[第六章.网络编程 2](#_Toc1774)

[1. 网络编程简介 2](#_Toc4674)

[1.1 五层因特网协议栈 2](#_Toc13581)

[1. 2 传输层和应用层的常见协议 2](#_Toc8393)

[2. 网络编程之TCP/UDP比较 4](#_Toc14828)

[2.1 TCP和UDP 原理上的区别 4](#_Toc3956)

[2.2 为何存在UDP协议 4](#_Toc18997)

[2.3 TCP/UDP网络通信大概交互图 4](#_Toc19794)

[3. 网络编程主要函数介绍 7](#_Toc14731)

[3.1 socket函数 7](#_Toc9505)

[3.2 bind函数 7](#_Toc24267)

[3.3 listen函数 8](#_Toc21757)

[3.4 accept函数 8](#_Toc32652)

[3.5 connect函数 8](#_Toc13403)

[3.6 send函数 9](#_Toc9383)

[3.7 recv函数 9](#_Toc25355)

[3.8 recvfrom函数 9](#_Toc1976)

[3.9 sendto函数 9](#_Toc4373)

[4. TCP编程简单示例 10](#_Toc2561)

[4.1 服务器端代码 10](#_Toc11945)

[4.2 客户端代码 13](#_Toc26391)

[4.3 Makefile文件 15](#_Toc3206)

[4.4 执行 15](#_Toc10815)

[5. UDP编程简单示例 16](#_Toc30837)

[5.1 服务器端代码 16](#_Toc25443)

[5.2客户端代码 18](#_Toc32467)

[5.2.1 客户端程序1 18](#_Toc16651)

[5.2.2 客户端程序2 20](#_Toc319)

[5.3 Makefile文件 22](#_Toc7868)

[5.4 执行 22](#_Toc5096)

**第六章.网络编程**

1. **网络编程简介**

要编写通过计算机网络通信的程序，首先要确定这些程序同通信的协议（protocol），在设计一个协议的细节之前，首先要分清程序是由哪个程序发起以及响应何时产生。

举例来说，一般认为WEB服务器程序是一个长时间运行的程序（守护进程deamon），它只在响应来自网络的请求时才发送网络消息。协议的另一端是web客户程序，如某种浏览器，与服务器进程的通信总是由客户进程发起。大多数网络应用就是按照划分为客户（clinet）和服务器（server）来组织的。

**1.1 五层因特网协议栈**

为了给网络协议的设计提供一个结构，网络设计者以分层（layer）的方式组织协议以及实现这些协议的网络硬件和软件。

分层提供了一种结构化方式来讨论系统组件。模块化使更新系统组件更为容易。

协议栈是各层所有协议的总和。



五层因特网协议栈

应用层：应用层是网络应用程序及它们的应用层协议存留的地方。

运输层：因特网的运输层在应用程序端点之间传从应用层报文。

网络层：因特网呃网络层负责将称为数据包（datagram）的网络层分组从一台主机移动到另一台主机。

链路层：因特网的网络层通过源和目的地之间的一系列路由器路由数据报。

物理层：虽然链路层的任务是将整个帧从一个网络元素移动到临近的网络元素，而物理层的任务是将该帧的一个一个比特从一个节点移动到下一个节点。

1. **2 传输层和应用层的常见协议**

我们重点介绍和应用层编程关系密切的应用层和运输层。

应用层：

因特网的应用层包含很多协议，例如HTTP，SMTP，和 FTP。我们看到的某些网络功能，比如将www.baidu.com这样对人友好的端系统名字转换为32比特网络地址，也是借助于特定的应用层协议即域名系统（DNS）完成的。

应用层的协议分布在多个端系统上，一个端系统中的应用程序使用协议与另一个端系统中的应用程序交换信息分组。

运输层：

在英特网中有两个运输协议，即TCP和UDP，利用其中的任何一个都能运输应用层报文。我们写应用程序的时候具体选择哪个运输层协议应该根据实际情况来确定（后面会具体讲解）。

1. **网络编程之TCP/UDP比较**

**2.1 TCP和UDP 原理上的区别**

TCP向它的应用程序提供了面向连接的服务。这种服务包括了应用层报文向目的地的确保传递和流量控制（即发送方/接收方速率匹配）。这种服务包括了应用层报文划分为短报文，并提供拥塞控制机制，因此当网络拥塞时源抑制其传输速率。

UDP协议向它的应用程序提供无连接服务。这是一种不提供不必要服务的服务，没有可靠性，没有流量控制，也没有拥塞控制。

**2.2 为何存在UDP协议**

既然TCP提供了可靠数据传输服务，而UDP不能提供，那么TCP是否总是首选呢？答案是否定的，因为有许多应用更适合用UDP，原因有以下几点：

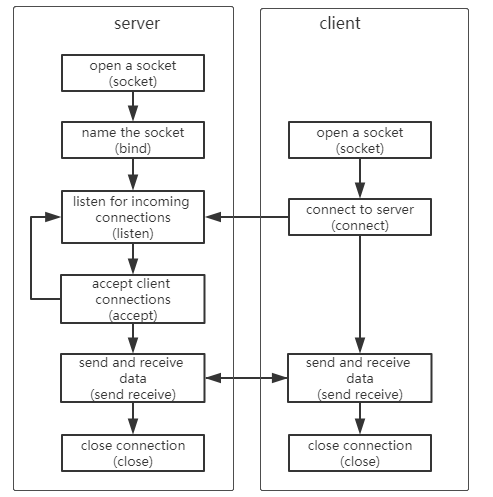
1. 关于何时发送什么数据控制的更为精细。

采用UDP时只要应用进程将数据传递给UDP，UDP就会立即将其传递给网络层。而TCP有重传机制，而不管可靠交付需要多长时间。但是实时应用通常不希望过分的延迟报文段的传送，且能容忍一部分数据丢失。

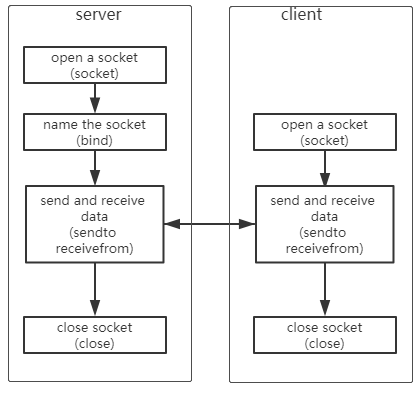
1. 无需建立连接，不会引入建立连接时的延迟。
2. 无连接状态，能支持更多的活跃客户。
3. 分组首部开销较小。

**2.3 TCP/UDP网络通信大概交互图**

下面我们分别画出运用TCP协议和运用UDP协议的客户端和服务器大概交互图。



面向连接的TCP流模式



UDP用户数据包模式

**3. 网络编程主要函数介绍**

**3.1 socket函数**

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

此函数用于创建一个套接字。

**domain**是网络程序所在的主机采用的通讯协族(AF\_UNIX和AF\_INET等)。

AF\_UNIX只能够用于单一的Unix 系统进程间通信，而AF\_INET是针对Internet的，因而可以允许远程通信使用。

**type**是网络程序所采用的通讯协议(SOCK\_STREAM,SOCK\_DGRAM等)。

SOCK\_STREAM表明用的是TCP 协议，这样会提供按顺序的，可靠，双向，面向连接的比特流。

SOCK\_DGRAM 表明用的是UDP协议，这样只会提不可靠，无连接的通信。

关于**protocol**，由于指定了type，所以这个地方一般只要用0来代替就可以了。

此函数执行成功时返回文件描述符，失败时返回-1,看errno可知道出错的详细情况。

**3.2 bind函数**

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

从函数用于将地址绑定到一个套接字。

**sockfd**是由socket函数调用返回的文件描述符。

**my\_addr**是一个指向sockaddr的指针。

**addrlen**是sockaddr结构的长度。

sockaddr的定义：

struct sockaddr{

unisgned short  as\_family;

char sa\_data[14];

};

不过由于系统的兼容性,我们一般使用另外一个结构(struct sockaddr\_in) 来代替。

sockaddr\_in的定义：

struct sockaddr\_in{

unsigned short          sin\_family;

unsigned short int      sin\_port;

struct in\_addr          sin\_addr;

unsigned char           sin\_zero[8];

}

如果使用Internet所以sin\_family一般为AF\_INET。

sin\_addr设置为INADDR\_ANY表示可以和任何的主机通信。

sin\_port是要监听的端口号。

bind将本地的端口同socket返回的文件描述符捆绑在一起.成功是返回0,失败的情况和socket一样。

**3.3 listen函数**

**int listen(int sockfd,int backlog);**

此函数宣告服务器可以接受连接请求。

**sockfd**是bind后的文件描述符。

**backlog**设置请求排队的最大长度。当有多个客户端程序和服务端相连时，使用这个表示可以介绍的排队长度。

listen函数将bind的文件描述符变为监听套接字，返回的情况和bind一样。

**3.4 accept函数**

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

服务器使用此函数获得连接请求，并且建立连接。

**sockfd**是listen后的文件描述符。

**addr，addrlen**是用来给客户端的程序填写的,服务器端只要传递指针就可以了， bind,listen和accept是服务器端用的函数。

accept调用时，服务器端的程序会一直阻塞到有一个客户程序发出了连接。 accept成功时返回最后的服务器端的文件描述符，这个时候服务器端可以向该描述符写信息了，失败时返回-1 。

**3.5 connect函数**

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

可以用connect建立一个连接，在connect中所指定的地址是想与之通信的服务器的地址。

**sockfd**是socket函数返回的文件描述符。

**serv\_addr**储存了服务器端的连接信息，其中sin\_add是服务端的地址。

**addrlen**是serv\_addr的长度

connect函数是客户端用来同服务端连接的.成功时返回0，sockfd是同服务端通讯的文件描述符，失败时返回-1。

**3.6 send函数**

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

**sockfd** 指定发送端套接字描述符；

**buf** 指明一个存放应用程序要发送数据的缓冲区；

**len** 指明实际要发送的数据的字节数；

**flags** 一般置0。

客户或者服务器应用程序都用send函数来向TCP连接的另一端发送数据

**3.7 recv函数**

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

**sockfd** 指定接收端套接字描述符；

**buf** 指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；

**len** 指明buf的长度；

**flags** 一般置0。

客户或者服务器应用程序都用recv函数从TCP连接的另一端接收数据。

**3.8 recvfrom函数**

**ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,**

**struct sockaddr \*src\_addr, socklen\_t \*addrlen);**

recvfrom通常用于无连接套接字，因为此函数可以获得发送者的地址。

**src\_addr** 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。

**addrlen** 常置为sizeof （struct sockaddr）。

**3.9 sendto函数**

**ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,**

**const struct sockaddr \*dest\_addr, socklen\_t addrlen);**

sendto和send相似，区别在于sendto允许在无连接的套接字上指定一个目标地址。

**dest\_addr** 表示目地机的IP地址和端口号信息，

**addrlen** 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。

1. **TCP编程简单示例**

服务器首先进行初始化操作：调用函数socket创建一个套接字，函数bind将这个套接字与服务器的公认地址绑定在一起，函数listen将这个套接字换成倾听套接字，然后调用函数accept来等待客户机的请求。过了一段时间后，客户机启动，调用socket创建一个套接字，然后调用函数connect来与服务器建立连接。连接建立之后，客户机和服务器通过读、写套接字来进行通信。

**4.1 服务器端代码**

参考：TCP/server\_line.c

1#include <stdio.h>

2#include <stdlib.h>

3#include <string.h>

4#include <sys/types.h>

5#include <sys/socket.h>

6#include <netinet/in.h>

7#include <arpa/inet.h>

8#include <unistd.h>

9#include <signal.h>

10

11#define SERVER\_PORT 8180

12#define C\_QUEUE 10

13

14/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15\*函数功能描述：从8180端口接收客户端数据

16\*输入参数：无

17\*输出参数：打印客户IP以及发来的信息

18\*返回值：无

19\*修改日期 版本号 修改人 修改内容

20\*2020/05/13 v1.0.0 zonghzha reat

21\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

22

23int main(int argc, char \*\*argv)

24{

25 char buf[512];

26 int len;

27 int duty\_socket;

28 int customer\_socket;

29 struct sockaddr\_in socket\_server\_addr;

30 struct sockaddr\_in socket\_client\_addr;

31 int ret;

32 int addr\_len;

33

34 signal(SIGCHLD, SIG\_IGN);

35

36 /\* 服务器端开始建立socket描述符 \*/

37 duty\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

38 if (duty\_socket == -1)

39 {

40 printf("socket error");

41 return -1;

42 }

43

44 /\* 服务器端填充 sockaddr\_in结构 \*/

45 socket\_server\_addr.sin\_family = AF\_INET;

46 /\*端口号转换为网络字节序\*/

47 socket\_server\_addr.sin\_port = htons(SERVER\_PORT);

48 /\*接收本机所有网口的数据\*/

49 socket\_server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

50 memset(socket\_server\_addr.sin\_zero, 0, 8);

51

52 /\* 捆绑sockfd描述符 \*/

53 ret = bind(duty\_socket, (const struct sockaddr \*)&socket\_server\_addr, sizeof(struct sockaddr));

54 if (ret == -1)

55 {

56 printf("bind error!\n");

57 return -1;

58 }

59 ret = listen(duty\_socket, C\_QUEUE);

60 if (ret == -1)

61 {

62 printf("listen error!\n");

63 return -1;

64 }

65

66 while (1)

67 {

68 addr\_len = sizeof(struct sockaddr);

69 /\* 服务器阻塞,直到客户程序建立连接 \*/

70 customer\_socket = accept(duty\_socket, (struct sockaddr \*)&socket\_client\_addr, &addr\_len);

71 if (customer\_socket != -1)

72 {

73 /\*inet\_ntoa的作用是将一个32位Ipv4地址转换为相应的点分十进制数串\*/

74 printf("Get connect from %s\n", inet\_ntoa(socket\_client\_addr.sin\_addr));

75 }

76 if (!fork())

77 {

78 while (1)

79 {

80 memset(buf, 512, 0);

81 /\*接收数据\*/

82 len = recv(customer\_socket, buf, sizeof(buf), 0);

83 buf[len] = '\0';

84 if (len <= 0)

85 {

86 close(customer\_socket);

87 return -1;

88 }

89 else

90 {

91 printf("Get connect from %s, Msg is %s\n", inet\_ntoa(socket\_client\_addr.sin\_addr), buf);

92 }

93 }

94 }

95 }

96

97 close(duty\_socket);

98 return 0;

99}

**4.2 客户端代码**

参考TCP/client\_line.c

1#include <stdio.h>

2#include <stdlib.h>

3#include <string.h>

4#include <sys/types.h>

5#include <sys/socket.h>

6#include <netinet/in.h>

7#include <arpa/inet.h>

8#include <unistd.h>

9

10#define SERVER\_PORT 8180

11/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12\*函数功能描述：向指定IP的8180端口发送数据

13\*输入参数：点分十进制服务器IP

14\*输出参数：无

15\*返回值：无

16\*修改日期 版本号 修改人 修改内容

17\*2020/05/13 v1.0.0 zonghzha creat

18\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

19

20int main(int argc, char \*\*argv)

21{

22 unsigned char buf[512];

23 int len;

24 struct sockaddr\_in socket\_server\_addr;

25 int ret;

26 int addr\_len;

27 int client\_socket;

28

29

30 if (argc != 2)

31 {

32 printf("Usage:\n");

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

34 return -1;

35 }

36

37 /\* 客户程序开始建立 sockfd描述符 \*/

38 client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

39 if (client\_socket == -1)

40 {

41 printf("socket error");

42 return -1;

43 }

44

45 /\* 客户程序填充服务端的资料 \*/

46 socket\_server\_addr.sin\_family = AF\_INET;

47 /\*主机字节序转换为网络字节序\*/

48 socket\_server\_addr.sin\_port = htons(SERVER\_PORT);

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

50 {

51 printf("invalid server ip\n");

52 return -1;

53 }

54 memset(socket\_server\_addr.sin\_zero, 0, 8);

55 /\* 客户程序发起连接请求 \*/

56 ret = connect(client\_socket, (const struct sockaddr \*)&socket\_server\_addr, sizeof(struct sockaddr));

57 if (ret == -1)

58 {

59 printf("connect error!\n");

60 return -1;

61 }

62

63

64 while (1)

65 {

66 if (fgets(buf, sizeof(buf), stdin))

67 {

68 len = send(client\_socket, buf, strlen(buf), 0);

69 if (len <= 0)

70 {

71 close(client\_socket);

72 return -1;

73 }

74 }

75 }

76

77 close(client\_socket);

78 return 0;

79}

**4.3 Makefile文件**

all:server client

server:server.c

gcc $^ -o $@

client:client.c

gcc $^ -o $@

clean:

rm server client -f

（注意：命令语句的开头要用“Tab”键。）

**4.4 执行**

服务器端：

./server

客户端：

./client 127.0.0.1

客户端输入：

good night

服务器端显示：

Get connect from 127.0.0.1

Get connect from 127.0.0.1, Msg is good night

1. **UDP编程简单示例**

UDP服务器首先进行初始化操作：调用函数socket创建一个数据报类型的套接字，函数bind将这个套接字与服务器的公认地址绑定在一起。然后调用函数recvfrom接收UDP客户机的数据报。UDP客户机首先调用函数socket创建一个数据报套接字，然后调用函数sendto向服务器发送数据报。在结束通信后，客户机调用close关闭UDP套接字，服务器继续使用这个UDP套接字接收其它客户机的数据报。

**5.1 服务器端代码**

参考UDP/server\_line.c

1#include <stdio.h>

2#include <stdlib.h>

3#include <string.h>

4//#include <sys/type.h>

5#include <sys/socket.h>

6#include <netinet/in.h>

7#include <arpa/inet.h>

8#include <unistd.h>

9#include <signal.h>

10

11/\*服务器端口为8180\*/

12#define SERVER\_PORT 8180

13

14/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15\*函数功能描述：从8180端口接收客户端数据

16\*输入参数：无

17\*输出参数：打印客户IP以及发来的信息

18\*返回值：无

19\*修改日期 版本号 修改人 修改内容

20\*2020/05/13 v1.0.0 zonghzha creat

21\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

22

23

24int main(int argc, char \*\*argv)

25{

26 unsigned char buf[512];

27 int len;

28 int duty\_socket;

29 int customer\_socket;

30 struct sockaddr\_in socket\_server\_addr;

31 struct sockaddr\_in socket\_client\_addr;

32 int ret;

33 int addr\_len;

34

35 /\*创建数据报套接字\*/

36 duty\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

37 if (duty\_socket == -1)

38 {

39 printf("socket error");

40 return -1;

41 }

42

43 /\* 服务器端填充 sockaddr\_in结构 \*/

44 socket\_server\_addr.sin\_family = AF\_INET;

45 socket\_server\_addr.sin\_port = htons(SERVER\_PORT);

46 socket\_server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

47 memset(socket\_server\_addr.sin\_zero, 0, 8);

48

49 /\*绑定套接字\*/

50 ret = bind(duty\_socket, (const struct sockaddr \*)&socket\_server\_addr, sizeof(struct sockaddr));

51 if (ret == -1)

52 {

53 printf("bind error!\n");

54 return -1;

55 }

56

57

58 while (1)

59 {

60 addr\_len = sizeof(struct sockaddr);

61 /\* 接收客户端数据报，返回的为接收到的字节数 \*/

62 len = recvfrom(duty\_socket, buf, sizeof(buf), 0, (struct sockaddr \*)&socket\_client\_addr, &addr\_len);

63 if (len > 0)

64 {

65 buf[len] = '\0';

66 printf("Get Msg from %s : %s\n", inet\_ntoa(socket\_client\_addr.sin\_addr), buf);

67 }

68

69 }

70

71 close(duty\_socket);

72 return 0;

73}

74

**5.2 客户端代码**

**5.2.1 客户端程序1**

参考UDP/client\_line\_1.c

1#include <stdio.h>

2#include <stdlib.h>

3#include <string.h>

4#include <sys/socket.h>

5#include <netinet/in.h>

6#include <arpa/inet.h>

7#include <unistd.h>

8

9#define SERVER\_PORT 8180

10

11/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12\*函数功能描述：向指定IP的8180端口发送数据

13\*输入参数：点分十进制服务器IP

14\*输出参数：无

15\*返回值：无

16\*修改日期 版本号 修改人 修改内容

17\*2020/05/13 v1.0.0 zonghzha creat

18\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

19

20int main(int argc, char \*\*argv)

21{

22 unsigned char buf[512];

23 int len;

24 struct sockaddr\_in socket\_server\_addr;

25 int ret;

26 int addr\_len;

27 int client\_socket;

28

29

30 if (argc != 2)

31 {

32 printf("Usage:\n");

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

34 return -1;

35 }

36

37 /\*创建套接字\*/

38 client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

39 if (client\_socket == -1)

40 {

41 printf("socket error");

42 return -1;

43 }

44

45 /\* 填充服务端的资料 \*/

46 socket\_server\_addr.sin\_family = AF\_INET;

47 socket\_server\_addr.sin\_port = htons(SERVER\_PORT);

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

49 {

50 printf("invalid server ip\n");

51 return -1;

52 }

53 memset(socket\_server\_addr.sin\_zero, 0, 8);

54

55

56

57

58 while (1)

59 {

60 if (fgets(buf, sizeof(buf), stdin))

61 {

62 // len = send(client\_socket, buf, strlen(buf), 0);

63 /\*向服务器端发送数据报\*/

64 addr\_len = sizeof(struct sockaddr);

65 len = sendto(client\_socket, buf, sizeof(buf), 0, (struct sockaddr \*)&socket\_server\_addr, addr\_len);

66 if (len <= 0)

67 {

68 close(client\_socket);

69 return -1;

70 }

71 }

72 }

73

74 close(client\_socket);

75 return 0;

76}

77

问：用UDP协议写网络通讯程序不可以用connect函数吗？

答：非也。

**5.2.2 客户端程序2**

参考UDP/client\_line\_2.c

1#include <stdio.h>

2#include <stdlib.h>

3#include <string.h>

4#include <sys/socket.h>

5#include <netinet/in.h>

6#include <arpa/inet.h>

7#include <unistd.h>

8

9/\*服务器端口为8180\*/

10#define SERVER\_PORT 8180

11

12/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13\*函数功能描述：向指定IP的8180端口发送数据

14\*输入参数：点分十进制服务器IP

15\*输出参数：无

16\*返回值：无

17\*修改日期 版本号 修改人 修改内容

18\*2020/05/13 v1.0.0 zonghzha creat

19\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

20

21int main(int argc, char \*\*argv)

22{

23 unsigned char buf[512];

24 int len;

25 struct sockaddr\_in socket\_server\_addr;

26 int ret;

27 int addr\_len;

28 int client\_socket;

29

30

31 if (argc != 2)

32 {

33 printf("Usage:\n");

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

35 return -1;

36 }

37

38 /\*创建数据报套接字\*/

39 client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

40 if (client\_socket == -1)

41 {

42 printf("socket error");

43 return -1;

44 }

45

46 socket\_server\_addr.sin\_family = AF\_INET;

47 socket\_server\_addr.sin\_port = htons(SERVER\_PORT);

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

49 {

50 printf("invalid server ip\n");

51 return -1;

52 }

53 memset(socket\_server\_addr.sin\_zero, 0, 8);

54

55 ret = connect(client\_socket, (const struct sockaddr \*)&socket\_server\_addr, sizeof(struct sockaddr));

56 if (ret == -1)

57 {

58 printf("connect error!\n");

59 return -1;

60 }

61

62

63 while (1)

64 {

65 if (fgets(buf, sizeof(buf), stdin))

66 {

67 len = send(client\_socket, buf, strlen(buf), 0);

68 if (len <= 0)

69 {

70 close(client\_socket);

71 return -1;

72 }

73 }

74 }

75

76 close(client\_socket);

77 return 0;

78}

79

在客户端代码2中，connect函数并非真的在协议层建立了连接，它只是指定了服务器的地址和端口号信息。

因为在connect中指定了服务器的地址和端口号信息，所以后面的send就可以直接发送了，而不用再次指定地址和端口号。

**5.3 Makefile文件**

all:server client\_1 client\_2

server:server.c

gcc $^ -o $@

client\_1:client\_1.c

gcc $^ -o $@

client\_2:client\_2.c

gcc $^ -o $@

clean:

rm server client\_1 client\_2 -f

（注意：命令语句的开头要用“Tab”键。）

**5.4 执行**

服务器端执行：

./server

客户端执行：

./client\_1 127.0.0.1

客户端输入：

good night

服务器端显示：

Get Msg from 127.0.0.1 : good night