二、Tracert 与 Ping 程序设计与实现

课程设计题目

了解Tracert程序的实现原理，并调试通过。然后参考Tracert程序和教材4.4.2节，编写一个Ping程序，并能测试本局域网的所有机器是否在线。

课程设计内容

ICMP(Internet Control Message,网际控制报文协议)是为网关和目标主机而提供的一种差错控制机制，使它们在遇到差错时能把错误报告给报文源发方。ICMP协议是IP层的一个协议，但是由于差错报告在发送给报文源发方时可能也要经过若干子网，因此牵涉到路由选择等问题，所以ICMP报文需通过IP协议来发送。ICMP数据报的数据发送前需要两级封装：首先添加ICMP报头形成ICMP报文，再添加IP报头形成IP数据报。

课程设计结果及结果分析 心得体会

由于IP层协议是一种点对点的协议，而非端对端的协议，它提供无连接的数据报服务，没有端口的概念，因此很少使用bind()和connect()函数，若有使用也只是用于设置IP地址。发送数据使用sendto()函数，接收数据使用recvfrom()函数。

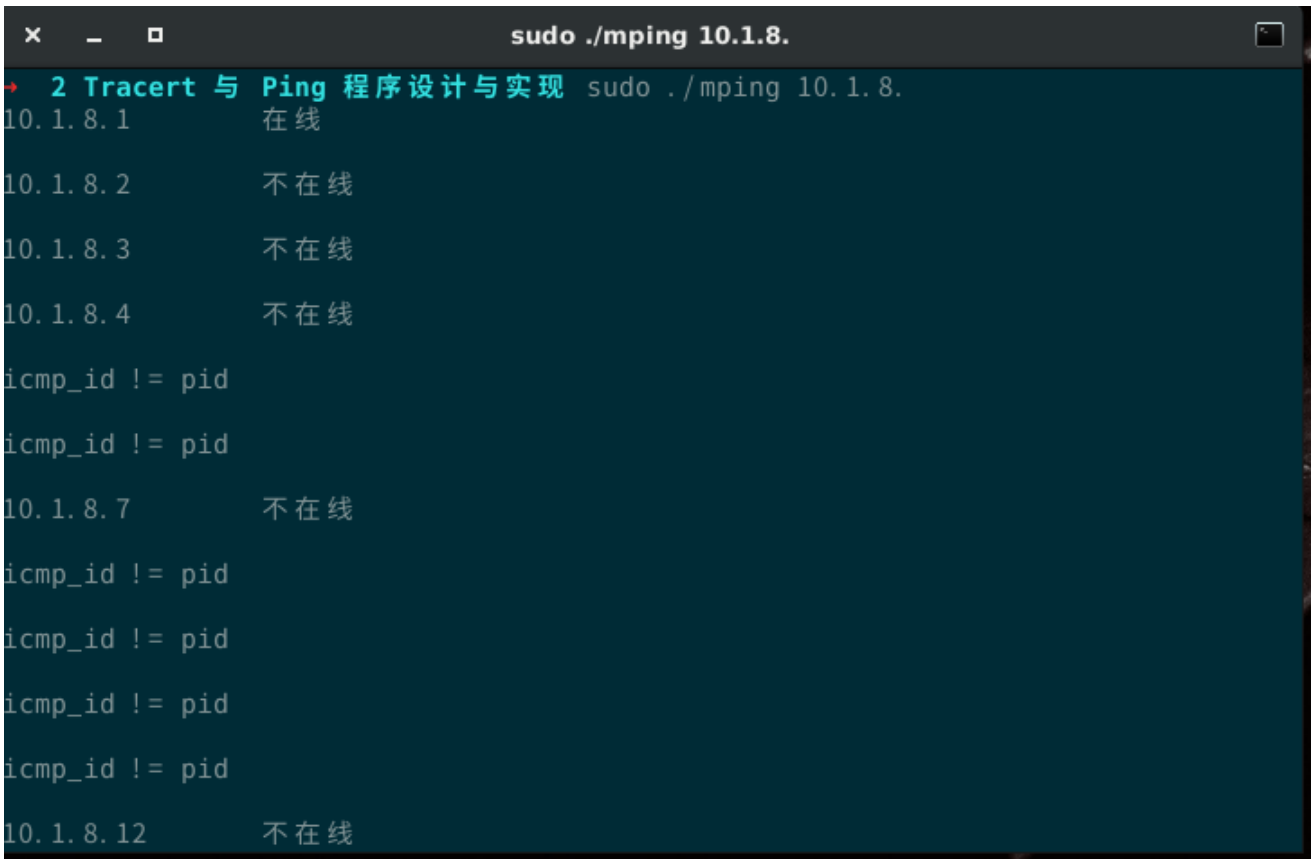
ICMP报文分为两种，一是错误报告报文，二是查询报文。每个ICMP报头均包含类型、编码和校验和这三项内容，长度为８位，８位和１６位，其余选项则随ICMP的功能不同而不同。

Ping命令只使用众多ICMP报文中的两种："请求回送'(ICMP_ECHO)和"请求回应'(ICMP_ECHOREPLY)。在Linux中定义如下：

```
#define ICMP_ECHO   0
#define ICMP_ECHOREPLY  8
```

程序源文件主要代码展示

程序运行截图



程序主要代码展示

```
int main(int argc, char **argv) /*argc表示隐形程序命令行中参数的数目，argv是一个指向字符串数组指针，其中每一个字符对应一个参数*/
{
    struct hostent *host; /*该结构体属于include<netdb.h>*/
    int on = 1;

    if (argc < 2)    /*判断是否输入了地址*/
    {
        printf("Usage: %s net number\n", argv[0]);
        exit(1);
    }

    /*
    if ((host = gethostbyname(argv[1])) == NULL) //gethostbyname()返回对应于给定主机名的包含主机名字和地址信息的结构指针
    {
        perror("can not understand the host name");
        exit(1);
    }
    */

    hostname = argv[1]; /*取出地址名*/

    int hosts_i = 1;
    for (; hosts_i < 255; hosts_i++)
```

```
    {
        /*
          构造 ICMP 报文并发送
        */
        memset(&dest, 0, sizeof dest); /*将dest中前sizeof(dest)个字节替换为0并返回s,此处为初始化,给最大内存清零*/
        dest.sin_family = PF_INET; /*PF_INET为IPV4, internet协议, 在<netinet/in.h>中, 地址族*/
        dest.sin_port = ntohs(0); /*端口号,ntohs()返回一个以主机字节顺序表达的数。*/
        // dest.sin_addr = *(struct in_addr *)host->h_addr_list[0]; /*host->h_addr_list[0]是地址的指针.返回IP地址, 初始化*/
        char addr_tmp[20];
        char host_i[5];
        memset(addr_tmp, '\0', sizeof(addr_tmp));
        memset(host_i, '\0', sizeof(host_i));
        strcpy(addr_tmp, argv[1]);
        // printf("after cpy:%s\n", addr_tmp);
        sprintf(host_i, "%d", hosts_i);
        // printf("after sprintf:%s\n", host_i);
        strcat(addr_tmp, host_i);
        // printf("after strcat:%s\n", addr_tmp);
        dest.sin_addr.s_addr = inet_addr(addr_tmp); /*host->h_addr_list[0]是地址的指针.返回IP地址, 初始化*/

        if ((sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0) /*PF_INEI套接字协议族, SOCK_RAW套接字类型, IPPROTO_ICMP使用协议, 调用socket函数来创建一个能够进行网络通信的套接字。这里判断是否创建成功*/
        {
            perror("raw socket created error");
            exit(1);
        }

        /*设置当前套接字选项特定属性值
          sockfd套接字;
          IPPROTO_IP协议层为IP层;
          IP_HDRINCL套接字选项条目;
          套接字接收缓冲区指针;
          sizeof(on)缓冲区长度的长度
        */
        setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
        // setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, (char*)&val_alarm, sizeof(val_alarm));

        /*getuid()函数返回一个调用程序的真实用户ID,setuid()是让普通用户可以以root用户的角色运行只有root帐号才能运行的程序或命令。*/
        setuid(getuid());
        /*getpid函数用来取得目前进程的进程识别码*/
        pid = getpid();

        set_sighandler();/*对信号处理*/

        // printf("mPing %s(%s): %d bytes data in ICMP packets.\n\n",
        //        addr_tmp, inet_ntoa(dest.sin_addr), datalen);
```

```
        if ((setitimer(ITIMER_REAL, &val_alarm, NULL)) == -1) /*定时函数*/
            bail("setitimer fails.");


        recv_reply();/*接收ping应答*/
    }

    return 0;
}
/*发送ping消息*/
void send_ping(void)
{
    struct iphdr *ip_hdr;   /*iphdr为IP头部结构体*/
    struct icmphdr *icmp_hdr;   /*icmphdr为ICMP头部结构体*/
    int len;
    int len1;
    /*ip头部结构体变量初始化*/
    ip_hdr = (struct iphdr *)sendbuf; /*字符串指针*/
    ip_hdr->hlen = sizeof(struct iphdr) >> 2; /*头部长度*/
    ip_hdr->ver = IPVERSION; /*版本*/
    ip_hdr->tos = 0; /*服务类型*/
    ip_hdr->tot_len = IP_HSIZE + ICMP_HSIZE + datalen; /*报文头部加数据的总长度*/
    ip_hdr->id = 0;  /*初始化报文标识*/
    ip_hdr->frag_off = 0; /*设置flag标记为0*/
    ip_hdr->protocol = IPPROTO_ICMP; /*运用的协议为ICMP协议*/
    ip_hdr->ttl = 255; /*一个封包在网络上可以存活的时间*/
    ip_hdr->daddr = dest.sin_addr.s_addr; /*目的地址*/
    len1 = ip_hdr->hlen << 2; /*ip数据长度*/

    /*ICMP头部结构体变量初始化*/
    icmp_hdr = (struct icmphdr *)(sendbuf + len1); /*字符串指针*/
    icmp_hdr->type = 8;  /*初始化ICMP消息类型type*/
    icmp_hdr->code = 0;  /*初始化消息代码code*/
    icmp_hdr->icmp_id = pid; /*把进程标识码初始给icmp_id*/
    icmp_hdr->icmp_seq = nsent++; /*发送的ICMP消息序号赋值给icmp序号*/
    memset(icmp_hdr->data, 0xff, datalen); /*将datalen中前datalen个字节替换为0xff并返回
icmp_hdr-dat*/

    gettimeofday((struct timeval *)icmp_hdr->data, NULL); /* 获取当前时间*/

    len = ip_hdr->tot_len; /*报文总长度赋值给len变量*/
    icmp_hdr->checksum = 0;  /*初始化*/
    icmp_hdr->checksum = checksum((u8 *)icmp_hdr, len); /*计算校验和*/

    sendto(sockfd, sendbuf, len, 0, (struct sockaddr *)&dest, sizeof (dest)); /*经socket
传送数据*/
}

/*接收程序发出的ping命令的应答*/
void recv_reply()
{
    int n, len;
```

```
    int errno;

    // nrecv: 接收的 ICMP 消息序号
    n = nrecv = 0;
    /*发送ping应答消息的主机IP
      from: 发送ping应答消息的主机 IP （结构体）
    */
    len = sizeof(from);

    // 只接收一次 ICMP 消息报文
    while (nrecv < 1)
    {
        /*经socket接收数据,如果正确接收返回接收到的字节数, 失败返回0.*/
        struct timeval structtimeval = {5, 0};
        setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&structtimeval,
sizeof(structtimeval));
        memset(&from, 0, sizeof from); /*将from中前sizeof(from)个字节替换为0并返回s,此处为初
始化,给最大内存清零*/
        if ((n = recvfrom(sockfd, recvbuf, sizeof recvbuf, 0, (struct sockaddr *)&from,
&len)) < 0)
        {
            if (errno == EINTR) /*EINTR表示信号中断*/
            {
                // printf("continue1\n");
                // printf("%s\t不在线\n", inet_ntoa(from.sin_addr));
                printf("%s\t不在线\n", inet_ntoa(dest.sin_addr));
                break;
                // continue;
            }
            // bail("recvfrom error");
        }

        // recvtime:15464374xx
        gettimeofday(&recvtime, NULL);  /*记录收到应答的时间*/

        /*接收到错误的ICMP应答信息*/
        if (handle_pkt())
        {
            printf("continue2\n");
            continue;
        }

        nrecv++;
    }

    get_statistics(nsent, nrecv);    /*统计ping命令的检测结果*/
}
/*计算校验和*/
u16 checksum(u8 *buf, int len)
{
    u32 sum = 0;
    u16 *cbuf;
```

```
    cbuf = (u16 *)buf;

    while (len > 1)
    {
        sum += *cbuf++;
        len -= 2;
    }

    if (len)
        sum += *(u8 *)cbuf;

    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);

    return ~sum;
}

/*ICMP应答消息处理*/
int handle_pkt()
{
    struct iphdr *ip;
    struct icmphdr *icmp;

    int ip_hlen;
    u16 ip_datalen; /*ip数据长度*/
    double rtt; /* 往返时间*/
    struct timeval *sendtime;

    ip = (struct iphdr *)recvbuf;

    ip_hlen = ip->hlen << 2;
    ip_datalen = ntohs(ip->tot_len) - ip_hlen;

    icmp = (struct icmphdr *)(recvbuf + ip_hlen);

    if (checksum((u8 *)icmp, ip_datalen)) /*计算校验和*/
    {
        printf("计算校验和失败\n");
        return -1;
    }

    // 打印 ICMP 报文 type 字段
    // printf("ICMP type:%d\tICMP code:%d\n", icmp->type, icmp->code);

    if (icmp->icmp_id != pid)
    {
        printf("icmp_id != pid\n");
        // printf("%s\t不在线\n", inet_ntoa(from.sin_addr));
        return 0;
    }

    sendtime = (struct timeval *)icmp->data; /*发送时间*/
```

```
    rtt = ((&recvtime)->tv_sec - sendtime->tv_sec) * 1000 + ((&recvtime)->tv_usec -
sendtime->tv_usec) / 1000.0; /* 往返时间*/

    /*打印结果*/
    // printf("%d bytes from %s:icmp_seq=%u ttl=%d rtt=%.3f ms\n",
    //        ip_datalen, /*IP数据长度*/
    //        inet_ntoa(from.sin_addr),    /*目的ip地址*/
    //        icmp->icmp_seq, /*icmp报文序列号*/
    //        ip->ttl,  /*生存时间*/
    //        rtt);    /*往返时间*/
    printf("%s\t在线\n", inet_ntoa(from.sin_addr));

    return 0;
}

/*设置信号处理程序*/
void set_sighandler()
{
    act_alarm.sa_handler = alarm_handler;
    if (sigaction(SIGALRM, &act_alarm, NULL) == -1) /*sigaction()会依参数signum指定的信号编
号来设置该信号的处理函数。参数signum指所要捕获信号或忽略的信号，&act代表新设置的信号共用体，NULL代表之
前设置的信号处理结构体。这里判断对信号的处理是否成功。*/
        bail("SIGALRM handler setting fails.");

    act_int.sa_handler = int_handler;
    if (sigaction(SIGINT, &act_int, NULL) == -1)
        bail("SIGALRM handler setting fails.");
}

/*统计ping命令的检测结果*/
void get_statistics(int nsent, int nrecv)
{
    printf("\n");
    /*
    printf("--- %s ping statistics ---\n", inet_ntoa(dest.sin_addr)); //将网络地址转换
成“.”点隔的字符串格式。
    printf("%d packets transmitted, %d received, %0.0f%% ""packet loss\n",
           nsent, nrecv, 1.0 * (nsent - nrecv) / nsent * 100);
    */
}

/*错误报告*/
void bail(const char * on_what)
{
    fputs(strerror(errno), stderr); /*:向指定的文件写入一个字符串（不写入字符串结束标记
符‘\0’）。成功写入一个字符串后，文件的位置指针会自动后移，函数返回值为0；否则返回EOR(符号常量，其值
为-1)。*/
    fputs(":", stderr);
    fputs(on_what, stderr);
    fputc('\n', stderr); /*送一个字符到一个流中*/
    exit(1);
}
```

```c
/*SIGINT（中断信号）处理程序*/
void int_handler(int sig)
{
    get_statistics(nsent, nrecv);   /*统计ping命令的检测结果*/
    close(sockfd);  /*关闭网络套接字*/
    exit(1);
}

/*SIGALRM（终止进程）处理程序*/
void alarm_handler(int signo)
{
    send_ping();    /*发送ping消息*/

}
```