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：

图 1: windows 终端执行结果

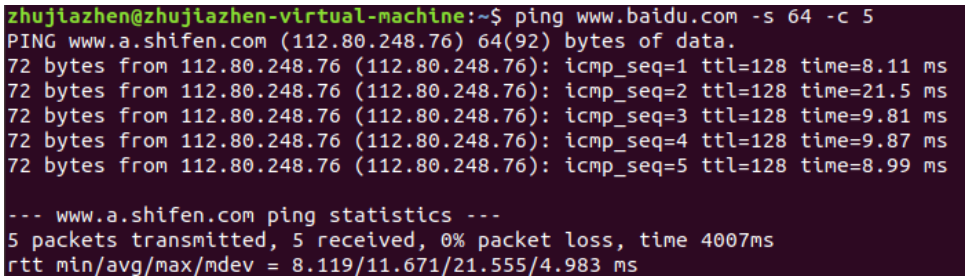

图 2: Linux 终端执行结果

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

```
unsigned short calcChecksum(unsigned short *addr, int len)
{
    unsigned int sum = 0;
    unsigned short answer = 0;
    unsigned short *w = addr;
    int nleft = len;

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

```
    }

    if (nleft == 1) {
        *(unsigned char *)(&answer) = *(unsigned char *)w;
        sum += answer;
    }

    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    answer = ~sum;
    return answer;
}
```

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

```
int sendPingRequest(int sockfd, struct sockaddr_in *dest_addr, int
   packet_size, int seq)
{
    struct icmp *icmp_packet;
    char send_packet[PACKET_SIZE];
    int packet_len;

    icmp_packet = (struct icmp *)send_packet;
    icmp_packet->icmp_type = ICMP_ECHO;
    icmp_packet->icmp_code = 0; // 询问报文
    icmp_packet->icmp_id = getpid();
    icmp_packet->icmp_seq = seq;
    memset(icmp_packet->icmp_data, 0xa5, packet_size);
    gettimeofday((struct timeval *)icmp_packet->icmp_data, NULL);

    packet_len = 8 + packet_size;
    icmp_packet->icmp_cksum = 0;
```

```
    icmp_packet->icmp_cksum = calcChecksum((unsigned short *)icmp_packet
        , packet_len);

    if (sendto(sockfd, send_packet, packet_len, 0,
        (struct sockaddr *)dest_addr, sizeof(struct sockaddr)) == -1) {
        perror("sendto error");
        return -1;
    }

    return 0;
}
```

发送函数中，我们使用了 icmp 结构体：

```
struct icmphdr {
    uint8_t type;        // ICMP 报文类型
    uint8_t code;        // ICMP 报文代码
    uint16_t checksum;   // ICMP 报文校验和
    // 其他字段根据不同的 ICMP 报文类型可以有不同的定义
};
```

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

```
int receivePingResponse(int sockfd, int seq)
{
    char recv_packet[PACKET_SIZE];
    struct sockaddr_in from;
    socklen_t from_len;
    int packet_len;

    while (1) {
        memset(recv_packet, 0, sizeof(recv_packet));
        from_len = sizeof(from);

        if ((packet_len = recvfrom(sockfd, recv_packet, sizeof(
            recv_packet), 0,
                (struct sockaddr *)&from, &from_len)) == -1) {
            perror("recvfrom error");
            return -1;
```

```
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：

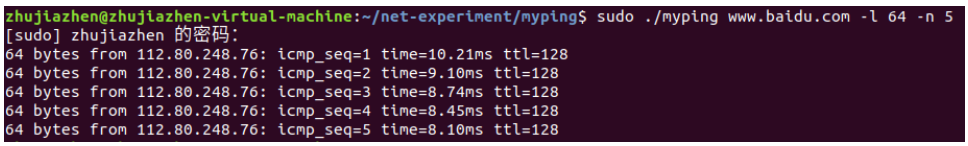

图 3: myping 终端执行结果

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。