**Webbench源码注释**

# webbench.c

/\*

\* (C) Radim Kolar 1997-2004

\* This is free software, see GNU Public License version 2 for

\* details.

\*

\* Simple forking WWW Server benchmark:

\*

\* Usage:

\* webbench --help

\*

\* Return codes:

\* 0 - sucess

\* 1 - benchmark failed (server is not on-line)

\* 2 - bad param

\* 3 - internal error, fork failed

\*

\*/

#include "socket.c"

#include <unistd.h>

#include <sys/param.h>

#include <rpc/types.h>

#include <getopt.h>

#include <strings.h>

#include <time.h>

#include <signal.h>

/\* values \*/

/\*\*注\*\*：volatile是易变的，不稳定的意思。volatile是关键字，是一种类型修饰符，用它修饰的变量表示可以被某些编译器未知的因素更改，比如操作系统、硬件或者其他线程等，遇到这个关键字声明的变量，编译器对访问该变量的代码不再进行优化，从而可以提供对特殊地址的稳定访问。\*\*/

volatile int timerexpired=0; //根据测试时间判断是否超时

int speed=0; //服务器响应数

int failed=0; //请求失败数

int bytes=0; //读取字节数

/\* globals \*/

int http10=1; //http协议版本；0为http/0.9, 1为http/1.0, 2为 http/1.1

/\* Allow: GET, HEAD, OPTIONS, TRACE \*/

#define METHOD\_GET 0

#define METHOD\_HEAD 1

#define METHOD\_OPTIONS 2

#define METHOD\_TRACE 3

#define PROGRAM\_VERSION "1.5"

//定义HTTP请求方法GET，此外还支持OPTIONS、HEAD、TRACE方法

int method=METHOD\_GET;

int clients=1; //并发数；由命令行参数-c指定，默认为1

int force=0; //是否等待服务器应答

int force\_reload=0; //是否使用cache，默认为0，使用

int proxyport=80; //代理服务器端口号，默认为80

char \*proxyhost=NULL; //代理服务器地址

int benchtime=30; //测试时间；由命令行参数-t指定，默认为30s

/\* internal \*/

int mypipe[2]; //创建管道；用于父子进程间通信，读写数据

char host[MAXHOSTNAMELEN]; //主机名

#define REQUEST\_SIZE 2048

char request[REQUEST\_SIZE]; //HTTP请求信息

/\*\*注\*\*：struct option 类型数组。

说明：该数据结构中的每个元素对应了一个长选项，并且每个元素是由四个域组成。通常情况下，可以按以下规则使用。第一个元素，描述长选项的名称；第二个选项，代表该选项是否需要跟着参数，需要参数则为1，反之为0；第三个选项，可以赋为NULL；第四个选项，是该长选项对应的短选项名称。另外，数据结构的最后一个元素，要求所有域的内容均为0，即{NULL,0,NULL,0}。

结构中的元素解释如下：

1）const char \*name：选项名，前面没有短横线。譬如"help"、"verbose"之类。

2）int has\_arg：描述长选项是否有选项参数，如果有，是哪种类型的参数，其值见下表:

符号常量 数值 含义

no\_argument 0 选项没有参数

required\_argument 1 选项需要参数

optional\_argument 2 选项参数是可选的

3）int \*flag：

如果该指针为NULL，那么getopt\_long返回val字段的值；

如果该指针不为NULL，那么会使得它所指向的结构填入val字段的值，同时getopt\_long返回0

4）int val：

如果flag是NULL，那么val通常是个字符常量，如果短选项和长选项一致，那么该字符就应该与optstring中出现的这个选项的参数相同；\*\*/

//struct option结构体，配合getopt\_long函数使用

static const struct option long\_options[]=

{

{"force",no\_argument,&force,1},

{"reload",no\_argument,&force\_reload,1},

{"time",required\_argument,NULL,'t'},

{"help",no\_argument,NULL,'?'},

{"http09",no\_argument,NULL,'9'},

{"http10",no\_argument,NULL,'1'},

{"http11",no\_argument,NULL,'2'},

{"get",no\_argument,&method,METHOD\_GET},

{"head",no\_argument,&method,METHOD\_HEAD},

{"options",no\_argument,&method,METHOD\_OPTIONS},

{"trace",no\_argument,&method,METHOD\_TRACE},

{"version",no\_argument,NULL,'V'},

{"proxy",required\_argument,NULL,'p'},

{"clients",required\_argument,NULL,'c'},

{NULL,0,NULL,0}

};

/\* prototypes \*/

static void benchcore(const char\* host,const int port, const char \*request);

static int bench(void);

static void build\_request(const char \*url);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：信号处理函数

\*/

static void alarm\_handler(int signal)

{

timerexpired=1; //timerexpired置1

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：程序使用说明

\*/

static void usage(void)

{

fprintf(stderr,

"webbench [option]... URL\n"

" -f|--force Don't wait for reply from server.\n"

" -r|--reload Send reload request - Pragma: no-cache.\n"

" -t|--time <sec> Run benchmark for <sec> seconds. Default 30.\n"

" -p|--proxy <server:port> Use proxy server for request.\n"

" -c|--clients <n> Run <n> HTTP clients at once. Default one.\n"

" -9|--http09 Use HTTP/0.9 style requests.\n"

" -1|--http10 Use HTTP/1.0 protocol.\n"

" -2|--http11 Use HTTP/1.1 protocol.\n"

" --get Use GET request method.\n"

" --head Use HEAD request method.\n"

" --options Use OPTIONS request method.\n"

" --trace Use TRACE request method.\n"

" -?|-h|--help This information.\n"

" -V|--version Display program version.\n"

);

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*主函数

\*/

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

{

int opt=0;

int options\_index=0;

char \*tmp=NULL;

/\*\*注\*\*：argc：指命令行输入参数的个数；argv：存储所有命令行参数，字符串数组\*\*/

if(argc==1) //不带参数时，输出使用说明

{

usage(); //调用usage()函数：使用说明

return 2;

}

/\*注：getopt\_long 函数：用来解析命令行参数，支持长命令选项。

参数1、2：main函数的argc、argv参数；

参数3：由该命令要处理的各个选项组成的字符串。选项后面带有冒号时，表示该选项是一个带参数的选项；

参数4：构造体struct option数组；

参数5：输出参数，函数getopt\_long（）返回时，该参数的值是struct option数组的索引

optarg：处理带输入参数的选项时，选项参数保存至char \*optarg中。

optind：下一个处理的选项在argv中的地址，所有选项处理完后，optind指向未识别的项。

optopt：最后一个已知项。

\*/

//检查输入参数，并设置对应选项

while((opt=getopt\_long(argc,argv,"912Vfrt:p:c:?h",long\_options,&options\_index))!=EOF )

{

switch(opt) //相应的命令行参数

{

case 0 : break;

case 'f': force=1;break;

case 'r': force\_reload=1;break;

case '9': http10=0;break;

case '1': http10=1;break;

case '2': http10=2;break;

case 'V': printf(PROGRAM\_VERSION"\n");exit(0); //输出版本号

//optarg表示命令后的参数，例如-c 100，optarg为100

case 't': benchtime=atoi(optarg);break;

case 'p':

/\* proxy server parsing server:port \*/

/\*\*注\*\*：strrchr函数。找一个字符c在另一个字符串str中末次出现的位置（也就是从str的右侧开始查找字符c首次出现的位置），并返回从字符串中的这个位置起，一直到字符串结束的所有字符。如果未能找到指定字符，那么函数将返回NULL\*\*/

tmp=strrchr(optarg,':');

proxyhost=optarg; //地址设定

if(tmp==NULL)

{

break;

}

if(tmp==optarg)

{

// fprintf函数：格式化输出到文件

fprintf(stderr,"Error in option --proxy %s: Missing hostname.\n",optarg);

return 2;

}

if(tmp==optarg+strlen(optarg)-1)

{

fprintf(stderr,"Error in option --proxy %s Port number is missing.\n",optarg);

return 2;

}

\*tmp='\0';

proxyport=atoi(tmp+1);break; //重设端口号

case ':':

case 'h':

case '?': usage();return 2;break;

case 'c': clients=atoi(optarg);break; //并发数

}

}

//optind：命令行参数中未读取的下一个元素下标

if(optind==argc) {

fprintf(stderr,"webbench: Missing URL!\n");

usage();

return 2;

}

//并发数和测试时间不能为0

if(clients==0) clients=1;

if(benchtime==0) benchtime=60;

/\* Copyright \*/

fprintf(stderr,"Webbench - Simple Web Benchmark "PROGRAM\_VERSION"\n"

"Copyright (c) Radim Kolar 1997-2004, GPL Open Source Software.\n"

);

build\_request(argv[optind]); //封装HTTP请求信息

/\* print bench info \*/

//输出提示信息

printf("\nBenchmarking: ");

switch(method)

{

case METHOD\_GET:

default:

printf("GET");break;

case METHOD\_OPTIONS:

printf("OPTIONS");break;

case METHOD\_HEAD:

printf("HEAD");break;

case METHOD\_TRACE:

printf("TRACE");break;

}

printf(" %s",argv[optind]);

switch(http10)

{

case 0: printf(" (using HTTP/0.9)");break;

case 2: printf(" (using HTTP/1.1)");break;

}

printf("\n");

if(clients==1) printf("1 client");

else

printf("%d clients",clients); //并发数

printf(", running %d sec", benchtime); //测试时间

if(force) printf(", early socket close");

if(proxyhost!=NULL) printf(", via proxy server %s:%d",proxyhost,proxyport);

if(force\_reload) printf(", forcing reload");

printf(".\n");

return bench(); //开始测试

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：封装HTTP请求信息

\*\*@url:URL地址

\*\*备注：封装好的HTTP请求信息存放在数组request

\*/

void build\_request(const char \*url)

{

char tmp[10];

int i;

/\*\*注\*\*：bzero函数。原型：extern void bzero(void \*s, int n);功能：置字节字符串s的前n个字节为零\*\*/

//请求地址和请求连接初始化清零

bzero(host,MAXHOSTNAMELEN);

bzero(request,REQUEST\_SIZE);

//设置HTTP协议版本

if(force\_reload && proxyhost!=NULL && http10<1) http10=1;

if(method==METHOD\_HEAD && http10<1) http10=1;

if(method==METHOD\_OPTIONS && http10<2) http10=2;

if(method==METHOD\_TRACE && http10<2) http10=2;

//设置HTTP请求方法

switch(method)

{

default:

case METHOD\_GET: strcpy(request,"GET");break;

case METHOD\_HEAD: strcpy(request,"HEAD");break;

case METHOD\_OPTIONS: strcpy(request,"OPTIONS");break;

case METHOD\_TRACE: strcpy(request,"TRACE");break;

}

strcat(request," ");

//判断URL地址是否合法

if(NULL==strstr(url,"://")) //判断URL地址是否包含“：//”

{

//错误信息

fprintf(stderr, "\n%s: is not a valid URL.\n",url);

exit(2);

}

if(strlen(url)>1500) //判断URL长度是否太长

{

//错误信息

fprintf(stderr,"URL is too long.\n");

exit(2);

}

if(proxyhost==NULL) //判断是否有代理服务器

if (0!=strncasecmp("http://",url,7)) //判断前7个字符串是否为http://

{

//错误信息

fprintf(stderr,"\nOnly HTTP protocol is directly supported, set --proxy for others.\n");

exit(2);

}

/\* protocol/host delimiter \*/

i=strstr(url,"://")-url+3; //指向http://后第一个位置，即主机名

if(strchr(url+i,'/')==NULL) { //判断URL地址是否以“/”结尾

//错误信息

fprintf(stderr,"\nInvalid URL syntax - hostname don't ends with '/'.\n");

exit(2);

}

if(proxyhost==NULL)

{

/\* get port from hostname \*/

/\*\*注\*\*：index()函数。功能：用来找出参数s字符串中第一个出现的参数c地址，然后将该字符出现的地址返回\*\*/

if(index(url+i,':')!=NULL &&index(url+i,':')<index(url+i,'/'))

{

strncpy(host,url+i,strchr(url+i,':')-url-i); //主机地址

bzero(tmp,10);

strncpy(tmp,index(url+i,':')+1,strchr(url+i,'/')-index(url+i,':')-1);//端口号

proxyport=atoi(tmp); //类型转换

if(proxyport==0) proxyport=80;

}

else

{

strncpy(host,url+i,strcspn(url+i,"/"));

}

/\*\*注\*\*：strcspn ：返回str1和str2中不同的元素的个数\*\*/

strcat(request+strlen(request),url+i+strcspn(url+i,"/"));

}

else

{

strcat(request,url); //URL地址

}

//开始封装HTTP请求信息

if(http10==1) //版本号

strcat(request," HTTP/1.0");

else if (http10==2)

strcat(request," HTTP/1.1");

strcat(request,"\r\n"); // \r\n：回车换行

if(http10>0) // User-Agent：

strcat(request,"User-Agent: WebBench "PROGRAM\_VERSION"\r\n");

if(proxyhost==NULL && http10>0) //Host：

{

strcat(request,"Host: ");

strcat(request,host); //URL地址

strcat(request,"\r\n");

}

if(force\_reload && proxyhost!=NULL) // Pragma:

{

strcat(request,"Pragma: no-cache\r\n");

}

if(http10>1) // Connection:

strcat(request,"Connection: close\r\n");

/\* add empty line at end \*/

if(http10>0) strcat(request,"\r\n");

// printf("Req=%s\n",request);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：创建管道，派生子进程，子进程测试HTTP请求

\*/

static int bench(void)

{

int i,j,k;

pid\_t pid=0;

FILE \*f;

/\* check avaibility of target server \*/

//调用Socket函数创建socket连接，测试地址是否可以正常访问

i=Socket(proxyhost==NULL?host:proxyhost,proxyport);

if(i<0) { //错误信息

fprintf(stderr,"\nConnect to server failed. Aborting benchmark.\n");

return 1;

}

/\*\*注\*\*：close（）函数。语法：close（fd）。功能：当使用完文件后若已不再需要则可使用 close()关闭该文件, close()会让数据写回磁盘, 并释放该文件所占用的资源。参数fd 为先前由open()或creat()所返回的文件描述词\*\*/

close(i);

/\* create pipe \*/

/\*\*注\*\*：pipe（）函数。原型：int pipe(int filedes[2]);

说明：pipe()会建立管道，并将文件描述词由参数filedes数组返回。

filedes[0]为管道里的读取端

filedes[1]则为管道的写入端

返回值：若成功则返回零，否则返回-1，错误原因存于errno中\*\*/

if(pipe(mypipe)) //创建管道；用于父子进程数据传输

{

/\*\*注\*\*：perror( ) 函数。功能：将上一个函数发生错误的原因输出到标准设备(stderr)\*\*/

//错误信息

perror("pipe failed.");

return 3;

}

/\* fork childs \*/

for(i=0;i<clients;i++)

{

/\*\*注\*\*：fork（）函数。功能：通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

返回值：1）在父进程中，fork返回新创建子进程的进程ID；

2）在子进程中，fork返回0；

3）如果出现错误，fork返回一个负值；\*\*/

pid=fork(); //根据并发数，派生相应数目的子进程

if(pid <= (pid\_t) 0)

{

/\* child process or error\*/

sleep(1); /\* make childs faster \*/

break; //子进程要跳出循环，防止子进程派生子子进程

}

}

if( pid< (pid\_t) 0) // fork调用失败返回负数

{

//错误信息

fprintf(stderr,"problems forking worker no. %d\n",i);

perror("fork failed.");

return 3;

}

if(pid== (pid\_t) 0) //子进程；fork返回0

{

/\* I am a child \*/

if(proxyhost==NULL) //是否使用proxyhost

benchcore(host,proxyport,request); //测试HTTP请求

else

benchcore(proxyhost,proxyport,request);

/\* write results to pipe \*/

/\*\*注\*\*：fdopen（）函数。原型：FILE \* fdopen(int filedes, const char \*type);

说明：用于在一个已经打开的文件描述符上打开一个流，其第1个参数表示一个已经打开的文件描述符，第2个参数type表示打开的方式，该值以一个字符串的形式传入。w ：打开只写文件，若文件存在则文件长度清为0，即该文件内容会消失。若文件不存在则建立该文件；r ：打开只读文件，该文件必须存在

返回值：转换成功时返回指向该流的文件指针。失败则返回NULL，并把错误代码存在errno中\*\*/

f=fdopen(mypipe[1],"w"); //子进程打开管道写

if(f==NULL)

{

//错误信息

perror("open pipe for writing failed.");

return 3;

}

fprintf(f,"%d %d %d\n",speed,failed,bytes); //子进程将测试结果写入管道

/\*\*注\*\*：fclose（）函数。说明：fclose函数的参数是一个FILE对象的指针，它指向需要关闭的流。如果关闭成功，fclose函数返回0，失败返回EOF。这个值是一个定义在stdio.h文件中的宏，其值是-1\*\*/

fclose(f);

return 0;

}

else //父进程

{

f=fdopen(mypipe[0],"r"); //父进程打开管道读

if(f==NULL)

{

//错误信息

perror("open pipe for reading failed.");

return 3;

}

/\*\*注\*\*：setvbuf（）函数。说明：设置文件缓冲区函数；使得打开文件后，用户可建立自己的文件缓冲区，而不使用fopen()函数打开文件设定的默认缓冲区。由malloc函数来分配缓冲区\*\*/

setvbuf(f,NULL,\_IONBF,0);

speed=0; //传输速度

failed=0; //失败请求数

bytes=0; //传输字节数

while(1)

{

pid=fscanf(f,"%d %d %d",&i,&j,&k); //父进程读取管道数据

if(pid<2){

//错误信息

fprintf(stderr,"Some of our childrens died.\n");

break;

}

speed+=i; //传输速度计数

failed+=j; //失败请求数计数

bytes+=k; //传输字节数计数

//判断是否读取完所有子进程数据，读取完则退出循环

if(--clients==0) break;

}

fclose(f);

//在屏幕上输出测试结果

printf("\nSpeed=%d pages/min, %d bytes/sec.\nRequests: %d susceed, %d failed.\n",

(int)((speed+failed)/(benchtime/60.0f)),

(int)(bytes/(float)benchtime),

speed,

failed);

}

return i;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：测试HTTP请求

\*\*@host：地址

\*\*@port：端口

\*\*@req：HTTP请求信息

\*/

void benchcore(const char \*host,const int port,const char \*req)

{

int rlen;

char buf[1500]; //存放服务器响应请求所返回的数据

int s,i;

struct sigaction sa;

/\* setup alarm signal handler \*/

sa.sa\_handler=alarm\_handler; //信号处理函数

sa.sa\_flags=0;

/\*\*注\*\*：sigaction（）函数。说明：检查或修改与指定信号相关联的处理动作（可同时两种操作）\*\*/

//超时将产生SIGALRM信号，调用alarm\_handler函数处理信号

if(sigaction(SIGALRM,&sa,NULL))

exit(3);

/\*\*注\*\*：alarm（）函数。说明：设置信号SIGALRM 在经过参数seconds 指定的秒数后传送给目前的进程。如果参数seconds 为0, 则之前设置的闹钟会被取消, 并将剩下的时间返回\*\*/

alarm(benchtime); //设置闹钟函数：计时开始

rlen=strlen(req);

nexttry:while(1) //带go-to语句的while循环

{

//计时结束，产生信号，信号处理函数将timerexpired置1，退出函数

if(timerexpired)

{

if(failed>0)

{

failed--;

}

return;

}

s=Socket(host,port); //建立socket连接，获取socket描述符

if(s<0) { failed++;continue;} //连接失败，failed加1

if(rlen!=write(s,req,rlen)) {failed++;close(s);continue;} //发出请求

if(http10==0) //针对HTTP0.9进行特殊处理

if(shutdown(s,1)) { failed++;close(s);continue;}

if(force==0) //是否等待服务器响应，-f选项为不等待

{

/\* read all available data from socket \*/

while(1)

{

if(timerexpired) break; //判断是否超时

i=read(s,buf,1500); //读取服务器响应数据，存放在数组buf

if(i<0) //读取数据失败

{

failed++;

close(s);

goto nexttry; //重新开始循环

}

else //读取数据成功

if(i==0) break;

else

bytes+=i; //读取字节数增加

}

}

if(close(s)) {failed++;continue;}

speed++; //HTTP测试成功，speed加1

}

}

# socket.c

/\* $Id: socket.c 1.1 1995/01/01 07:11:14 cthuang Exp $

\* This module has been modified by Radim Kolar for OS/2 emx

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module: socket.c

program: popclient

SCCS ID: @(#)socket.c 1.5 4/1/94

programmer: Virginia Tech Computing Center

compiler: DEC RISC C compiler (Ultrix 4.1)

environment: DEC Ultrix 4.3

description: UNIX sockets code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <sys/types.h>

#include <sys/socket.h>

#include <fcntl.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/time.h>

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*函数功能：建立Socket连接

\*\*@host:网络地址

\*\*@clientPort:端口

\*\*Return：建立的socket连接；如果返回-1，表示建立连接失败

\*/

int Socket(const char \*host, int clientPort)

{

int sock;

unsigned long inaddr;

struct sockaddr\_in ad;

struct hostent \*hp;

memset(&ad, 0, sizeof(ad)); //初始化地址

ad.sin\_family = AF\_INET;

/\*\*注\*\*：inet\_addr（）函数。功能：将一个点分十进制的IP转换成一个长整数型数。如果传入的字符串不是一个合法的IP地址，将返回INADDR\_NONE \*\*/

/\*\*注\*\*：memcpy（）函数。功能：从源src所指的内存地址的起始位置开始拷贝n个字节到目标dest所指的内存地址的起始位置中\*\*/

inaddr = inet\_addr(host); //将点分的十进制的IP转为无符号长整形

if (inaddr != INADDR\_NONE) //判断是否为合法IP地址

memcpy(&ad.sin\_addr, &inaddr, sizeof(inaddr));

else //如果是域名

{

/\*\*注\*\*：gethostbyname（）函数。功能：返回对应于给定主机名的包含主机名字和地址信息的hostent结构指针。结构的声明与gethostaddr()中一致\*\*/

hp = gethostbyname(host); //通过域名获取IP地址

if (hp == NULL)

return -1;

memcpy(&ad.sin\_addr, hp->h\_addr, hp->h\_length);

}

/\*\*注\*\*：htons（）函数。功能：将一个无符号短整型的主机数值转换为网络字节顺序。备注：网络字节顺序是TCP/IP中规定好的一种数据表示格式，它与具体的CPU类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释，网络字节顺序采用big-endian排序方式\*\*/

ad.sin\_port = htons(clientPort);

/\*\*注\*\*：socket（）函数。功能：系统调用socket()来获取文件描述符，返回一个套接口描述符，如果出错，则返回-1 \*\*/

sock = socket(AF\_INET, SOCK\_STREAM, 0); //获取socket描述符

if (sock < 0)

return sock;

/\*\*注\*\*：connect（）函数。功能：用于建立与指定socket的连接

参数1：标识一个未连接socket ；

参数2：指向要连接套接字的sockaddr结构体的指针

参数3：sockaddr结构体的字节长度\*\*/

if (connect(sock, (struct sockaddr \*)&ad, sizeof(ad)) < 0) //建立连接

return -1;

return sock;

}