# 网络编程基础API

## socket地址API

### 主机字节和网络字节转换API

目前CPU的累加器一次能够装在4个字节，那么这4个字节在内存中的排列顺序影响累加器累加的值，这就是字节序的问题。目前，PC都采用小端字节序。但是，格式化的数据在两台不同字节序的主机之间传递时，接收端必然会错误地解释。解决的办法就是发送端总是将字节序转换为大端字节序，然后发送，接收端知道传过来的是大端字节序，因此接收端总是能正确地接收数据。

#include<netinet/in.h>

//主机字节转换成网络字节

unsigned short int htons(unsigned short int hostshrot);

//网络字节转换成主机字节

unsigned short int ntohs(unsigned short int netshrot);

### IP转换函数

专用socket地址结构体：

//TCP/IP协议族专用socket地址结构体

struct sockaddr\_in

{

sa\_family\_t sin\_family; //地址族：AF\_INET

u\_int16\_t sin\_port; //端口号，用于网络字节序表示

struct in\_addr sin\_addr; //IPv4地址结构体，是u\_int32\_t s\_addr

};

#include <arpa/inet.h>

//将字符串IP转换为整数IP

int inet\_pton(int af, const char\* src,void\* dst);

//将整数IP转换成字符串IP

const char\* inet\_ntop(int af, const void\* dst,socklen\_t cnt);

## 创建socket

#include<sys/types>

#include<sys/socket.h>

/\*

domain：底层协议族类型：对于TCP/IP而言AF\_INET/AF\_INET6

type：指定服务类型

protocol：具体的协议

\*/

int socket(int domain, int type, int protocol);

命名socket：

//服务器绑定网络地址

int bind(int sockfd, const struct sockeaddr\* addr,socketlen\_t addrlen);

bind函数成功返回0，否则为-1，常见error类型：

EACCES：被绑定的地址收到保护

EADDRINUSE：被绑定的地址正在使用中

## 监听socket

int listen(int sockfd,int backlog);

## 接收连接accept

### 原型

int accept(int sockfd,struct sockaddr\* addr,socklen\_t\* addrlen);

从listen监听队列中接收一个连接

### 连接过程

## 发起连接connect



### 原型

int connect(int sockfd,const struct sockaddr\* serv\_addr,socklen\_t

addrlen);

connect函数成功返回0，否则为-1，常见error类型：

ECONNEFUSED：目标端口不存在

ETIMEOUT：连接超时

### 发送过程



## 关闭连接close/shutdown

### 原型

int close(int fd);

说明：close并不是真正关闭连接，而是将文件描述符fd的引用数-1，只有当fd引用次数为0时，才会关闭连接。如果想立即关闭连接，可以使用shutdown。

int shutdown(int sockfd,int howto);

shutdown函数中howto参数选项：

SHUT\_RE：关闭读这一半

SHUT\_WR：关闭写这一半

SHUT\_RDWR：同时关闭读和写

### 关闭过程



## 数据读写recv/send

对文件的读写，系统调用read、write同样适用于socket，但是这里提供了专用于socket的recv和send。

TCP数据读写API：

ssize\_t recv(int sockfd,void\* buf,size\_t len,int flags);

ssize\_t send(int sockfd,cons void\* buf,size\_t len,int flags);

recv函数和send函数的flag参数选项：

MSG\_WAITALL：读操作读取到一定数量字节后返回

MSG\_PEEK：查看读缓冲找那个的数据而不取出

UDP数据读写API：

ssize\_t recvfrom(int sockfd,void\* buf,size\_t len,int flags,struct sockaddr\* src\_addr,socklen\_t\* addrlen);

ssize\_t sendto(int sockfd,cons void\* buf,size\_t len,int flags,const struct sockaddr\* dest\_addr,sockelen\_t addrlen);

因为UDP通信没有连接的概念，所以每次读取数据都需要获取发送端数据的大小。

# 套接字选项

## SOL\_SOCKET协议族选项

### 获取和设置套接字选项

#### 函数

#include<sys/socket.h>

int getsockopt(int s, int level, int optname, void \*optval.socklen\_t\*optlen);

int setsockopt(int s, int level, int optname, void\*optval.socklen\_t optlen);

参数：

s：将要设置或获取的套接字描述符

optname：操作内存缓冲区

optval.socklen\_t optlen：第4个参数长度

#### 错误代码

getsockopt()和setsockopt函数错误代码含义：

EBADF：参数s不是有效的文件描述符

EFAULT：optval指向的内存并非有效的进程空间

EINVAL：在调用setsockopt函数时，optlen无效

ENOPROTOOPT：指定的协议层不能识别选项

ENOTSOCK：s描述的不是套接字描述符

#### level

按照参数选项界别level值的不同，套接字选项可以分为3类：

通用套接选项：level的值为SOL\_SOCKET

IP选项：level的值为IPPROTO\_IP

TCP选项：level的值为IPPROTO\_TCP

### SO\_KEEPALIVE选项

SO\_KEEPALIVE 保持连接选项，用于设置TCP/IP的连接，2小时没有数据交互，发送探测报文，有三种回应：

回应一个ACK报文

回应一个RST报文

没有任何回应

注：通常在服务端设置，用于长时间没有响应的TCP连接场景。

使用：

int optval=1;

setsockopt(sockfd, SOL\_SOCKET , SO\_KEEPALIVE ，&optval, sizeof(iptval));

### SO\_LINGER选项

SO\_LINGER 缓冲区处理方式选项，主要用于设置TCP连接关闭（即系统调用close）时的方式，它的操作通过一个结构体进行：

struct linger{

int l\_onoff; //开启（非0）还是关闭（0）该选项

int l\_linger; //滞留时间

};

SO\_LINGER选项使用方法有三种：

l\_onoff等于0，此时SO\_LINGER选项不起作用

l\_onoff不为0，l\_linger为0：close系统调用立即返回，TCP模块将丢弃被关闭的套接字对应的TCP发送缓冲区中残留的数据，同时给对方发送一个复位报文段

l\_onoff不为0，l\_linger不为0：此时close行为取决于两个条件：1、被关闭的套接字对应的TCP发送缓冲区中是否含有残留的数据；2、该套接字是阻塞还是非阻塞，对于阻塞的，close将等待l\_linger时间，直到TCP模块发送完所有缓冲区残留数据，并得到对方确认，否则close系统调用返回-1，如果是非阻塞的，则返回-1

使用方法：

optval.l\_onoff = 1;

optval.l\_linger = 60;

setsockopt(sockfd, SOL\_SOCKET, SO\_LINGER, &optval, sizeof(optval));

### SO\_RCVBUF和SO\_SNDBUF选项

SO\_RCVBUF和SO\_SNDBUF缓冲区大小选项：

1、设置TCP/UDP连接的接收/发送缓冲区的含义：在UDP连接中，由于是无连接，在发送缓冲区在数据通过网络设备发送后就可以丢弃，不用保存，而接收缓冲区需要保存数据，直到应用程序读取，由于UDP没有流量控制，当缓冲区过小时，局部时间内会产生爆发性数据传输，因此在使用UDP时需要将接收缓冲区调整到比较大的值；在TCP连接中，接收缓冲区的大小就是滑动窗口的大小，TCP的接收缓冲区不可能溢出，因为不允许对方发送超过接收缓冲区大小的数据。

2、在connect()函数调用之前设置：滑动窗口的协商是在建立连接时，通过同步报文获得的，对于客户端程序而言，接收缓冲区的大小在connect系统调用之前设置，因为connect需要通过同步报文段进行连接，而对于 服务器而言需要在listen之前设置接收缓冲区的大小，因为accept返回的套接字描述符是继承了listen的描述符属性。

发送缓冲区设置：

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#define BUFFER\_SIZE 1024

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

{

if(argc <= 2)

{

printf("please input ip address and port number!\r\n");

return -1;

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

struct sockaddr\_in server\_address;

bzero(&server\_address,sizeof(server\_address));

server\_address.sin\_family = AF\_INET;

inet\_pton(AF\_INET,ip,&server\_address.sin\_addr);

server\_address.sin\_port = htons(port);

int sock = socket(AF\_INET,SOCK\_STREAM,0);

if(sock < 0)

{

printf("create socket failed !\r\n");

return -1;

}

int sendbuf = atoi(argv[3]);

int len = sizeof(sendbuf);

setsockopt(sock,SOL\_SOCKET,SO\_SNDBUF,&sendbuf,sizeof(sendbuf));

getsockopt(sock,SOL\_SOCKET,SO\_SNDBUF,&sendbuf,(socklen\_t\*)&len);

printf("tcp send buffer size after setting is %d\r\n",sendbuf);

if(connect(sock,(struct sockaddr\*)&server\_address,sizeof(server\_address)) != -1)

{

char buffer[BUFFER\_SIZE];

memset(buffer,'a',BUFFER\_SIZE);

send(sock,buffer,BUFFER\_SIZE,0);

}

close(sock);

return 0;

}

接收缓冲区设置：

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#define BUFFER\_SIZE 1024

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

{

if(argc <= 2)

{

printf("please input ip address and port number!\r\n");

return -1;

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

struct sockaddr\_in server\_address;

bzero(&server\_address,sizeof(server\_address));

server\_address.sin\_family = AF\_INET;

inet\_pton(AF\_INET,ip,&server\_address.sin\_addr);

server\_address.sin\_port = htons(port);

int sock = socket(AF\_INET,SOCK\_STREAM,0);

if(sock < 0)

{

printf("create socket failed !\r\n");

return -1;

}

int recvbuf = atoi(argv[3]);

int len = sizeof(recvbuf);

setsockopt(sock,SOL\_SOCKET,SO\_RCVBUF,&recvbuf,sizeof(recvbuf));

getsockopt(sock,SOL\_SOCKET,SO\_RCVBUF,&recvbuf,(socklen\_t\*)&len);

printf("the tcp receive buffer size after setting is %d\r\n",recvbuf);

int ret = bind(sock,(struct sockaddr\*)&server\_address,sizeof(server\_address));

if(ret == -1)

{

printf("bind failed !\r\n");

return -1;

}

ret = listen(sock,5);

if(ret == -1)

{

printf("listen failed !\r\n");

return -1;

}

struct sockaddr\_in client;

socklen\_t client\_len = sizeof(client);

int connfd = accept(sock,(struct sockaddr\*)&client,&client\_len);

if(connfd < 0)

{

printf("accept failed !\r\n");

}

else

{

char buffer[BUFFER\_SIZE];

memset(buffer,0x00,sizeof(buffer));

while(recv(connfd,buffer,BUFFER\_SIZE,0) > 0) {}

close(connfd);

}

close(sock);

return 0;

}

### SO\_RCVLOWAT和SO\_SNDLOWAT选项

SO\_RCVTIMEO表示接收超时，SO\_SNDTIMEO表示发送超时（发送超时与接收超时与IO复用中select部分功能是一致的），设置是通过一个结构体来实现：

struct timeval{

time\_t tv\_sec;

seconds\_t tv\_usec;

};

### SO\_RCVTIMEO和SO\_SNDTIMEO选项

SO\_RCVLOWAT和SO\_SNDLOWAT选项表示接收缓冲区和发送缓冲区的低水位标记（一般被IO复用调用，判断socket是否可读/可写）。

当TCP接收缓冲区可读数据的总数大于其低水位标记时，IO复用系统调用将通知应用程序可以对对应的socket上读取数据；当TCP发送缓冲区中的空闲空间，即可写入数据的空间大于其低水位标记时，IO复用系统调用将通知应用程序可以对对应的socket上写入数据。

在默认情况下，TCP接收缓冲区的低水位标记和TCP发送缓冲区的低水位标记都是为1字节。

### SO\_REUSEADDR选项

SO\_REUSEADDR地址可重用选项，设置此选项可以使用被处于TIME\_WAIT状态的连接占用的socket地址：

int reuse=1；

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse));

实例：

int sock = socket(AF\_INET,SOCK\_STREAM,0);

assert(sock >=0);

int reuse = 1;

setsockopt(sock,SOL\_SOCKET,SO\_REUSEADDR,&reuse,sizeof(reuse));

struct sockaddr\_in address;

bzero(&address,sizeof(address));

address.sin\_family = AF\_INET;

inet\_pton(AF\_INET,ip,&address.sin\_addr);

address.sin\_port = htons(port);

int ret = bind(sock,(struct sockaddr\*)&address,sizeof(address));

## IPPROTO\_TCP和IPPROTO\_IP选项

### TCP\_KEEPALIVE选项

TCP\_KEEPALIVE用来获取或设置存活探测的时间间隔（在SO\_KEEPALIVE设置的情况下才有效，默认情况下2小时）：

int alivetime = 60;

int fd = socket(AF\_INET, SOCK\_STREAM, 0);

setsockopt(fd, IPPOTO\_TCP, TCP\_KEPALIVE, &alivetime. Sizeof(alivetime));

### TCP\_NODELAY和TCP\_CORK选项

TCP\_NODELAY和TCP\_CORK针对Nagle算法，此算法的基本原理如下：

1. 将小分组封装成大的分组/帧再发送
2. 使用延迟确认

TCP\_NODELAY选项的作用：客户端的请求不需要和其他分组合并

TCP\_CORK选项作用：需要等到发送的数据量达到最大值，一次性发送全部数据

### IP\_TOS选项

IP\_TOS可以设置服务类型，服务类型选项：

IPTOS\_LOWDELAY：表示最小延迟

IPTOS\_RELIABLITY：表示最大可靠性

IPTOS\_THROUGHPUT：表示最大吞吐量

IPTOS\_LOWCOST：表示最小成本

### IP\_TTL选项

IP\_TTL是一种生存时间选项，而生成时间选项：

可以获取或设置发送报文的TTL值（一般为64.对于套接字为255）

可以调整网络数据发送速度

实例：

int current\_TTL =0;

int set\_TTL = 32;

int length\_TTL =sizeof(int);

getsockopt(s,IPPROTO\_IP,IP\_TTL,&current\_TTL,&lenght\_TTL);

setsockopt(s,IPPROTO\_IP,IP\_TTL,&set\_TTL,length\_TTL);

sendto(s,buffer,buffer\_len,0,(struct sockaddr\*)&remote\_ip,sizeof(struct sockaddr));

setsockopt(s,IPPROTO\_IP,IP\_TTL,&current\_TTL,length\_TTL);

## ioctl()函数

### ioctl函数命令选项

ioctl函数和协议栈交互，主要的交互类型：

IO

文件

网络接口

路由

ARP

### ioctl函数IO请求

套接字IO操作的命令请求有6个，第3个参数为整形指针：

SIOCATMARK：查看TCP连接是否有带外数据

SIOCSPGRP和FIOSETOWN：处理套接字的SIGIO和SIGURG信号

SIOCGPGRP和FIOGETOWN：设置接收SIGIO信号的进程ID

SIOCGSTAMP：获取最后一个数据报达到的事件

### ioctl函数文件请求

文件请求的命令有3个：

FIONBIO：用于设置或清除套接字的非阻塞标志

FIOASYNC：设置或者清除套接字的异步信号

FIONREAD：获得当前套接字接收缓冲区的字节数

### ioctl函数网络接口请求

获取网络接口的不同参数的不同命令选项：

SIOCGIFADDR和SIOCSIFADDR：获取和设置本地IP地址

SIOCGIFMTU和SIOCSIFMTU：获取和设置MTU

SIOCGIFHWADDR和SIOCSIFHADDR：获取和设置硬件地址

### ioctl函数对ARP告诉缓存操作

ioctl函数获取ARP高速缓存，有三个命令字：

SIOCDARP：删除高速缓存中的一个记录

SIOCARP：设置或者修改一个记录

SIOCGARP：获得一个记录

# TCP服务端/客户端开发

## 流程



## 示例

### 客户端

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <assert.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

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

{

if(argc <= 2)

{

printf("please input server ip and port number\r\n");

return -1;

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

struct sockaddr\_in server\_address;

bzero(&server\_address,sizeof(server\_address));

server\_address.sin\_family = AF\_INET;

inet\_pton(AF\_INET,ip,&server\_address.sin\_addr);

server\_address.sin\_port = htons(port);

int sockfd = socket(AF\_INET,SOCK\_STREAM,0);

if(sockfd < 0)

{

printf("socket create failed \r\n");

return -1;

}

if(connect(sockfd,(struct sockaddr\*)&server\_address,sizeof(server\_address))<0)

{

printf("connect failed \r\n");

return -1;

}

else

{

char data[64]="Where are you from ?";

char buffer[64];

int i,len;

for(i=0;i<10;i++)

{

memset(buffer,0x00,sizeof(buffer));

send(sockfd,data,strlen(data),0);

len = recv(sockfd,buffer,sizeof(buffer),0);

sleep(1);

printf("index is %d len is %d : %s\r\n",i,len,buffer);

}

}

close(sockfd);

return 0;

}

### 服务端

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <assert.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

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

{

if(argc <=2 )

{

printf("please input ipaddress and port number\r\n");

return -1;

}

const char\* ip = argv[1];

int port = atoi(argv[2]);

struct sockaddr\_in address;

bzero(&address,sizeof(address));

address.sin\_family = AF\_INET;

inet\_pton(AF\_INET,ip,&address.sin\_addr);

address.sin\_port = htons(port);

int sock = socket(AF\_INET,SOCK\_STREAM,0);

if(sock <0)

{

printf("create sock failed \r\n");

return -1;

}

int ret = bind(sock,(struct sockaddr\*)&address,sizeof(address));

if(ret == -1)

{

printf("bind failed\r\n");

return -1;

}

ret = listen(sock,5);

if(ret == -1)

{

printf("listen failed \r\n");

return -1;

}

struct sockaddr\_in client;

socklen\_t client\_addrlength = sizeof(client);

int connfd = accept(sock,(struct sockaddr\*)&client,&client\_addrlength);

if(connfd < 0)

{

printf("accept failed \r\n");

}

else

{

char data[64]="I am from jikexueyuan!";

char buffer[64];

int i,len;

for(i=0;i<10;i++)

{

memset(buffer,0x00,sizeof(buffer));

len = recv(connfd,buffer,sizeof(buffer),0);

printf("index is %d len is %d: %s\r\n",i,len,buffer);

sleep(1);

send(connfd,data,strlen(data),0);

}

close(connfd);

}

close(sock);

return 0;

}

# UDP服务端/客户端开发

## 流程

## 示例

### 客户端

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define PORT\_SERVER 8888

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

{

int s;

struct sockaddr\_in addr\_server,addr\_client;

const char\* ip = "192.168.80.130";

bzero(&addr\_server,sizeof(addr\_server));

inet\_pton(AF\_INET,ip,&addr\_server.sin\_addr);

addr\_server.sin\_family = AF\_INET;

addr\_server.sin\_port = htons(PORT\_SERVER);

s = socket(AF\_INET,SOCK\_DGRAM,0);

printf("s is %d \r\n",s);

char data[64]="Where are you from ?";

char buffer[64];

int i;

int len = sizeof(addr\_server);

for(i=0;i<10;i++)

{

memset(buffer,0x00,sizeof(buffer));

int ret = sendto(s,data,strlen(data),0,(struct sockaddr\*)&addr\_server,len);

recvfrom(s,buffer,sizeof(buffer),0,(struct sockaddr\*)&addr\_client,&len);

printf("index is %d : %s \r\n",i,buffer);

sleep(1);

}

close(s);

return 0;

}

### 服务端

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define PORT\_SERVER 8888

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

{

int s;

struct sockaddr\_in addr\_server,addr\_client;

const char\* ip = "192.168.80.130";

s = socket(AF\_INET,SOCK\_DGRAM,0);

memset(&addr\_server,0x00,sizeof(addr\_server));

addr\_server.sin\_family = AF\_INET;

addr\_server.sin\_addr.s\_addr = htonl(INADDR\_ANY);

addr\_server.sin\_port = htons(PORT\_SERVER);

bind(s,(struct sockaddr\*)&addr\_server,sizeof(addr\_server));

char data[64]="I am from jikexueyuan!";

char buffer[64];

int i;

int len = sizeof(addr\_server);

for(i=0;i<10;i++)

{

memset(buffer,0x00,sizeof(buffer));

recvfrom(s,buffer,sizeof(buffer),0,(struct sockaddr\*)&addr\_client,&len);

printf("index is %d : %s \r\n",i,buffer);

sendto(s,data,strlen(data),0,(struct sockaddr\*)&addr\_client,len);

sleep(1);

}

close(s);

return 0;

}