湘潭大学信息工程学院
The College of Information Engineering of Xiangtan University

TCPIP网络编程基础教程

王雷

博士、教授

教学大纲（四个部分）：

1.基于套接字的TCP/IP网络通信原理与模型

套接字的基本概念、基于套接字的网络通信实现流程、BSD UNIX套接字API函数、 Windows 套接字API函数、 TCP/IP网络通信模型及其C语言实现方法

2.循环服务器软件的实现原理与方法

循环服务器软件的实现原理及其C语言实现方法

3.服务器与客户进程中的并发机制与实现方法

服务器与客户进程中的并发机制、多进程并发机制的实现原理与方法、多线程并发TCP服务器软件的实现原理与方法、单线程并发机制的实现原理与方法、基于POOL和EPOLL的并发机制与实现方法

4.客户/服务器系统中的死锁问题与解决方法

死锁的定义、产生死锁的原因、处理死锁的基本方法

第二章 循环服务器软件的实现原理与方法

2.1客户/服务器模型中服务器软件实现的复杂性

2.1.1服务器设功能需求的复杂性

为了完成计算和返回结果给客户，服务器软件通常需要访问受操作系统保护的对象（如：文件、数据库、设备或协议端口等），因此，服务器软件的执行通常需要带有一些系统特权。由于服务区具有系统特权并应包含处理安全问题的代码，从而导致了服务器软件的设计与实现比客户软件的设计与实现要更加的困难和复杂。

第二章 循环服务器软件的实现原理与方法

2.1.2 服务器类型的复杂性

在客户/服务器模型中，服务器模型可依据不同方式进行分类。

例如：按照使用的传输协议不同可以分为无连接（UDP）的服务器与面向连接（TCP）的服务器；

按照是否维护与客户交互活动的信息可以分为有状态服务器与无状态服务器；

按照处理与客户交互的机制不同又可以分为循环服务器与并发服务器。

第二章 循环服务器软件的实现原理与方法

2.2循环服务器的进程结构

2.2.1循环UDP服务器的进程结构

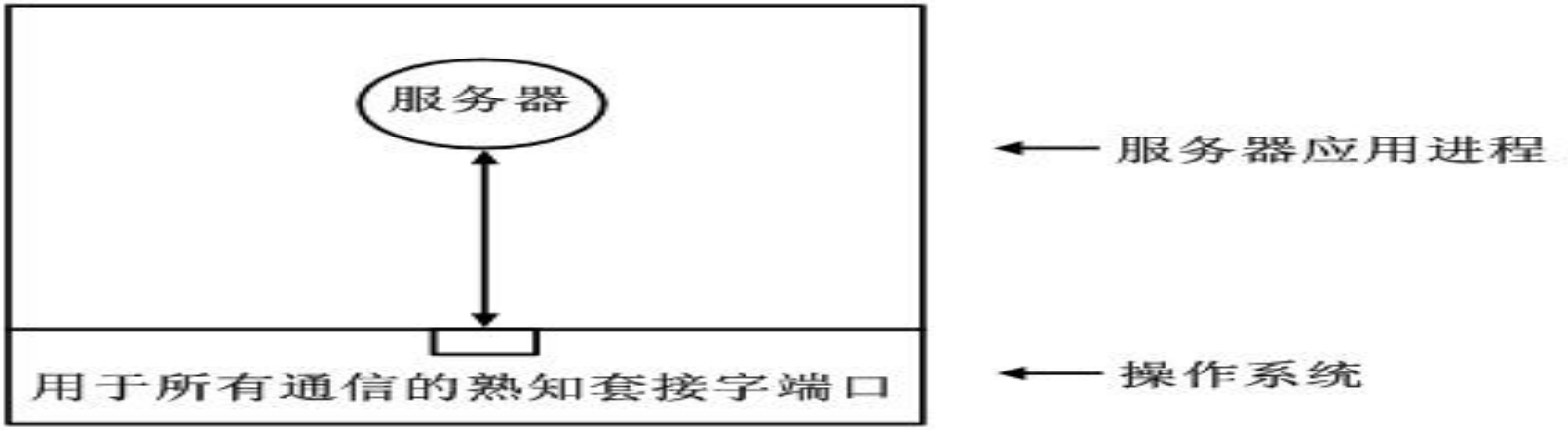
虽然循环的无连接服务器采用的是循环的方式来处理来自多个客户端的请求，当服务器每次从套接字上读取了一个新的客户请求之后，均需要在将该客户请求处理完毕并将结果返回给了该客户之后才能读取下一个客户请求，但由于采用了无连接的UDP方式来进行通信，从而使得没有一个客户端可以长时间占据服务器不放，因此，循环UDP服务器不但设计、编程、排错以及修改等工作都非常简单，而且只要处理过程没有被设计成死循环，就总能够满足每一个客户的请求。

循环UDP服务器的进程结构如下图2.1所示：

第二章 循环服务器软件的实现原理与方法

1.1.3 TCP/IP参考模型的通信原理

由图2.1易知，循环UDP服务器只需要一个单线程的进程即可实现，它仅使用一个被动套接字，该套接字绑定到所提供服务的熟知端口上，服务器从该套接字上循环获取新的客户请求，并计算出响应、然后再通过把该客户请求中包含的源地址作为应答中的目的地址来将响应返回给该客户。



第二章 循环服务器软件的实现原理与方法

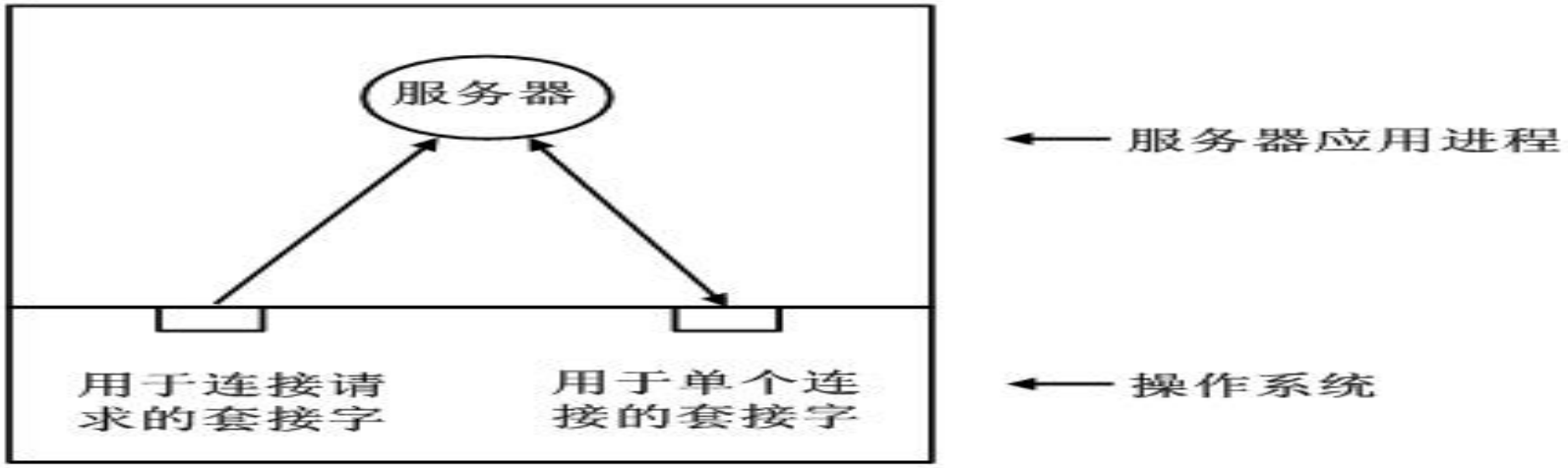
2.2.2循环TCP服务器的进程结构

虽然循环的面向连接的服务器也是采用的循环的方式来处理来自多个客户端的请求，但由于采用了面向连接的TCP方式来进行通信，因此当服务器每次从套接字上读取了一个新的客户连接请求之后，服务器将首先与该客户端之间建立一个连接，然后再通过该连接与该客户进行交互，当交互结束之后再关闭该连接，然后再次等候/读取了下一个新的客户连接请求。

循环TCP服务器的进程结构如下图2.2所示：

第二章 循环服务器软件的实现原理与方法

由图2.2易知，循环TC服务器也只需要一个单线程的进程即可实现，但它使用了两个套接字：其中，一个套接字用于循环地接受来自客户的连接请求，该套接字绑定到所提供服务的熟知端口上，服务器从该套接字上循环地获取新的客户连接请求，该套接字为**永久套接字**。



而另一个套接字则为**临时套接字**，该临时套接字用于处理单个连接，当服务器接受一个新的客户请求之后，将建立一个与该客户端的TCP连接，并创建一个临时套接字来负责在该TCP连接上与该客户端进行通信（计算出响应并将响应返回给该客户），当与该客户端之间的交互完毕之后，该临时套接字与TCP连接均将被关闭，然后服务器将再次基于永久套接字获取来自下一个新的客户的连接请求。

2.3循环服务器软件的设计流程

2.3.1循环UDP服务器软件的设计流程

依据上述给出的循环UDP服务器的进程结构，可以给出循环UDP服务器算法的设计流程如下：

第二章 循环服务器软件的实现原理与方法

1. UNIX/LINUX环境下的循环UDP服务器软件设计流程

步骤1：调用socket()函数创建服务器端UDP套接字；

步骤2：调用bind()函数将套接字绑定到本机的一个可用端点地址；

步骤3：调用while(1)函数设置无限循环；

步骤4：在循环体内：

步骤4.1：调用recvfrom()函数读取来自客户的请求；

步骤4.2：然后构造响应；

步骤4.3：再调用sendto()函数按照应用协议将响应发回给客户。

步骤5：调用close()函数关闭套接字，释放相关资源；

第二章 循环服务器软件的实现原理与方法

基于以上给出的算法流程，可以给出用C语言实现的伪代码描述如下：

```
socket(...);   //对应于步骤1
bind(...);    //对应于步骤2
while(1){   //对应于步骤3
      recvfrom(...);   //对应于步骤4.1
      process(...);    //对应于步骤4.2
      sendto(...);    //对应于步骤4.3
 }
close(…); //对应于步骤5
```

第二章 循环服务器软件的实现原理与方法

2. Windows环境下的循环UDP服务器软件设计流程

步骤1：调用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建服务器端UDP套接字；

步骤3：调用bind()函数将套接字绑定到本机的一个可用端点地址；

步骤4：调用while(1)函数设置无限循环；

步骤5：在循环体内：

步骤5.1：调用recvfrom()函数读取来自客户的请求；

步骤5.2：然后构造响应；

步骤5.3：再调用sendto()函数按照应用协议将响应发回给客户。

步骤6：调用closesocket()函数关闭套接字，释放相关资源；

步骤7：调用WSACleanup()函数结束Winsock Socket API。

第二章 循环服务器软件的实现原理与方法

基于以上给出的算法流程，可以给出用C语言实现的伪代码描述如下：

```
 WSAStartup(...);   //对应于步骤1
socket(...);   //对应于步骤2
 bind(...);    //对应于步骤3
while(1){   //对应于步骤4
      recvfrom(...);   //对应于步骤5.1
      process(...);    //对应于步骤5.2
      sendto(...);    //对应于步骤5.3
}
closesocket(…); //对应于步骤6
WSACleanup(); //对应于步骤7
```

第二章 循环服务器软件的实现原理与方法

2.3.2循环TCP服务器软件的设计流程

依据上述给出的循环TCP服务器的进程结构，可以给出循环TCP服务器算法的设计流程如下：

1. UNIX/LINUX环境下的循环TCP服务器软件设计流程

步骤1：调用socket()函数创建服务器端TCP主套接字；

步骤2：调用bind()函数将套接字绑定到本机的一个可用的端点地址；

步骤3：调用listen()函数将套接字设置为被动模式，并设置等待队列的长度；

步骤4：调用while(1)函数设置无限循环；

步骤5：在循环体内：

第二章 循环服务器软件的实现原理与方法

步骤5.1：调用accept()函数接受来自客户的连接请求并创建一个用于处理该连接的临时套接字；

步骤5.2：调用recv/send()函数基于新创建的临时套接字与客户进行交互；

步骤5.3：与客户交互完毕，调用close()函数将临时套接字关闭。

步骤6：调用close()函数将服务器端TCP主套接字关闭。

第二章 循环服务器软件的实现原理与方法

基于以上给出的算法流程，可以给出用C语言实现的伪代码描述如下：

```
socket(...);   //对应于步骤1
 bind(...);    //对应于步骤2
listen(...);   //对应于步骤3
while(1){   //对应于步骤4
        accept() (...);   //对应于步骤5.1
        process(...);    //对应于步骤5.2
        close(...);     //对应于步骤5.3
 }
 close(...);     //对应于步骤6
```

第二章 循环服务器软件的实现原理与方法

2. Windows环境下的循环TCP服务器软件设计流程

步骤1：调用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建服务器端TCP主套接字；

步骤3：调用bind()函数将套接字绑定到本机的一个可用的端点地址；

步骤4：调用listen()函数将套接字设置为被动模式，并设置等待队列的长度；

步骤5：调用while(1)函数设置无限循环；

步骤6：在循环体内：

第二章 循环服务器软件的实现原理与方法

步骤6.1：调用accept()函数接受来自客户的连接请求并创建一个用于处理该连接的临时套接字；

步骤6.2：调用recv/send()函数基于新创建的临时套接字与客户进行交互；

步骤6.3：与客户交互完毕，调用closesocket()函数将临时套接字关闭。

步骤7：调用closesocket()函数将服务器端TCP主套接字关闭。

步骤8：调用WSACleanup()函数结束Winsock Socket API。

第二章 循环服务器软件的实现原理与方法

基于以上给出的算法流程，可给出C语言实现的伪代码描述如下：

```
 WSAStartup(...); //对应于步骤1
socket(...);   //对应于步骤2
bind(...);    //对应于步骤3
listen(...);   //对应于步骤4
while(1){   //对应于步骤5
    accept() (...);   //对应于步骤6.1
    process(...);    //对应于步骤6.2
    close(...);     //对应于步骤6.3
}
closesocket(...);     //对应于步骤7
WSACleanup(); //对应于步骤8
```

第二章 循环服务器软件的实现原理与方法

2.4基于循环服务器的网络通信例程剖析

2.4.1相关系统函数及其调用方法简介

在实际给出循环服务器例程并对其进行深入剖析之前，首先详细介绍例程中在本书中首次出现的系统函数及其调用方法如下：

1. sizeof()操作符

sizeof()是C/C++中的一个操作符（operator），作用是返回一个对象或类型所占用的内存字节数。sizeof()操作符的返回值类型为size_t（即unsigned int），该类型可以保证能够容纳实现所建立的最大对象的字节大小。sizeof()操作符的调用方法如下（两种原型等价）：

第二章 循环服务器软件的实现原理与方法

```
#include <stdlib.h>
size_t sizeof(object);      //sizeof(对象)
size_t sizeof(type_name);  //sizeof(类型)
```

例： int i;

size_t sz;

```
sz=sizeof(i);              //返回对象i所占用的内存字节数
sz=sizeof(int);            /*返回类型int所占用的内存字节数，由于i的类
                            型为int，因此sizeof(int)等价于sizeof(i)*/
```

第二章 循环服务器软件的实现原理与方法

例：char *c="abcdef";

char d[]="abcdef";

则sizeof(c)的返回值为4，strlen(c)的返回值为6；sizeof(d)的返回值为7，strlen(d)的返回值为6。其中，由于c是一个指向字符串"abcdef"的指针，而指针一般分配4个字节，因此sizeof(c)的结果就是4；再由于指针c指向的字符串的长度为6个字节，因此strlen(c) 的结果就是6；其次，由于d是一个未指定大小的字符串，其大小将根据后面初始化的内容来自动分配，而后面初始化的实际内容是一个6字节的字符串"abcdef"，因此strlen(c)的结果就是6；再由于字符串最后还包括有一个终止符'\0'，因此sizeof(d) 的结果就是6+1=7。

第二章 循环服务器软件的实现原理与方法

3. printf()函数

printf()函数是一个可变参数函数，其主要功能是用来向标准输出设备按规定格式输出信息。printf()函数的原型如下：

```
#include <stdio.h>
int printf(const char *format[,argument,...]);
```

在上述printf()函数的原型中，各参数的含义如下：

※ **format**："格式控制"字符串，用于指明输出的格式，其完整形式为：% - 0 m.n l或h 格式字符，其中：

（1）%：表示格式说明的起始符号，不可缺少。

（2）-：有-表示左对齐输出，如省略表示右对齐输出。

第二章 循环服务器软件的实现原理与方法

（3）0：有0表示指定空位填0，如省略表示指定空位不填。

（4）m.n：其中，m表示域宽，即对应的输出项在输出设备上所占的字符数。n指精度。用于说明输出的实型数的小数位数。未指定n时，隐含的精度为n=6位。

（5）l或h：其中，l用于对整型指long型，对实型指double型。h用于将整型的格式字符修正为short型。

（6）格式字符：

① d格式：用来输出十进制整数。

② o格式：以无符号八进制形式输出整数。

第二章 循环服务器软件的实现原理与方法

③ x格式：以无符号十六进制形式输出整数。

④ u格式：以无符号十进制形式输出整数。

⑤ c格式：输出一个字符。

⑥ s格式：用来输出一个串。

⑦ f格式：用来输出实数（包括单双精度），以小数形式输出。

⑧ e格式：以指数形式输出实数。

⑨ g格式：自动选f格式或e格式中较短的一种输出，且不输出无意义的零。

注：若想输出字符"%"，则需在"格式控制"字符串中用连续两个%表示。

第二章 循环服务器软件的实现原理与方法

※ **argument**：指向需要输出的字符串的指针。

例：printf("%.3f ",12.3456); //输出结果为12.346

printf("%.9f ",12.3456); //输出结果为12.345600000，不足位补0

4. fprintf()函数

fprintf()函数用来将输出的内容输出到硬盘上的文件或是相当于文件的设备上。fprintf()函数的原型如下：

#include <stdio.h> //标准输入输出头文件，包含了fprintf()等函数的定义

int fprintf(FILE *stream,const char *format[,argument,...]);

第二章 循环服务器软件的实现原理与方法

※ 调用fprintf()函数向显示器输出错误信息时其调用方法为：

fprintf(stderr,"错误信息");

在上述fprintf()函数的原型中，各参数的含义如下：

※ **stream**：指向用于接受输出的设备或文件的指针；

※ **format**：参数的输出格式，具体可参见printf()函数；

※ **argument**：指向需要输出的字符串的指针。

5. memset()函数

memset()函数用来对一段内存地址空间全部设置为某个值或清空（否则，可能会在测试当中出现野值），一般用在对定义的字符串进行初始化，也可用来方便地清空一个结构类型的变量或数组；

memset()函数的原型如下：

第二章 循环服务器软件的实现原理与方法

#include <mem.h>　　//该头文件中提供了memset()函数原型的定义

void* memset(void* s, int c, size_t n);

在上述memset()函数的原型中，各参数的含义如下：

※ **s**：指向目标内存地址空间的起始地址的指针；

※ **c**：要赋的值；

※ **n**：要赋值的长度（字节数）。注：由该参数可知，memset()函数是以字节为单位来进行赋值的。

例1. 调用memset()函数给数组赋值：

调用memset()函数给整型数组赋值：

```
int buf[50];
memset(buf, 0, 50*sizeof(int));
```

第二章 循环服务器软件的实现原理与方法

调用memset()函数给字符型数组赋值：

```
char buf[50];
memset( buf, '\0', 50);  /*转义符'\0'为C语言中的字符串结束符，在数值类型里代表数字0，即8位的00000000*/
```

例2. 调用memset()函数给结构赋值：

```
struct sample_struct {
    char csName[16];
    int iSeq;
    int iType;
};
struct sample_strcut stTest;
memset(&stTest,0,sizeof(struct sample_struct));
```

第二章 循环服务器软件的实现原理与方法

#include <string.h> //该头文件中提供了字符串函数的原型定义

int strcmp(const char *s1,const char * s2);

在上述strcmp()函数的原型中，各参数的含义如下：

※ **s1**：指向用于比较的第一个字符串的指针；

※ **s2**：指向用于比较的第二个字符串的指针。

当s1<s2时，strcmp()函数的返回值<0；当s1=s2时，strcmp()函数的返回值=0；当s1>s2时，strcmp()函数的返回值>0。

7. atoi()函数

atoi()函数用于将一个字符串转换成一个整型数值，若成功转换将返回转换后得到的整型数值，若失败则返回0。atoi()函数的原型如下：

第二章 循环服务器软件的实现原理与方法

#include <stdlib.h> //C标准库头文件，包含了exit()、atoi()等函数原型的定义

int atoi(const char *str);

在上述atoi()函数的原型中，各参数的含义如下：

※ **str**：待转换为整型数值的字符串。

atoi()函数会扫描待转换为整型数值的字符串str，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而等到再遇到非数字或字符串结束符'\0'时就结束转换并将结果返回。

例：char *a = "-100abc";

```
char *b = "456.12";
int c;
c = atoi(a) + atoi(b);          //c的值为356（= -100+456）
```

8. 可变参数函数

可变参数函数的参数个数是可变的。一般情况下，所编的程序中的函数的参数个数都是固定的，但是有时候需要用到可变参数的函数。要在函数中包含可变个数的参数，首先应该在头文件中包含<stdarg.h>，即#include <stdarg.h>。该头文件声明了一个va_list类型和四个操作可变参数的函数：

```
void va_start(va_list ap, argN);
void va_copy(va_list dest, va_list src);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

第二章 循环服务器软件的实现原理与方法

9. main()函数

C程序是从main()函数开始执行，其原型如下：

int main(int argc, char* argv[]);

在上述main()函数的原型中，各参数的含义如下：

※ **argc**：指用命令行方式输入的命令行参数的个数；

※ **argv**：实际存储了所有输入的命令行参数。

假如编写的程序是hello.exe，如果在命令行方式下运行该程序（首先应该在命令行方式下用 cd 命令进入到 hello.exe 文件所在目录），则运行改程序的命令为：hello.exe a b c；此时，argc的值将为4，其中，argv[0]的值是"hello.exe"，argv[1]的值是"a"，argv[2]的值是"b"，argv[3]的值是"c"。

第二章 循环服务器软件的实现原理与方法

10. switch-case分支判断语句

```
switch(表达式) {
    case 表达式条件1:
        执行相应操作处理1;
        break;
    case 表达式条件2:
        执行相应操作处理2;
        break;
    …
    case 表达式条件N:
        执行相应操作处理N;
        break;
    default:  /*除以上三个条件外的其它条件*/
        执行相应其它操作处理;
        break;
}
```

第二章 循环服务器软件的实现原理与方法

例：
```
switch(a){
    case 1: /*若变量a等于1*/
            printf("1");
            break;
    case 2: /*若变量a等于2*/
            printf("2");
            break;
   case 4: /*若变量a等于4*/
           printf("4");
           break;
  default: /*若变量a不等于1、2或4*/
           printf("other nums");
           break;
}
```

第二章 循环服务器软件的实现原理与方法

11. time()函数

time()函数的功能是用于返回当前的日历时间，若调用该函数时发生错误则返回零。time()函数的原型如下：

```
#include <time.h>            //提供time()函数原型的定义
time_t time(time_t *time);
```

在上述time()函数的原型之中，各参数的含义如下：

※ **time**：指向用于存储当前时间的缓冲区的指针。

12. ctime()函数

ctime()函数返回一个字符串指针，其功能是用于将日历时间转换为字符串形式的本地时间。ctime()函数的原型如下：

第二章 循环服务器软件的实现原理与方法

```
#include <time.h>
char *ctime(const time_t *timer);
```

在上述ctime()函数的原型之中，各参数的含义如下：

※ **timer**：指向存储有当前日历时间的缓冲区的指针，该参数一般是通过调用函数time()获得。

例：

```
time_t t;
    time(&t);
    printf("Today's date and time: %s\n", ctime(&t));
```

13. strerror()函数

strerror()函数的功能是用于返回对应于某个错误编号的错误原因的描述字符串，从而得到一个可读的出错提示信息，而不再只是得到一个冷冰冰的错误编号数字。strerror()函数的原型如下：

```
#include<string.h>
char * strerror(int errnum);
```

在上述strerror()函数的原型之中，各参数的含义如下：

※ **errnum**：错误编号。

注：C语言中在头文件<errno.h>中定义有一个全局变量errno来记录程序出错时的对应错误编号。

14. fputs()函数

fputs()函数的功能是用于向指定的文件写入一个字符串，如果调用成功将返回 0，否则将返回-1。fputs()函数的原型如下：

```
#include<stdio.h>
int fputs(char *string, FILE *stream);
```

在上述fputs()函数的原型之中，各参数的含义如下：

※ **string**：需送入流的字符串指针；

※ **stream**：一个FILE型的指针。

14. fputs()函数

fputs()函数的功能是用于向指定的文件写入一个字符串，如果调用成功将返回 0，否则将返回-1。fputs()函数的原型如下：

```
#include<stdio.h>
int fputs(char *string, FILE *stream);
```

在上述fputs()函数的原型之中，各参数的含义如下：

※ **string**：需送入流的字符串指针；

※ **stream**：一个FILE型的指针。

15. malloc()函数

malloc()函数的功能是用于进行动态内存分配，如果调用成功将返回指向被分配内存的指针（此存储区中的初始值不确定），否则返回空指针NULL。当内存不再使用时，应使用free()函数将内存块释放。malloc()函数的原型如下：

```
#include <malloc.h>    /*动态存储分配函数头文件，包含了对内存区进行操作的相关函数的定义*/
void *malloc(size_t size);
```

在上述malloc()函数的原型之中，各参数的含义如下：

※ **size**：指明动态分配的内存块长度为size字节。

16. free()函数

free()函数的功能是用于释放malloc函数申请的动态内存，与malloc()函数配对使用，free()函数无返回值。free()函数的原型如下：

```
#include <malloc.h>
void free(void *ptr)
```

在上述free()函数的原型之中，各参数的含义如下：

※ **ptr**：指向申请释放的动态内存空间的指针。

17. OpenCV图像处理函数

OpenCV的全称是Open Source Computer Vision Library，是一个基于BSD许可（开源）发行的跨平台计算机视觉库，OpenCV开发包提供了读取各种类型的图像文件、视频内容以及摄像机输入的功能。

OpenCV开发包可以安装运行在Linux、Windows和Mac OS操作系统上，主要由一系列 C 函数和少量 C++ 类构成，同时提供了Python、Ruby、MATLAB等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

第二章 循环服务器软件的实现原理与方法

（1）cvLoadImage()函数

cvLoadImage()函数的功能是用于从指定文件读入图像，返回读入图像的指针。目前cvLoadImage()函数支持的文件格式包括Windows位图文件- BMP、DIB；JPEG文件- JPEG、JPG、JPE；便携式网络图片- PNG；便携式图像格式- PBM、PGM、PPM；TIFF文件- TIFF、TIF；以及JPEG 2000 图片-JP2等。

cvLoadImage()函数的原型如下：

IplImage* cvLoadImage(const char* filename, int flags);

在上述cvLoadImage()函数的原型之中，各参数的含义如下：

第二章 循环服务器软件的实现原理与方法

※**filename**：要被读入的文件的文件名（包括后缀）。

※**flags**：表示指定读入图像的颜色和深度，其使用方法如下：

cvLoadImage(filename, -1); //表示默认读取图像的原通道数

cvLoadImage(filename, 0); //表示强制转化读取图像为灰度图

cvLoadImage(filename, 1); //表示读取彩色图

（2）**cvCreateImage()函数**

cvCreateImage()函数的功能是用于为图像创建首地址并分配存储空间。cvCreateImage()函数的原型如下：

IplImage* cvCreateImage(CvSize size, int depth, int channels);

在上述cvCreateImage()函数的原型之中，各参数的含义如下：

第二章 循环服务器软件的实现原理与方法

※**size**：图像的宽、高。

※**depth**：图像元素的位深度，可取下列值项之一：

① IPL_DEPTH_8U：无符号8位整型。

② IPL_DEPTH_8S：有符号8位整型。

③ IPL_DEPTH_16U：无符号16位整型。

④ IPL_DEPTH_16S：有符号16位整型。

⑤ IPL_DEPTH_32S：有符号32位整型。

⑥ IPL_DEPTH_32F：单精度浮点数。

⑦ IPL_DEPTH_64F：双精度浮点数。

※**channels**：每个元素（像素）通道数.可以是 1, 2, 3 或 4。通道是交叉存取的，例如，通常的彩色图像数据排列是：b0 g0 r0 b1 g1 r1 ...。

（3）**cvNamedWindow()函数**

cvNamedWindow()函数的功能是用于创建指定的窗口。

cvNamedWindow()函数的原型如下：

int cvNamedWindow(const char* name, int flags);

在上述cvNamedWindow()函数的原型之中，各参数的含义如下：

※**name**：窗口名字，用来区分不同窗口，并被显示为窗口标题。

※**flags**：窗口属性标志。0表示用户可以手动调节窗口大小，且显示的图像尺寸随之变化。1表示用户不能手动改变窗口大小，窗口大小会自动调整以适合被显示图像。

第二章 循环服务器软件的实现原理与方法

（4）cvCvtColor()函数

cvCvtColor()函数的功能是用于图像的颜色空间转换，可以实现BGR颜色向HSV、HSI等颜色空间的转换，也可以转换为灰度图像。cvCvtColor()函数的原型如下：

```
void cvCvtColor(const CvArr* src, CvArr* dst, int code);
```

在上述cvCvtColor()函数的原型之中，各参数的含义如下：

※**src**：输入的 8-bit、16-bit或 32-bit单倍精度浮点数影像。

※**dst**：输出的8-bit、16-bit或 32-bit单倍精度浮点数影像。

※**code**：色彩空间转换的模式，可实现不同类型的颜色空间转换。例如取值CV_BGR2GRAY时表示转换为灰度图，取值CV_BGR2HSV时表示将图片从RGB空间转换为HSV空间。其中，当code选用CV_BGR2GRAY时，dst需要是单通道图片。当code选用CV_BGR2HSV时，对于8位图，需要将RGB值归一化到0-1之间，这样才能使得最终得到HSV图中的H范围才是0-360，S和V的范围是0-1。

（5）**cvShowImage()函数**

用于在指定窗口中显示图像，如果窗口创建的时候被设定标志CV_WINDOW_AUTOSIZE，那么图像将以原始尺寸显示；否则图像将被伸缩以适合窗口大小。cvShowImage()函数的原型如下：

void cvShowImage(const char* name, const CvArr* image);

在上述cvShowImage()函数的原型之中，各参数的含义如下：

※**name**：窗口的名字。

※**image**：被显示的图像。

（6）**cvWaitKey()函数**

cvWaitKey()函数的功能是用于不断刷新图像，频率时间为delay，单位为ms，其返回值为当前键盘按键值。当delay>0时，在当前状态下等待"delay"ms，若超过指定时间则返回-1；当delay<=0时，如果没有键盘触发则一直等待，否则返回值为键盘按下的码字。cvWaitKey()函数的原型如下：

第二章 循环服务器软件的实现原理与方法

int cvWaitKey(int delay);

在上述cvWaitKey()函数的原型之中，各参数的含义如下：

※**delay**：刷新图像的频率时间。

用法举例：

① 在显示图像时，若在cvShowImage("xxxx.bmp",image)后加上while(cvWaitKey(n)==key){}循环，其中n为大于等于0的数，则程序将会停在显示函数处不运行其他代码，直到键盘值为key的响应之后。

② 在条件语句if(cvWaitKey(10)>=0){}中，cvWaitKey(10)表示在当前状态下等待十毫秒，而整个条件语句的意思就是：如果在十毫秒内按下任意键就将进入到if子句之中。

第二章 循环服务器软件的实现原理与方法

（7）**cvDestroyWindow()函数**

用于销毁一个窗口。cvDestroyWindow()函数的原型如下：

```
void cvDestroyWindow(const char* name);
```

在上述cvDestroyWindow()函数的原型之中，各参数的含义如下：

※**name**：窗口的名字。

（8）**cvReleaseImage()函数**

用于释放图像指针变量占用的内存资源。其原型如下：

```
void cvReleaseImage(IplImage** image);
```

在上述cvReleaseImage()函数的原型之中，各参数的含义如下：

※**image**：调用cvLoadImage()函数或cvCreateImage()函数创建的IplImage图像指针。

第二章 循环服务器软件的实现原理与方法

（9）**cvSaveImage()函数**

用于保存图像到指定文件，图像格式的的选择依赖于filename的后缀（扩展名，如.JPG、.BMP等），只有8位单通道或3通道（通道顺序为'BGR'）的图像才可以使用这个函数保存。如果格式，深度或者通道不符合要求，需要先用cvCvtScale()函数和cvCvtColor()函数进行转换；或者使用通用的cvSave()函数来保存图像为XML或YAML格式。cvSaveImage()函数的原型如下：

int cvSaveImage(const char* filename, const CvArr* image);

在上述cvSaveImage()函数的原型之中，各参数的含义如下：

※**filename**：图像文件名。

※**image**：要保存的图像。

第二章 循环服务器软件的实现原理与方法

（10）**OpenCV图像处理函数应用举例**

```
//从图像文件Bird.jpg加载图像
IplImage
*img=cvLoadImage(“Bird.jpg”,CV_LOAD_IMAGE_COLOR);
if(img){
    cvNamedWindow(“OpenCV Demo”, 1); //创建显示窗口
    cvShowImage(“OpenCV Demo”, img); //显示图像
    cvWaitKey(0); //等待用户按任意键退出
    cvSaveImage(“Bird.png”, img); //将图像保存为png格式
    cvSaveImage(“Bird.bmp”, img); //将图像保存为bmp格式
    cvDestroyWindow(“OpenCV Demo”); //关闭显示窗口
cvReleaseImage(&img); //释放图像
}
```

18. OpenCV视频处理函数

（1）cvCreateFileCapture()函数

cvCreateFileCapture()函数的功能是用于从文件中获取视频，并给指定文件中的视频流分配和初始化CvCapture结构，当分配的结构不再使用的时候，应使用cvReleaseCapture函数释放掉。

cvCreateFileCapture()函数的原型如下：

```
CvCapture* cvCreateFileCapture(const char* filename);
```

在上述cvCreateFileCapture()函数的原型之中，各参数含义如下：

※**filename**：视频文件名。

第二章 循环服务器软件的实现原理与方法

（2）cvCreateCameraCapture()函数

cvCreateCameraCapture()函数的功能是用于从摄像头中获取视频，并给从摄像头的视频流分配和初始化CvCapture结构，当分配的结构不再使用的时候，应使用cvReleaseCapture()函数释放掉。cvCreateCameraCapture()函数的原型如下：

```
CvCapture* cvCreateCameraCapture(int index);
```

在上述cvCreateCameraCapture()函数的原型之中，各参数的含义如下：

※**index**：要使用的摄像头索引。如果只有一个摄像头或者用哪个摄像头均无所谓，则可设置该参数为-1。如果要同时使用多个摄像头，则摄像头的索引是按序号排列的，第一个摄像头索引号为0，另一个就是1，以此类推。

第二章 循环服务器软件的实现原理与方法

（3）**cvReleaseCapture()函数**

用于释放由cvCreateFileCapture()或cvCreateCameraCapture()函数所分配的CvCapture结构。cvReleaseCapture()函数的原型如下：

void cvReleaseCapture(CvCapture** capture);

在上述cvReleaseCapture()函数的原型之中，各参数的含义如下：

※**capture**：视频获取结构指针。

（4）**cvGrabFrame()函数**

用于从摄像头或者视频文件中抓取帧。该函数的目的是快速抓取帧，这对需要同时从多个摄像头读取数据时的同步而言是非常重要的。函数cvGrabFrame()从摄像头或者文件中抓取的帧是在内部被存储的，如果要取回获取的帧，还需要调用cvRetrieveFrame()函数。cvGrabFrame()函数的原型如下：

第二章 循环服务器软件的实现原理与方法

int cvGrabFrame(CvCapture* capture);

在上述cvGrabFrame()函数的原型之中，各参数的含义如下：

※**capture**：视频获取结构。

（5）**cvRetrieveFrame()函数**

用于取回由函数cvGrabFrame抓取的图像，返回的图像不可以被用户释放或者修改。cvRetrieveFrame()函数的原型如下：

IplImage* cvRetrieveFrame(CvCapture* capture);

在上述cvRetrieveFrame()函数的原型之中，各参数的含义如下：

※**capture**：视频获取结构。

（6）**cvQueryFrame()函数**

cvQueryFrame()函数的功能是用于从摄像头或者文件中抓取并返回一帧，然后解压并返回这一帧。该函数仅仅是函数cvGrabFrame()和函数cvRetrieveFrame()的组合调用，返回的图像不可以被用户释放或者修改。抓取后，capture被指向下一帧，可用cvSetCaptureProperty()函数调整capture到合适的帧。

cvQueryFrame()函数的原型如下：

IplImage* cvQueryFrame(CvCapture* capture);

在上述cvQueryFrame()函数的原型之中，各参数的含义如下：

※**capture**：视频获取结构。

第二章 循环服务器软件的实现原理与方法

（7）cvGetCaptureProperty()函数

用于获得视频获取结构的属性。有时候这个函数在cvQueryFrame被调用一次后，再调用cvGetCaptureProperty才会返回正确的数值。这是一个bug，因此，建议在调用此函数前先调用cvQueryFrame()函数。函数原型如下：

```
double cvGetCaptureProperty(CvCapture* capture, int property_id);
```

在上述cvGetCaptureProperty()函数的原型中，各参数含义如下：

※**capture**：视频获取结构。

※**property_id**：属性标识。可以是下面之一：

第二章 循环服务器软件的实现原理与方法

① CV_CAP_PROP_POS_MSEC：影片目前位置，为毫秒数或者视频的获取时间戳。

② CV_CAP_PROP_POS_FRAMES：将被下一步解压/获取的帧索引，以0为起点。

③ CV_CAP_PROP_POS_AVI_RATIO：视频文件的相对位置（0表示影片的开始，1表示影片的结尾）。

④ CV_CAP_PROP_FRAME_WIDTH：视频流中的帧宽度。

⑤ CV_CAP_PROP_FRAME_HEIGHT：视频流中的帧高度。

⑥ CV_CAP_PROP_FPS：帧速率。

⑦ CV_CAP_PROP_FOURCC：表示codec的四个字符。

⑧ CV_CAP_PROP_FRAME_COUNT：视频文件中帧的总数。

第二章 循环服务器软件的实现原理与方法

（8）cvSetCaptureProperty()函数

cvSetCaptureProperty()函数的功能是用于设置视频获取属性。该函数的原型如下：

```
int cvSetCaptureProperty(CvCapture* capture, int property_id, double value);
```

在上述cvSetCaptureProperty()函数的原型之中，各参数的含义如下：

※**capture**：视频获取结构。

※**property_id**：属性标识。可以是下面之一：

第二章 循环服务器软件的实现原理与方法

① CV_CAP_PROP_POS_MSEC：从文件开始的位置，单位为毫秒。
② CV_CAP_PROP_POS_FRAMES：单位为帧数的位置（只对视频文件有效）。
③ CV_CAP_PROP_POS_AVI_RATIO：视频文件的相对位置（0表示影片的开始，1表示影片的结尾）。
④ CV_CAP_PROP_FRAME_WIDTH：视频流的帧宽度（只对摄像头有效）。
⑤ CV_CAP_PROP_FRAME_HEIGHT：视频流的帧高度（只对摄像头有效）。
⑥ CV_CAP_PROP_FPS：帧速率（只对摄像头有效）。
⑦ CV_CAP_PROP_FOURCC：表示codec的四个字符（只对摄像头有效）。

第二章 循环服务器软件的实现原理与方法

※**value**：属性的值。目前，cvSetCaptureProperty()函数对视频文件只支持CV_CAP_PROP_POS_MSEC、CV_CAP_PROP_POS_FRAMES和CV_CAP_PROP_POS_AVI_RATIO。

（9）**cvCreateVideoWriter()函数**

cvCreateVideoWriter()函数的功能是用于创建视频文件写入器。该函数的原型如下：

```
typedef struct CvVideoWriter CvVideoWriter;
CvVideoWriter* cvCreateVideoWriter(const char* filename, int fourcc, double fps, CvSize frame_size, int is_color);
```

在上述cvCreateVideoWriter()函数的原型之中，各参数含义如下：

第二章 循环服务器软件的实现原理与方法

※**filename**：输出视频文件名。

※**fourcc**：四个字符用来表示压缩帧的codec 例如，CV_FOURCC('P','I','M','1')是MPEG-1 codec，CV_FOURCC('M','J','P','G')是motion-jpeg codec等。

※**fps**：被创建视频流的帧速率。

※**frame_size**：视频流的大小。

※**is_color**：如果非零，编码器将希望得到彩色帧并进行编码；否则，是灰度帧（仅在Windows下支持该标志）。

（10）**cvReleaseVideoWriter()函数**

cvReleaseVideoWriter()函数的功能是用于释放视频写入器。该函数的函数原型如下：

void cvReleaseVideoWriter(CvVideoWriter** writer);

void cvReleaseVideoWriter(CvVideoWriter** writer);

在上述cvReleaseVideoWriter()函数的原型之中，各参数的含义如下：

※**writer**：指向视频写入器的指针。

（11）**cvWriteFrame()函数**

cvWriteFrame()函数的功能是用于写入一帧到一个视频文件中。该函数的原型如下：

int cvWriteFrame(CvVideoWriter* writer, const IplImage* image);

在上述cvWriteFrame()函数的原型之中，各参数的含义如下：

※**writer**：视频写入器结构。

※**image**：被写入的帧。

第二章 循环服务器软件的实现原理与方法

（12）OpenCV视频处理函数应用举例

例 1：播放硬盘中的视频文件

```
#include "highgui.h"
int main( int argc, char** argv ) {
    /**以下语句用于调用 cvNamedWindow()函数创建图像显示窗口**/
    cvNamedWindow("Example2", CV_WINDOW_AUTOSIZE);
    /**以下语句用于调用 cvCreateFileCapture()函数读入 AVI 视频文件**/
    CvCapture* capture = cvCreateFileCapture(argv[1]);
    IplImage* frame;
    /**以下 while(1)循环用于从视频中循环获取帧图像并显示**/
```

第二章 循环服务器软件的实现原理与方法

```
while(1) {
    frame = cvQueryFrame(capture);  //从视频中获取1帧图像
    if( !frame ) break;
    cvShowImage("Example2", frame);  //在窗口显示该帧图像
    char c = cvWaitKey(33);          //当前帧被显示后等待 33 ms
    if( c == 27 ) break;  //用户按ESC键(ASCII值为27)退出循环
 }
cvReleaseCapture(&capture);  //释放CvCapture 结构占用资源
cvDestroyWindow("Example2");  //关闭图像显示窗口
return 0;
}
```

第二章 循环服务器软件的实现原理与方法

例 2： 播放摄像机采集的视频数据

```
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

int main(int argc, char** argv){
    IplImage* pFrame=NULL;                          //声明 IplImage 指针
    CvCapture* pCapture=cvCreateCameraCapture(-1);  //获取任意摄像头
    cvNamedWindow("video", 1);                      //创建图像显示窗口
    /**以下 while(1)循环用于从视频中循环获取帧图像并显示**/
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
    pFrame=cvQueryFrame(pCapture);          //从视频中获取 1 帧图像
    if(!pFrame)break;
    cvShowImage("video",pFrame);            //在窗口中显示该帧图像
    char c=cvWaitKey(33);                   //当前帧被显示后等待 33 ms
    if(c==27)break;          //用户按 ESC 键（ASCII 值为 27）退出循环
}
cvReleaseCapture(&pCapture);       //释放为 CvCapture 结构占用的资源
cvDestroyWindow("video");          //关闭图像显示窗口
return 0;
}
```

第二章 循环服务器软件的实现原理与方法

例 3：读入一个彩色视频文件并以灰度格式输出这个视频文件

```
#include "cv.h"
#include <highgui.h>
int main(int argc, char* argv[]){
    CvCapture* capture = 0;
    capture=cvCreateFileCapture(argv[1]);    //argv[1]中保存输入视频文件名
    if(!capture){
        return -1;
    }
    IplImage *bgr_frame=cvQueryFrame(capture); //从视频中获取 1 帧
    /*以下语句用于获取视频的帧速率*/
    double fps = cvGetCaptureProperty(capture,CV_CAP_PROP_FPS);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于获取视频的帧高和帧宽*/
CvSize size = cvSize((int)cvGetCaptureProperty(capture, CV_CAP_PROP_
FRAME_WIDTH), (int)cvGetCaptureProperty(capture, CV_CAP_PROP_ FRAME_
HEIGHT));
/*以下语句用于创建视频文件写入器，argv[2]中保存输出视频文件名*/
CvVideoWriter *writer = cvCreateVideoWriter(argv[2], CV_FOURCC
('M', 'J', 'P', 'G'), fps,size);
/*以下语句用于创建用于保存灰度图像的首地址并分配存储空间*/
IplImage* logpolar_frame = cvCreateImage(size,IPL_DEPTH_8U,3);
while((bgr_frame=cvQueryFrame(capture)) != NULL){
    cvLogPolar(bgr_frame,logpolar_frame,cvPoint2D32f(bgr_frame->widt
h/2,bgr_frame->height/2),40,CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS);
```

第二章 循环服务器软件的实现原理与方法

```
    //调用 cvLogPolar()函数将帧图像映射到极指数空间
    cvWriteFrame(writer, logpolar_frame);      //写入一帧到视频文件中
  }
  cvReleaseVideoWriter(&writer);
  cvReleaseImage(&logpolar_frame);
  cvReleaseCapture(&capture);
  return(0);
}
```

第二章 循环服务器软件的实现原理与方法

2.4.2 UNIX/LINUX环境下基于TCP套接字的例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号10000）发送连接请求，连接建立之后，向服务器发送“HELP”请求，在得到服务器的应答之后，首先在显示屏上回显服务器的应答消息，然后发送两个数字和一个算术四则运算符（+，–，*，/）给服务器请求服务器给出计算结果，最后，在收到服务器回送的计算结果之后，将计算结果在显示屏上回显，然后中断本次通信过程。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“HELP”请求之后，则回送“OK”作为应答，在收到了客户发送的两个数字和一个算术四则运算符之后，则首先计算出相关结果，然后再将结果作为应答回送给客户并中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

1. TCP 服务器端例程剖析

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include <malloc.h>
#define SERVER_PORT 10000          //定义端口号为 10000
#define QUEUE 20                   //定义等待队列长度为 20

int main(){
        int msock;                 //声明主套接字描述符变量
        int ssock;                 //声明从套接字描述符变量
```

第二章 循环服务器软件的实现原理与方法

```
int ret, num=0;
char buf[1024];                    //声明保存 HELP 消息的变量
char *buffer ="ok";                //声明保存 OK 消息的变量
double *i;                  //声明保存客户发送的第 1 个数字消息的变量
i = (double*)malloc(sizeof(double));
char *j;                    //声明保存客户发送的四则运算符消息的变量
j = (char*)malloc(sizeof(char));
double *k;                  //声明保存客户发送的第 2 个数字消息的变量
k = (double*)malloc(sizeof(double));
```

第二章 循环服务器软件的实现原理与方法

```
double result=0;          //声明保存计算结果的变量
↵
struct sockaddr_in servaddr; //声明服务器端套接字端点地址结构体变量
struct sockaddr_in clientaddr; //声明客户端套接字端点地址结构体变量
msock = socket(AF_INET , SOCK_STREAM , 0 );        //创建主套接字
if (msock<0){                                      //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
memset(&servaddr,0,sizeof(servaddr));
/*以下语句用于给服务器端主套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                     //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给 IP 地址字段赋值
servaddr.sin_port=htons(SERVER_PORT);              //给端口号字段赋值
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于调用 bind()函数将套接字与端点地址绑定*/
ret=bind(msock,(struct sockaddr*)&servaddr,sizeof(struct sockaddr_in));
if(ret<0){                                          //调用 bind()函数出错
        printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
        exit(-1);
}
/*以下语句调用 listen()函数设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
if(ret<0){                                          //调用 listen()函数出错
        printf("Listen Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下 while(1)无限循环用于循环读取来自不同客户端的连接请求*/
while(1){
    memset(&clientaddr,0,sizeof(clientaddr));
    int len = sizeof(clientaddr);
    /*以下语句调用 accept()函数接受客户连接请求并创建从套接字*/
    ssock = accept(msock,(struct sockaddr *)&clientaddr,&len);
    if(ssock<0){                                    //调用 accept()函数出错
        printf("Accept Failed!\n");
        break;
    }
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于调用 recv()/send()函数基于从套接字与客户交互*/
memset(buf, '\0', sizeof(buf));
num=0;
num=recv(ssock,buf,sizeof(buf),0);    //接收客户发送的 HELP 消息.
if (num<0){
        printf("Recieve Data Failed!\n");
        break;
}
printf("%s\n", buf);
num=0;
num=send(ssock,buffer,strlen(buffer),0);//服务器端回送 OK 作为应答
if(num!=strlen(buffer)){                          //调用 send()函数出错
        printf("Send Data Failed!\n");
        break;
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=recv(ssock, i, sizeof (double),0); //接收客户发的第 1 个数字
if (num<0){
        printf("Recieve Data Failed!\n");
        break;
}
num=0;
num=recv(ssock, j, sizeof (char),0);        //接收客户发的四则运算符
if (num<0){
        printf("Recieve Data Failed!\n");
        break;
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=recv(ssock, k, sizeof (double),0);      //接收客户发的第 2 个数字
if (num<0){
        printf("Recieve Data Failed!\n");
        break;
}
printf("接收到的客户端数据为： i = %f, j = '%c', k = %f.\n",*i,*j,*k);

/*以下语句根据四则运算符计算相应结果*/
switch(*j) {
    case '+':                          //加法
        result = *i + *k;
        break;
```

第二章 循环服务器软件的实现原理与方法

```
        case '-':                       //减法
            result = *i - *k;
            break;
        case '*':                       //乘法
            result = (*i) * (*k);
            break;
        default:                        //除法
            result = (*i) / (*k);
            break;
    }
    num=0;
```

第二章 循环服务器软件的实现原理与方法

```
        num=send(ssock,&result,sizeof(double),0);      //结果回送给客户
        if(num<0){                                     //调用 send()函数出错
            printf("Send Data Failed!\n");
            break;
        }
        close(ssock);                                  //与客户互完毕，将从套接字关闭
    }
    close(msock);                                      //服务器端退出，将主套接字关闭
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2. TCP 客户端例程剖析

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
#include <malloc.h>
#define SERVERIP "172.0.0.1"          //定义 IP 地址常量
#define SERVERPORT 10000              //定义端口号常量

int main(){
    int ret, num=0;
    int tsock;                        //声明客户端套接字描述符变量
```

第二章 循环服务器软件的实现原理与方法

```
double *i = 0;                          //声明保存第 1 个数字消息的变量
i = (double*)malloc(sizeof(double));
char *j = '\0';                         //声明保存四则运算符消息的变量
j = (char*)malloc(sizeof(char));
double *k = 0;                          //声明保存第 2 个数字消息的变量
k = (double*)malloc(sizeof(double));
double result = 0;                      //声明保存计算结果消息的变量
char *buffer = "HELP";                  //声明保存 HELP 消息的变量
char buf[20];                           //声明接收服务器应答消息的变量
struct sockaddr_in servaddr;            //声明服务器套接字端点地址结构体变量
```

第二章 循环服务器软件的实现原理与方法

```
tsock=socket(AF_INET,SOCK_STREAM,0);        //创建客户端套接字
if (tsock<0){                               //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
memset(&servaddr,0,sizeof(servaddr));
/*以下语句用于给服务器端套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;              //给协议族字段赋值
servaddr.sin_port = htons(SERVERPORT);      //给端口号字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);     //给 IP 地址字段赋值
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句调用 connect()函数向远程服务器发起 TCP 连接建立请求*/
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                          //调用 connect()函数出错
        printf("Connect Failed!\n");
        exit(-1);
}
/*以下语句用于调用 send()函数利用从套接字发送数据给服务器端*/
num=send(tsock,buffer,strlen(buffer),0);    //发送 HELP 消息给服务器端
if(num!=strlen(buffer)){                            //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句调用 recv()函数从套接字读取服务器端发送过来的数据*/
num=0;
num=recv(tsock,buf,sizeof(buf),0);      //接收服务器端的应答消息 OK
if (num<0){
        printf("Recieve Data Failed!\n");
        exit(-1);
}
printf("%s\n", buf);

*i = 1.0;
*j = '+';
*k = 2.0;
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=send(tsock,i,sizeof(double),0);          //发送第 1 个数字给服务器
if(num<0){                                    //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
num=0;
num=send(tsock,j,sizeof(char),0);            //发送四则运算符给服务器
if(num<0){                                    //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=send(tsock,k,sizeof(double),0);        //发送第 2 个数字给服务器
if(num<0){                                  //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
num=0;
num=recv(tsock,&result,sizeof(double),0); //接收服务器回送的计算结果
```

第二章 循环服务器软件的实现原理与方法

```
if (num<0){
        printf("Recieve Data Failed!\n");
        exit(-1);
}
printf("result : %f\n",result);

close(tsock);

free(i);
free(j);
free(k);
return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2.4.3 Windows环境下基于TCP套接字的例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号10000）发送连接请求，连接建立之后，向服务器发送“HELP”请求，在得到服务器的应答之后，首先在显示屏上回显服务器的应答消息，然后发送两个数字和一个算术四则运算符（+，–，*，/）给服务器请求服务器给出计算结果，最后，在收到服务器回送的计算结果之后，将计算结果在显示屏上回显，然后中断本次通信过程。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“HELP”请求之后，则回送“OK”作为应答，在收到了客户发送的两个数字和一个算术四则运算符之后，则首先计算出相关结果，然后再将结果作为应答回送给客户并中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

1. TCP 服务器端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
#include <windows.h>
#include <winsock2.h>
#include<string.h>
#include <malloc.h>

#pragma comment(lib,"ws2_32.lib")

int main(){
    SOCKET msock;                    //声明服务器端主套接字描述符变量
    SOCKET ssock;                    //声明服务器端从套接字描述符变量
```

第二章 循环服务器软件的实现原理与方法

```
int ret,num=0;
char buf [1024];                    //声明保存客户 HELP 消息的变量
char * buffer ="ok";                //声明保存 OK 应答消息的变量

double *i;                  //声明保存客户发送的第 1 个数字消息的变量
i = (double*)malloc(sizeof(double));
char *j;                    //声明保存客户发送的四则运算符消息的变量
j = (char*)malloc(sizeof(char));
double *k;                  //声明保存客户发送的第 2 个数字消息的变量
k = (double*)malloc(sizeof(double));
double result=0;            //声明保存计算结果的变量
struct sockaddr_in servaddr; //声明服务器套接字端点地址结构体变量
struct sockaddr_in clientaddr; //声明客户端套接字端点地址结构体变量
/*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
```

第二章 循环服务器软件的实现原理与方法

```
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
}
msock=socket(AF_INET,SOCK_STREAM,0); //创建服务器端主套接字
if(msock == INVALID_SOCKET){                    //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下语句用于给服务器端主套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                          //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);     //给 IP 地址字段赋值
servaddr.sin_port = htons(10000);                       //给端口号字段赋值
/*以下语句用于调用 bind()函数将套接字绑定到端点地址*/
ret=bind(msock,(struct sockaddr*)&servaddr,sizeof(struct sockaddr_in));
if(ret<0){                                                      //调用 bind()函数出错
        printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
        exit(-1);
}
listen(msock,20);
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
    ZeroMemory(&clientaddr,sizeof(clientaddr));
    int len = sizeof(clientaddr);
    ssock = accept(msock,(struct sockaddr *)&clientaddr,&len);
    if(ssock == INVALID_SOCKET){          //调用 accept()函数出错
        printf("Accept Failed!\n");
        exit(-1);
    }
    /*以下语句调用 recv/send()函数基于从套接字与客户交互*/
    ZeroMemory(buf,sizeof(buf));
    num=0;
    num=recv(ssock,buf,sizeof(buf),0);   //接收客户发送的 HELP 消息
```

第二章 循环服务器软件的实现原理与方法

```
if (num<0){
        printf("Recieve Data Failed!\n");
        exit(-1);
}
printf("%s.\n",buf);
num=0;
num=send(ssock,buffer,strlen(buffer),0);          //回送 OK 作为应答
if(num<0){                                        //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=recv(ssock, i, sizeof (double),0);      //接收客户发的第 1 个数字
if(num<0){                                  //调用 send()函数出错
        printf("Recieve Data Failed!\n");
        exit(-1);
}
num=0;
num=recv(ssock, j, sizeof (char),0);        //接收客户发的四则运算符
if(num<0){                                  //调用 send()函数出错
        printf("Recieve Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=recv(ssock, k, sizeof (double),0);      //接收客户发的第 2 个数字
if(num<0){                                  //调用 send()函数出错
        printf("Recieve Data Failed!\n");
        exit(-1);
}
printf("接收到的数据为： i = %f, j = '%c', k = %f\n",*i,*j,*k);

/*以下语句根据四则运算符计算相应结果*/
switch(*j) {
    case '+':                               //加法
        result = *i + *k;
        break;
```

第二章 循环服务器软件的实现原理与方法

```
        case '-':                          //减法
            result = *i - *k;
            break;
        case '*':                          //乘法
            result = (*i) * (*k);
            break;
        default:                           //除法
            result = (*i) / (*k);
            break;
    }
    num=0;
    num=send(ssock,&result,sizeof(double),0); //将计算结果回送给客户
```

第二章 循环服务器软件的实现原理与方法

```
        if(num<0){                                  //调用 send()函数出错
            printf("Send Data Failed!\n");
            exit(-1);
        }
        closesocket(ssock);                         //与客户交互完毕后关闭从套接字
    }
    closesocket(msock);                             //与所有客户交互完毕后关闭主套接字
    WSACleanup();                                   //结束 Winsock Socket API
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2. TCP 客户端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
#include <windows.h>
#include <winsock2.h>
#include<string.h>
#include <malloc.h>

#pragma comment(lib,"ws2_32.lib")

#define SERVERIP "172.0.0.1" //定义 IP 地址常量
#define SERVERPORT 10000   //定义端口号常量
```

第二章 循环服务器软件的实现原理与方法

```
int main(){
    SOCKET tsock;                          //声明客户端套接字描述符变量
    double *i = 0;                         //声明保存第 1 个数字消息的变量
    i = (double*)malloc(sizeof(double));
    char *j = '\0';                        //声明保存四则运算符消息的变量
    j = (char*)malloc(sizeof(char));
    double *k = 0;                         //声明保存第 2 个数字消息的变量
    k = (double*)malloc(sizeof(double));
    double result = 0;                     //声明保存计算结果消息的变量
    char *buffer = "HELP";                 //声明保存 HELP 消息的变量
    char buf[1024];                        //声明保存服务器应答消息的变量
    struct sockaddr_in servaddr;           //声明服务器套接字端点地址结构体变量
    int ret, num=0;
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
  WORD sockVersion = MAKEWORD(2,2);
  WSADATA wsaData;
  ret=WSAStartup(sockVersion, &wsaData);
  if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
  }
tsock=socket(AF_INET,SOCK_STREAM,0);      //创建客户端套接字
```

第二章 循环服务器软件的实现原理与方法

```
if (tsock == INVALID_SOCKET){                    //调用 socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下语句用于给服务器套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                   //给协议族字段赋值
servaddr.sin_port = htons(SERVERPORT);           //给端口号字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);          //给 IP 地址字段赋值
/*以下语句调用 connect()函数向远程服务器发起连接建立请求*/
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                       //调用 connect()函数出错
    printf("Connect Failed!\n");
    exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于调用 send()函数利用从套接字发送数据给服务器端*/
num=0;
num=send(tsock,buffer,strlen(buffer),0);        //发送 HELP 消息给服务器端
if(num!=strlen(buffer)){                        //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
/*以下语句调用 recv()函数从套接字读取服务器端发送过来的数据*/
num=0;
num=recv(tsock,buf,sizeof(buf),0);          //接收服务器端的应答消息 OK
if (num<0){
      printf("Recieve Data Failed!\n");
      exit(-1);
}
printf("%s.\n", buf);
```

第二章 循环服务器软件的实现原理与方法

```
printf("请输入参与计算的第 1 个数字：\n");
scanf("%lf", i);
printf("请输入四则运算符：\n");
scanf("%c", j);
printf("请输入参与计算的第 2 个数字：\n");
scanf("%lf", k);
num=0;
num=send(tsock,i,sizeof(double),0);          //发送第 1 个数字给服务器端
if(num<0){                                   //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=send(tsock,j,sizeof(char),0);          //发送四则运算符给服务器端
if(num<0){                                 //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
num=0;
num=send(tsock,k,sizeof(double),0);        //发送第 2 个数字给服务器端
if(num<0){                                 //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
num=recv(tsock,&result,sizeof(double),0); //接收服务器回送的计算结果
if(num<0){                                 //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
printf("result：  %f.\n",result);

closesocket(tsock);
```

```
        free(i);
        free(j);
        free(k);
        WSACleanup(); //结束 Winsock Socket API
        return 0;
    }
```

2.4.4 UNIX/LINUX环境下基于UDP套接字的例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号10000）发送连接请求，连接建立之后，向服务器发送“HELP”请求，在得到服务器的应答之后，首先在显示屏上回显服务器的应答消息，然后发送两个数字和一个算术四则运算符（+，–，*，/）给服务器请求服务器给出计算结果，最后，在收到服务器回送的计算结果之后，将计算结果在显示屏上回显，然后中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“HELP”请求之后，则回送“OK”作为应答，在收到了客户发送的两个数字和一个算术四则运算符之后，则首先计算出相关结果，然后再将结果作为应答回送给客户并中断本次通信过程。

1. UDP 服务器端例程剖析

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include <malloc.h>
```

第二章 循环服务器软件的实现原理与方法

```
/*定义存储客户发送的 2 个数字和 1 个四则运算符消息的结构体 Node*/
typedef struct{
        double i;
        char j;
        double k;
} Node;

int main(){
```

第二章 循环服务器软件的实现原理与方法

```
int msock;                          //声明服务器端 UDP 套接字描述符变量
char buf[20];                       //声明保存客户 HELP 消息的变量
char * buffer ="ok";                //声明保存 OK 应答消息的变量
int ret,num=0;
/*以下语句用于声明存储客户端发送的 2 个数字和 1 个四则运算符消息
  的结构体变量，并对其赋初值*/
Node node;
node.i = 0;
node.j = '\0';
node.k = 0;
double result=0; //声明用于保存计算结果的变量
struct sockaddr_in servaddr; //声明服务器端套接字端点地址结构体变量
struct sockaddr_in clientaddr; //声明客户端套接字端点地址结构体变量
```

第二章 循环服务器软件的实现原理与方法

```
msock=socket(AF_INET,SOCK_DGRAM,0); //创建服务器端 UDP 套接字
if (msock<0){                       //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
memset(&servaddr,0,sizeof(servaddr));
/*以下语句用于给服务器套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                  //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给 IP 地址字段赋值
servaddr.sin_port = htons(10000);               //给端口号字段赋值
/*以下语句用于调用 bind()函数将套接字与端点地址绑定*/
ret=bind(msock,(struct sockaddr*)&servaddr,sizeof(struct sockaddr_in));
if(ret<0){                                      //调用 bind()函数出错
      printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
      exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
    memset(&clientaddr,0,sizeof(clientaddr));
    int len = sizeof(clientaddr);
    memset(buf,'\0',sizeof(buf));
    num=0;
    /*以下语句用于服务器接收客户发送的 HELP 请求消息*/
    num=recvfrom(msock,buf,sizeof(buf),0,(struct sockaddr*)&clientaddr,
&len);
    if(num<0){                                    //调用 recvfrom()函数出错
        printf("Receive Data Failed!\n");
        break;
    }
    printf("%s.\n",buf);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于服务器回送 OK 给客户作为应答*/
num=0;
num = sendto(msock, buffer, strlen(buffer), 0, (struct sockaddr *)
&clientaddr, &len);
if(num != strlen(buffer)){                    //调用 sendto()函数出错
        printf("Send Data Failed!\n");
        break;
}
num=0;
num = recvfrom(msock, &node, sizeof(Node), 0, (struct sockaddr *)
&clientaddr, &len); //接收客户发送的 2 个数字和 1 个运算符消息
if(num<0){                                    //调用 recvfrom()函数出错
        printf("Receive Data Failed!\n");
        break;
}
printf("接收到数据：i = %f, j = '%c', k = %f\n", node.i, node.j, node.k);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句根据四则运算符计算相应结果*/
switch(node.j) {
        case '+':
                result = node.i + node.k;
                break;
        case '-':
                result = node.i - node.k;
                break;
        case '*':
                result = node.i * node.k;
                break;
        default:
                result = node.i / node.k;
                break;
}
```

第二章 循环服务器软件的实现原理与方法

```
            num=0;
            num = sendto(msock, &result, sizeof(double), 0, (struct sockaddr *)
&clientaddr, &len); //将计算结果作为应答回送给客户
            if(num<0){                                    //调用 sendto()函数出错
                printf("Send Data Failed!\n");
                break;
            }
        }
        close(msock);
        return 0;
    }
```

第二章 循环服务器软件的实现原理与方法

2. UDP 客户端例程剖析

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
#include <malloc.h>
#define SERVERIP "172.0.0.1" //定义 IP 地址常量
#define SERVERPORT 10000    //定义端口号常量

/*定义存储客户发送的 2 个数字和 1 个四则运算符消息的结构体 Node*/
typedef struct{
```

第二章 循环服务器软件的实现原理与方法

```
        double i;
        char j;
        double k;
} Node;

int main(){
    int tsock;                                  //声明客户端套接字描述符变量
    /*以下语句用于声明存储客户端发送的 2 个数字和 1 个四则运算符消息
      的结构体变量，并对其赋初值*/
    Node node;
    node.i = 0;
    node.j = '\0';
    node.k = 0;
```

第二章 循环服务器软件的实现原理与方法

```
double result = 0;                          //声明保存计算结果消息的变量
char *buffer = "HELP";                      //声明保存 HELP 消息的变量
char buf[1024]                              //声明保存服务器应答消息的变量
struct sockaddr_in servaddr;     //声明服务器套接字端点地址结构体变量

tsock=socket(AF_INET, SOCK_DGRAM,0);    //创建客户端面 UDP 套接字
if (tsock<0){                                //调用 socket()函数出错
      printf("Create Socket Failed!\n");
      exit(-1);
}
memset(&servaddr,0,sizeof(servaddr));
/*以下语句用于给服务器套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                  //给协议族字段赋值
servaddr.sin_port = htons(SERVERPORT);      //给端口号字段赋值
inet_aton(SERVERIP,& servaddr.sin_addr);     //给 IP 地址字段赋值
```

第二章 循环服务器软件的实现原理与方法

```
int ret, num=0;
int len=sizeof(servaddr);
/*以下语句用于客户端发送 HELP 消息给服务器*/

num = sendto(tsock, buffer, strlen(buffer), 0, (struct sockaddr *)&servaddr,
&len);
/*以下语句用于客户端接收服务器的应答消息. OK*/
num=0;
num=recvfrom(tsock,buf,sizeof(buf),0,(struct sockaddr *)&servaddr,&len);
if(num<0){                                          //调用 recvfrom()函数出错
        printf("Receive Data Failed!\n");
        exit(-1);
}
printf("%s.\n", buf);
```

第二章 循环服务器软件的实现原理与方法

```
node.i = 1.0;
node.j = '-';
node.k = 3.0;
/*以下语句用于客户端发送第 1 个数字给服务器*/
num=0;
num=sendto(tsock, &node, sizeof(Node), 0, (struct sockaddr *)&servaddr,
&len);
if(num<0){                                          //调用 sendto()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
        num=0;
        num=recvfrom(tsock,&result,sizeof(double),0, (struct sockaddr *)&servaddr,
&len);                                          //接收服务器回送的计算结果
        if(num<0){                              //调用 recvfrom()函数出错
            printf("Receive Data Failed!\n");
            exit(-1);
        }
        printf("resul：  %f\n", result);
        close(tsock);
        return 0;
    }
```

第二章 循环服务器软件的实现原理与方法

2.4.5 Windows环境下基于UDP套接字的例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号10000）发送连接请求，连接建立后，向服务器发送“HELP”请求，在得到服务器的应答后，首先在显示屏上回显服务器的应答消息，然后发送两个数字和一个算术四则运算符（+，–，*，/）给服务器请求服务器给出计算结果，最后，在收到服务器回送的计算结果后，将计算结果在显示屏上回显，然后中断本次通信过程。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“HELP”请求之后，则回送“OK”作为应答，在收到了客户发送的两个数字和一个算术四则运算符之后，则首先计算出相关结果，然后再将结果作为应答回送给客户并中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

1. UDP 服务器端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
#include <windows.h>
#include <winsock2.h>
#include<string.h>
#include <malloc.h>

#pragma comment(lib,"ws2_32.lib")

/*定义存储客户发送的 2 个数字和 1 个四则运算符消息的结构体 Node*/
typedef struct{
        double i;
        char j;
        double k;
} Node;
```

第二章 循环服务器软件的实现原理与方法

```
int main(){
    SOCKET msock;                //声明服务器端 UDP 套接字描述符变量
    char buf[1024];              //声明保存 HELP 请求消息的变量
    char * buffer ="ok";         //声明保存 OK 应答消息的变量

    /*以下语句用于声明存储客户端发送的 2 个数字和 1 个四则运算符消息
      的结构体变量，并对其赋初值*/
    Node node;
    node.i = 0;
    node.j = '\0';
    node.k = 0;
```

第二章 循环服务器软件的实现原理与方法

```
double result=0;                                    //声明保存计算结果的变量
struct sockaddr_in servaddr;      //声明服务器套接字端点地址结构体变量
struct sockaddr_in clientaddr;    //声明客户端套接字端点地址结构体变量
/*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
int ret, num;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
msock=socket(AF_INET,SOCK_DGRAM,0); //创建服务器端 UDP 套接字
if(msock == INVALID_SOCKET){          //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下语句用于给服务器套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                    //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);       //给 IP 地址字段赋值
servaddr.sin_port = htons(10000);                 //给端口号字段赋值
/*以下语句用于调用 bind()函数将套接字与端点地址绑定*/
ret=bind(msock,(struct sockaddr*)&servaddr,sizeof(struct sockaddr_in));
if(ret<0){                                        //调用 bind()函数出错
        printf("Bind Socket Failed!\n");
```

第二章 循环服务器软件的实现原理与方法

```
            exit(-1);
        }
        while(1){
            ZeroMemory(&clientaddr,sizeof(clientaddr));
            ZeroMemory(buf,sizeof(buf));
            int len = sizeof(clientaddr);
            /*以下语句用于服务器接收客户发送的 HELP 请求消息*/
            num=0;
            num = recvfrom(msock, buf, sizeof(buf), 0, (struct sockaddr *)
&clientaddr, &len);
            if(num<0){                                  //调用 recvfrom()函数出错
                printf("Receive Data Failed!\n");
                break;
            }
            printf("%s\n",buf);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于服务器回送 OK 给客户作为应答*/
num=0;
num = sendto(msock, buffer, strlen(buffer), 0, (struct sockaddr *)
&clientaddr, &len);
if(num != strlen(buf)){                          //调用 sendto()函数出错
        printf("Send Data Failed!\n");
        break;
}
num=0;
num=recvfrom(msock, &node, sizeof(Node), 0, (struct sockaddr *)
&clientaddr, &len);                  //接收客户发送的 2 个数字和 1 个运算符消息
if(num<0){                                       //调用 recvfrom()函数出错
        printf("Receive Data Failed!\n");
        break;
}
printf("接收到数据：i = %f, j = '%c', k = %f\n", node.i, node.j, node.k);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句根据四则运算符计算相应结果*/
switch(node.j) {
        case '+':
                result = node.i + node.k;
                break;
        case '-':
                result = node.i - node.k;
                break;
        case '*':
                result = node.i * node.k;
                break;
        default:
                result = node.i / node.k;
                break;
}
```

第二章 循环服务器软件的实现原理与方法

```
            num=0;
            num=sendto(msock, &result, sizeof(double), 0, (struct sockaddr *)
&clientaddr, &len);                          //将计算结果作为应答回送给客户
            if(num<0){                       //调用 sendto()函数出错
                printf("Send Data Failed!\n");
                break;
            }
        }
        closesocket(msock);                  //关闭套接字
        WSACleanup();                        //结束 Winsock Socket API
        return 0;
    }
```

第二章 循环服务器软件的实现原理与方法

2. UDP 客户端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
#include <windows.h>
#include <winsock2.h>
#include<string.h>
#include <malloc.h>

#pragma comment(lib,"ws2_32.lib")

#define SERVERIP "172.0.0.1" //定义 IP 地址常量
#define SERVERPORT 10000   //定义端口号常量
```

第二章 循环服务器软件的实现原理与方法

```
/定义存储客户发送的 2 个数字和 1 个四则运算符消息的结构体 Node*/
typedef struct{
        double i;
        char j;
        double k;
} Node;

int main(){
    SOCKET tsock;                          //声明客户端套接字描述符变量
    /*以下语句用于声明存储客户端发送的 2 个数字和 1 个四则运算符消息
     的结构体变量，并对其赋初值*/
     Node node;
     node.i = 0;
     node.j = '\0';
     node.k = 0;
```

第二章 循环服务器软件的实现原理与方法

```
double result = 0;                          //声明保存计算结果消息的变量
char *buffer = "HELP";                      //声明保存 HELP 消息的变量
char buf[1024];                             //声明保存服务器应答消息的变量
struct sockaddr_in servaddr;     //声明服务器套接字端点地址结构体变量
/*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
int ret, num;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
```

第二章 循环服务器软件的实现原理与方法

```
}
tsock=socket(AF_INET, SOCK_DGRAM,0);      //创建客户端 UDP 套接字
if (tsock == INVALID_SOCKET){              //调用 socket()函数出错
        printf("Create Socket Failed!\n");
        exit(-1);
}
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下语句用于给服务器套接字端点地址变量赋值*/
servaddr.sin_family = AF_INET;                    //给协议族字段赋值
servaddr.sin_port = htons(SERVERPORT);            //给端口号字段赋值
inet_aton(SERVERIP,& servaddr.sin_addr);          //给 IP 地址字段赋值
ZeroMemory(buf,sizeof(buf));
int len=sizeof(servaddr);
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句调用客户端发送 HELP 消息给服务器端*/
num=0;
num=sendto(tsock,buf,strlen(buf),0, (struct sockaddr *)&servaddr,&len);
if(num != strlen(buf)){                                    //调用 sendto()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
/*以下语句用于客户端接收服务器端的应答消息, OK*/
num=0;
num=recvfrom(tsock,buf,sizeof(buf),0, (struct sockaddr *)&servaddr,&len);
if(num<0){                                                 //调用 recvfrom()函数出错
        printf("Receive Data Failed!\n");
        exit(-1);
}
printf("%s.\n", buf);
```

第二章 循环服务器软件的实现原理与方法

```
node.i = 1.0;
node.j = '-';
node.k = 3.0;
/*以下语句用于客户端发送第 1 个数字给服务器端*/
num=0;
num = sendto(tsock, &node, sizeof(Node), 0, (struct sockaddr *)
&servaddr, &len);
if(num<0){                                          //调用 sendto()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
num=0;
```

第二章 循环服务器软件的实现原理与方法

```
        num = recvfrom(tsock, &result, sizeof(double), 0, (struct sockaddr *)
&servaddr, &len);                                    //接收服务器回送的计算结果
        if(num<0){                                   //调用 recvfrom()函数出错
            printf("Receive Data Failed!\n");
            exit(-1);
        }
        printf("result: %f\n", result);

        closesocket(clientfd);
        WSACleanup();                                //结束 Winsock Socket API
        return 0;
    }
```

第二章 循环服务器软件的实现原理与方法

2.4.6 UNIX/LINUX环境下基于TCP套接字的文件传输例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号6666）发送连接请求，连接建立之后，向服务器发送“希望下载的文件的文件名“请求信息，然后，在接收完毕服务器回送的文件内容并保存到本地文件之中，中断本次与客户的通信过程。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“希望下载的文件的文件名”请求之后，则打开该文件，然后再将该文件的内容作为应答回送给客户并中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

1. TCP 服务器端例程剖析

```
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define SERVER_PORT 10000
#define QUEUE 20
#define BUFFER_SIZE 1024
#define FILE_NAME_MAX_SIZE 512
```

第二章 循环服务器软件的实现原理与方法

```
int main(int argc, char **argv){
    struct sockaddr_in   servaddr;     //声明套接字端点地址结构体变量
    memset(&servaddr, 0, sizeof(servaddr));
    /*以下 3 条语句用于给端点地址结构体变量 servaddr 赋值*/
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htons(INADDR_ANY);
    servaddr.sin_port = htons(SERVER_PORT);

    int msock = socket(AF_INET, SOCK_STREAM, 0);   //创建 TCP 套接字
    if (msock < 0){
        printf("Create Socket Failed!\n");
        exit(-1);
    }
```

第二章 循环服务器软件的实现原理与方法

```
int ret, num;
/*以下语句用于调用 bind()函数将主套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));
if(ret<0){                                          //调用 bind()函数出错
        printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
        exit(-1);
}
/*以下语句用于调用 listen()函数设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
if(ret<0){                                          //调用 listen()函数出错
        printf("Listen Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
    /*定义客户端的 socket 地址结构 clientaddr，当收到来自客户端的请
      求后，调用 accept()函数接受此请求，同时将 client 端的地址和端
      口等信息写入 clientaddr 中*/
    struct sockaddr_in clientaddr;
    int len = sizeof(clientaddr);
    int ssock = accept(msock, (struct sockaddr*) &clientaddr, &len);
    if (ssock < 0){
        printf("Accept Failed!\n");
        break;
    }
```

第二章 循环服务器软件的实现原理与方法

```
char buffer[BUFFER_SIZE];
memset(buffer, '\0', sizeof(buffer));
/*以下语句用于接收客户端传送过来的文件名并存储到缓存区中*/
num=0;
num = recv(ssock, buffer, BUFFER_SIZE, 0);
if (num< 0){
        printf("Recieve Data Failed!\n");
        break;
}
char file_name[FILE_NAME_MAX_SIZE + 1];
memset(file_name, '\0', sizeof(file_name));
/*将文件名从缓存区 buffer 拷贝到数组 file_name 中*/
```

第二章 循环服务器软件的实现原理与方法

```
        strncpy(file_name, buffer, strlen(buffer) > FILE_NAME_MAX_SIZE ?
FILE_NAME_MAX_SIZE : strlen(buffer));
        FILE *fp = fopen(file_name, "r");    //打开客户指定要发送的该文件
        if (fp == NULL){
            printf("File:\t%s Not Found!\n", file_name);
        }
        else {
            memset(buffer, '\0', BUFFER_SIZE);
            int file_block_length = 0;
            /*以下 while 循环用于反复读取文件内容并存储到 buffer 中*/
            while((file_block_length = fread(buffer, sizeof(char), BUFFER_
SIZE, fp)) > 0) {
```

第二章 循环服务器软件的实现原理与方法

```
                printf("file_block_length = %d\n", file_block_length);
                /*将 buffer 中的字符串发送给客户端*/
                if(send(ssock, buffer, file_block_length,0)< 0){
                    printf("Send File:\t%s Failed!\n", file_name);
                    break;
                }
                memset(buffer, '\0', sizeof(buffer));
            }
            fclose(fp);                    //文件发送完毕，关闭文件描述符
            printf("File:\t%s Transfer Finished!\n", file_name);
        }
        close(ssock);
    }
    close(msock);
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2. TCP 客户端例程剖析

```
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define SERVER_PORT   10000
#define BUFFER_SIZE 1024
#define FILE_NAME_MAX_SIZE   512
```

第二章 循环服务器软件的实现原理与方法

```
int main(int argc, char **argv){
    if (argc != 2){
        printf("Usage: ./%s ServerIPAddress\n", argv[0]);
        exit(1);
    }
    int tsock = socket(AF_INET, SOCK_STREAM, 0); //创建客户端套接字
    if (tsock<0){                                //调用 socket()函数出错
       printf("Create Socket Failed!\n");
       exit(-1);
    }
    /*声明服务器端 socket 地址结构变量并给其赋值*/
```

第二章 循环服务器软件的实现原理与方法

```
struct sockaddr_in    servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;                          //给协议族字段赋值
if (inet_aton(argv[1], &servaddr.sin_addr) == 0){       //给 IP 地址字段赋值
        printf("Server IP Address Error!\n");
        exit(-1);
}
server_addr.sin_port = htons(SERVER_PORT);      //给端口号字段赋值
int len = sizeof(servaddr);
/*以下语句用于客户端向远程服务器发起 TCP 连接建立请求*/
```

第二章 循环服务器软件的实现原理与方法

```
int ret, num;
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                          //调用 connect()函数出错
        printf("Connect Failed!\n");
        exit(-1);
}
char file_name[FILE_NAME_MAX_SIZE + 1];
memset(file_name, '\0', sizeof(file_name));
printf("Please Input File Name On Server.\t");
scanf("%s", file_name);
char buffer[BUFFER_SIZE];
memset(buffer, '\0', sizeof(buffer));
```

第二章 循环服务器软件的实现原理与方法

```
        strncpy(buffer, file_name, strlen(file_name)>BUFFER_SIZE? BUFFER_
SIZE : strlen(file_name));
        /*向服务器发送 buffer 中的数据，此时 buffer 中存放的是客户端需要接
          收的文件的名字*/
        num=0;
        num=send(tsock, buffer, BUFFER_SIZE, 0);
        if(num!=strlen(buffer)){                    //调用 send()函数出错
            printf("Send Data Failed!\n");
            exit(-1);
        }
        FILE *fp = fopen(file_name, "w");           //创建并打开该文件
        if (fp == NULL){
            printf("File:\t%s Can Not Open To Write!\n", file_name);
            exit(-1);
        }
```

第二章 循环服务器软件的实现原理与方法

```
/*从服务器端接收数据到 buffer 中*/
memset(buffer, '\0', sizeof(buffer));
num=0;
/*若调用 recv()函数接收到的服务器应答内容长度>0，则循环接收服务器的应答。若等于 0 则退出循环，表示服务器的文件传输结束。若小于 0 则表示接收出错*/
while(num=recv(tsock, buffer, BUFFER_SIZE, 0) != 0){
    if (num < 0){
        printf("Recieve Data From Server %s Failed!\n", argv[1]);
        break;
    }
```

第二章 循环服务器软件的实现原理与方法

```
        //将收到的服务器应答内容写入文件
        int write_len = fwrite(buffer, sizeof(char), num, fp);
        if (write_len != num){
            printf("File:\t%s Write Failed!\n", file_name);
            break;
        }
        memset(buffer, '\0', BUFFER_SIZE);
    }//while 循环结束
    printf("RecieveFile:\t %s FromServer[%s]Finished!\n",file_name,argv[1]);

    fclose(fp);                 //传输完毕后关闭文件
    close(tsock);               //关闭套接字
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2.4.7 UNIX/LINUX环境下基于TCP套接字的音频传输例程剖析

客户端功能需求描述：向服务器（IP地址127.0.0.1，端口号6666）发送连接请求，在连接建立之后，首先，向服务器发送“希望下载的音频文件的文件名”请求信息，然后，在接收完毕服务器回送的音频内容并保存到本地的音频文件中之后，中断本次通信过程，最后，将收到的音频文件发送至声卡播放。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的“希望下载的音频文件的文件名”请求之后，首先创建一个以该文件名命名的音频文件，然后，从麦克风录制一段音频并保存到该音频文件，最后，在播放该音频文件之后，再将其通过套接字传送给客户端并中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

1. TCP 服务器端例程剖析

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
#include <linux/soundcard.h>
#include <termios.h>
#include<string.h>

#include<netinet/in.h>
#include<sys/socket.h>
```

第二章 循环服务器软件的实现原理与方法

```
#define LENGTH   10                    //录音时间（秒）
#define RATE   88200                   //采样频率
#define SIZE   16                      //量化位数
#define CHANNELS   2                   //声道数目
#define RSIZE   8                      //buf 的大小，

#define SERVER_PORT   10000
#define QUEUE    20
#define BUFFER_SIZE   1024
#define FILE_NAME_MAX_SIZE    512
```

第二章 循环服务器软件的实现原理与方法

```
/*定义 WAVE 文件的文件头结构体 wfhead，长度为 44 个字节*/
struct wfhead{
    /*RIFF WAVE CHUNK*/
    unsigned char a[4]; //4 字节，存放'R','I','F','F'
    long int b;    /*整个 WAVE 文件的长度减去 8 个字节，4 字节*/
    unsigned char c[4]; //4 字节，存放'W','A','V','E'
    /*Format CHUNK*/
    unsigned char d[4]; //4 字节，存放'f','m','t',''
    long int e;    //4 字节
    short int f;   //2 字节，编码方式，一般为 0x0001;
    short int g;   //2 字节，声道数目，1 为单声道，2 为双声道;
    long int h;    //4 字节，采样频率;
    long int i;    //4 字节，每秒所需字节数;
```

第二章 循环服务器软件的实现原理与方法

```
    short int j;    //2 字节，每个采样需要多少字节，若是 2 声道，则乘以 2
    short int k;    //2 字节，即量化位数
    /*Data Chunk**/
     unsigned char p[4]; //4 字节，存放'd','a','t','a'
     long int q;    //4 字节，语音数据部分长度，不包括文件头
}wavefilehead;

int main(int argc, char **argv){
    /*声明 socket 地址结构变量 server_addr 并给其赋值*/
    struct sockaddr_in servaddr;
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htons(INADDR_ANY);
    servaddr.sin_port = htons(SERVER_PORT);
```

第二章 循环服务器软件的实现原理与方法

```
int msock= socket(AF_INET, SOCK_STREAM, 0);      //创建 TCP 套接字
if (msock< 0){
        printf("Create Socket Failed!\n");
        exit(-1);
}
/*以下语句用于调用 bind()函数将主套接字与端点地址绑定*/
int ret, num;
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));
if(ret<0){                                          //调用 bind()函数出错
        printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
if(ret<0){                                    //调用 listen()函数出错
        printf("Listen Failed!\n");
        exit(-1);
}
/*定义客户端的 socket 地址结构 client_addr，当收到来自客户端的请求
  后，调用 accept()函数接受此请求，同时将 client 端的地址和端口等信
  息写入 client_addr 中*/
struct sockaddr_in clientaddr;
int len = sizeof(clientaddr);
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
        memset(&clientaddr, 0, len);
        int ssock=accept(msockt, (struct sockaddr*) &client_addr, &len);
        if (ssock<0){
                printf("Accept Failed!\n");
                break;
        }
        char buffer[BUFFER_SIZE];
        memset(buffer, '\0', sizeof(buffer));
```

第二章 循环服务器软件的实现原理与方法

```
num=0;
/*接收客户端传来的音频文件名并存储到缓存区 buffer 中*/
num = recv(ssock, buffer, BUFFER_SIZE, 0);
if (num<0){
        printf("Recieve Data Failed!\n");
        break;
}
char file_name[FILE_NAME_MAX_SIZE + 1];
memset(file_name, '\0', sizeof(file_name));
/*将音频文件名从缓存区 buffer 拷贝到数组 file_name 中*/
strncpy(file_name, buffer, strlen(buffer) > FILE_NAME_MAX_SIZE ?
FILE_NAME_MAX_SIZE : strlen(buffer));
```

第二章 循环服务器软件的实现原理与方法

```
FILE *fp = fopen(file_name, "w"); /*创建并打开客户指定要发送的
                                    音频文件*/
if (fp == NULL){
    printf("File:\t%s Can Not Create!\n", file_name);
    break;
}
int i;
unsigned char buf[RSIZE];  /*从麦克风每次循环获取 RSIZE 大小
                             的数据放入 buf 中，然后再写入文件；
                             放音则相反*/
/*打开声卡设备，只读方式；并对声卡进行设置*/
int fd_dev= open("/dev/dsp", O_RDONLY, 0777);
```

第二章 循环服务器软件的实现原理与方法

```
if (fd_dev < 0){
        printf("Cannot open /dev/dsp device");
        break;
}
/*以下语句用于设置量化位数*/
int arg = SIZE;
if (ioctl(fd_dev,SOUND_PCM_WRITE_BITS, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_BITS ");
        break;
}
 /*以下语句用于设置声道数*/
```

第二章 循环服务器软件的实现原理与方法

```
arg = CHANNELS;
if(ioctl(fd_dev,SOUND_PCM_WRITE_CHANNELS,&arg)== -1){
        printf("Cannot set SOUND_PCM_WRITE_CHANNELS");
        break;
}
  /*以下语句用于设置采样率*/
arg = RATE;
if (ioctl(fd_dev,SOUND_PCM_WRITE_RATE, &arg) == -1){
         printf("Cannot set SOUND_PCM_WRITE_WRITE");
         break;
}
```

第二章 循环服务器软件的实现原理与方法

```
/*开始使用麦克风录音，并写入到客户指定要发送的音频文件*/
/*首先，将 WAVE 文件的文件头写入音频文件*/
/*给 WAVE 文件头结构体变量赋值*/
memset(&wavefilehead, 0, sizeof(wavefilehead));
wavefilehead.a[0]='R';
wavefilehead.a[1]='I';
wavefilehead.a[2]='F';
wavefilehead.a[3]='F';
wavefilehead.b=LENGTH*RATE*CHANNELS*SIZE/8-8;
wavefilehead.c[0]='W';
wavefilehead.c[1]='A';
wavefilehead.c[2]='V';
wavefilehead.c[3]='E';
wavefilehead.d[0]='f';
```

第二章 循环服务器软件的实现原理与方法

```
wavefilehead.d[1]='m';
wavefilehead.d[2]='t';
wavefilehead.d[3]=' ';
wavefilehead.e=16;
wavefilehead.f=1;
wavefilehead.g=CHANNELS;
wavefilehead.h=RATE;
wavefilehead.i=RATE*CHANNELS*SIZE/8;
wavefilehead.j=CHANNELS*SIZE/8;
wavefilehead.k=SIZE;
wavefilehead.p[0]='d';
wavefilehead.p[1]='a';
wavefilehead.p[2]='t';
wavefilehead.p[3]='a';
wavefilehead.q=LENGTH*RATE*CHANNELS*SIZE/8;
```

第二章 循环服务器软件的实现原理与方法

```
/*赋值完毕后，将文件头结构体变量写入到音频文件*/
if(fwrite(&wavefilehead, sizeof(wavefilehead),1,fp)==-1){
        printf("write the file head to wave file error!!");
        break;
}
/*从麦克风循环获取语音数据，每次获得 RSIZE 大小的数据，共循
   环“语音长度/RSIZE”次*/
for(i=0;i<(LENGTH*RATE*SIZE*CHANNELS/8)/RSIZE;i++){
    if (read(fd_dev, buf, sizeof(buf)) !=sizeof(buf)){//读取麦克风录音
            printf("read wrong number of Bytes from microphone!");
            break;
    }
```

第二章 循环服务器软件的实现原理与方法

```
        /*将录音数据写入 WAVE 文件*/
        if(fwrite(buf, sizeof(char), RSIZE,fp)!=RSIZE){
            printf("write to wave file error!!");
            break;
        }
    }//for 循环结束，录音并生成音频文件的过程结束
    close(fd_dev);                          //关闭声卡设备
    close(fp);                              //关闭 wave 文件
}//for 循环结束，录音并生成音频文件的过程结束
/*首先播放 WAVE 文件，然后再将其通过套接字传送给客户端*/
/*以只写方式打开声卡设备并对声卡进行设置*/
int fd_dev = open("/dev/dsp", O_WRONLY,0777);
if (fd_dev < 0){
        printf("Cannot open /dev/dsp device");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于设置量化位数*/
arg = SIZE;
if (ioctl(fd_dev, SOUND_PCM_WRITE_BITS, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_BITS ");
        exit(-1);
}
/*以下语句用于设置声道数*/
arg = CHANNELS;
if (ioctl(fd_dev, SOUND_PCM_WRITE_CHANNELS, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_CHANNELS");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于设置采样率*/
arg = RATE;
if (ioctl(fd_dev, SOUND_PCM_WRITE_RATE, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_WRITE");
        exit(-1);
}
/*首先打开并播放 WAVE 文件，然后再将其传送给客户端*/
if(( fp = fopen(file_name, "r")) == NULL){
        printf("cannot open the wave file");
        exit(-1);
}
lseek(fp, 44, SEEK_SET);          //过滤掉 WAVE 文件头 44 个字节
```

第二章 循环服务器软件的实现原理与方法

```
/*从 WAVE 文件中循环读取语音数据并送声卡播放*/
for(i=0;i<(LENGTH*RATE*SIZE*CHANNELS/8)/RSIZE;i++){
        memset(buf, '\0', sizeof(buf));
        /*读 WAVE 文件数据*/
        if (fread(buf, sizeof(char), RSIZE,fp) != RSIZE){
                printf("read wave file error!");
                break;
        }
        if (write(fd_dev, buf, sizeof(buf)) != sizeof(buf)){ //送声卡播放
                printf("play wave file error!");
                break;
        }
}//for 循环结束，WAVE 文件的播放过程结束
```

第二章 循环服务器软件的实现原理与方法

```
/*将 WAVE 文件通过套接字传送给客户端*/
lseek(fp, 0, SEEK_SET);                    //返回 WAVE 文件的起始位置
/*读取 WAVE 文件头部分的 44 个字节数据*/
memset(&wavefilehead, 0, sizeof(wavefilehead));
if (fread(&wavefilehead,sizeof(wavefilehead),1,fp) !=1){
        printf("read wave file head error!");
        exit(-1);
}
//把 wave 文件头通过套接字发送给客户端
if(send(ssock,&wavefilehead,sizeof(wavefilehead),0)<0){
```

第二章 循环服务器软件的实现原理与方法

```
        printf("Send Wave File Head Failed!\n");
        exit(-1);
    }
    /*把 WAVE 文件中的音频数据通过套接字发送给客户端*/
    for(i=0;i<(LENGTH*RATE*SIZE*CHANNELS/8)/RSIZE;i++){
        memset(buf, '\0', sizeof(buf));
        if (fread(buf,sizeof(char),RSIZE,fp)!=RSIZE){//读 WAVE 文件数据
            printf("read wave file error!");
            break;
        }
        if(send(new_server_socket, buf, strlen(buf),0)< 0){//回送客户端
            printf("Send Wave File Failed!\n");
            break;
        }
    }//for 循环结束，通过套接字发送 WAVE 文件给客户端的过程结束
```

第二章 循环服务器软件的实现原理与方法

```
            fclose(fp);                          //关闭 WAVE 文件
            close(fd_dev);                       //关闭声卡设备
            close(ssock);       //关闭临时套接字
            printf("Wave File Transfer Finished!\n");
    }//while 循环结束
    close(msock);
    return 0;
}
```

2. TCP 客户端例程剖析

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
```

第二章 循环服务器软件的实现原理与方法

```
#include <linux/soundcard.h>
#include <termios.h>
#include<string.h>

#include<netinet/in.h>
#include<sys/socket.h>

#define LENGTH 10                    //录音时间（秒）
#define RATE 88200                   //采样频率
#define SIZE 16                      //量化位数
#define CHANNELS 2                   //声道数目
#define RSIZE 8                      //buf 的大小

#define SERVER_PORT   10000
#define BUFFER_SIZE 1024
#define FILE_NAME_MAX_SIZE   512
```

第二章 循环服务器软件的实现原理与方法

```
/*定义 WAVE 文件的文件头结构体，长度为 44 个字节*/
struct wfhead{
    /*RIFF WAVE CHUNK*/
    unsigned char a[4]; //4 字节，存放'R','I','F','F'
    long int b;    /*整个 WAVE 文件的长度减去 8 个字节，4 字节*/
    unsigned char c[4]; //4 字节，存放'W','A','V','E'
    /*Format CHUNK*/
    unsigned char d[4]; //4 字节，存放'f','m','t',''
    long int e;    //4 字节
    short int f;   //2 字节，编码方式，一般为 0x0001;
    short int g;   //2 字节，声道数目，1 为单声道，2 为双声道;
    long int h;    //4 字节，采样频率;
    long int i;    //4 字节，每秒所需字节数;
```

第二章 循环服务器软件的实现原理与方法

```
    short int j;    //2 字节，每个采样需要多少字节，若是 2 声道，则乘以 2
    short int k;    //2 字节，即量化位数
    /*Data Chunk**/
    unsigned char p[4]; //4 字节，存放'd','a','t','a'
    long int q;    //4 字节，语音数据部分长度，不包括文件头
}wavefilehead;

int main(int argc, char **argv){
    if (argc != 2){
        printf("Usage: ./%s ServerIPAddress\n", argv[0]);
        exit(1);
    }
```

第二章 循环服务器软件的实现原理与方法

```
int tsock = socket(AF_INET, SOCK_STREAM, 0);    //创建客户端套接字
if (tsock< 0){
        printf("Create Socket Failed!\n");
        exit(-1);
}
/*声明服务器端 socket 地址结构变量 server_addr 并给其赋值*/
struct sockaddr_in    servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family=AF_INET;                          //给协议族字段赋值
if (inet_aton(argv[1],&servaddr.sin_addr)==0){    //给 IP 地址字段赋值
        printf("Server IP Address Error!\n");
        exit(1);
}
```

第二章 循环服务器软件的实现原理与方法

```
servaddr.sin_port = htons(SERVER_PORT);          //给端口号字段赋值
int len = sizeof(servaddr);
/*以下语句用于向远程服务器端发起 TCP 连接建立请求*/
int ret, num;
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                       //调用 connect()函数出错
        printf("Connect Failed!\n");
        exit(-1);
}
char file_name[FILE_NAME_MAX_SIZE + 1];
memset(file_name, '\0', sizeof(file_name));
printf("Please Input Wave File Name On Server.\t");
scanf("%s", file_name); //用户输入 WAVE 文件的名字
```

第二章 循环服务器软件的实现原理与方法

```
char buffer[BUFFER_SIZE];
memset(buffer, '\0', sizeof(buffer));
strncpy(buffer,file_name,strlen(file_name)>BUFFER_SIZE?
BUFFER_SIZE : strlen(file_name));
/*向服务器发送 buffer 中的数据，此时 buffer 中存放的是客户端需要接
  收的 WAVE 文件的名字*/
num=0;
num=send(tsock, buffer, strlen(buffer), 0);
if(num!=strlen(buffer)){                          //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
FILE *fp = fopen(file_name, "w");            //生成并打开该 WAVE 文件
if (fp == NULL){
    printf("File:\t%s Can Not Open To Write!\n", file_name);
    exit(-1);
}
/*从服务器端接收数据并写入到该 WAVE 文件中*/
/*从服务器端接收 WAVE 文件头数据并写入到该 WAVE 文件中*/
memset(&wavefilehead, 0, sizeof(wavefilehead));
num=0;
```

第二章 循环服务器软件的实现原理与方法

```
num=recv(tsock, &wavefilehead, sizeof(wavefilehead), 0);
if (num<0){
        printf("Recieve Data Failed!\n");
        exit(-1);
}
//从服务器端接收 WAVE 音频数据并写入到该 wave 文件中
unsigned char buf[RSIZE]; /*从套接字每次循环获取 RSIZE 大小的数据，
                            放入 buf 中，然后再写入到该 WAVE 文件中*/
memset(buf, '\0', sizeof(buf));
/*反复调用 recv 函数接收到服务器的应答，若接收到的内容长度大于 0，
 则循环接收服务器的应答。若等于 0，则退出循环，表示服务器的文件
 传输结束。若小于 0，则表示接收出错*/
```

第二章 循环服务器软件的实现原理与方法

```
num = 0;
while(num=recv(tsock, buf, RSIZE, 0) != 0){
    if (num<0){
        printf("Recieve Data From Server %s Failed!\n", argv[1]);
        break;
    }
    //将接收到的数据写入到 WAVE 文件中
    int write_len = fwrite(buf, sizeof(char), num, fp);
    if (write_len != num){
        printf("File:\t%s Write Failed!\n", file_name);
        break;
    }
    memset(buf, '\0', RSIZE);
} //while 循环结束，接收服务器回送 WAVE 文件的过程结束
```

第二章 循环服务器软件的实现原理与方法

```
fclose(fp);                          //关闭 WAVE 文件
close(tsock);                        //关闭套接字

/*播放该 WAVE 文件*/
/*以只写方式打开声卡设备并对声卡进行设置*/
int fd_dev = open("/dev/dsp", O_WRONLY, 0777);
if (fd_dev < 0){
        printf("Cannot open /dev/dsp device");
        exit(-1);
}
/*以下语句用于设置量化位数*/
arg = SIZE;
if (ioctl(fd_dev, SOUND_PCM_WRITE_BITS, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_BITS ");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于设置声道数*/
arg = CHANNELS;
if (ioctl(fd_dev, SOUND_PCM_WRITE_CHANNELS, &arg) == -1){
        printf("Cannot set SOUND_PCM_WRITE_CHANNELS");
        exit(-1);
}
/*以下语句用于设置采样率*/
arg = RATE;
if (ioctl(fd_dev, SOUND_PCM_WRITE_RATE, &arg) == -1){
       printf("Cannot set SOUND_PCM_WRITE_WRITE");
       exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*首先打开并播放 WAVE 文件，然后再将其传送给客户端*/
if(( fp = fopen(file_name, "r")) == NULL){
        printf("cannot open the wave file");
        exit(-1);
}

lseek(fp, 44, SEEK_SET);        //过滤掉 WAVE 文件头 44 个字节
/*从 WAVE 文件中循环读取语音数据并送声卡播放*/
for(i=0;i<(LENGTH*RATE*SIZE*CHANNELS/8)/RSIZE;i++){
          memset(buf, '\0', sizeof(buf));
          /*读取 WAVE 文件中的数据*/
          if (fread(buf, sizeof(char), RSIZE,fp) != RSIZE){
                printf("read wave file error!");
                break;
           }
```

第二章 循环服务器软件的实现原理与方法

```
            if (write(fd_dev, buf, sizeof(buf)) != sizeof(buf)){ //送声卡播放
                printf("play wave file error!");
                break;
            }
        }//for 循环结束，WAVE 文件的播放过程结束
        fclose(fp);              //关闭 WAVE 文件
        close(fd_dev);           //关闭声卡设备
        return 0;
    }
```

- **2.4.8 Windows 环境下基于 TCP 套接字的图像传输例程剖析**

客户端功能需求描述：向服务器（IP 地址 127.0.0.1，端口号 8888）发送连接请求，连接建立之后，向服务器发送图片 wall.jpg，并将该图片 wall.jpg 在显示屏上显示该图片，然后中断本次通信过程。

第二章 循环服务器软件的实现原理与方法

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的图片 wall.jpg 之后，则在显示屏上显示该图片并中断本次通信过程。

1. TCP 服务器端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <winsock2.h>
#include < cv.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#pragma comment(lib,"ws2_32.lib")
```

第二章 循环服务器软件的实现原理与方法

```
int main(int argc, char* argv[]){
    /*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
    int ret, num;
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    ret=WSAStartup(sockVersion, &wsaData);
    if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
    }
    SOCKET msock=socket(AF_INET, SOCK_STREAM, 0);   //创建套接字
    if(msock == INVALID_SOCKET){
        printf("Create Socket Failed!\n");
        exit(-1);
    }
```

第二章 循环服务器软件的实现原理与方法

```
//绑定 IP 和端口
struct sockaddr_in servaddr;                    //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下 3 条语句用于给端点地址结构体变量 servaddr 赋值*/
servaddr.sin_family=AF_INET;                    //给协议族字段赋值
servaddr.sin_port=htons(10000);                 //给端口号字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);     //给 IP 地址字段赋值
/*以下语句用于调用 bind()函数将主套接字与端点地址绑定*/
```

第二章 循环服务器软件的实现原理与方法

```
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));
if(ret<0){                                                    //调用 bind()函数出错
        printf("Bind Port Failed!\n");
        exit(-1);
}
/*以下语句用于设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, 20);
if(ret<0){                                                    //调用 listen()函数出错
        printf("Listen Failed!\n");
        exit(-1);
}
/*以下 while(1)循环用于循环接收不同客户端发送的 TCP 连接请求*/
```

第二章 循环服务器软件的实现原理与方法

```
SOCKET ssock;
sockaddr_in clientaddr;
int len = sizeof(clientaddr);
printf("等待连接...\n");
while(1){
    ZeroMemory(&clientaddr,sizeof(clientaddr));
    ssock = accept(msock, (struct sockaddr *)&clientaddr, &len);
    if(ssock == INVALID_SOCKET){          //调用 accept()函数出错
        printf("Accept Failed!\n");
        break;
    }
    printf("接收到一个连接： %s \r\n", inet_ntoa(remoteAddr.sin_addr));
    char revData[1000000] = "";
    /*以下语句用于创建用于接收灰度图像的首地址并分配存储空间*/
    IplImage *image_src = cvCreateImage(cvSize(640, 480), 8, 1);
```

第二章 循环服务器软件的实现原理与方法

```
int i, j;
int ret;
cvNamedWindow("server", 1);                    //创建图像的显示窗口
while(ret>0){
    ret=0;
    /*以下语句调用 recv()函数从套接字接收灰度图像数据*
    ret = recv(sClient, revData, 1000000, 0);
    if(ret > 0){
        revData[ret] = 0x00;
        /*以下 for 循环用于将灰度图像转换为原始图像*/
        for(i = 0; i < image_src->height; i++){        //外层 for 循环
            for (j = 0; j < image_src->width; j++){    //内层 for 循环
                ((char *)(image_src->imageData + i * image_src->
widthStep))[j] = revData[image_src->width * i + j];
            }//内层 for 循环结束
        }//外层 for 循环结束
```

第二章 循环服务器软件的实现原理与方法

```
                cvShowImage("server", image_src);          //显示图像
                cvWaitKey(1);
            }//if(ret > 0)条件语句结束
        }//while(ret>0)循环结束，亦即表示接收客户数据结束
        closesocket(ssock);                                //关闭从套接字
        cvDestroyWindow("server");                         //关闭显示窗口
        cvReleaseImage(&image_src);                        //释放图像
    }//while(1)循环结束
    closesocket(masock);                                   //关闭主套接字
    WSACleanup();                                  //结束 Winsock Socket API
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

2. TCP 客户端例程剖析

```
#include "stdafx.h"
#include <windows.h>
#include <winsock2.h>
#include <iostream>
#include <stdio.h>
#include <cv.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#define SERVERIP "172.0.0.1"                    //定义 IP 地址常量
#define SERVERPORT 10000                        //定义端口号常量

#pragma    comment(lib,"ws2_32.lib")
```

第二章 循环服务器软件的实现原理与方法

```
int main(int argc, char* argv[]){
    /*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
    int ret, num;
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    ret=WSAStartup(sockVersion, &wsaData);
    if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
    }
    SOCKET tsock = socket(AF_INET, SOCK_STREAM, 0); //创建套接字
    if(tsock == INVALID_SOCKET){
        printf("Create Socket Failed!\n");
        exit(-1);
    }
```

第二章 循环服务器软件的实现原理与方法

```
sockaddr_in servaddr;
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下 3 条语句用于给端点地址结构体变量 servaddr 赋值*/
servaddr.sin_family=AF_INET;                    //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);         //给 IP 地址字段赋值
servaddr.sin_port = htons(SERVERPORT);          //给端口号字段赋值
/*以下语句用于向远程服务器发起 TCP 连接建立请求*/
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                      //调用 connect()函数出错
        printf("Connect Failed!\n");
        exit(-1);
}
/*以下语句用于读取图像并发送*/
IplImage *image_src = cvLoadImage("wall.jpg"); //首先，载入原始图像
```

第二章 循环服务器软件的实现原理与方法

```
/*其次，创建用于存储灰度图像的首地址并分配存储空间*/
IplImage *image_dst = cvCreateImage(cvSize(640, 480), 8, 1);
int i, j;
char sendData[1000000] = "";
cvNamedWindow("client", 1);                    //创建图像的显示窗口
/*然后，将原始图像转换为灰度图像*/
cvCvtColor(image_src, image_dst, CV_RGB2GRAY);
/*同时，将灰度图像保存到 sendData 数组之中*/
for(i = 0; i < image_dst->height; i++){
    for (j = 0; j < image_dst->width; j++){
        sendData[image_dst->width * i + j] = ((char
*)(image_dst->imageData + i * image_dst->widthStep))[j];
    }
}
```

第二章 循环服务器软件的实现原理与方法

```
    cvShowImage("client", image_dst);                    //在窗口显示图像
    cvWaitKey(0);                                        //任意键发送
    num=0;
    num=send(tsock,sendData,strlen(sendData), 0);        //最后， 发送灰度图像
    if(num!=strlen(sendData)){                           //调用 send()函数出错
        printf("Send Data Failed!\n");
        exit(-1);
    }
    cvDestroyWindow("client");                           //关闭图像的显示窗口
    cvReleaseImage(&image_src);                          //释放原始图像
    cvReleaseImage(&image_dst);                          //释放灰度图像
    closesocket(tsock);                                  //关闭套接字
    WSACleanup();                                        //结束 Winsock Socket API
    return 0;
}
```

第二章 循环服务器软件的实现原理与方法

▪ 2.4.9 Windows 环境下基于 TCP 套接字的视频传输例程剖析

客户端功能需求描述：向服务器（IP 地址 127.0.0.1，端口号 8888）发送连接请求，连接建立之后，读取摄像头的视频帧，并在将该视频帧在显示屏上显示之后发送给服务器，然后中断本次通信过程。

服务器端功能需求描述：反复读取来自客户的任何请求，在收到了客户发送的视频帧之后，则在显示屏上显示该视频帧并中断本次通信过程。

1. TCP 服务器端例程剖析

```
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <winsock2.h>
#include <cv.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
```

第二章 循环服务器软件的实现原理与方法

```
#pragma comment(lib,"ws2_32.lib")

int main(void){
    /*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
    int ret;
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    ret=WSAStartup(sockVersion, &wsaData);
    if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
    }
    /*以下语句用于创建套接字*/
    SOCKET msock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

第二章 循环服务器软件的实现原理与方法

```
if(msock == INVALID_SOCKET) {                    //调用 socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
struct sockaddr_in servaddr;                     //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下 3 条语句用于给端点地址结构体变量 servaddr 赋值*/
servaddr.sin_family=AF_INET;                     //给协议族字段赋值
servaddr.sin_port=htons(10000);                  //给端口号字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);      //给 IP 地址字段赋值
/*以下语句用于调用 bind()函数将主套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));
if(ret<0){                                       //调用 bind()函数出错
    printf("Bind Port Failed!\n");
    exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
if(ret<0){                                          //调用 listen()函数出错
        printf("Listen Failed!\n");
        exit(-1);
}
/*以下 while(1)循环用于循环接收来不同客户端的 TCP 连接建立请求*/
SOCKET ssock;
sockaddr_in clientaddr;
int len = sizeof(clientaddr);
printf("等待连接...\n");
while(1){
      ZeroMemory(&clientaddr,sizeof(clientaddr));
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于接受客户连接请求并创建从套接字*/
ssock = accept(msock, (struct sockaddr *)&clientaddr, &len);
if(ssock == INVALID_SOCKET){                    //调用 accept()函数出错
        printf("Accept Failed!\n");
        break;
}
printf("接收到一个连接： %s \r\n", inet_ntoa(clientaddr.sin_addr));
char revData[1000000] = "";
/*假定收到的图像是 640x480 的，如不同请根据实际情况修改*/
IplImage *image_src = cvCreateImage(cvSize(640, 480), 8, 1);
int i, j;
int ret=1;
cvNamedWindow("server", 1);                     //创建图像显示窗口
```

第二章 循环服务器软件的实现原理与方法

```
while(ret>0){
    ret=0;
    /*调用 recv()函数从套接字接收灰度图像数据*/
    ret = recv(ssock, revData, 1000000, 0);
    if(ret > 0){
        revData[ret] = 0x00;
        for(i = 0; i < image_src->height; i++){
            for (j = 0; j < image_src->width; j++){
                ((char *)(image_src->imageData + i * image_src->
widthStep))[j] = revData[image_src->width * i + j];
            }//for 循环结束
        }//for 循环结束
        cvShowImage("server",image_src); //显示收到的灰度图像
        cvWaitKey(1);
    }//if(ret > 0)条件语句结束
}//while(ret>0)循环结束
```

第二章 循环服务器软件的实现原理与方法

```
            closesocket(ssock);                    //关闭从套接字
            cvDestroyWindow("server");             //关闭图像显示窗口
            cvReleaseImage(&image_src);            //释放图像
        }//while(1)循环结束
        closesocket(msock);                        //关闭主套接字
        WSACleanup();                              //结束 Winsock Socket API
        return 0;
}
```

2. TCP 客户端例程剖析

```
#include "stdafx.h"
#include <windows.h>
#include <winsock2.h>
#include <iostream>
#include <stdio.h>
#include <cv.h>
```

第二章 循环服务器软件的实现原理与方法

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#define SERVERIP "172.0.0.1"                    //定义 IP 地址常量
#define SERVERPORT 10000                        //定义端口号常量

#pragma    comment(lib,"ws2_32.lib")

int main(int argc, char* argv[]){
    /*以下语句调用 WSAStartup()函数初始化 Winsock DLL*/
    int ret;
    WORD sockVersion = MAKEWORD(2,2);
    WSADATA wsaData;
    ret=WSAStartup(sockVersion, &wsaData);
```

第二章 循环服务器软件的实现原理与方法

```
if (ret != 0){
        printf("Couldn't Find a Useable Winsock.dll!\n");
        exit(-1);
}
/*以下语句用于创建套接字*/
SOCKET tsock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if(tsock == INVALID_SOCKET) {              //调用 socket()函数出错
      printf("Create Socket Failed!\n");
      exit(-1);
}
struct sockaddr_in servaddr;                //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
```

第二章 循环服务器软件的实现原理与方法

```
/*以下 3 条语句用于给端点地址结构体变量 servaddr 赋值*/
servaddr.sin_family=AF_INET;                    //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);         //给 IP 地址字段赋值
servaddr.sin_port = htons(SERVERPORT);          //给端口号字段赋值
/*以下语句用于向远程服务器发起 TCP 连接建立请求*/
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                                      //调用 connect()函数出错
        printf("Connect Failed!\n");
        exit(-1);
}
```

第二章 循环服务器软件的实现原理与方法

```
/*以下语句用于读取图像并发送*/
IplImage *image_src = NULL;
/*假定摄像头分辨率为 640x480，如不同请根据实际情况修改*/
IplImage *image_dst = cvCreateImage(cvSize(640, 480), 8, 1);
CvCapture *capture = cvCreateCameraCapture(0);                    //打开摄像头
if (!capture){
        printf("摄像头打开失败，请检查设备!\n");
}
int i, j;
char sendData[1000000] = "";
cvNamedWindow("client", 1);                                       //创建图像显示窗口
```

第二章 循环服务器软件的实现原理与方法

```
while(1){
    image_src = cvQueryFrame(capture);          //从摄像头获取 1 帧图像
    /*以下语句用于将图像转换为灰度图像*/
    cvCvtColor(image_src, image_dst, CV_RGB2GRAY);
    /*以下语句用于将灰度图像存储到发送缓存数组 sendData 之中*/
    for(i = 0; i < image_dst->height; i++){
        for (j = 0; j < image_dst->width; j++){
            sendData[image_dst->width * i + j] = ((char *)(image_dst->
imageData + i * image_dst->widthStep))[j];
        }//for 循环结束
    }//for 循环结束
    cvShowImage("client", image_src);                          //显示原图
    char c=cvWaitKey(30);        /*延时 30ms,若服务器端收到的视频比
                                   较卡，此处延时可适当改大一点*/
    if(c==27)break;              /*用户按 ESC 键（ASCII 值为 27）退出
                                   while(1)循环*/
```

第二章 循环服务器软件的实现原理与方法

```
            send(sclient, sendData, 1000000, 0); //将该帧灰度图像发送给客户端
        }//while(1)循环结束
        cvReleaseCapture(&capture);
        cvDestroyWindow("client");
        closesocket(tsock);
        WSACleanup();
        return 0;
    }
```

- **2.5 本章小结**

本章主要对 UNIX/LINUX 与 Windows 环境下的循环服务器进程结构以及循环服务器软件的算法设计流程分别进行了详细介绍，并在此基础上分别给出了 UNIX/LINUX 与 Windows 环境下的 2 个循环 UDP 服务器与客户端的完整 C 语言例程以及 6 个循环 TCP 服务器与客户端的完整 C 语言例程。通过本章学习，需要了解循环服务器的进程结构；需要熟习循环服务器软件的算法设计流程；需要掌握 UNIX/LINUX 与 Windows 环境下的客户端与循环服务器软件的 C 语言实现方法。

本章作业

第1-8题

谢谢！