

# 网络高级编程

## 实验报告

实验名称: TCP 建立连接及过程分析

TCP 客户-服务器并发程序开发

UDP 客户-服务器程序开发

报告日期: 2019.5.28

学生姓名: 陈诚

学生学号: 09016429

# TCP 建立连接及过程分析

## 一、实验目的

掌握 TCP 连接的建立和终止。使用 ethereal/TCPDump 等抓包工具，截取 TCP 建立过程中产生的数据包，分析连接建立过程。

## 二、实验内容

### 1. 调试过程

服务器端 IP 地址为 223.3.89.102，使用 Ubuntu 操作系统。使用 13 号端口时客户端无法与其连接。13 号端口为 DAYTIME 协议端口，可能会造成冲突，因此实验中改用 8888 端口。客户端 IP 地址为 10.203.81.174，使用 Mac OS Mojave 操作系统，兼容绝大部分的 UNIX 操作命令。

### 2. 程序运行结果截屏

Server 端:

```
[cheng16@beilun16-WD12:~$ vi datetimes.c
[cheng16@beilun16-WD12:~$ gcc datetimes.c -o datetimes
datetimes.c: In function 'main':
datetimes.c:30:3: warning: implicit declaration of function 'write'; did you mean 'fwrite'? [-Wimplicit-function-declaration]
    write( connfd , buff , strlen( buff ) );
    ^~~~~
    fwrite
datetimes.c:31:3: warning: implicit declaration of function 'close'; did you mean 'pclose'? [-Wimplicit-function-declaration]
    close( connfd );
    ^~~~~
    pclose
[cheng16@beilun16-WD12:~$ ./datetimes
```

Client 端:

```
((base) chenchengdeMacBook-Pro:第一章 charles$ ls
ACNP-HW01.ppt  datetime.h  datetimed.c  datetimes  datetimes.dSYM
ACNP01.ppt    datetimed  datetimed.dSYM  datetimes.c
((base) chenchengdeMacBook-Pro:第一章 charles$ ./datetimed 223.3.89.102
Tue May 28 16:23:51 2019
((base) chenchengdeMacBook-Pro:第一章 charles$
```

### 3. 数据抓包

Ethereal 目前已改名为 Wireshark，是一个网络封包分析软件用于抓取网络封包，并尽可能显示出最为详细的网络封包资料。

为过滤抓取信息，实验中使用过滤条件 `ip.addr==223.3.89.102 and ip.addr==10.203.81.174`。前者为服务器端 IP 地址，后者为客户端 IP 地址。

抓取结果如下：

Capturing from Wi-Fi: en0

ip.addr==223.3.89.102 and ip.addr==10.203.81.174

No.	Time	Source	Destination	Protocol	Length	Info
213	24.817107	10.203.81.174	223.3.89.102	TCP	78	65439 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1102656906 TSecr=0 ...
216	24.836808	223.3.89.102	10.203.81.174	TCP	74	8888 → 65439 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=42...
217	24.836917	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1102656925 TSecr=4284383950
218	24.855085	223.3.89.102	10.203.81.174	TCP	92	8888 → 65439 [PSH, ACK] Seq=1 Ack=1 Win=29056 Len=26 TSval=4284383971 TSecr=11026...
219	24.855092	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439 [FIN, ACK] Seq=27 Ack=1 Win=29056 Len=0 TSval=4284383971 TSecr=11026...
220	24.855208	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=27 Win=131712 Len=0 TSval=1102656943 TSecr=4284383971
221	24.855209	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=28 Win=131712 Len=0 TSval=1102656943 TSecr=4284383971
222	24.855771	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [FIN, ACK] Seq=1 Ack=28 Win=131712 Len=0 TSval=1102656943 TSecr=4284...
223	24.876651	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439 [ACK] Seq=28 Ack=2 Win=29056 Len=0 TSval=4284383992 TSecr=1102656943

Frame 213: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0

Ethernet II, Src: Apple\_97:11:94 (38:f9:d3:97:11:94), Dst: IETF-VRRP-VRID\_65 (00:00:5e:00:01:65)

Destination: IETF-VRRP-VRID\_65 (00:00:5e:00:01:65)

Address: IETF-VRRP-VRID\_65 (00:00:5e:00:01:65)

...0... = LG bit: Globally unique address (factory default)

...0... = IG bit: Individual address (unicast)

Source: Apple\_97:11:94 (38:f9:d3:97:11:94)

Address: Apple\_97:11:94 (38:f9:d3:97:11:94)

0000 00 00 5e 00 01 65 38 f9 d3 97 11 94 08 00 45 00 ...e8...E:

0010 00 40 00 00 40 00 40 06 a5 d5 0a cb 51 ae df 03 ...@...@...Q...

0020 59 66 ff 9f 22 b8 8f 01 4d d8 00 00 00 00 b0 02 Yf...M...

0030 ff ff 2b a4 00 00 02 04 05 b4 01 03 03 06 01 01 ...+...+...

0040 08 0a 41 b9 35 8a 00 00 00 00 04 02 00 00 ...A.5...

Wi-Fi: en0: <live capture in progress> Packets: 324 · Displayed: 9 (2.8%) Profile: Default

从图中可以看出客户端由系统自动分配 65439 端口。数据传输过程是[PSH, ACK], 强制传输缓冲区的内容。

#### 4.抓包分析

TCP/IP 通过三次握手建立一个连接。这一过程中的三种报文是：SYN, SYN/ACK, ACK。

第一次握手：客户端发送一个 TCP，标志位为 SYN，序列号为 0，代表客户端请求建立连接。如下图：

Capturing from

ip.addr==223.3.89.102 and ip.addr==10.203.81.174

No.	Time	Source	Destination	Protocol	Length	Info
213	24.817107	10.203.81.174	223.3.89.102	TCP	78	65439 → 8888
216	24.836808	223.3.89.102	10.203.81.174	TCP	74	8888 → 65439
217	24.836917	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
218	24.855085	223.3.89.102	10.203.81.174	TCP	92	8888 → 65439
219	24.855092	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439
220	24.855208	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
221	24.855209	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
222	24.855771	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
223	24.876651	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439

Transmission Control Protocol, Src Port: 65439, Dst Port: 8888, Seq: 0, Len: 0

Source Port: 65439

Destination Port: 8888

[Stream index: 13]

[TCP Segment Len: 0]

Sequence number: 0 (relative sequence number)

[Next sequence number: 0 (relative sequence number)]

Acknowledgment number: 0

1011 .... = Header Length: 44 bytes (11)

Flags: 0x002 (SYN)

Window size value: 65535

[Calculated window size: 65535]

Checksum: 0x2ba4 [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, N



第二次握手：服务器发回确认包，标志位为 SYN,ACK. 将确认序号 (Acknowledgement Number) 设置为客户的 ISN 加 1. 即  $0+1=1$ ，如下图。

No.	Time	Source	Destination	Protocol	Length	Info
213	24.817107	10.203.81.174	223.3.89.102	TCP	78	65439 → 8888
216	24.836808	223.3.89.102	10.203.81.174	TCP	74	8888 → 65439
217	24.836917	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
218	24.855085	223.3.89.102	10.203.81.174	TCP	92	8888 → 65439
219	24.855092	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439
220	24.855208	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
221	24.855209	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
222	24.855771	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
223	24.876651	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439

Transmission Control Protocol, Src Port: 8888, Dst Port: 65439, Seq: 0, Ack: 1, Len: 0	
Source Port:	8888
Destination Port:	65439
[Stream index:	13]
[TCP Segment Len:	0]
Sequence number:	0 (relative sequence number)
[Next sequence number:	0 (relative sequence number)]
Acknowledgment number:	1 (relative ack number)
1010 ....	= Header Length: 40 bytes (10)
Flags:	0x012 (SYN, ACK)
Window size value:	28960
[Calculated window size:	28960]
Checksum:	0xf68f [unverified]
[Checksum Status:	Unverified]
Urgent pointer:	0
Options:	(20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (N

Wi-Fi: en0: <live capture in progress>	
0000	38 f9 d3 97 11 94 00 00 5e 00 01 65 08 00 45 00 8.....^.....E.
0010	00 3c 00 00 40 00 3f 06 a6 d9 df 03 59 66 0a cb ..<..@.?.....Yf..
0020	51 ae 22 b8 ff 9f 4e 5a 04 5f 8f 01 4d d9 a0 12 Q."...NZ ..M...

第三次握手：客户端再次发送确认包(ACK) SYN 标志位为 0，ACK 标志位为 1。并且把服务器发来 ACK 的序号字段+1，放在确定字段中发送给对方。并且在数据段放写 ISN 的+1，如下图。

No.	Time	Source	Destination	Protocol	Length	Info
213	24.817107	10.203.81.174	223.3.89.102	TCP	78	65439 → 8888
216	24.836808	223.3.89.102	10.203.81.174	TCP	74	8888 → 65439
217	24.836917	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
218	24.855085	223.3.89.102	10.203.81.174	TCP	92	8888 → 65439
219	24.855092	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439
220	24.855208	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
221	24.855209	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
222	24.855771	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888
223	24.876651	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439

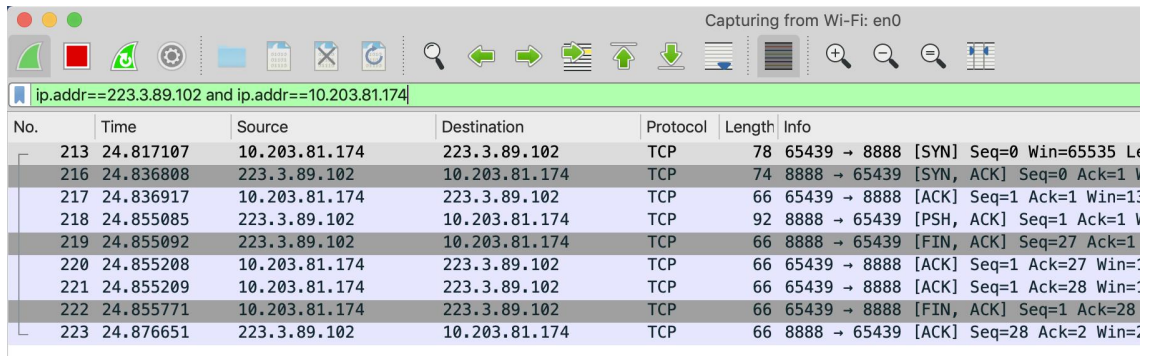
Transmission Control Protocol, Src Port: 65439, Dst Port: 8888, Seq: 1, Ack: 1, Len: 0	
Source Port:	65439
Destination Port:	8888
[Stream index:	13]
[TCP Segment Len:	0]
Sequence number:	1 (relative sequence number)
[Next sequence number:	1 (relative sequence number)]
Acknowledgment number:	1 (relative ack number)
1000 ....	= Header Length: 32 bytes (8)
Flags:	0x010 (ACK)
Window size value:	2058
[Calculated window size:	131712]
[Window size scaling factor:	64]
Checksum:	0x8e5f [unverified]
[Checksum Status:	Unverified]
Urgent pointer:	0

Wi-Fi: en0: <live capture in progress>	
0000	00 00 5e 00 01 65 38 f9 d3 97 11 94 08 00 45 00 ..^.....e8: .....E.
0010	00 34 00 00 40 00 00 06 a5 e1 0a cb 51 ae df 03 .4..@.@.....Q...
0020	59 66 ff 9f 22 b8 8f 01 4d d9 4e 5a 04 60 80 10 Yf.....M.NZ...



TCP/IP 通过 4 次握手来断开连接，这一过程中的 3 种报文为：FIN,FIN/ACK,ACK。如下图所示。



No.	Time	Source	Destination	Protocol	Length	Info
213	24.817107	10.203.81.174	223.3.89.102	TCP	78	65439 → 8888 [SYN] Seq=0 Win=65535 Len=0
216	24.836808	223.3.89.102	10.203.81.174	TCP	74	8888 → 65439 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
217	24.836917	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=1 Win=1 Len=0
218	24.855085	223.3.89.102	10.203.81.174	TCP	92	8888 → 65439 [PSH, ACK] Seq=1 Ack=1 Win=0 Len=0
219	24.855092	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439 [FIN, ACK] Seq=27 Ack=1 Win=0 Len=0
220	24.855208	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=27 Win=0 Len=0
221	24.855209	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [ACK] Seq=1 Ack=28 Win=0 Len=0
222	24.855771	10.203.81.174	223.3.89.102	TCP	66	65439 → 8888 [FIN, ACK] Seq=1 Ack=28 Win=0 Len=0
223	24.876651	223.3.89.102	10.203.81.174	TCP	66	8888 → 65439 [ACK] Seq=28 Ack=2 Win=0 Len=0

首先服务器 8888 端口向 65439 端口发出 FIN 的断开连接请求, 然后客户端 65439 端口收到请求之后向服务器 8888 端口回复了一个 ACK, 接着客户端 65439 端口向服务器 8888 端口发送断开请求[FIN, ACK], 最后服务器 8888 端口向客户端发送断开的回复 ACK。这样四次握手之后, 服务器和客户端都确认了断开连接, 可以看到断开连接是双向的。

### 三、源程序代码

见报告最后附录 1:实验一源代码

## TCP 客户-服务器并发程序开发

### 一、实验目的

掌握 TCP 并发程序设计和 C/S 结构程序的开发。

### 二、实验内容

#### 1.Fork()函数原理

进程包括代码、数据和分配给进程的资源。Fork()函数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。

#### 2.调试过程

服务器端 IP 地址为 10.203.81.174, 使用 Mac OS Mojave 操作系统, 兼容绝大部分的 UNIX 操作命令。实验中服务器端使用 13 号端口。客户端 IP 地址也为 10.203.81.174。

#### 3.程序运行结果

## 服务器启动

```
实验二代码&可执行文件 — datetimes - datetimes — 80x24
Last login: Tue May 28 21:25:05 on ttys003
[(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ls
client-1      datetime.h      datetimes
client-2      datetime.c      datetimes.c
[(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ./datetimes
```

## Client-1 访问

```
实验二代码&可执行文件 — client-1 127.0.0.1 — 80x24
(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ./client-1 127.0.0.1
服务端子进程 id13855
时间: Tue May 28 21:36:54 2019
```

## Client-2 访问

```
实验二代码&可执行文件 — client-2 127.0.0.1 — 80x24
(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ./client-2 127.0.0.1
服务端子进程 id13857
时间: Tue May 28 21:37:01 2019
```

## 此时的 Server 端

```
实验二代码&可执行文件 — datetimes - datetimes — 80x24
[(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ls
client-1      datetime.h      datetimes
client-2      datetime.c      datetimes.c
[(base) chenchengdeMacBook-Pro:实验二代码&可执行文件 charles$ ./datetimes
Initializeing new connection...
服务端子进程 id13855
时间: Tue May 28 21:36:54 2019
Initializeing new connection...
服务端子进程 id13857
时间: Tue May 28 21:37:01 2019
```

## 三、源程序代码

见报告最后附录 2:实验二源代码

## UDP 客户-服务器程序开发

### 一、实验目的

掌握 UDP C/S 结构程序的开发。看懂 ping 指令的原理，编写一个 ping 程序。

## 二、实验内容

### 1.ping 指令原理

ping 命令是基于 ICMP 协议来工作的，ICMP 全称为 Internet 控制报文协议 (Internet Control Message Protocol)。ping 命令会发送一份 ICMP 回显请求报文给目标主机，并等待目标主机返回 ICMP 回显应答。因为 ICMP 协议会要求目标主机在收到消息之后，必须返回 ICMP 应答消息给源主机，如果源主机在一定时间内收到了目标主机的应答，则表明两台主机之间网络是可达的。

假定主机 A 的 IP 地址是 192.168.1.1，主机 B 的 IP 地址是 192.168.1.2，都在同一子网内。首先，Ping 命令会构建一个固定格式的 ICMP 请求数据包，然后由 ICMP 协议将这个数据包连同地址 192.168.1.2 一起交给 IP 层协议（和 ICMP 一样，实际上是一组后台运行的进程），IP 层协议将以地址 192.168.1.2 作为目的地址，本机 IP 地址作为源地址，加上一些其他的控制信息，构建一个 IP 数据包，并在一个映射表中查找出 IP 地址 192.168.1.2 所对应的物理地址（也叫 MAC 地址），一并交给数据链路层。后者构建一个数据帧，目的地址是 IP 层传过来的物理地址，源地址则是本机的物理地址，还要附加上一些控制信息，依据以太网的介质访问规则，将它们传出去。

主机 B 收到这个数据帧后，先检查它的目的地址，并和本机的物理地址对比，如符合，则接收；否则丢弃。接收后检查该数据帧，将 IP 数据包从帧中提取出来，交给本机的 IP 层协议。同样，IP 层检查后，将有用的信息提取后交给 ICMP 协议，后者处理后，马上构建一个 ICMP 应答包，发送给主机 A，其过程和主机 A 发送 ICMP 请求包到主机 B 一模一样。

### 2.调试过程

实验中，源主机 IP 地址为 10.203.81.174，使用 Mac OS Mojave 操作系统。目标主机为 www.baidu.com，实验时的 IP 为 220.181.38.150。实验共发送 3 个 ICMP 报文，每个 1 秒发送一个。

### 3.程序运行结果

```
(base) chenchengdeMacBook-Pro:第三章 charles$ sudo ./uping www.baidu.com
PING www.baidu.com(220.181.38.150): 56 bytes data in ICMP packets.
64 byte from 220.181.38.150: icmp_seq=1 ttl=49 rtt=3011.000 ms
64 byte from 220.181.38.150: icmp_seq=2 ttl=49 rtt=2006.000 ms
64 byte from 220.181.38.150: icmp_seq=3 ttl=49 rtt=1003.000 ms

-----PING statistics-----
3 packets transmitted, 3 received , %0 lost
(base) chenchengdeMacBook-Pro:第三章 charles$
```

从图中可以看出源主机发出的 3 个报文全部被正确接收，因此表明两台主机之间是可达的。

### 4.数据抓包

The image shows a Wireshark interface capturing traffic from a Wi-Fi adapter (en0). The filter bar at the top displays the expression `ip.addr==220.181.38.150 and ip.addr==10.203.81.174`. The main packet list contains six entries:

No.	Time	Source	Destination	Protocol	Length	Info
34	3.301837	10.203.81.174	220.181.38.150	ICMP	98	Echo (ping) request id=0x9322, seq=256/1, ttl=64 (reply in 35)
35	3.339833	220.181.38.150	10.203.81.174	ICMP	98	Echo (ping) reply id=0x9322, seq=256/1, ttl=49 (request in 34)
38	4.306511	10.203.81.174	220.181.38.150	ICMP	98	Echo (ping) request id=0x9322, seq=512/2, ttl=64 (reply in 39)
39	4.337220	220.181.38.150	10.203.81.174	ICMP	98	Echo (ping) reply id=0x9322, seq=512/2, ttl=49 (request in 38)
40	5.310200	10.203.81.174	220.181.38.150	ICMP	98	Echo (ping) request id=0x9322, seq=768/3, ttl=64 (reply in 41)
41	5.349179	220.181.38.150	10.203.81.174	ICMP	98	Echo (ping) reply id=0x9322, seq=768/3, ttl=49 (request in 40)

The details pane for the selected packet (No. 34) shows the following information:

- Ethernet II, Src: Apple\_97:11:94 (38:f9:d3:97:11:94), Dst: IETF-VRRP-VRID\_65 (00:00:5e:00:01:65)
- Internet Protocol Version 4, Src: 10.203.81.174, Dst: 220.181.38.150
- Internet Control Message Protocol**
  - Type: 8 (Echo (ping) request)
  - Code: 0
  - Checksum: 0xc866 [correct]
  - [Checksum Status: Good]
  - Identifier (BE): 37666 (0x9322)
  - Identifier (LE): 8851 (0x2293)
  - Sequence number (BE): 256 (0x0100)
  - Sequence number (LE): 1 (0x0001)
  - [\[Response frame: 35\]](#)
  - Timestamp from icmp data: May 28, 2019 16:53:04.000000000 CST
  - [Timestamp from icmp data (relative): 0.074486000 seconds]
  - Data (48 bytes)
    - Data: bd220100

The packet bytes pane shows the raw data of the selected packet, with hex values and their corresponding ASCII representations.

从图中可以看出共生成了 6 个报文（3 个 request 报文和 3 个 reply 报文）。从 Ping 的工作过程，我们可以知道，主机 A 收到了主机 B 的一个应答包，说明两台主机之间的去、回通路均正常。也就是说，无论从主机 A 到主机 B，还是从主机 B 到主机 A，都是正常的。

### 三、源程序代码

见报告最后附录 3:实验三源代码

## 附录 1:实验一源代码

datetime.h

```
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define MAXLINE 4096
```

# datetimes.c

```

/*****
/***** datetime Example Server *****/

```





```
/* **** */
#include "datetime.h"
#include <time.h>

int
main( int argc , char * * argv )
{
    int listenfd , connfd;
    struct sockaddr_in servaddr;
    char buff[ MAXLINE ];
    time_t ticks;

    listenfd = socket( AF_INET , SOCK_STREAM , 0 );

    memset( &servaddr , 0 , sizeof( servaddr ) );
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl( INADDR_ANY );
    servaddr.sin_port = htons( 8888 );

    bind( listenfd , (struct sockaddr *)&servaddr , sizeof( servaddr ) );
    listen( listenfd , 1024 );

    for( ; ; ){
        connfd = accept( listenfd , (struct sockaddr *)NULL , NULL );
        ticks = time( NULL );
        snprintf( buff , sizeof( buff ) , "%.24s\r\n" , ctime( &ticks ) );
        write( connfd , buff , strlen( buff ) );
        close( connfd );
    }
}
```

## datetimec.c

```
/* **** */
/* **** datetime Example Client **** */
/* **** */

#include "datetime.h"

int main( int argc , char * * argv )
{
    int sockfd , n ;
    char recvline[ MAXLINE + 1 ];
    struct sockaddr_in servaddr;

    if( argc != 2 ) {
```



```
printf( "usage : a.out <IP address>\n" );
exit( 1 );
}

if( ( sockfd = socket( AF_INET , SOCK_STREAM , 0 ) ) < 0 ) {
    printf( "socket error\n" );
    exit( 1 );
}

memset( &servaddr , 0 , sizeof( servaddr ) );
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons( 8888 );

if( inet_pton( AF_INET , argv[ 1 ] , &servaddr.sin_addr ) <= 0 ) {
    printf( "inet_pton error for %s\n" , argv[ 1 ] );
    exit( 1 );
}

if( connect( sockfd , (struct sockaddr *)&servaddr , sizeof( servaddr ) ) < 0 ) {
    printf( "connect error\n" );
    exit( 1 );
}

while( ( n = read( sockfd , recvline , MAXLINE ) ) > 0 ) {
    recvline[ n ] = 0;
    if( fputs( recvline , stdout ) == EOF ) {
        printf( "fputs error\n" );
        exit( 1 );
    }
}

if( n < 0 ) {
    printf( "read error\n" );
    exit( 1 );
}
exit( 0 );
}
```

## 附录 2:实验二源代码

datetimes.c

```
/*
*****
***** datetime Example Server *****
*/
```



```
/******
```

```
#include "datetime.h"
```

```
#include <time.h>
```

```
int
```

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

```
{
```

```
    int listenfd , connfd, idx;
```

```
    struct sockaddr_in servaddr;
```

```
    char buff[ MAXLINE ];
```

```
    time_t ticks;
```

```
    pid_t pid;
```

```
    char recvline[ MAXLINE + 1];
```

```
    listenfd = socket( AF_INET , SOCK_STREAM , 0 );
```

```
    memset( &servaddr , 0 , sizeof( servaddr ) );
```

```
    servaddr.sin_family = AF_INET;
```

```
    servaddr.sin_addr.s_addr = htonl( INADDR_ANY );
```

```
    servaddr.sin_port = htons( 13 );
```

```
    bind( listenfd , (struct sockaddr *)&servaddr , sizeof( servaddr ) );
```

```
    listen( listenfd , 1024 );
```

```
    for( ; )
```

```
    {
```

```
        connfd = accept( listenfd , (struct sockaddr *)NULL , NULL );
```

```
        printf("Initialzeing new connection...\n");
```

```
        if((pid = fork()) == 0)
```

```
        {
```

```
            close(listenfd);
```

```
            ticks = time( NULL );
```

```
            snprintf( buff , sizeof( buff ) , "服务端子进程 id%d\n 时间: %.24s\r\n" , getpid() ,  
ctime( &ticks ) );
```

```
            printf("%s" , buff);
```

```
            write( connfd , buff , strlen( buff ) );
```

```
            if( ( idx = read( connfd , recvline , MAXLINE ) ) > 0 ) {
```

```
                recvline[ idx ] = 0;
```

```
                if( fputs( recvline , stdout ) == EOF ) {
```

```
                    printf( "fputs error\n" );
```

```
                    exit( 1 );
```

```
                }
```

```
    }

    close( connfd );
    exit(0);
}

close( connfd );
}
}
```

## 附录 3:实验三源代码

```
#include <stdio.h>
#include <signal.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <setjmp.h>
#include <errno.h>

#define PACKET_SIZE      4096
#define MAX_WAIT_TIME    5
#define MAX_NO_PACKETS  3

char sendpacket[PACKET_SIZE];
char recvpacket[PACKET_SIZE];
int sockfd,dataalen=56;
int nsend=0,nreceived=0;
struct sockaddr_in dest_addr;
pid_t pid;
struct sockaddr_in from;
struct timeval tvrecv;

void statistics(int signo);
unsigned short cal_chksum(unsigned short *addr,int len);
int pack(int pack_no);
void send_packet(void);
void recv_packet(void);
int unpack(char *buf,int len);
```





```
void tv_sub(struct timeval *out,struct timeval *in);
```

```
void statistics(int signo)
```

```
{    printf("\n-----PING statistics-----\n");
    printf("%d packets transmitted, %d received , %%%d lost\n",nsend,nreceived,
           (nsend-nreceived)/nsend*100);

    close(sockfd);
    exit(1);
}
```

```
/*校验和算法*/
```

```
unsigned short cal_chksum(unsigned short *addr,int len)
```

```
{    int nleft=len;
    int sum=0;
    unsigned short *w=addr;
    unsigned short answer=0;
```

```
/*把 ICMP 报头二进制数据以 2 字节为单位累加起来*/
```

```
    while(nleft>1)
    {        sum+=*w++;
            nleft-=2;
    }
```

```
/*若 ICMP 报头为奇数个字节，会剩下最后一字节。把最后一个字节视为一个 2 字节数据的  
高字节，这个 2 字节数据的低字节为 0，继续累加*/
```

```
    if( nleft==1)
    {        *(unsigned char *)&answer=*(unsigned char *)w;
            sum+=answer;
    }
    sum=(sum>>16)+(sum&0xffff);
    sum+=(sum>>16);
    answer=~sum;
    return answer;
```

```
}
```

```
/*设置 ICMP 报头*/
```

```
int pack(int pack_no)
```

```
{    int i,packsize;
    struct icmp *icmp;
    struct timeval *tval;

    icmp=(struct icmp*)sendpacket;
    icmp->icmp_type=ICMP_ECHO;
    icmp->icmp_code=0;
    icmp->icmp_cksum=0;
    icmp->icmp_seq=pack_no;
    icmp->icmp_id=pid;
    packsize=8+datalen;
```



```
tval= (struct timeval *)icmp->icmp_data;
gettimeofday(tval,NULL);    /*记录发送时间*/
icmp->icmp_cksum=cal_chksum( (unsigned short *)icmp,packsize); /*校验算法*/
return packsize;
}

/*发送三个 ICMP 报文*/
void send_packet()
{
    int packsize;
    while( nsend<MAX_NO_PACKETS)
    {
        nsend++;
        packsize=pack(nsend); /*设置 ICMP 报头*/
        if( sendto(sockfd,sendpacket,packsize,0,
                    (struct sockaddr *)&dest_addr,sizeof(dest_addr) )<0 )
        {
            perror("sendto error");
            continue;
        }
        sleep(1); /*每隔一秒发送一个 ICMP 报文*/
    }
}

/*接收所有 ICMP 报文*/
void recv_packet()
{
    int n,fromlen;
    extern int errno;

    signal(SIGALRM,statistics);
    fromlen=sizeof(from);
    while( nreceived<nsend)
    {
        alarm(MAX_WAIT_TIME);
        if( (n=recvfrom(sockfd,recvpacket,sizeof(recvpacket),0,
                        (struct sockaddr *)&from,&fromlen)) <0)
        {
            if(errno==EINTR)continue;
            perror("recvfrom error");
            continue;
        }
        gettimeofday(&tvrecv,NULL); /*记录接收时间*/
        if(unpack(recvpacket,n)==-1)continue;
        nreceived++;
    }
}

/*剥去 ICMP 报头*/
int unpack(char *buf,int len)
{
    int i,iphdrlen;
```



```
struct ip *ip;
struct icmp *icmp;
struct timeval *tvsend;
double rtt;

ip=(struct ip *)buf;
iphdrlen=ip->ip_hl<<2;    /*求 ip 报头长度,即 ip 报头的长度标志乘 4*/
icmp=(struct icmp *) (buf+iphdrlen); /*越过 ip 报头,指向 ICMP 报头*/
len-=iphdrlen;           /*ICMP 报头及 ICMP 数据报的总长度*/
if( len<8)               /*小于 ICMP 报头长度则不合理*/
{
    printf("ICMP packets\'s length is less than 8\n");
    return -1;
}
/*确保所接收的是我所发的 ICMP 的回应*/
if( (icmp->icmp_type==ICMP_ECHOREPLY) && (icmp->icmp_id==pid) )
{
    tvsend=(struct timeval *)icmp->icmp_data;
    tv_sub(&tvrecv,tvsend); /*接收和发送的时间差*/
    rtt=tvrecv.tv_sec*1000+tvrecv.tv_usec/1000; /*以毫秒为单位计算 rtt*/
    /*显示相关信息*/
    printf("%d byte from %s: icmp_seq=%u ttl=%d rtt=%.3f ms\n",
           len,
           inet_ntoa(from.sin_addr),
           icmp->icmp_seq,
           ip->ip_ttl,
           rtt);
}
else    return -1;
}

main(int argc,char *argv[])
{
    struct hostent *host;
    struct protoent *protocol;
    unsigned long inaddr=0l;
    int waittime=MAX_WAIT_TIME;
    int size=50*1024;

    if(argc<2)
    {
        printf("usage:%s hostname/IP address\n",argv[0]);
        exit(1);
    }

    if( (protocol=getprotobyname("icmp"))==NULL)
    {
        perror("getprotobyname");
        exit(1);
    }
}
```



```
/*生成使用 ICMP 的原始套接字,这种套接字只有 root 才能生成*/
if( (sockfd=socket(AF_INET,SOCK_RAW,protocol->p_proto) )<0)
{
    perror("socket error");
    exit(1);
}
/* 回收 root 权限,设置当前用户权限*/
setuid(getuid());
/*扩大套接字接收缓冲区到 50K 这样做主要为了减小接收缓冲区溢出的
   可能性,若无意中 ping 一个广播地址或多播地址,将会引来大量应答*/
setsockopt(sockfd,SOL_SOCKET,SO_RCVBUF,&size,sizeof(size) );
bzero(&dest_addr,sizeof(dest_addr));
dest_addr.sin_family=AF_INET;

/*判断是主机名还是 ip 地址*/
if( inaddr=inet_addr(argv[1])==INADDR_NONE)
{
    if((host=gethostbyname(argv[1]) )==NULL) /*是主机名*/
    {
        perror("gethostbyname error");
        exit(1);
    }
    memcpy( (char *)&dest_addr.sin_addr,host->h_addr,host->h_length);
}
else /*是 ip 地址*/
    memcpy( (char *)&dest_addr,(char *)&inaddr,host->h_length);
/*获取 main 的进程 id,用于设置 ICMP 的标志符*/
pid=getpid();
printf("PING %s(%s): %d bytes data in ICMP packets.\n",argv[1],
        inet_ntoa(dest_addr.sin_addr),datalen);
send_packet(); /*发送所有 ICMP 报文*/
recv_packet(); /*接收所有 ICMP 报文*/
statistics(SIGALRM); /*进行统计*/

return 0;

}
/*两个 timeval 结构相减*/
void tv_sub(struct timeval *out,struct timeval *in)
{
    if( (out->tv_usec-=in->tv_usec)<0)
    {
        --out->tv_sec;
        out->tv_usec+=1000000;
    }
    out->tv_sec-=in->tv_sec;
}
```