socket编程

使用TCP协议的流程图

TCP通信的基本步骤如下：

服务端：socket---bind---listen---while(1){---accept---recv---send---close---}---close 客户端：socket----------connect---send---recv----------close

服务器端：

头文件包含：

#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <arpa/inet.h> #include <unistd.h> #include <string.h> #include <stdio.h> #include <stdlib.h>

socket函数：生成一个套接口描述符。

```
原型：int socket(int domain,int type,int protocol);
```

参数： domain { AF_INET：Ipv4网络协议 AF_INET6：IPv6网络协议} type {tcp：SOCK_STREAM udp：SOCK_DGRAM} protocol指定socket所使用的传输协议编号。通常为0. 返回值：成功则返回套接口描述符，失败返回-1。 常用实例：int sfd = socket(AF_INET, SOCK_STREAM, 0); if(sfd == -1){perror("socket");exit(-1);}

bind函数：用来绑定一个端口号和IP地址，使套接口与指定的端口号和IP地址相关联。

原型：int bind(int sockfd,struct sockaddr * my_addr,int addrlen); 参数：sockfd为前面socket的返回值。

my_addr为结构体指针变量 对于不同的socket domain定义了一个通用的数据结构 struct sockaddr //此结构体不常用 { unsigned short int sa_family; //调用socket（）时的domain参数，即AF_INET值。
char sa_data[14]; //最多使用14个字符长度 }; 此sockaddr结构会因使用不同的socket domain而有不同结构定义， 例如使用AF_INET domain，其socketaddr结构定义便为

struct sockaddr_in //常用的结构体 {
unsigned short int sin_family; //即为sa_family èAF_INET
uint16_t sin_port; //为使用的port编号
struct in_addr sin_addr; //为IP 地址
unsigned char sin_zero[8]; //未使用 }; struct in_addr {
uint32_t s_addr; }; addrlenàsockaddr的结构体长度。通常是计算sizeof(struct sockaddr);

返回值：成功则返回0，失败返回-1

常用实例：struct sockaddr_in my_addr; //定义结构体变量

memset(&my_addr, 0, sizeof(struct sockaddr)); //将结构体清空 my_addr.sin_family = AF_INET; //表示采用Ipv4网络协议 my_addr.sin_port = htons(8888); //表示端口号为8888，通常是大于1024的一个值。 //htons()用来将参数指定的16位hostshort转换成网络字符顺序 my_addr.sin_addr.s_addr = inet_addr("192.168.0.101"); //

inet_addr()用来将IP地址字符串转换成网络所使用的二进制数字，如果为INADDR_ANY，这表示服务器自动填充本机IP地址。 if(bind(sfd, (struct sockaddr*)&my_str, sizeof(struct socketaddr)) == -1) {perror("bind");close(sfd);exit(-1);} （注：通过将my_addr.sin_port置为0，函数会自动为你选择一个未占用的端口来使用。同样，通过将my_addr.sin_addr.s_addr置为INADDR_ANY，系统会自动填入本机IP地址。）

listen函数：

使服务器的这个端口和IP处于监听状态，等待网络中某一客户机的连接请求。如果客户端有连接请求，端口就会接受这个连接。

原型：int listen(int sockfd, int backlog); 参数：sockfdà为前面socket的返回值.即sfd backlogà指定同时能处理的最大连接要求，通常为10或者5。 最大值可设至128 返回值：成功则返回0，失败返回-1 常用实例：if(listen(sfd, 10) == -1) {perror("listen");close(sfd);exit(-1);}

accept函数

接受远程计算机的连接请求，建立起与客户机之间的通信连接。服务器处于监听状态时，如果某时刻获得客户机的连接请求，此时并不是立即处理这个请求，而是将这个请求放在等待队列中，当系统空闲时再处理客户机的连接请求。当accept函数接受一个连接时，会返回一个新的socket标识符，以后的数据传输和读取就要通过这个新的socket编号来处理，原来参数中的socket也可以继续使用，继续监听其它客户机的连接请求。（也就是说，类似于移动营业厅，如果有客户打电话给10086，此时服务器就会请求连接，处理一些事务之后，就通知一个话务员接听客户的电话，也就是说，后面的所有操作，此时已经于服务器没有关系，而是话务员跟客户的交流。对应过来，客户请求连接我们的服务器，我们服务器先做了一些绑定和监听等等操作之后，如果允许连接，则调用accept函数产生一个新的套接字，然后用这个新的套接字跟我们的客户进行收发数据。也就是说，服务器跟一个客户端连接成功，会有两个套接字。）

原型：int accept(int s,struct sockaddr * addr,int * addrlen);

参数：sà为前面socket的返回值.即sfd addrà为结构体指针变量，和bind的结构体是同种类型的，系统会把远程主机的信息（远程主机的地址和端口号信息）保存到这个指针所指的结构体中。 addrlenà表示结构体的长度，为整型指针 返回值：成功则返回新的socket处理代码new_fd，失败返回-1

常用实例：struct sockaddr_in clientaddr; memset(&clientaddr, 0, sizeof(struct sockaddr)); int addrlen = sizeof(struct sockaddr); int new_fd = accept(sfd, (struct sockaddr*)&clientaddr, &addrlen); if(new_fd == -1) {perror("accept");close(sfd);exit(-1);} printf("%s %d success connect\n",inet_ntoa(clientaddr.sin_addr),ntohs(clientaddr.sin_port));

recv函数：

用新的套接字来接收远端主机传来的数据，并把数据存到由参数buf 指向的内存空间 原型：int recv(int sockfd,void *buf,int len,unsigned int flags); 参数：sockfdà为前面accept的返回值.即new_fd，也就是新的套接字。 bufà表示缓冲区 lenà表示缓冲区的长度 flagsà通常为0 返回值：成功则返回实际接收到的字符数，可能会少于你所指定的接收长度。失败返回-1

常用实例：char buf[512] = {0};

if(recv(new_fd, buf, sizeof(buf), 0) == -1)

{perror("recv");close(new_fd);close(sfd);exit(-1);}

puts(buf);

send函数：

用新的套接字发送数据给指定的远端主机

原型：int send(int s,const void * msg,int len,unsigned int flags);

参数：sà为前面accept的返回值.即new_fd

msgà一般为常量字符串

lenà表示长度

flagsà通常为0

返回值：成功则返回实际传送出去的字符数，可能会少于你所指定的发送长度。失败返回-1

常用实例：if(send(new_fd, "hello", 6, 0) == -1)

{perror("send");close(new_fd);close(sfd);exit(-1);}

close函数：

当使用完文件后若已不再需要则可使用close()关闭该文件，并且close()会让数据写回磁盘，并释放该文件所占用的资源

原型：int close(int fd);

参数：fdà为前面的sfd,new_fd

返回值：若文件顺利关闭则返回0，发生错误时返回-1

常用实例：close(new_fd);

close(sfd);

客户端：

connect函数：

用来请求连接远程服务器，将参数sockfd 的socket 连至参数serv_addr 指定的服务器IP和端口号上去。

原型：int connect (int sockfd,struct sockaddr * serv_addr,int addrlen);

参数：sockfdà为前面socket的返回值，即sfd

serv_addrà为结构体指针变量，存储着远程服务器的IP与端口号信息。

addrlenà表示结构体变量的长度

返回值：成功则返回0，失败返回-1

常用实例：struct sockaddr_in seraddr;//请求连接服务器

memset(&seraddr, 0, sizeof(struct sockaddr));

seraddr.sin_family = AF_INET;

seraddr.sin_port = htons(8888); //服务器的端口号

seraddr.sin_addr.s_addr = inet_addr("192.168.0.101"); //服务器的ip

if(connect(sfd, (struct sockaddr*)&seraddr, sizeof(struct sockaddr)) == -1)

{perror("connect");close(sfd);exit(-1);}

将上面的头文件以及各个函数中的代码全部拷贝就可以形成一个完整的例子，此处省略。 还可以不写客户端程序，使用telnet远程登录来检测我们的服务器端程序。比如我们的服务器程序在监听8888端口，我们可以用telnet 192.168.0.101 8888来查看服务端的状况。 Example：将一些通用的代码全部封装起来，以后要用直接调用函数即可。如下： 通用网络封装代码头文件：

tcp_net_socket.h

```
#ifndef __TCP__NET__SOCKET__H
#define __TCP__NET__SOCKET__H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>

extern int tcp_init(const char* ip,int port);
extern int tcp_accept(int sfd);
extern int tcp_connect(const char* ip,int port);
extern void signalhandler(void);
#endif
```

具体的通用函数封装如下： tcp_net_socket.c

```
#include "tcp_net_socket.h"

int tcp_init(const char* ip, int port)   //用于初始化操作
{
    int sfd = socket(AF_INET, SOCK_STREAM, 0);//首先创建一个socket，向系统申请
    if(sfd == -1)
    {
        perror("socket");
        exit(-1);
    }
    struct sockaddr_in serveraddr;
    memset(&serveraddr, 0, sizeof(struct sockaddr));
    serveraddr.sin_family = AF_INET;
```

```
    serveraddr.sin_port = htons(port);
    serveraddr.sin_addr.s_addr = inet_addr(ip);//或INADDR_ANY
if(bind(sfd, (struct sockaddr*)&serveraddr, sizeof(struct sockaddr)) == -1)
//将新的socket与制定的ip、port绑定
    {
        perror("bind");
        close(sfd);
        exit(-1);
    }
    if(listen(sfd, 10) == -1)//监听它，并设置其允许最大的连接数为10个
    {
        perror("listen");
        close(sfd);
        exit(-1);
    }
    return sfd;
}


int tcp_accept(int sfd)   //用于服务端的接收
{
    struct sockaddr_in clientaddr;
    memset(&clientaddr, 0, sizeof(struct sockaddr));
    int addrlen = sizeof(struct sockaddr);
    int new_fd = accept(sfd, (struct sockaddr*)&clientaddr, &addrlen);
    //sfd接受客户端连接，并创建新的socket为new_fd，将请求连接的客户端的ip、port保存在结
构体clientaddr中
    if(new_fd == -1)
    {
        perror("accept");
        close(sfd);
        exit(-1);
    }
    printf("%s %d success
connect...\n",inet_ntoa(clientaddr.sin_addr),ntohs(clientaddr.sin_port));
    return new_fd;
}

int tcp_connect(const char* ip, int port)   //用于客户端的连接
{
    int sfd = socket(AF_INET, SOCK_STREAM, 0);//向系统注册申请新的socket
    if(sfd == -1)
    {
        perror("socket");
        exit(-1);
    }
    struct sockaddr_in serveraddr;
    memset(&serveraddr, 0, sizeof(struct sockaddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(port);
    serveraddr.sin_addr.s_addr = inet_addr(ip);
    if(connect(sfd, (struct sockaddr*)&serveraddr, sizeof(struct sockaddr)) == -1)
    //将sfd连接至制定的服务器网络地址serveraddr
    {
```

```
    perror("connect");
    close(sfd);
    exit(-1);
    }
    return sfd;
}

void signalhandler(void)    //用于信号处理，让服务端在按下Ctrl+c或Ctrl+\的时候不会退出
{
    sigset_t sigSet;
    sigemptyset(&sigSet);
    sigaddset(&sigSet,SIGINT);
    sigaddset(&sigSet,SIGQUIT);
    sigprocmask(SIG_BLOCK,&sigSet,NULL);
}
```

服务器端： tcp_net_server.c

```
#include "tcp_net_socket.h"
int main(int argc, char* argv[])
{
if(argc < 3)
{
printf("usage:./servertcp  ip  port\n");
exit(-1);
}
signalhandler();
int sfd = tcp_init(argv[1], atoi(argv[2]));  //或int sfd =
tcp_init("192.168.0.164", 8888);
while(1) //用while循环表示可以与多个客户端接收和发送，但仍是阻塞模式的
{
int cfd = tcp_accept(sfd);
char buf[512] = {0};
if(recv(cfd, buf, sizeof(buf), 0) == -1)//从cfd客户端接收数据存于buf中
{
perror("recv");
close(cfd);
close(sfd);
exit(-1);
}
puts(buf);
if(send(cfd, "hello world", 12, 0) == -1)//从buf中取向cfd客户端发送数据
{
perror("send");
close(cfd);
close(sfd);
exit(-1);
}
close(cfd);
}
close(sfd);
}
```

客户端： tcp_net_client.c

```
#include "tcp_net_socket.h"
int main(int argc, char* argv[])
{
    if(argc < 3)
    {
    printf("usage:./clienttcp  ip  port\n");
    exit(-1);
    }
    int sfd = tcp_connect(argv[1],atoi(argv[2]));
    char buf[512] = {0};
    send(sfd, "hello", 6, 0);      //向sfd服务端发送数据
    recv(sfd, buf, sizeof(buf), 0); //从sfd服务端接收数据
    puts(buf);
    close(sfd);
}
```

#gcc –o tcp_net_server tcp_net_server.c tcp_net_socket.c #gcc –o tcp_net_client tcp_net_client.c tcp_net_socket.c #./tcp_net_server 192.168.0.164 8888 #./tcp_net_client 192.168.0.164 8888 /* 备注 可以通过将上述经常用到的函数做成动态库，这样以后再用到的时候就可以直接用。步骤如下： gcc –fpic –c tcp_net_socket.c –o tcp_net_socket.o gcc –shared tcp_net_socket.o –o libtcp_net_socket.so cp lib*.so /lib //这样以后就可以直接使用该库了 cp tcp_net_socket.h /usr/include/ //这样头文件包含可以用include <tcp_net_socket.h>了 gcc –o main main.c –ltcp_net_socket //其中main.c要包含头文件 : include <tcp_net_socket.h> ./main */ 注：上面的虽然可以实现多个客户端访问，但是仍然是阻塞模式（即一个客户访问的时候会阻塞不让另外的客户访问）。解决办法有：

多进程（因为开销比较大，所以不常用）

```
#include <tcp_net_socket.h>
int main()
{
int sfd = tcp_init("192.168.0.101", 8888);
while(1)
{
int cfd = tcp_accept(sfd);
if(fork() == 0)
{
send(cfd, "hello", 6, 0);
sleep(10);
close(cfd);
}
else
{
close(cfd);
}
}
```

```
close(sfd);
return 0;
}
```

多线程

```
#include <tcp_net_socket.h>
#include <pthread.h>

void* pthfunc(void* arg)
{
int cfd = (int)arg;
send(cfd, "hello", 6, 0);
sleep(10);
close(cfd);
}
int main()
{
int sfd = tcp_init("192.168.0.101", 8888);
pthread_t pthid = 0;
while(1)
{
    int cfd = tcp_accept(sfd);
    pthread_create(&pthid, NULL, pthfunc, (void*)cfd);
}
    close(sfd);
return 0;
}
```

/* 备注 读写大容量的文件时，通过下面的方法效率很高

ssize_t readn(int fd, char *buf, int size)//读大量内容 { char *pbuf = buf; int total ,nread; for(total = 0; total < size;) { nread=read(fd,pbuf,size-total); if(nread==0) return total; if(nread == -1) { if(errno == EINTR) continue; else return -1; } total += nread; pbuf += nread; } return total; }

ssize_t writen(int fd, char *buf, int size)//写大量内容 { char *pbuf=buf; int total ,nwrite; for(total = 0; total < size;) { nwrite=write(fd,pbuf,size-total); if(nwrite <= 0) { if(nwrite == -1 && errno == EINTR) continue; else return -1; } total += nwrite; pbuf += nwrite; }

return total;

}

*/

调用fcntl将sockfd设置为非阻塞模式。（不常见）

#include <unistd.h> #include <fcntl.h>

sockfd = socket(AF_INET,SOCK_STREAM,0); iflags = fcntl(sockfd, F_GETFL, 0);
fcntl(sockfd,F_SETFL,O_NONBLOCK | iflags);

```
多路选择select
```

#include <sys/select.h> #include "tcp_net_socket.h" #define MAXCLIENT 10 main() { int sfd = tcp_init("192.168.0.164", 8888); int fd = 0; char buf[512] = {0}; fd_set rdset; while(1) { FD_ZERO(&rdset); FD_SET(sfd,&rdset); if(select(MAXCLIENT + 1, &rdset, NULL, NULL, NULL) < 0) continue; for(fd = 0; fd < MAXCLIENT; fd++) { if(FD_ISSET(fd,&rdset)) { if(fd == sfd) { int cfd = tcp_accept(sfd); FD_SET(cfd,&rdset); //..... } else { bzero(buf, sizeof(buf)); recv(fd, buf, sizeof(buf), 0); puts(buf); send(fd, "java", 5, 0); //FD_CLR(fd, &rdset); close(fd); } } } } close(sfd); }

具体例子请参考《网络编程之select.doc》或《tcp_select》

使用UDP协议的流程图

UDP通信流程图如下：

服务端：socket---bind---recvfrom---sendto---close

客户端：socket----------sendto---recvfrom---close

·sendto()函数原型：

int sendto(int sockfd, const void *msg,int len,unsigned int flags,const struct sockaddr *to, int tolen);

该函数比send()函数多了两个参数，to表示目地机的IP地址和端口号信息，而tolen常常被赋值为sizeof (struct sockaddr)。sendto 函数也返回实际发送的数据字节长度或在出现发送错误时返回-1。

·recvfrom()函数原型：

int recvfrom(int sockfd,void *buf,int len,unsigned int flags,struct sockaddr *from,int *fromlen);

from是一个struct sockaddr类型的变量，该变量保存连接机的IP地址及端口号。fromlen常置为sizeof (struct sockaddr)。当recvfrom()返回时，fromlen包含实际存入from中的数据字节数。Recvfrom()函数返回接收到的字节数或 当出现错误时返回-1，并置相应的errno。

Example:UDP的基本操作

服务器端：

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

main()

{

```
int sfd = socket(AF_INET, SOCK_DGRAM, 0);

if(sfd == -1)

{

    perror("socket");

    exit(-1);

}



struct sockaddr_in saddr;

bzero(&saddr, sizeof(saddr));

saddr.sin_family = AF_INET;

saddr.sin_port = htons(8888);

saddr.sin_addr.s_addr = INADDR_ANY;

if(bind(sfd, (struct sockaddr*)&saddr, sizeof(struct sockaddr)) == -1)

{

    perror("bind");

    close(sfd);

    exit(-1);

}



char buf[512] = {0};

while(1)

{
```

```
    struct sockaddr_in fromaddr;

    bzero(&fromaddr, sizeof(fromaddr));

    int fromaddrlen = sizeof(struct sockaddr);

    if(recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr*)&fromaddr,
&fromaddrlen) == -1)

    {

        perror("recvfrom");

        close(sfd);

        exit(-1);

    }
```

printf("receive from %s %d,the message is:%s\n", inet_ntoa(fromaddr.sin_addr), ntohs(fromaddr.sin_port), buf);

```
    sendto(sfd, "world", 6, 0, (struct sockaddr*)&fromaddr, sizeof(struct
sockaddr));
```

}

```
close(sfd);
```

}

客户端：

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

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

{

```
int sfd = socket(AF_INET, SOCK_DGRAM, 0);

if(sfd == -1)

{

    perror("socket");

    exit(-1);

}



struct sockaddr_in toaddr;

bzero(&toaddr, sizeof(toaddr));

toaddr.sin_family = AF_INET;

toaddr.sin_port = htons(atoi(argv[2])); //此处的端口号要跟服务器一样

toaddr.sin_addr.s_addr = inet_addr(argv[1]); //此处为服务器的ip

sendto(sfd, "hello", 6, 0, (struct sockaddr*)&toaddr, sizeof(struct sockaddr));



char buf[512] = {0};

struct sockaddr_in fromaddr;

bzero(&fromaddr, sizeof(fromaddr));

int fromaddrlen = sizeof(struct sockaddr);

if(recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr*)&fromaddr, &fromaddrlen)
== -1)

{

    perror("recvfrom");

    close(sfd);

    exit(-1);
```

```
}
```

printf("receive from %s %d,the message is:%s\n", inet_ntoa(fromaddr.sin_addr), ntohs(fromaddr.sin_port), buf);

```
close(sfd);
```

}

Example：UDP发送文件 先发文件大小 再发文件内容

服务器端：

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

main()

{

```
int sfd = socket(AF_INET, SOCK_DGRAM, 0);
if(sfd == -1)
{
    perror("socket");
    exit(-1);
}
```

```
struct sockaddr_in saddr;

bzero(&saddr, sizeof(saddr));

saddr.sin_family = AF_INET;

saddr.sin_port = htons(8888);

saddr.sin_addr.s_addr = INADDR_ANY;

if(bind(sfd, (struct sockaddr*)&saddr, sizeof(struct sockaddr)) == -1)

{

    perror("bind");

    close(sfd);

    exit(-1);

}



char buf[512] = {0};

struct sockaddr_in fromaddr;

bzero(&fromaddr, sizeof(fromaddr));

int fromaddrlen = sizeof(struct sockaddr);

if(recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr*)&fromaddr, &fromaddrlen)
== -1)

{

    perror("recvfrom");

    close(sfd);

    exit(-1);

}

printf("receive from %s %d,the message is:%s\n", inet_ntoa(fromaddr.sin_addr),
ntohs(fromaddr.sin_port), buf);
```

```
FILE* fp = fopen("1.txt","rb");

struct stat st;  //用于获取文件内容的大小

stat("1.txt", &st);

int filelen = st.st_size;

sendto(sfd, (void*)&filelen, sizeof(int), 0, (struct sockaddr*)&fromaddr,
sizeof(struct sockaddr));

while(!feof(fp))   //表示没有到文件尾

{

    int len = fread(buf,1,sizeof(buf),fp);

    sendto(sfd, buf, len, 0, (struct sockaddr*)&fromaddr, sizeof(struct
sockaddr));
```

}

```
close(sfd);
```

}

客户端：

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#define BUFSIZE 512

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

{

```
int sfd = socket(AF_INET, SOCK_DGRAM, 0);

if(sfd == -1)

{

    perror("socket");

    exit(-1);

}



struct sockaddr_in toaddr;

bzero(&toaddr, sizeof(toaddr));

toaddr.sin_family = AF_INET;

toaddr.sin_port = htons(atoi(argv[2]));

toaddr.sin_addr.s_addr = inet_addr(argv[1]);

sendto(sfd, "hello", 6, 0, (struct sockaddr*)&toaddr, sizeof(struct sockaddr));



char buf[BUFSIZE] = {0};

struct sockaddr_in fromaddr;

bzero(&fromaddr, sizeof(fromaddr));

int fromaddrlen = sizeof(struct sockaddr);

int filelen = 0;   //用于保存文件长度

FILE* fp = fopen("2.txt","w+b");
```

//接收文件的长度

recvfrom(sfd, (void*)&filelen, sizeof(int), 0, (struct sockaddr*)&fromaddr, &fromaddrlen);

```
printf("the length of file is %d\n",filelen);

printf("Create a new file!\n");
```

```
printf("begin to reveive file content!\n");

//接收文件的内容
```

while(1)

```
{

    int len = recvfrom(sfd, buf, sizeof(buf), 0, (struct sockaddr*)&fromaddr,
&fromaddrlen);

    if(len < BUFSIZE)
```

//如果接收的长度小于BUFSIZE，则表示最后一次接收，此时要用break退出循环

```
    {

        fwrite(buf,sizeof(char),len,fp);

        break;

    }

    fwrite(buf,sizeof(char),len,fp);

}

printf("receive file finished!\n");

close(sfd);
```

} 3.3. 设置套接口的选项setsockopt的用法

函数原型：

#include <sys/types.h >

#include <sys/socket.h>

int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

sockfd：标识一个套接口的描述字

level：选项定义的层次；支持SOL_SOCKET、IPPROTO_TCP、IPPROTO_IP和IPPROTO_IPV6

optname：需设置的选项

optval：指针，指向存放选项值的缓冲区

optlen：optval缓冲区长度

全部都必须要放在bind之前，另外通常是用于UDP的。

1. 如果在已经处于 ESTABLISHED状态下的socket(一般由端口号和标志符区分）调用closesocket（一般不会立即关闭而经历TIME_WAIT的过程）后想继续重用该socket：

```
int reuse=1;

setsockopt(s,SOL_SOCKET ,SO_REUSEADDR,(const char*)& reuse,sizeof(int));
```

2. 如果要已经处于连接状态的soket在调用closesocket后强制关闭，不经历TIME_WAIT的过程：

```
int reuse=0;

setsockopt(s,SOL_SOCKET ,SO_REUSEADDR,(const char*)& reuse,sizeof(int));
```

3.在send(),recv()过程中有时由于网络状况等原因，发收不能预期进行,而设置收发时限：

```
int nNetTimeout=1000; // 1秒

// 发送时限

setsockopt(socket，SOL_S0CKET,SO_SNDTIMEO，(char *)&nNetTimeout,sizeof(int));

// 接收时限

setsockopt(socket，SOL_S0CKET,SO_RCVTIMEO，(char *)&nNetTimeout,sizeof(int));
```

4.在send()的时候，返回的是实际发送出去的字节(同步)或发送到socket缓冲区的字节(异步),系统默认的状态发送和接收一次为8688字节(约为8.5K)；在实际的过程中发送数据和接收数据量比较大，可以设置socket缓冲区，而避免了send(),recv()不断的循环收发：

```
// 接收缓冲区

int nRecvBuf=32*1024; // 设置为32K

setsockopt(s,SOL_SOCKET,SO_RCVBUF,(const char*)&nRecvBuf,sizeof(int));

// 发送缓冲区

int nSendBuf=32*1024; // 设置为32K

setsockopt(s,SOL_SOCKET,SO_SNDBUF,(const char*)&nSendBuf,sizeof(int));
```

5. 如果在发送数据时，希望不经历由系统缓冲区到socket缓冲区的拷贝而影响程序的性能：

```
int nZero=0;

setsockopt(socket，SOL_SOCKET,SO_SNDBUF，(char *)&nZero,sizeof(int));
```

6.同上在recv()完成上述功能(默认情况是将socket缓冲区的内容拷贝到系统缓冲区)：

```
int nZero=0;

setsockopt(socket，SOL_SOCKET,SO_RCVBUF，(char *)&nZero,sizeof(int));
```

7.一般在发送UDP数据报的时候，希望该socket发送的数据具有广播特性：

```
int bBroadcast = 1;
setsockopt(s,SOL_SOCKET,SO_BROADCAST,(const char*)&bBroadcast,sizeof(int));
```