

PING 程序的实现

162020321 朱家震

2023 年 5 月 19 日

1 实验内容

1.1 实验目的

理解 ping 程序的概念，熟练使用原始套接字

1.2 实验环境

Linux, C

1.3 实验内容

1. 设计一个简单的 PING 程序，每隔 1 秒钟使用 ICMP 报文向目的 IP 地址发一个 ICMP 请求（长度由 length 指定），对方将返回一个 ICMP 应答，应答数据包通过循环调用函数 `recvfrom` 来接收。发送 ICMP 报文的次数由 counts 指定
2. `ping dstIP -l length -n counts`

2 实验设计

2.1 ping 程序执行

首先我们执行 ping 程序，了解我们要做的输出是什么样的，执行出来的效果如下：

```
C:\Users\Administrator>ping www.baidu.com -l 64 -n 5

正在 Ping www.a.shifen.com [112.80.248.76] 具有 64 字节的数据:
来自 112.80.248.76 的回复: 字节=64 时间=12ms TTL=57
来自 112.80.248.76 的回复: 字节=64 时间=8ms TTL=57
来自 112.80.248.76 的回复: 字节=64 时间=10ms TTL=57
来自 112.80.248.76 的回复: 字节=64 时间=8ms TTL=57
来自 112.80.248.76 的回复: 字节=64 时间=9ms TTL=57

112.80.248.76 的 Ping 统计信息:
    数据包: 已发送 = 5, 已接收 = 5, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 8ms, 最长 = 12ms, 平均 = 9ms
```

图 1: windows 终端执行结果

```
zhujiazhen@zhujiazhen-virtual-machine:~$ ping www.baidu.com -s 64 -c 5
PING www.a.shifen.com (112.80.248.76) 64(92) bytes of data.
72 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=1 ttl=128 time=8.11 ms
72 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=2 ttl=128 time=21.5 ms
72 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=3 ttl=128 time=9.81 ms
72 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=4 ttl=128 time=9.87 ms
72 bytes from 112.80.248.76 (112.80.248.76): icmp_seq=5 ttl=128 time=8.99 ms

--- www.a.shifen.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 8.119/11.671/21.555/4.983 ms
```

图 2: Linux 终端执行结果

我们接下来就根据 Linux 的输出结果进行一个复现。

2.2 程序分块

根据题意，我们可以将代码分为几个重要部分：计算校验和、发送 ICMP 请求以及接受 ICMP 请求。各个函数的具体实现在下一部分讲解。

3 关键程序

3.1 计算校验和

```
1 unsigned short calcChecksum(unsigned short *addr, int len)
2 {
3     unsigned int sum = 0;
4     unsigned short answer = 0;
5     unsigned short *w = addr;
6     int nleft = len;
7
8     while (nleft > 1) {
9         sum += *w++;
10        nleft -= 2;
```

```

11     }
12
13     if (nleft == 1) {
14         *(unsigned char *)(&answer) = *(unsigned char *)w;
15         sum += answer;
16     }
17
18     sum = (sum >> 16) + (sum & 0xFFFF);
19     sum += (sum >> 16);
20     answer = ~sum;
21     return answer;
22 }

```

校验和是为了确保数据在传输过程中的完整性。接收方可以通过校验和验证数据是否被篡改或损坏。

‘calcChecksum’函数使用了标准的互联网校验和算法，对每个 16 位的字进行求和，并进行溢出的处理。同时，计算校验和时使用无符号短整型（unsigned short）是为了确保计算的结果符合互联网校验和算法的规范。如果使用有符号短整型（short），可能会导致计算结果出现溢出或不符合预期，因为有符号短整型的范围是从负数到正数，而不是从 0 到正数。

3.2 发送 ICMP 请求

```

1 int sendPingRequest(int sockfd, struct sockaddr_in *dest_addr, int
   packet_size, int seq)
2 {
3     struct icmp *icmp_packet;
4     char send_packet[PACKET_SIZE];
5     int packet_len;
6
7     icmp_packet = (struct icmp *)send_packet;
8     icmp_packet->icmp_type = ICMP_ECHO;
9     icmp_packet->icmp_code = 0; // 询问报文
10    icmp_packet->icmp_id = getpid();
11    icmp_packet->icmp_seq = seq;
12    memset(icmp_packet->icmp_data, 0xa5, packet_size);
13    gettimeofday((struct timeval *)icmp_packet->icmp_data, NULL);
14
15    packet_len = 8 + packet_size;
16    icmp_packet->icmp_cksum = 0;

```

```

17     icmp_packet->icmp_cksum = calcChecksum((unsigned short *)icmp_packet
18         , packet_len);
19
20     if (sendto(sockfd, send_packet, packet_len, 0,
21         (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) == -1) {
22         perror("sendto error");
23         return -1;
24     }
25
26     return 0;
27 }

```

发送函数中，我们使用了 icmp 结构体：

```

1 struct icmp_hdr {
2     uint8_t type;           // ICMP 报文类型
3     uint8_t code;           // ICMP 报文代码
4     uint16_t checksum;      // ICMP 报文校验和
5     // 其他字段根据不同的 ICMP 报文类型可以有不同的定义
6 };

```

数据部分被填充为 0xa5，并使用 ‘memset’ 函数将数据部分的字节设置为指定的值。这是为了在发送方和接收方之间提供一致的数据，以便检查传输的正确性。

3.3 接收 ICMP 请求

```

1 int receivePingResponse(int sockfd, int seq)
2 {
3     char recv_packet[PACKET_SIZE];
4     struct sockaddr_in from;
5     socklen_t from_len;
6     int packet_len;
7
8     while (1) {
9         memset(recv_packet, 0, sizeof(recv_packet));
10        from_len = sizeof(from);
11
12        if ((packet_len = recvfrom(sockfd, recv_packet, sizeof(
13            recv_packet), 0,
14            (struct sockaddr *)&from, &from_len)) == -1) {
15            perror("recvfrom error");
16            return -1;
17        }
18    }
19 }

```

```

16     }
17     // printf("Received ICMP response from: %s\n", inet_ntoa(from.
        sin_addr));
18
19     // 解析ICMP应答
20     struct ip *ip_packet = (struct ip *)recv_packet;
21     struct icmp *icmp_packet = (struct icmp *) (recv_packet + (
        ip_packet->ip_hl << 2));
22
23     if (icmp_packet->icmp_type == ICMP_ECHOREPLY && icmp_packet->
        icmp_id == getpid()
24         && icmp_packet->icmp_seq == seq) {
25         struct timeval *st = (struct timeval *)icmp_packet->icmp_data;
26         struct timeval ct;
27         gettimeofday(&ct, NULL);
28
29         double rtt = (ct.tv_sec - st->tv_sec) * 1000.0 + (ct.tv_usec -
        st->tv_usec) / 1000.0;
30         printf("%d bytes from %s: icmp_seq=%d time=%.2fms ttl=%d\n",
31             packet_len, inet_ntoa(from.sin_addr), seq, rtt, ip_packet
        ->ip_ttl);
32         // printf("ICMP response received.\n");
33         break;
34     }
35 }
36
37 return 0;
38 }

```

在接受 icmp 函数中，首先，根据 IP 首部的长度字段找到 ICMP 首部的位置。然后，检查 ICMP 类型是否为 ICMP 回显应答，并验证 ICMP 标识和序列号与发送的请求是否匹配。最后，计算往返时间（RTT）并输出相关信息。

3.4 主函数

在主函数解析地址时，‘inet_pton’函数用于将点分十进制表示的 IPv4 地址转换为二进制形式。而 ‘gethostbyname’函数用于解析主机名，将主机名转换为 IP 地址。

4 实验结果与分析

最后，myping 的运行结果如下：

```
zhujiazhen@zhujiazhen-virtual-machine:~/net-experiment/myping$ sudo ./myping www.baidu.com -l 64 -n 5
[sudo] zhujiazhen 的密码:
64 bytes from 112.80.248.76: icmp_seq=1 time=10.21ms ttl=128
64 bytes from 112.80.248.76: icmp_seq=2 time=9.10ms ttl=128
64 bytes from 112.80.248.76: icmp_seq=3 time=8.74ms ttl=128
64 bytes from 112.80.248.76: icmp_seq=4 time=8.45ms ttl=128
64 bytes from 112.80.248.76: icmp_seq=5 time=8.10ms ttl=128
```

图 3: myping 终端执行结果

可以看出结果较为清晰，比较真实的还原了 Linux 上 ping 的输出结果。