组的大小/长度
   a. sizeof关键字
      sizeof(arr) : 获取数组在内存中占用的字节大小
      sizeof(arr[0]) : 获取数组第一个元素在内存中占用的字节大小

则数组长度为:

sizeof(arr)/sizeof(arr[0])

4. for循环遍历数组元素

5. 字符数组
   a. char arr[] = { 'a' , 'b' , 'c' };
   b. 字符串是一个字符数组,结尾有/0是一个结束标志
   c. gets() : 输入字符串
   d. puts() : 打印字符数组
   e. strlen() : 获取字符串长度,以/0为结束标志
   f. strcmp(str1 , str2) : 比较两个字符串,返回一个整形数据
   g. 代码示例:

```
void strFunction(){
    char a[20]="Hello";
    char b[20]="World";
    strcat(a,b);
    puts(a);
    strcpy(a,b);
    puts(a);
    printf("%d\n",strlen(b));
    if(0==strcmp(a,b))
        printf("Same");
    else if(strcmp(a,b)>0)
        printf("a is bigger");
    else
        printf("b is bigger");
}
```

6. 代码示例:

```
#include "static/arrayLearning.h"
#include <iostream>
using namespace std;

void test1(){
    int arr[] = {1,2,3,4,5};
    int* p = arr;
    cout<<*(p+1)<<endl;
    cout<<sizeof(arr)/sizeof(arr[0])<<endl;
}

void bubbleSort( int arr[] , int arrLength ){
    for (int i = 0; i < arrLength; i++) {
        for (int j = i+1; j < arrLength; j++) {
            if (arr[i]<arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    for (int i = 0; i < arrLength; ++i) {
        cout<<arr[i]<<endl;
    }
}

void maxNumOfArray(){
    int arrLength;
    cout<<"Please input the length of array..."<<endl;
    cin>>arrLength;
    int arr[arrLength];
    cout<<"Please input the elements of array..."<<endl;
    for (int i = 0; i < arrLength; ++i) {
        printf("The %d element is :",i);
        cin>>arr[i];
    }
    cout<<"The elements by sorted :"<<endl;
    bubbleSort(arr , arrLength);
}

void arrayLearning(){
    maxNumOfArray();
}
```

三. 本日总结

今天回顾了C语言的相关基础知识 : 控制语句与数组以及一些字符串函数 .

在控制语句的学习中重点学习了循环语句while和for的使用,并进行了相关实例代码的实现,如:在控制台打印三角形,99乘法表和烧烤点菜,有点难度但也十分有趣,在此次学习过程中,我对循环嵌套有了一个清晰的认识并对for语句的理解更上一层楼,受益匪浅.

数组的学习过程中遇到了数据与结构课程中学习的排序算法:冒泡排序,我利用两层for语句嵌套虚拟出两个指针用以比较大小并完成排序操作,在老师的教导中顺利地写出了代码,

C语言高级

2021年7月21日 9:02

一. 指针

1. 概述:
   a. *地址:*
      i. 是内存空间中计算机对内存的编号
      ii. 指针是用来保存地址信息的

   b. *指针大小:*
      i. 64位系统8字节，32位系统4字节（映射内存全部地址，同cache)

   c. *定义:*
      i. 数据类型 *变量名 （如:int *p , char *q ）

   d. *解引用:*
      i. 取出指针指向的值（如: int a = 10 ; int *p = &a ; 则 *p的值为10 ）

   e. *野指针*
      i. 指针游离或访问未授权区域
      ii. 空指针 : 有指向但指向为空

2. 使用:
   a. *指针运算*
      i. 例子:

```
int a[10] = {1,2,3,4,5,6,7} ;
int *p = a ;
printf("a[4] = %d\n" , a[4]);
printf("&a[0] = %p\n" , &a[0]);
printf("a = %p\n" , a);
printf("a+1 = %p\n" , a+1);
printf("p = %p\n" , p);
printf("p+1 = %p\n" , p+1);
```

      输出结果:

```
&a[0] == p == a , a+1 == p+1
```

      ii. 指针的运算以该指针数据类型的大小为基本单位, 如: int *p : +1表示偏移四个字节单位 ; char *p : +1表示偏移一个字节单位

   b. *示例代码:*
      i. 冒泡排序

```
#include <stdio.h>
void bubbleSort(int *p , int len){
    for(int i = 0 ; i < len ; i++){
        for(int j = i+1 ; j < len ; j++){
            if(*(p+i) < *(p+j)){
                int temp = *(p+i);
                *(p+i) = *(p+j);
                *(p+j) = temp;
            }
        }
    }
}
void main(){
    int arr[10] = {2,3,5,6,77,55,99,34,56,34};
    bubbleSort(arr,10);
    for(int i = 0 ; i<10 ; i++){
        printf("%d ",arr[i]);
    }
}
```

3. 高级:
   a. *常量指针与指针常量*

i. 常量指针:

1) const int *p : 指针指向值不可更改,指向可更改

2) 示例:

```
int num1 = 10;
int num2 =20;
const int *p = &num1;
*p = 30; //error
p = &num2; //allow
```

ii. 指针常量:

1) int const *p : 指针指向不可更改,指向值可更改

2) 示例:

```
int num1 = 10;
int num2 = 20;
int const *p = &num1;
*p = 30; //allow
p = &num2;//error
```

iii. 引用 (C++内容)

1) 引用声明时必须初始化

2) 本质 : 引用本质为指针常量

3) 示例:

```
int a = 10;
int &b = a;
```

(b的值与地址都与a相同,其值为10,同int const *b,相当于起别名)

二. 函数

1. 概述:

a. *函数定义格式:*

i. 返回值数据类型 函数名 (形式参数...) { codes... }

ii. 返回值类型:

1) char
2) short
3) int
4) float
5) long
6) double
7) void

b. *函数传值:*

i. 值传递

1) 形参改变不会影响实参的值,原因是形参属于局部变量,存储在栈区,函数调用完会被自动释放
2) 函数调用时,将实参的值复制一份给形参,他们的地址是不同的

ii. 地址传递

1) 形参为地址,通过传入指针实现,因此会操作同一份数据,形参改变实参也会改变
2) 形参的地址与实参的地址不同(指针地址不同,但指针保存的值也就是地址是一样的)

2. 内存模型:

a. *代码区:*

i. 保存代码片段的区域

b. *全局区 :*

i. 变量类型

1) 全局变量
2) 全局常量
3) 静态变量

ii. 共享的区域,所有位置都可以操作

c. *栈区:*

i. 变量类型

1) 形参
2) 局部变量
3) 局部常量

ii. 独立的区域,属于每一个方法,空间由编译器自动管理,外部不能操作与访问

d. 堆区:

i. 由编程人员管理的一段内存区域,需手动分配与释放

3. 头文件(分布式):

三.代码

1. 指针与函数的运用实现冒泡排序,统计字符数组中数字与字母个数

```
#include <stdio.h>
void bubbleSort(int *p , int len){
    for(int i = 0 ; i < len ; i++){
        for(int j = i+1 ; j < len ; j++){
            if(*(p+i) < *(p+j)){
                int temp = *(p+i);
                *(p+i) = *(p+j);
                *(p+j) = temp;
            }
        }
    }
    for(int i = 0 ; i<len ; i++){
        printf("%d ",*(p+i));
    }
    printf("\n");
}
void countOfChar(char *arr){
    int numCount,letterCount;
    int i = 0;
    while(*(arr+i) != '\0'){
        if((*(arr+i)>='0')&&(*(arr+i)<='9')){
            numCount++;
        }
        if((*(arr+i)>65 && *(arr+i)<91) || (*(arr+i)>97 && *(arr+i)<123)){
            letterCount++;
        }
        i++;
    }
    printf("numCount = %d , letterCount = %d",numCount,letterCount);
}
void main(){
    int arr1[10] = {2,3,5,6,77,55,99,34,56,34};
    bubbleSort(arr1,10);

    char arr2[5];
    printf("Please input char array...\n");
    gets(arr2);
    countOfChar(arr2);
}
```

四.本日总结

本日所学内容为指针与函数,属于C语言高级编程,是C语言学习过程中的一大难点,涉及到内存空间的分配与回收等相关概念.但是因为在本学期开设的操作系统课程中学习过存储器的管理等内容,已事先了解了内存编址,内存的连续分配与离散分配(分页分段以及虚拟存储器)等内容,所以在今日的课程学习过程中,还算顺利.我利用指针将冒泡排序与统计数字等操作封装在函数中,然后再从主函数中调用,使得主函数清晰简洁,各个操作也显得条理清晰,使得我充分认识到了指针与函数的便捷性与强大.

文件IO

2021年7月22日 9:38

1. 文件定义:

a. 文件是一组有序数据的集合
b. Linux系统下一切皆为文件,对文件的操作需要有一个标识文件的符号称为 : 文件描述符
    i. 文件描述符: 用来描述文件的非负整数
        1) 0---标准输入
        2) 1---标准输出
        3) 2---标准错误输出

2. 相关库函数

a. 函数:
    i. close() :
        1) 关闭某个文件描述符
        2) 需添加的库函数---<unistd.h>

    ii. open() :
        1) 打开某个文件描述符
        2) 需添加的头文件---<sys/stat.h>,<sys/types.h>,<fcntl.h>
        3) 函数原型:
            a) int open(const char *path , int flags)
            b) int open(const char *path , int flags , mode_t mode)
            c) 参数说明:

                i) 返回值: 打开的文件描述符
                One. -1---文件打开失败
                Two. 3...---文件打开成功

                ii) 形参:
                One. path---文件路径
                Two. flags---打开文件的权限(O_RDONLY , O_RDWR , O_WDONLY, O_CREAT) //只读,读写,只写,创建
                Three. mode---创建文件给予的权限

                open("文件名" , O_RDWD | O_CREAT , 664);

                First. 注:
                1. 传入一个八进制数 (以0开头的数) , 用以表示创建这个文件给予的权限,一般给664.

                2. 补充知识:
                    1. 每一个文件有三组权限:创建者,同组者,其他
                    2. 例子:
                        1. -rw-rw-r-- 表示这是一个文件,创建者与同组者持有读写权限,其他人只有读权限.
                            1. 二进制表示为 : 110 110 100
                            2. 八进制为 : 664
                3. unix系统更改文件访问权限命令为 : chmod 八进制数 文件名 .

    iii. read():
        1) 将文件读取到缓冲区
        2) 需添加的头文件---<unistd.h>
        3) 函数原型
            a) ssize_t read(int fd , void *buf , size_t count);
            b) 参数说明:

i) 返回值 : 成功读取到的字节数 , 错误时返回-1
ii) 形参:
One. fd---传入需要读取文件的文件描述符
Two. buf---缓冲区,数组
Three. count---每次读取的字节数

iv. write():
1) 将数据从缓冲区写入到文件
2) 需添加头文件---<unistd.h>
3) 函数原型:
a) ssize_t write(int fd , const void *buf , size-t count)
b) 参数说明:
i) 返回值 : 成功为写入的字节数,错误返回-1
ii) 形参:
One. fd---传入需要写入文件的文件描述符
Two. buf---缓冲区
Three. count---每次写入的字节数

v. lseek():
1) 移动文件指针
2) 函数原型:
a) off_t lseek(int filedes, off_t offset, int whence);
b) 参数说明:
i) 返回值 : 成功为指针偏移量 =,错误返回-1
ii) 形参:
One. filedes---文件描述符
Two. offset---指定指针偏移量
Three. whence:
First. SEEK_SET---文件开头
Second. SEEK_END---文件结尾

vi. perror("提示信息名"):
1) 打印错误信息

3. 数据流:

a. 指程序与数据的交互是以流的形式进行的
b. 文件存取时,先进行打开文件操作,即打开数据流
c. 关闭文件即关闭数据流

4. 缓冲区:

a. 指程序执行时所提供的额外内存,可用来暂存准备执行的数据,用以提高存取效率,通过数组实现
b. 文件存取操作分为,设置缓冲区与不设置缓冲区

5. 文件存取流程:

a. 打开文件: (程序从文件读取数据)
i. 磁盘---将文件复制到缓冲区--->缓冲区---从缓冲区读取数据--->程序
b. 关闭文件:(程序将数据写入到文件)

i. 程序---写入数据到--->缓冲区---将缓冲区数据写入到磁盘--->磁盘

6. 文件类型:

a. *文本文件:*

i. 文件内容以字符编码方式保存

b. *二进制文件:*

i. 文件内容以二进制形式保存,将内存中数据原封不动保存至文件中,适用于非字符为主数据,存取速度快,占用空间小,可随机读写

7. 课程代码:

a. main.c

```
#include "static/file_copy.h"

int main(intargs,char*argr[]){
file_copy(argr[1],argr[2]);
return0;
}
```

b. file_copy.c

```
#ifndef INTELLIGENSE_HOUSEKEEPER_FILE_COPY_H
#define INTELLIGENSE_HOUSEKEEPER_FILE_COPY_H
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int file_copy( const char *path1 , const char *path2 );

#endif//INTELLIGENSE_HOUSEKEEPER_FILE_COPY_H
```

c. file_copy.c

```
#include "static/file_copy.h"

int file_copy( const char *path1 , const char *path2 ){
    printf("------------copy start--------------");
    int fd1,fd2;
    int ret;

    fd1 = open(path1 , O_RDONLY);
    if(fd1<0){
        perror("openFile1Error");
        return -1;
    }
    fd2 = open(path2 , O_RDWR);
    if(fd2 < 0){
        perror("openFile2Error");
        return -1;
    }

    int len = lseek(fd1,0,SEEK_END);
    lseek(fd1,0,SEEK_SET);

    char buf[len];
    ret = read(fd1 , buf , len);
    if(ret<0){
        perror("readFileError");
     return -1;
    }

    ret = write(fd2 , buf , len);
    if(ret<0){
     perror("writeFileError");
        return -1;
    }
```

```
    if(fd1>0) close(0);
  if(fd2>0) close(1);

}
```

网络编程

2021年7月23日 8:55

一. 协议

1. *TCP (Transfer Control Protocol) : 传输控制协议*
   a. 面向连接 (三次握手,四次挥手)
   b. 基于字节流 (块传输,超时重传)
   c. 稳定可靠

2. *UDP (User Datagram Protocol) : 用户数据报协议*
   a. 无连接 (不可靠)
   b. 延迟低
   c. 数据传输效率高

二. 相关概念

1. *Socket : 套接字*
   a. 定义:

      所谓套接字(Socket)，就是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象。一个套接字就是网络上进程通信的一端，提供了应用层进程利用网络协议交换数据的机制。从所处的地位来讲，套接字上联应用进程，下联网络协议栈，是应用程序通过网络协议进行通信的接口，是应用程序与网络协议根进行交互的接口.

   b. 分类:
      i. *流式套接字 (SOCK_STREAM)*

         流套接字用于提供面向连接、可靠的数据传输服务。该服务将保证数据能够实现无差错、无重复送，并按顺序接收。流套接字之所以能够实现可靠的数据服务，使用TCP协议.

      ii. *数据报套接字 (SOCK_DGRAM)*

         数据报套接字提供一种无连接的服务。该服务并不能保证数据传输的可靠性,数据有可能在传输过程中丢失或出现数据重复，且无法保证顺序地接收到数据。数据报套接字使用UDP(User DatagramProtocol)协议进行数据的传输。由于数据报套接字不能保证数据传输的可靠性，对于有可能出现的数据丢失情况，需要在程序中做相应的处理.

      iii. *原始套接字 (SOCK-RAW)*

         原始套接字与标准套接字(标准套接字指的是前面介绍的流套接字和数据报套接字)的区别在于：原始套接字可以读写内核没有处理的IP数据包，而流套接字只能读取TCP协议的数据，数据报套接字只能读取UDP协议的数据。因此，如果要访问其他协议发送的数据必须使用原始套接.

2. 套接字工作流程

   要通过互联网进行通信，至少需要一对套接字，其中一个运行于客户端，我们称之为 Client Socket，另一个运行于服务器端，我们称之为 Server Socket.

   a. *服务器监听*
   b. *客户端请求*
   c. *连接确认*

三. 编程

1. 网络编程所需函数
   a. *socket():*
      i. 创建套接字
      ii. 函数原型:
         1) int socket(int domain , int type , int protocol)
         2) 需添加头文件 --- <sys/socket.h>
         3) 参数说明:
            a) 返回值 : 文件描述符,失败返回-1
            b) 形参:
               i) domain --- 选择通信协议族 (本次选 AF_INET : IPv4 Internet protocols)
               ii) type --- 选择套接字类型 (选TCP协议就选流式套接字 : SOCK_STREAM)
               iii) protocol --- 选择协议 (编译器可以通过前两个参数推断出使用何种协议 ,这里可填 0)

   b. *bind():*
      i. 绑定ip地址和端口号
      ii. 函数原型:
         1) int bind(int sockfd , const struct sockaddr *addr , socklen_t addrlen)
         2) 所需添加的头文件 --- <netinet/in.h>,<netinet/ip.h>,<arpa/inet.h>
         3) 参数说明:
            a) 返回值
            b) 形参:

i) sockfd --- 套接字文件句柄(文件描述符)
ii) addr --- 初始化ip地址和端口号的结构体
iii) addrlen --- 结构体大小

c. 套接字相关结构体
i. struct sockaddr:
1) 结构体原型:
a) struct sockaddr{
sa_family_t sin _family; //协议族
char sa_data[14]; //14字节,包含套接字中的目标地址和端口信息
};
2) 缺陷 : 将目标地址与端口信息混在一起
3) 使用struct sockaddr_in结构体强转,,不直接使用

ii. struct sockaddr_in:
1) 结构体原型:
a) struct sockaddr_in{
sa_family_t sin_family; //协议族,这里使用IPv4即 : AF_INET
unit16_t sin_port; //16位TCP/UDP端口号
struct in_addr sin_addr; //32位IP地址
};
b) struct in_addr{
In_addr_t s_addr; //32位IPv4地址
};

d. listen():
i. 服务端开启监听
ii. 函数原型
1) int listen(int sockfd , int backlog)
2) 参数说明:
a) 返回值 : 文件描述符,错误返回-1
b) 形参:
i) sockfd --- 套接字文件描述符
ii) backlog --- 最大同时可连接的数量

e. accept():
i. 服务端开启接收
ii. 函数原型
1) int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
2) 参数说明:
a) 返回值 : 文件描述符,错误返回-1
b) 形参:
i) sockfd --- 套接字文件描述符
ii) addr --- 初始化ip地址与端口信息
iii) addrlen --- 结构体大小的地址

f. connect():
i. 客户端请求连接
ii. 函数原型
1) int connect(int sockfd , const struct sockaddr *addr , socklen_t addrlen);
2) 参数说明:
a) 返回值 : 0代表成功,-1代表失败

四.代码

1. networkProgramming.c

```c
#include "static/networkProgramming.h"
int main(){
    int listenFd = server_init();
    int connFd = server_wait_connect(listenFd);
    char buff1[10];
    char buff2[1024*1024];
    while(1){
        read(connFd , buff1 , sizeof(buff1));
        printf("buf = %s\n",buff1);
        if(strcmp(buff1,"file") == 0){
            printf("---------Transport--------\n");
            int fd = open("./server_init.c" , O_RDWR);
```

```
            int size = lseek(fd , 0 , SEEK_END);
            lseek(fd , 0 , SEEK_SET);
            read(fd , buff2 , size);
            write(connFd , &size , 4);
            write(connFd , buff2 , size);
        }else if(strcmp(buff1 , "pic")==0){
            int pic_fd;
            pic_fd = open("./pic.png" , O_RDWR);
            int pic_size = lseek(pic_fd , 0 ,SEEK_END);
            lseek(pic_fd , 0 ,SEEK_SET);
            read(pic_fd , buff2 , pic_size);
            write(connFd , &pic_size , 4);
            write(connFd , buff2 , pic_size);
            int ret = write(connFd , buff2 , pic_size);
            printf("ret = %d\n" , ret);
        }else{
            write(connFd , buff1 , sizeof(buff1));
        }
    }
}
```

2. server_init.c

```
#include "static/networkProgramming.h"

int server_init(){
    int listenFd;
    //1. 创建监听套接字
    listenFd = socket(AF_INET , SOCK_STREAM , 0);//利用socket函数创建一个监听套接字,函数返回值为这个套接字的文件描述符
    if(listenFd < 0){perror("server_init->socket");return -1;}//如果创建失败则函数返回-1
    //2. 绑定ip地址和端口号
    struct sockaddr_in ser_addr;//创建一个用来保存服务器ip和端口号的结构体
    memset(&ser_addr , 0 , sizeof(ser_addr));//清空结构体
    //结构体赋值
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_port = htons(8888);
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret;
    ret = bind(listenFd , (const struct sockaddr*)&ser_addr , sizeof(ser_addr));//利用bind函数完成服务器ip地址和端口号的绑定
    if(ret < 0){ perror("server_init->bind");return -1;}
    //3.开启监听
    ret = listen(listenFd , 16);//利用listen函数开启监听操作
    if(ret < 0){ perror("server_init->listen");return -1;}
    printf("server_init->success\n");
    return listenFd;//返回创建成功的监听套接字的文件描述符
}
int server_wait_connect(int listenFd){
    int connFd; //连接文件描述符,通过accept函数赋值
    struct sockaddr_in con_addr;
    memset(&con_addr , 0 ,sizeof(con_addr));
    socklen_t len = sizeof (con_addr);
    connFd = accept(listenFd , (const struct sockaddr*)&con_addr , &len);
    if(connFd < 0){ perror("server_wait_connect");return -1;}
    printf("connect->success\n");
    return connFd;
}
```

3. client_init.c

```
#include "../static/networkProgramming.h"
#include <stdlib.h>
int main(int argc , char *argv[]){
    int sock;
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if(sock<0){
        perror("client_sock");
        return -1;
    }
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr =  inet_addr(argv[1]);
    int ret;
    ret = connect(sock , (const struct sockaddr*)&addr , sizeof(addr));
    if(ret<0){perror("client_connect");return -1;}
    char cmd_buf[10];
    char recv_buf[10];
    char buf[1024*1024];
    int size;
    char *p = buf;
    ret = 0;
    while(1){
        printf("cmd_buf: %s" , cmd_buf);
        fgets(cmd_buf , 10 , stdin);
        if(strcmp(cmd_buf,"file")==0){
            write(sock , cmd_buf , sizeof(cmd_buf));
```

```
            read(sock , &size , 4);
            read(sock , buf , size);
            puts(buf);
        }else if(strcmp(cmd_buf , "pic")==0){
            write(sock , cmd_buf , sizeof(cmd_buf));
            read(sock , &size , 4);
            printf("size = %d\n" , size);
            while(ret < size){
                ret += read(sock , p+ret , size);
                printf("ret = %d\n" , ret);
            }
            int fd = open("pic.png" , O_RDWR|O_CREAT , 0664);
            ret = write(fd , buf , size);
            printf("ret = %d\n" , ret);
            system("eog pic.png");
        }else{
                write(sock , cmd_buf , sizeof(cmd_buf));
                read(sock , recv_buf , sizeof(recv_buf));
                printf("recv : \n%s" , recv_buf);
        }
    }
}
```

4. *networkProgramming.h*

```
#ifndef INTELLIGENSE_HOUSEKEEPER_NETWORKPROGRAMMING_H
#define INTELLIGENSE_HOUSEKEEPER_NETWORKPROGRAMMING_H
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <memory.h>
#include <stdio.h>
//server
int server_init();
int server_wait_connect(int listenFd);


#endif//INTELLIGENSE_HOUSEKEEPER_NETWORKPROGRAMMING_H
```

四. 总结

今天学习了网络编程,并结合了昨天学习的文件IO操作,难度相对来讲提升了一个层次,颇具难度和挑战性.在去年上个学期,学习了java的TCP/UDP网络编程,令我庆幸的是与今日所学有异曲同工之妙,总归是将学习的难度消减一些,只是多个函数的使用颇为复杂,需要时间的锤炼,方能掌握的牢固一点.总的来讲,今日之所学,令我收益颇丰,但也使我感受到了挑战性,我会再接再厉的.

QT

2021年7月26日 18:34

一. 概述

QT是一款跨平台开发的C++图形用户界面应用程序开发框架.既可以开发GUI程序,也可以用于开发非GUI程序,比如控制台工具和服务器.QT是面向对象的框架,使用特殊的代码生成扩展(称为元对象编译器(Meta Object Compiler , moc)) 以及一些宏 , QT很容易扩展 , 并且允许真正的组件编程.

1. qt项目构建步骤
   a. 新建一个项目,选中桌面应用程序
   b. 填写项目名和创建路径
   c. 构建系统可选取
      i. Qmake:
         1) 能够自动生成MakeFile专案文件(.pro),不管源代码是否是QT写的,都能使用qmake,因此qmake能用于很多软件的构建过程.
         2) 手写Makefile是比较困难而且容易出错，尤其在进行跨平台开发时必须针对不同平台分别撰写Makefile，会增加跨平台开发复杂性与困难度。qmake会根据专案文件（.pro）里面的信息自动生成适合平台的 Makefile。开发者能够自行撰写专案文件或是由qmake本身产生。qmake包含额外的功能来方便 Qt 开发，如自动的包含moc 和 uic 的编译规则。

      ii. Cmake:

         CMake是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装(编译过程)。他能够输出各种各样的makefile或者**project文件，能测试编译器所支持的C++特性,类似UNIX下的automake。只是 CMake 的组态档取名为 CMakeLists.txt。Cmake 并不直接建构出最终的软件，而是产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++ 的 projects/workspaces），然后再依一般的建构方式使用。这使得熟悉某个集成开发环境（IDE）的开发者可以用标准的方式建构他的软件，这种可以使用各平台的原生建构系统的能力是 CMake 和 SCons 等其他类似系统的区别之处。**

      iii. Qbs
   d. 填写类信息
      i. 本次基类选取Qwidgt,其他不变
   e. 后续步骤默认

二. 具体代码方案

1. 在项目工程文件(.pro或CMakeList.txt)中添加网络模块
   a. .pro
      i. 首行添加 QT += core gui network
   b. CMakeList
      i. 添加如下代码
         1) 添加模块 : find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Core Network Gui REQUIRED)
         2) 目标链接库 :

```
target_link_libraries(connect PRIVATE
    Qt${QT_VERSION_MAJOR}::Widgets
    Qt${QT_VERSION_MAJOR}::Core
    Qt${QT_VERSION_MAJOR}::Gui
    Qt${QT_VERSION_MAJOR}::Network)
```

2. 在 widget.h 中添加头文件
   a. ---#include <QHostAddress>---
      i. 提供填写ip地址的函数
   b. ---#include <QTcpSocket>---
      i. 网络通信所用套接字
   c. ---#include <QPixmap>---
      i. 传输图片
   d. ---#include <QTimer>---
      i. 定时器

3. widget.h文件中的类Widget
    a. private slots
        i. 声明槽函数

    b. private中填写私有字段

4. widget.cpp文件中实现槽函数与各种界面绘制后的动作
    a. 构造函数
        Widget::Widget(QWidget *parent) : QWidget(*parent*) , ui(new Ui::Widget)
        可创建套接字,绑定槽与信号的连接等一系列前置操作

    b. 实现槽函数如void Widget::on_pushButton_clicked()

5. .ui文件中设计界面

三 .实现代码

1. 功能 : 连接服务端,获取图片并循环展示
2. 项目结构:
    a. main.cpp
    b. widget.cpp
    c. widgt.ui
    d. widget.h
    e. CMakeList.txt

3. 主要具体代码
    a. CMakeList.txt

```
cmake_minimum_required(VERSION 3.5)

project(connect VERSION 0.1 LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# QtCreator supports the following variables for Android, which are identical to qmake Android variables.
# Check https://doc.qt.io/qt/deployment-android.html for more information.
# They need to be set before the find_package( ...) calls below.

#if(ANDROID)
#   set(ANDROID_PACKAGE_SOURCE_DIR "${CMAKE_CURRENT_SOURCE_DIR}/android")
#   if (ANDROID_ABI STREQUAL "armeabi-v7a")
#      set(ANDROID_EXTRA_LIBS
#         ${CMAKE_CURRENT_SOURCE_DIR}/path/to/libcrypto.so
#         ${CMAKE_CURRENT_SOURCE_DIR}/path/to/libssl.so)
#   endif()
#endif()

find_package(QT NAMES Qt6 Qt5 COMPONENTS Widgets REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Widgets REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Core Network Gui REQUIRED)

set(PROJECT_SOURCES
      main.cpp
      widget.cpp
      widget.h
      widget.ui
)

if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)
   qt_add_executable(connect
      MANUAL_FINALIZATION
      ${PROJECT_SOURCES}
   )
else()
   if(ANDROID)
```

```
        add_library(connect SHARED
          ${PROJECT_SOURCES}
        )
    else()
        add_executable(connect
          ${PROJECT_SOURCES}
        )
    endif()
endif()

target_link_libraries(connect PRIVATE
    Qt${QT_VERSION_MAJOR}::Widgets
    Qt${QT_VERSION_MAJOR}::Core
    Qt${QT_VERSION_MAJOR}::Gui
    Qt${QT_VERSION_MAJOR}::Network)

set_target_properties(connect PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER my.example.com
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}
    MACOSX_BUNDLE_SHORT_VERSION_STRING ${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}
)

if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(connect)
endif()
```

b. widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QHostAddress>
#include <QPixmap>
#include <QTcpSocket>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void my_connect();
    void pic_show();
    void cmd_send();

    void on_pushButton_3_clicked();

private:
    Ui::Widget *ui;
    QTcpSocket *my_sock;
    QTimer *my_timer;
    int pic_len;
    char pic_buf[1024 * 1024];
    int flag;
};
#endif // WIDGET_H
```

c. widget.cpp

```
#include "widget.h"
#include "./ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    my_sock = new QTcpSocket;
    my_timer = new QTimer;
      //界面初始化后开始连接服务器
      //my_sock->connectToHost(QHostAddress("192.168.192.140"), 8888);
    ui->lineEdit->setText("192.168.192.140");

    //绑定信号与槽
```

```
    connect(my_sock, SIGNAL(connected()), this, SLOT(my_connect()));
    connect(my_sock, SIGNAL(readyRead()), this, SLOT(pic_show()));
    connect(my_timer, SIGNAL(timeout()), this, SLOT(cmd_send()));
    flag=0;

}

Widget::~Widget()
{
    delete ui;
}


void Widget::on_pushButton_clicked()
{
     my_sock->connectToHost(QHostAddress(ui->lineEdit->text()), 8888);
}


void Widget::on_pushButton_2_clicked()
{
    my_timer->start(1000);
}

void Widget::on_pushButton_3_clicked()
{
    my_timer->stop();
}

void Widget::cmd_send(){
    char buf[10] = "pic";
    my_sock->write(buf , sizeof(buf));
}

void Widget::my_connect(){
     ui->lineEdit->setText("my_connect->success");
}



void Widget::pic_show() {
    if (flag == 0) {
        my_sock->read((char *) &pic_len, 4);
        flag = 1;
    } else if (flag == 1) {
        if (my_sock->bytesAvailable() < pic_len) return;
        my_sock->read(pic_buf, pic_len);

        QPixmap pic;
        pic.loadFromData((uchar *) pic_buf, pic_len);
        pic = pic.scaled((ui->label->size()));
        ui->label->setPixmap(pic);
        flag=0;
    }
}
```

智能管家项目

2021年7月27日 14:04

一. 项目组成

1. QT用户界面客户端:
   a. 通信模块：通过套接字实现
   b. 语音识别：调用百度智能云SDK
   c. 摄像头与环境信息展示：通过接收服务器传送回的数据实现

2. 服务端
   a. 多线程创建多个服务器
      i. 实现步骤
         1) 添加头文件 ---<pthread.h>
         2) 函数:
            a) int pthread_create(pthread_t *thread , const pthread_attr_t *attr , void *(*start_routine)(void*) , void *arg) --- 开启线程
            b) pthread_detach(pthread *thread) --- 分离线程
            c) pthread_exit() --- 中断线程
         3) 编译make:
            a) 创建一个Makefile文件填写编译命令:
               i) all:
                  gcc server.c -lpthread -L ./ -lSmartHome -o app

   b. 智能家居相关函数:

```
int serial_init(char *devpath, int baudrate);//串口设备初始化函数
ssize_t serial_recv_exact_nbytes(int fd, void *buf, size_t count);//串口设备数据接收函数
ssize_t serial_send_exact_nbytes(int fd,  unsigned char *buf, size_t count);//串口设备数据写入函数
int serial_exit(int fd);//串口设备退出函数
```

参数说明:

1.char *devpath:串口设备打开的文件路径
2.baudrate:串口通信的波特率
3.fd:串口初始化返回的串口设备的文件描述符
4.buf:想要写入或者读出的缓冲区
5.count:缓冲区大小

```
//摄像头设备初始化函数
int camera_init(char *devpath, unsigned int *width, unsigned int *height, unsigned int *size);
int camera_start(int fd);//摄像头设备开启函数
int camera_dqbuf(int fd, void **buf, unsigned int *size, unsigned int *index);//摄像头图像数据出队函数，即用这个函数进行拍照操作
int camera_eqbuf(int fd, unsigned int index);//摄像头数据入队函数
int camera_stop(int fd);//摄像头停止函数
int camera_exit(int fd);//摄像头退出函数
```

参数说明:

1.char *devpath:摄像头设备的文件路径
2.width：摄像头采集图片的宽度
3.height：摄像头采集图片的高度
4.size：摄像头采集图片的大小
5.index：摄像头出队位置

   c. 串口控制:

```
unsigned char buf[5]   = {0xdd,0x02,0x24,0x00,0x00};    控制灯开启
unsigned char buf1[5] = {0xdd,0x02,0x24,0x00,0x01};    控制灯关闭
unsigned char buf2[5] = {0xdd,0x02,0x24,0x00,0x02};    控制蜂鸣器开启
unsigned char buf3[5] = {0xdd,0x02,0x24,0x00,0x03};    控制蜂鸣器关闭
unsigned char buf4[5] = {0xdd,0x02,0x24,0x00,0x04};    控制风扇开启
unsigned char buf5[5] = {0xdd,0x02,0x24,0x00,0x08};    控制风扇关闭
```

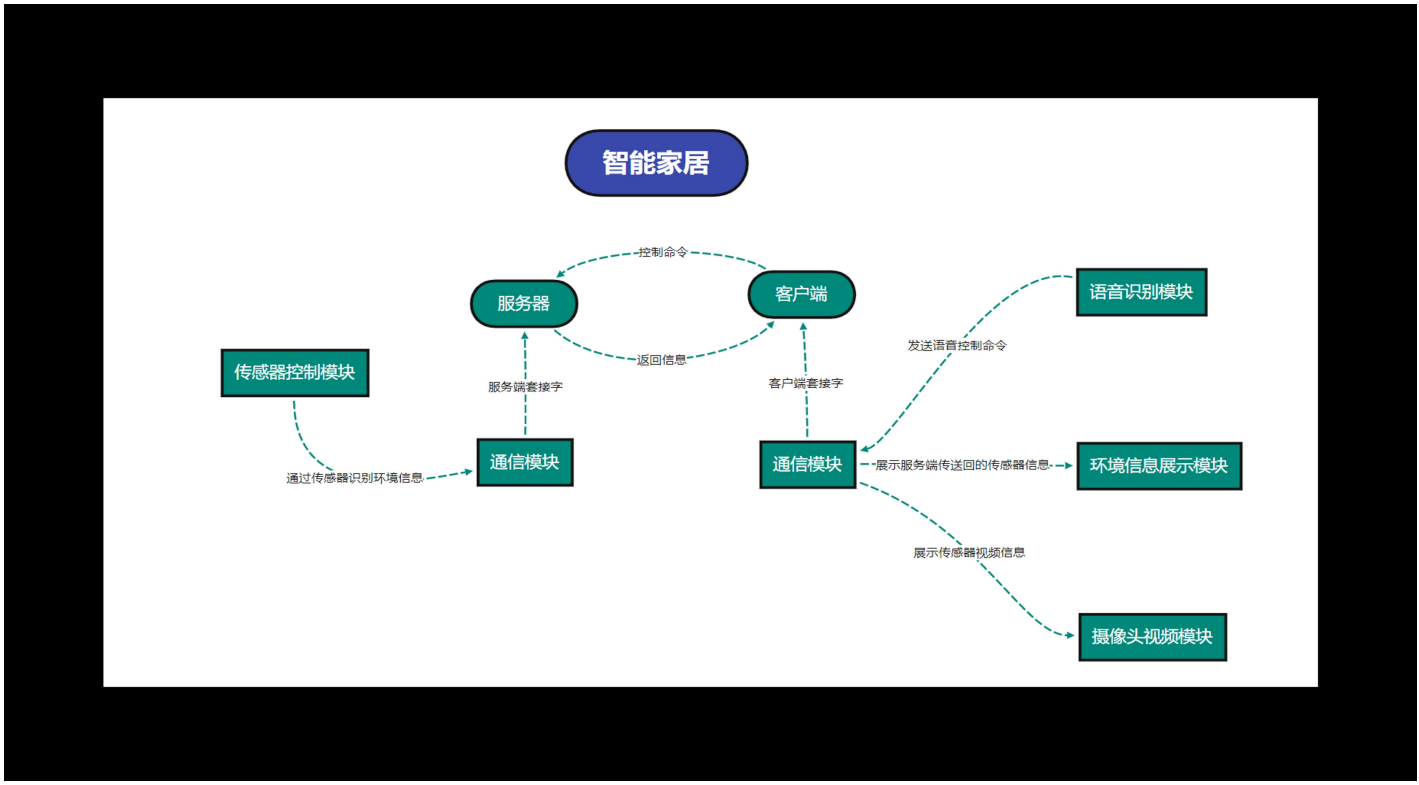
环境信息接收:

总共36个字节的数据

```
int id = buf[1];                                        //得到板子的id
int tem =  (int)buf[5] + buf[4] ;                       //将得到的温度数值
```

```
int hum =  (int)buf[7] + buf[6] ;                                   //将得到的湿度信息
int light =  (int)((buf[23] << 24) + (buf[22] << 16) + (buf[21] << 8) + buf[20]);       //将得到的光照信息
```

3. 各个模块之间关系



二. 代码

1. main.c

```c
/*------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>
//open
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
//read
#include <unistd.h>
//socket
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/in.h>
int server_init(short port)
{
    int listenfd;
    listenfd = socket(AF_INET,SOCK_STREAM,0);
    if(listenfd < 0)
    {
        perror("socket");
        return -1;
    }
    struct sockaddr_in ser_addr;
    memset(&ser_addr,0,sizeof(ser_addr));
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_port = htons(port);
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret = bind(listenfd,(const struct sockaddr*)&ser_addr,sizeof(ser_addr));
    if(ret < 0)
    {
        perror("bind");
        return -1;
    }
```

```
    ret = listen(listenfd,16);
    if(ret < 0)
    {
        perror("listen");
        return -1;
    }
    printf("server_init->success\n");
    return listenfd;
}
int server_wait_connect(int listenfd)
{
    int connfd;
    connfd = accept(listenfd,NULL,NULL);
    if(connfd < 0)
    {
        perror("accept");
        return -1;
    }
    printf("connect->success\n");
    return connfd;
}

void *func(void * lis_fd)
{
    int listenfd = *(int *)lis_fd;
    int connfd = server_wait_connect(listenfd);
    char pic_buf[1300*1024];
    int pic_len;
    int pic[8];
    pic[0] = open("../../../pic/1.jpg",O_RDWR);
    pic[1] = open("../../../pic/2.jpg",O_RDWR);
    pic[2] = open("../../../pic/3.jpg",O_RDWR);
    pic[3] = open("../../../pic/4.jpg",O_RDWR);
    pic[4] = open("../../../pic/5.jpg",O_RDWR);
    pic[5] = open("../../../pic/7.jpg",O_RDWR);
    pic[6] = open("../../../pic/8.jpg",O_RDWR);
    pic[7] = open("../../../pic/13.jpg",O_RDWR);
    int i=0;
    char buf[10];
    while(1)
    {
        read(connfd,buf,sizeof(buf));
        if(strcmp(buf,"pic")==0)
        {
            pic_len = lseek(pic[i],0,SEEK_END);
            lseek(pic[i],0,SEEK_SET);
            write(connfd,&pic_len,4);
            read(pic[i],pic_buf,pic_len);
            write(connfd,pic_buf,pic_len);
            i++;
            if(i==8)
            {
                i=0;
            }
        }
    }
}
int main()
{
    int listenfd = server_init(8888);
    int listenfd1 = server_init(7777);
    int listenfd2 = server_init(6666);
    pthread_t tid,tid2,tid3;
    int ret = pthread_create(&tid,NULL,func,(void *)&listenfd);
    if(ret < 0)
    {
        perror("pthread_create");
        return -1;
    }
    ret = pthread_detach(tid);
    if(ret < 0)
    {
        perror("pthread_detach");
        return -1;
    }
    ret = pthread_create(&tid2,NULL,func,(void *)&listenfd1);
    if(ret < 0)
    {
        perror("pthread_create");
        return -1;
    }
    ret = pthread_detach(tid2);
    if(ret < 0)
```

```
    {
        perror("pthread_detach");
        return -1;
    }
    ret = pthread_create(&tid3,NULL,func,(void *)&listenfd2);
    if(ret < 0)
    {
        perror("pthread_create");
        return -1;
    }
    ret = pthread_detach(tid3);
    if(ret < 0)
    {
        perror("pthread_detach");
        return -1;
    }
    while(1);
}
```

2. SmartHomeLib.h

```
/*================================================
*   文件名称：SmartHomeLib.h
*   描    述：关于智能家居的相关函数
================================================*/
#ifndef __SMARTHOMELIB_H__
#define __SMARTHOMELIB_H__
#include <stdio.h>      /*标准输入输出定义*/
#include <stdlib.h>     /*标准函数库定义*/
#include <unistd.h>     /*Unix 标准函数定义*/
#include <string.h>
#include <fcntl.h>      /*文件控制定义*/
#include <termios.h>    /*PPSIX 终端控制定义*/
#include <errno.h>      /*错误号定义*/
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <assert.h>
#include <poll.h>
#include <linux/videodev2.h>

int serial_init(char *devpath, int baudrate);
ssize_t serial_recv_exact_nbytes(int fd, void *buf, size_t count);
ssize_t serial_send_exact_nbytes(int fd,  unsigned char *buf, size_t count);
int serial_exit(int fd);


int camera_init(char *devpath, unsigned int *width, unsigned int *height, unsigned int *size);
int camera_start(int fd);
int camera_dqbuf(int fd, void **buf, unsigned int *size, unsigned int *index);
int camera_eqbuf(int fd, unsigned int index);
int camera_stop(int fd);
int camera_exit(int fd);


#endif
```