Linux之socket详解

目录

一、什么是Socket

1.1 Socket建立

1.2 Socket配置(绑定)

1.3 连接建立

1.4 数据传输

1.5 结束传输

二、编程DEMO

2.1 TCP编程

2.2 UDP编程

一、什么是Socket

Socket接口是TCP/IP网络的API，Socket接口定义了许多函数或例程，程序员可以用它们来开发TCP/IP网络上的应用程序。要学Internet上的TCP/IP网络编程，必须理解Socket接口

Socket接口设计者最先是将接口放在Unix操作系统里面的。如果了解Unix系统的输入和输出的话，就很容易了解Socket了。网络的 Socket数据传输是一种特殊的I/O，Socket也是一种文件描述符。Socket也具有一个类似于打开文件的函数调用Socket()，该函数返 回一个整型的Socket描述符，随后的连接建立、数据传输等操作都是通过该Socket实现的。常用的Socket类型有两种：流式Socket （SOCK\_STREAM）和数据报式Socket（SOCK\_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；数据 报式Socket是一种无连接的Socket，对应于无连接的UDP服务应用

1.1 Socket建立

为了建立Socket，程序可以调用Socket函数，该函数返回一个类似于文件描述符的句柄。socket函数原型为：

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

domain指明所使用的协议族，通常为AF\_INET，表示互联网协议族（TCP/IP协议族）；type参数指定socket的类型： SOCK\_STREAM 或SOCK\_DGRAM，Socket接口还定义了原始Socket（SOCK\_RAW），允许程序使用低层协议；protocol通常赋值"0"。 Socket()调用返回一个整型socket描述符，你可以在后面的调用使用它

domain的选项，AF\_INET\_IPv4：因特网域；AF\_INET6\_IPv6因特网域；AF\_UNIX：Unix域；AF\_ROUTE路由套接字；AF\_KEY密钥套接字；AF\_UNSPEC：未指定；

type的选项，SOCK\_STREAM：流式套接字提供可靠的、面向连接的通信流:它使用TCP协议，从而保证了数据传输的正确性和顺序性(TCP：可靠的、重传、有连接的，一般用于控制命令)；SOCK\_DGRAM：数据报套接字定义了一种无连接的服，数据通过相互独立的报文进行传输，是无序的，并且不保证是可靠、无差错的。它使用数据报协议(UDP：不可靠的、无连接，大数据传输，数据可能会丢失)

protocol的选项，0选择type类型对应的默认协议；IPPROTO\_TCP：TCP传输协议；IPPROTO\_UDP：UDP传输协议；IPPROTO\_SCTP：SCTP传输协议；IPPROTO\_TIPC：TIPC传输协议

Socket描述符是一个指向内部数据结构的指针，它指向描述符表入口。调用Socket函数时，socket执行体将建立一个Socket，实际上"建立一个Socket"意味着为一个Socket数据结构分配存储空间。Socket执行体为你管理描述符表

两个网络程序之间的一个网络连接包括五种信息：通信协议、本地协议地址、本地主机端口、远端主机地址和远端协议端口。Socket数据结构中包含这五种信息

1.2 Socket配置(绑定)

通过socket调用返回一个socket描述符后，在使用socket进行网络传输以前，必须配置该socket。面向连接的socket客户端通过 调用Connect函数在socket数据结构中保存本地和远端信息。无连接socket的客户端和服务端以及面向连接socket的服务端通过调用 bind函数来配置本地信息。

Bind函数将socket与本机上的一个端口相关联，随后你就可以在该端口监听服务请求。Bind函数原型为：

int bind(int sockfd,struct sockaddr \*my\_addr, int addrlen);

sockfd是调用socket函数返回的socket描述符，my\_addr是一个指向包含有本机IP地址及端口号等信息的sockaddr类型的指针；addrlen常被设置为sizeof(struct sockaddr)

struct sockaddr {

unsigned short sa\_family; /\* 地址族， AF\_xxx \*/

char sa\_data[14]; /\* 14 字节的协议地址 \*/

};

对于sockaddr中的sa\_data包含了端口号加IP地址，我们可以等同于如下结构体，两个结构体的大小是相等的，其中主要是把sa\_data的IP和端口号分出来了，对于sin\_zero数组只是填充用，并没有实际意义，只是为了跟sockaddr结构内存对齐，这样两者才能相互转换

struct sockaddr\_in {

sa\_family\_t sin\_family;

in\_port\_t sin\_port;

struct in\_addr sin\_addr;

unsigned char sin\_zero[8];

}

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，也可以自己来指定端口号。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(注意在使用bind函数是需要将sin\_port和sin\_addr转换成为网络字节优先顺序；而sin\_addr则不需要转换，如果sin\_add是自己指定的IP地址就需要我们自己来转换为网络字节序)

计算机数据存储有两种字节优先顺序：高位字节优先和低位字节优先。Internet上数据以高位字节优先顺序在网络上传输，所以对于在内部是以低位字节优先方式存储数据的机器，在Internet上传输数据时就需要进行转换，否则就会出现数据不一致。下面有几个字节顺序转换函数

htonl() //把32位值从主机字节序转换成网络字节序

htons() //把16位值从主机字节序转换成网络字节序

ntohl() //把32位值从网络字节序转换成主机字节序

ntohs() //把16位值从网络字节序转换成主机字节序

Bind()函数在成功被调用时返回0；出现错误时返回"-1"并将errno置为相应的错误号。需要注意的是，在调用bind函数时一般不要将端口号置为小于1024的值，因为1到1024是保留端口号，你可以选择大于1024中的任何一个没有被占用的端口号

地址转换API

把字符串形式的"192.168.1.123"转为网络能识别的格式

int inet\_aton(const char\* straddr,struct in\_addr \*addrp)

把网络格式的p地址转为字符串形式

char\* inet\_ntoa(struct in\_addr inaddr);

1.3 连接建立

面向连接的客户程序使用Connect函数来配置socket并与远端服务器建立一个TCP连接，其函数原型为：

int connect(int sockfd, struct sockaddr \*serv\_addr,int addrlen);

sockfd是socket函数返回的socket描述符；serv\_addr是包含远端主机IP地址和端口号的指针；addrlen是远端地质结构的长度。 Connect函数在出现错误时返回-1，并且设置errno为相应的错误码

listen函数使socket处于被动的监听模式，并为该socket建立一个输入数据队列，将到达的服务请求保存在此队列中，直到程序处理它们

int listen(int sockfd， int backlog);

sockfd是Socket系统调用返回的socket描述符；backlog指定在请求队列中允许的最大请求数，进入的连接请求将在队列中等待accept()它们。Backlog对队列中等待 服务的请求的数目进行了限制，大多数系统缺省值为20。如果一个服务请求到来时，输入队列已满，该socket将拒绝连接请求，客户将收到一个出错信息。

当出现错误时listen函数返回-1，并置相应的errno错误码

accept()函数让服务器接收客户的连接请求。在建立好输入队列后，服务器就调用accept函数，然后睡眠并等待客户的连接请求

int accept(int sockfd, void \*addr, int \*addrlen);

sockfd是被监听的socket描述符，addr通常是一个指向sockaddr\_in变量的指针，该变量用来存放提出连接请求服务的主机的信息（某 台主机从某个端口发出该请求）；addrten通常为一个指向值为sizeof(struct sockaddr\_in)的整型指针变量。出现错误时accept函数返回-1并置相应的errno值

1.4 数据传输

send()和recv()这两个函数用于面向连接的socket上进行数据传输。

send()函数原型为：

int send(int sockfd, const void \*msg, int len, int flags);

sockfd是你想用来传输数据的socket描述符；msg是一个指向要发送数据的指针；Len是以字节为单位的数据的长度；flags一般情况下置为0（关于该参数的用法可参照man手册）。

Send()函数返回实际上发送出的字节数，可能会少于你希望发送的数据。在程序中应该将send()的返回值与欲发送的字节数进行比较。当send()返回值与len不匹配时，应该对这种情况进行处理。

recv()函数原型为：

int recv(int sockfd,void \*buf,int len,unsigned int flags);

sockfd是接受数据的socket描述符；buf 是存放接收数据的缓冲区；len是缓冲的长度。Flags也被置为0。Recv()返回实际上接收的字节数，当出现错误时，返回-1并置相应的errno值。sendto()和recvfrom()用于在无连接的数据报socket方式下进行数据传输。由于本地socket并没有与远端机器建立连接，所以在发送数据时应指明目的地址。

sendto()函数原型为：

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()函数原型为：

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，该变量保存源机的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。如果你对数据报socket调用了connect()函数时，你也可以利用send()和recv()进行数据传输，但该socket仍然是数据报socket，并且利用传输层的UDP服务。但在发送或接收数据报时，内核会自动为之加上目地和源地址信息。

1.5 结束传输

当所有的数据操作结束以后，你可以调用close()函数来释放该socket，从而停止在该socket上的任何数据操作：

close(sockfd);

你也可以调用shutdown()函数来关闭该socket。该函数允许你只停止在某个方向上的数据传输，而一个方向上的数据传输继续进行。如你可以关闭某socket的写操作而允许继续在该socket上接受数据，直至读入所有数据

int shutdown(int sockfd,int how);

二、编程DEMO

2.1 TCP编程

对于server来说需要4步：socket、bind、listen、accept；对于client来说需要两步：socket、connect

可以用read/write来传输数据，也可以用send/recv来传输数据

server.c：

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdio.h>

#include <signal.h>

/\* socket

\* bind

\* listen

\* accept

\* send/recv

\*/

#define SERVER\_PORT 8888

#define BACKLOG 10

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

{

int iSocketServer;

int iSocketClient;

struct sockaddr\_in tSocketServerAddr;

struct sockaddr\_in tSocketClientAddr;

int iRet;

int iAddrLen;

int iRecvLen;

unsigned char ucRecvBuf[1000];

int iClientNum = -1;

signal(SIGCHLD,SIG\_IGN);

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

if (-1 == iSocketServer)

{

printf("socket error!\n");

return -1;

}

tSocketServerAddr.sin\_family = AF\_INET;

tSocketServerAddr.sin\_port = htons(SERVER\_PORT); /\* host to net, short \*/

tSocketServerAddr.sin\_addr.s\_addr = INADDR\_ANY;

memset(tSocketServerAddr.sin\_zero, 0, 8);

iRet = bind(iSocketServer, (const struct sockaddr \*)&tSocketServerAddr, sizeof(struct sockaddr));

if (-1 == iRet)

{

printf("bind error!\n");

return -1;

}

iRet = listen(iSocketServer, BACKLOG);

if (-1 == iRet)

{

printf("listen error!\n");

return -1;

}

while (1)

{

iAddrLen = sizeof(struct sockaddr);

iSocketClient = accept(iSocketServer, (struct sockaddr \*)&tSocketClientAddr, &iAddrLen);

if (-1 != iSocketClient)

{

iClientNum++;

printf("Get connect from client %d : %s\n", iClientNum, inet\_ntoa(tSocketClientAddr.sin\_addr));

if (!fork())

{

/\* 子进程的源码 \*/

while (1)

{

/\* 接收客户端发来的数据并显示出来 \*/

iRecvLen = recv(iSocketClient, ucRecvBuf, 999, 0);

if (iRecvLen <= 0)

{

close(iSocketClient);

return -1;

}

else

{

ucRecvBuf[iRecvLen] = '\0';

printf("Get Msg From Client %d: %s\n", iClientNum, ucRecvBuf);

}

}

}

}

}

close(iSocketServer);

return 0;

}

client.c：

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdio.h>

/\* socket

\* connect

\* send/recv

\*/

#define SERVER\_PORT 8888

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

{

int iSocketClient;

struct sockaddr\_in tSocketServerAddr;

int iRet;

unsigned char ucSendBuf[1000];

int iSendLen;

if (argc != 2)

{

printf("Usage:\n");

printf("%s <server\_ip>\n", argv[0]);

return -1;

}

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

tSocketServerAddr.sin\_family = AF\_INET;

tSocketServerAddr.sin\_port = htons(SERVER\_PORT); /\* host to net, short \*/

//tSocketServerAddr.sin\_addr.s\_addr = INADDR\_ANY;

if (0 == inet\_aton(argv[1], &tSocketServerAddr.sin\_addr))

{

printf("invalid server\_ip\n");

return -1;

}

memset(tSocketServerAddr.sin\_zero, 0, 8);

iRet = connect(iSocketClient, (const struct sockaddr \*)&tSocketServerAddr, sizeof(struct sockaddr));

if (-1 == iRet)

{

printf("connect error!\n");

return -1;

}

while (1)

{

if (fgets(ucSendBuf, 999, stdin))

{

iSendLen = send(iSocketClient, ucSendBuf, strlen(ucSendBuf), 0);

if (iSendLen <= 0)

{

close(iSocketClient);

return -1;

}

}

}

return 0;

}

2.2 UDP编程

对于server来说只需要2步：socket、bind；对于client来说可一步也可以两步：socket、connect(非必须)

用sendto/recvfrom来传输数据，对于client有connect可以用send/recv来传输数据

server.c：

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdio.h>

#include <signal.h>

/\* socket

\* bind

\* sendto/recvfrom

\*/

#define SERVER\_PORT 8888

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

{

int iSocketServer;

int iSocketClient;

struct sockaddr\_in tSocketServerAddr;

struct sockaddr\_in tSocketClientAddr;

int iRet;

int iAddrLen;

int iRecvLen;

unsigned char ucRecvBuf[1000];

int iClientNum = -1;

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

if (-1 == iSocketServer)

{

printf("socket error!\n");

return -1;

}

tSocketServerAddr.sin\_family = AF\_INET;

tSocketServerAddr.sin\_port = htons(SERVER\_PORT); /\* host to net, short \*/

tSocketServerAddr.sin\_addr.s\_addr = INADDR\_ANY;

memset(tSocketServerAddr.sin\_zero, 0, 8);

iRet = bind(iSocketServer, (const struct sockaddr \*)&tSocketServerAddr, sizeof(struct sockaddr));

if (-1 == iRet)

{

printf("bind error!\n");

return -1;

}

while (1)

{

iAddrLen = sizeof(struct sockaddr);

iRecvLen = recvfrom(iSocketServer, ucRecvBuf, 999, 0, (struct sockaddr \*)&tSocketClientAddr, &iAddrLen);

if (iRecvLen > 0)

{

ucRecvBuf[iRecvLen] = '\0';

printf("Get Msg From %s : %s\n", inet\_ntoa(tSocketClientAddr.sin\_addr), ucRecvBuf);

}

}

close(iSocketServer);

return 0;

}

client.c：

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <stdio.h>

/\* socket

\* connect

\* send/recv

\*/

#define SERVER\_PORT 8888

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

{

int iSocketClient;

struct sockaddr\_in tSocketServerAddr;

int iRet;

unsigned char ucSendBuf[1000];

int iSendLen;

int iAddrLen;

if (argc != 2)

{

printf("Usage:\n");

printf("%s <server\_ip>\n", argv[0]);

return -1;

}

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

tSocketServerAddr.sin\_family = AF\_INET;

tSocketServerAddr.sin\_port = htons(SERVER\_PORT); /\* host to net, short \*/

//tSocketServerAddr.sin\_addr.s\_addr = INADDR\_ANY;

if (0 == inet\_aton(argv[1], &tSocketServerAddr.sin\_addr))

{

printf("invalid server\_ip\n");

return -1;

}

memset(tSocketServerAddr.sin\_zero, 0, 8);

#if 0

iRet = connect(iSocketClient, (const struct sockaddr \*)&tSocketServerAddr, sizeof(struct sockaddr));

if (-1 == iRet)

{

printf("connect error!\n");

return -1;

}

#endif

while (1)

{

if (fgets(ucSendBuf, 999, stdin))

{

#if 0

iSendLen = send(iSocketClient, ucSendBuf, strlen(ucSendBuf), 0);

#else

iAddrLen = sizeof(struct sockaddr);

iSendLen = sendto(iSocketClient, ucSendBuf, strlen(ucSendBuf), 0,

(const struct sockaddr \*)&tSocketServerAddr, iAddrLen);

#endif

if (iSendLen <= 0)

{

close(iSocketClient);

return -1;

}

}

}

return 0;

}

————————————————

原文链接：https://blog.csdn.net/weixin\_45775710/article/details/115770691