计算机网络课程设计要求

一、实验内容

利用实验室提供的网络环境，完成以下两个实验：

1）实验一（必做）

2）从实验二和实验三中选择一个完成。

具体实验要求见后。

二、实验要求

实验分组进行，每组 3-4 人，自愿组织，每组选择一名组长，共同完成实验内容。实验要求如下：

1) 以上实验按照要求完成，具有演示效果，并由指导教师验收；

2) 按要求撰写实验报告，每组提交一份；

三、实验报告格式

见附录 C。

实验一 数据包的捕获与分析

1.1 课程设计目的

数据包捕获技术是网络管理系统的关键技术。本实验通过 Wireshark 软件的安装使用，监控局域网的状态，捕获在局域网中传输的数据包，并结合在计算机网络课程中学习到的理论知识，对常用网络协议的数据包做出分析，加深网络课程知识的理解和掌握。

1.2 课程设计内容

Wireshark 是一种开源的网络数据包的捕获和分析软件，本实验通过 Wireshark 软件的安装使用，监控局域网的状态，捕获在局域网中传输的数据包，并结合在计算机网络课程中学习到的理论知识，对常用网络协议的数据包做出分析，加深网络课程知识的理解和掌握。具体内容及要求如下：

- Wireshark 软件的安装；
- Wireshark 软件的启动，并设置网卡的状态为混杂状态，使得 Wireshark 可以监控局域网的状态；
- 启动数据包的捕获，跟踪 PC 之间的报文，并存入文件以备重新查；
- 设置过滤器过滤网络报文以检测特定数据流；
- 对常用协议的数据包的报文格式进行分析，利用协议分析软件的统计工具显示网络报文的各种统计信息。

1.3 相关知识

数据包捕获技术是网络管理系统的关键技术。以太网（Ethernet）具有共享介质的特征，信息是以明文的形式在网络上传输，当网络适配器设置为监听模式（混杂模式，Promiscuous）时，由于采用以太网广播信道争用的方式，使得监听系统与正常通信的网络能够并联连接，并可以捕获任何一个在同一冲突域上传输的数据包。IEEE802.3 标准的以太网采用的是持续 CSMA 的方式，正是由于以太网采用这种广播信道争用的方式，使得各个站点可以获得其他站点发送的数据。运用这一原理使信息捕获系统能够拦截的我们所要的信息，这是捕获数据包的物理基础。

以太网是一种总线型的网络，从逻辑上来看是由一条总线和多个连接在总线上的站点所组成各个站点采用上面提到的 CSMA/CD 协议进行信道的争用和共享，每个站点（这里特指计算机通过的接口卡）网卡来实现这种功能。网卡主要的工作是完成对于总线当前状态的探测，确定是否进行数据的传送，判断每个物理数据帧目的地是否为本站地址，如果不匹配，则说明不是发送到本站的而将它丢弃。如果是的话，接收该数据帧，进行物理数据帧的 CRC 校验，然后将数据帧提交给 LLC 子层。

网卡具有如下的几种工作模式：

1) 广播模式（Broad Cast Model）：它的物理地址（MAC）地址是 0Xffffff 的帧为广播帧，工作在广播模式的网卡接收广播帧。

2）多播传送（MultiCast Model）：多播传送地址作为目的物理地址的帧可以被组内的其

它主机同时接收，而组外主机却接收不到。但是，如果将网卡设置为多播传送模式，它可以接收所有的多播传送帧，而不论它是不是组内成员。

3）直接模式（Direct Model）：工作在直接模式下的网卡只接收目地址是自己 MAC 地址的帧。

4）混杂模式（Promiscuous Model）：工作在混杂模式下的网卡接收所有的流过网卡的帧，信包捕获程序就是在这种模式下运行的。

网卡的缺省工作模式包含广播模式和直接模式，即它只接收广播帧和发给自己的帧。如果采用混杂模式，一个站点的网卡将接受同一网络内所有站点所发送的数据包这样就可以到达对于网络信息监视捕获的目的。

1.4 课程设计分析

1、在 PC 中安装协议分析软件（如：Wireshark）。具体安装过程详见 Wireshark 用户指南。
2、启动 Wireshark 协议分析软件，选择抓包菜单项启动实时监视器，开始实时跟踪显示网络数据报文。可根据系统提示修改显示方式，详见 Wireshark 用户指南。
3、调出跟踪存储的历史报文，选择有代表性的 ETHERNETⅡ，IEEE802.3，IP，ICMP，TCP，UDP 报文，对照有关协议逐个分析报文各字段的含义及内容。

EHERNET 报文格式

DSTADDR	SRCADDR	TYPE	INFO
6 字节	6 字节	2 字节	
最大长度 1518 字节			

IEEE802.3 报文格式

DSTADDR	SRCADDR	LEN	DSAP	SSAP	CONTROL	INFO
6 字节	6 字节	2 字节	1 字节	1 字节	1/2 字节	信息
最大长度 1518 字节						

IP 报文格式

VERSION	IHL	TOS	TOTAL LENGTN
IDENTIFICATION		FLAGS	FRAGMENT OFFSET
TTL	PROTOWT	HEADER CHECLCSUM	
SOWRCE ADDRESS			
DRSTINWTION ADDRESS			
OPTIONS		PADDING	
INFO			

4、设置过滤器属性，如目的地址，源地址，协议类型等。如过滤不需要的网络报文，过滤

器允许设置第二层，第三层或第四层的协议字段。

过滤器有两种工作方式：

1）捕获前过滤：协议分析软件用过滤器匹配网络上的数据报文，仅当匹配通过时才捕获报文。

2）捕获后过滤：协议分析软件捕获所有报文，但仅显示匹配符合过滤条件的报文。

选择统计菜单项可以显示网络中各种流量的统计信息，如：关于字节数，广播中报文数，出错数等。详见：Wireshark 用户指南。

实验二 网络层实验—Ping 程序的设计与实现

2.1 课程设计目的

本实验目的是使学生掌握网络层协议的原理及实现方法。

2.2 课程设计内容

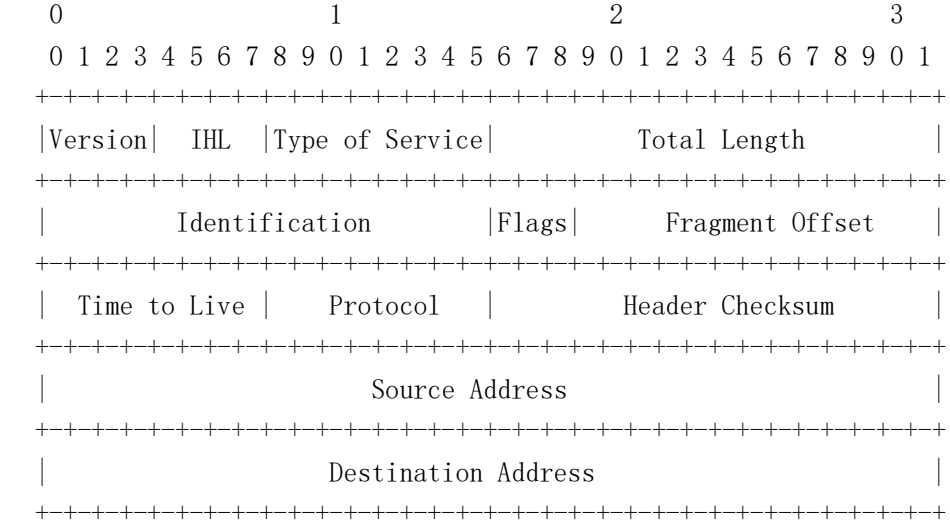
本实验为 ICMP 实验。实验内容：Ping 命令实现的扩充，在给定的 Ping 程序的基础上做如下功能扩充：

- -h 显示帮助信息
- -b 允许ping一个广播地址，只用于IPv4
- -t 设置ttl值，只用于IPv4
- -q 安静模式。不显示每个收到的包的分析结果，只在结束时，显示汇总结果

Ping 命令的基本描述

Ping 的操作是向某些 IP 地址发送一个 ICMP Echo 消息,接着该节点返回一个 ICMP Echo replay 消息。

ICMP 消息使用 IP 头作为基本控制。IP 头的格式如下



Version=4

IHL Internet 头长

Type of Service = 0

Total Length IP 包的总长度

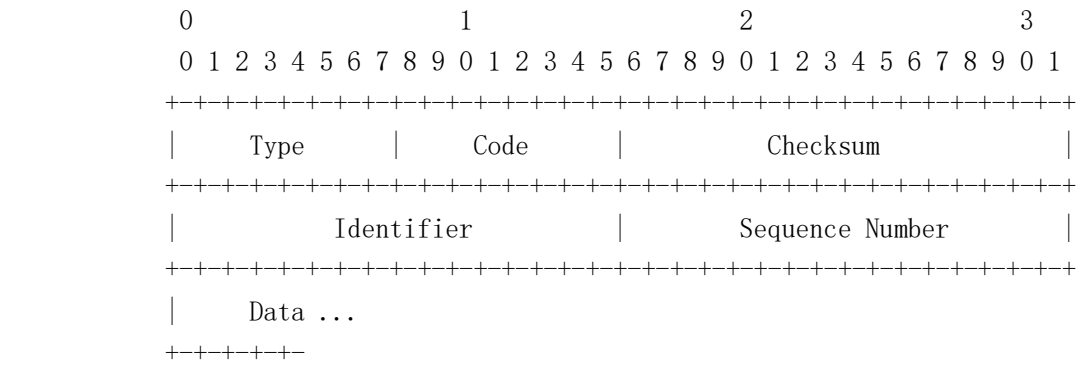
Identification, Flags, Fragment Offset 用于 IP 包分段

Time to Live IP 包的存活时长

Protocol ICMP = 1

Addresses 发送 Echo 消息的源地址是发送 Echo reply 消息的目的地址,相反,发送 Echo 消息的目的地址是发送 Echo reply 消息的源地址。

Ping 实际上是使用 ICMP 中的 ECHO 报文来实现的。Echo 或 Echo Reply 消息格式如下:



Type

echo 消息的类型为 8

echo reply 的消息类型为 0。

Code=0

Checksum

为从 TYPE 开始到 IP 包结束的校验和

Identifier

如果 code = 0, identifier 用来匹配 echo 和 echo reply 消息

Sequence Number

如果 code = 0, identifier 用来匹配 echo 和 echo reply 消息

功能描述:

收到 echo 消息必须回应 echo reply 消息。

identifier 和 sequence number 可能被发送 echo 的主机用来匹配返回的 echo reply 消息。例如: identifier 可能用于类似于 TCP 或 UDP 的 port 用来标示一个会话, 而 sequence number 会在每次发送 echo 请求后递增。

收到 echo 的主机或路由器返回同一个值与之匹配

1、数据结构的描述

1) IP 包格式

```
struct ip {
    BYTE Ver_ihl;     //版本号与包头长度
    BYTE TOS;         //服务类型
    WORD Leng;        //IP 包长度
    WORD Id;          //IP 包标示,用于辅助 IP 包的拆装,本实验不用,置零
    WORD Flg_offset; //偏移量,也是用于 IP 包的拆装,本实验不用,置零
    BYTE TTL;         //IP 包的存活时间
```

```
    BYTE Protocol;    //上一层协议,本实验置 ICMP
    WORD Checksum; //包头校验和,最初置零,等所有包头都填写正确后,计算并替换。
    BYTE Saddr[4];    //源端 IP 地址
    BYTE Daddr[4];    //目的端 IP 地址
    BYTE Data[1];     //IP 包数据
};
```

2）ICMP 包格式

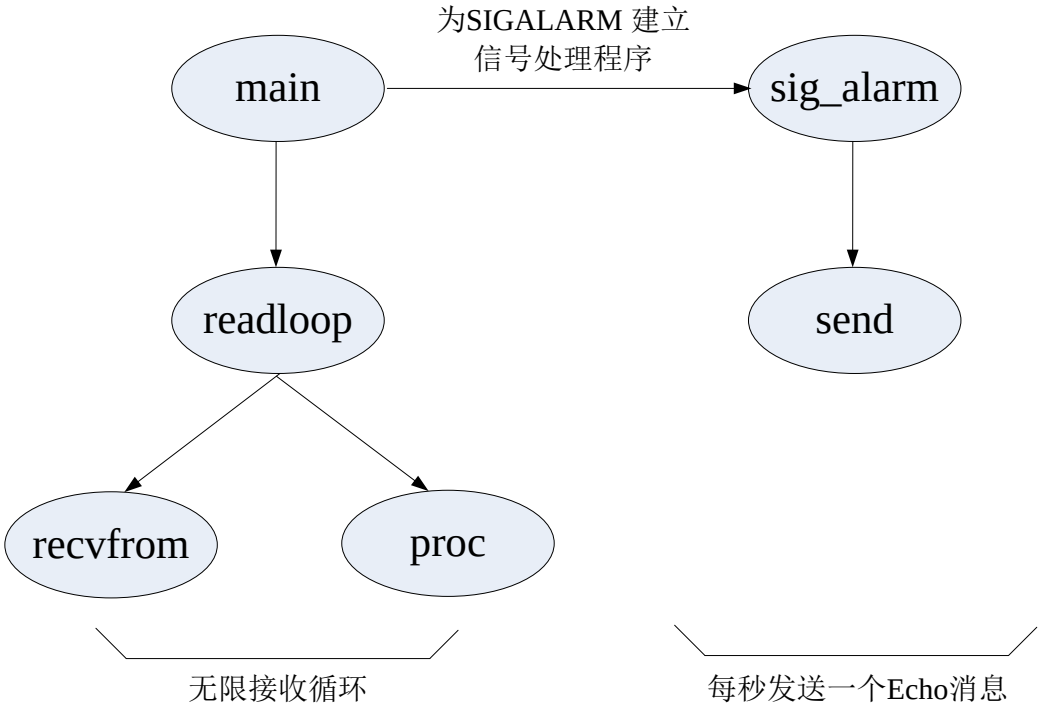
```
struct icmp {
    BYTE Type;        //ICMP 类型,本实验用 8: ECHO  0:ECHO  REPLY
    BYTE Code;        //本实验置零
    WORD Checksum;    //ICMP 包校验和,从 TYPE 开始,直到最后一位用户数据,如果为
                        字节数为奇数则补充一位
    WORD ID;          //用于匹配 ECHO 和 ECHO REPLY 包
    WORD Seq;         //用于标记 ECHO 报文顺序
    BYTE Data[1];     //用户数据
};
```

2.3 课程设计分析

1、 总体设计

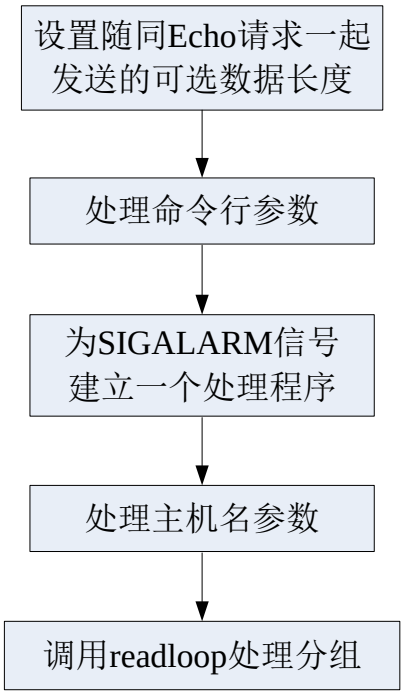
程序分为两大部分：一部分读取收到的所有消息，并输出 ICMP Echo replay 消息，另一部分每个一秒钟发送一个 Echo 消息。另一部分由 SIGALARM 信号每秒驱动一次。

2、 详细设计

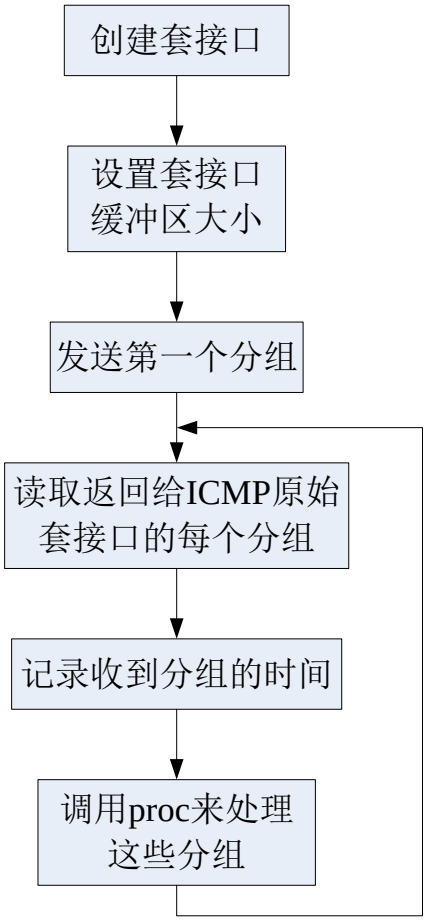


ping 程序函数概貌

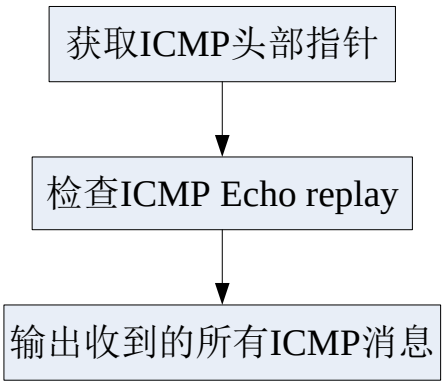
1）main 函数



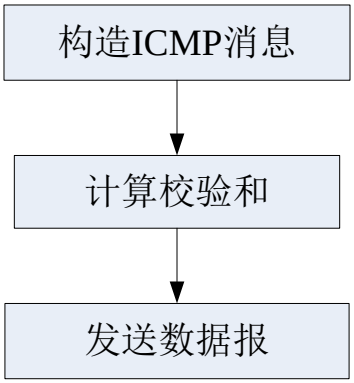
2）readloop 函数



3）proc 函数



4）send 函数


构造ICMP消息
计算校验和
发送数据报

实验三 应用层实验—简单聊天程序的设计与实现

3.1 课程设计目的

聊天程序是上网时经常使用的网络程序，通过它大家彼此之间可以交流信息。本实验要求完成一个聊天程序的简单设计与实现。通过聊天程序的设计与实现，可以达到如下目的：

- 掌握网络编程的知识和技能；
- 熟悉网络软件的开发过程，锻炼解决实际问题的能力。

3.2 课程设计内容

本实验要求设计并实现一个简单的聊天程序，包括服务器实现和客户端实现，具体内容和要求如下：

- 使用 MSN 或者 QQ，分析聊天程序的功能需求；
- 在给定的参考程序的基础上，参考 MSN 或者 QQ，对功能做出扩充，不局限于以下的范围：
  - 在客户端界面上显示所有联入聊天服务器的用户；
  - 支持两个用户之间的聊天；
  - 支持增加好友的功能，好友上线时如果该用户在线，则做出提醒；
  - 增加用户的个人信息修改、保存和查询；
  - 在聊天内容中支持中文；
  - 在聊天内容中支持图片等多媒体信息；
  - 支持在用户之间传输文件等附件；
  - 其他扩展。

3.3 相关知识

套接字编程的基本知识参见附录 B.

3.4 课程设计分析

聊天程序主要利用 Java 网络包中的服务器套接字和客户端套接字实现，其核心部分的实现思路如下：

- 服务器根据指定的端口创建服务器套接字，并在该端口侦听连接请求；
- 每一个新加入的客户端创建客户端套接字，与服务器进行连接；
- 每一个连接在服务器有一个连接池保持连接；
- 当其中的一个客户端发出聊天信息后，对应的连接的服务器程序在接收到该消息后，向其所维持的所有的客户连接广播该消息。

附录 A Linux 的安装与编程

1 安装 Linux

1. 将 Red Hat Linux9 的第一张光盘放到光驱中，重启计算机，出现以下界面后按“回车键”

2. 选“简体中文”，并按“Next（N）”按钮

3. 按“下一步（N）”按钮

4. 选“服务器”，并按“下一步（N）”按钮

5. 选“自动分区（A）”，并按“下一步（N）”按钮

6. 选“是”按钮

7. 选“删除系统内所有的 Linux 分区”，并按“下一步（N）”按钮

8. 选“是”按钮

9. 按“下一步（N）”按钮

10. 选 “下一步（N）”按钮

11. 按“编辑（E）”，在“编辑接口 eth0”窗口中，选“引导式激活（A）”，并输入机器的“IP 地址”和“子网掩码”，然后按“确定”按钮

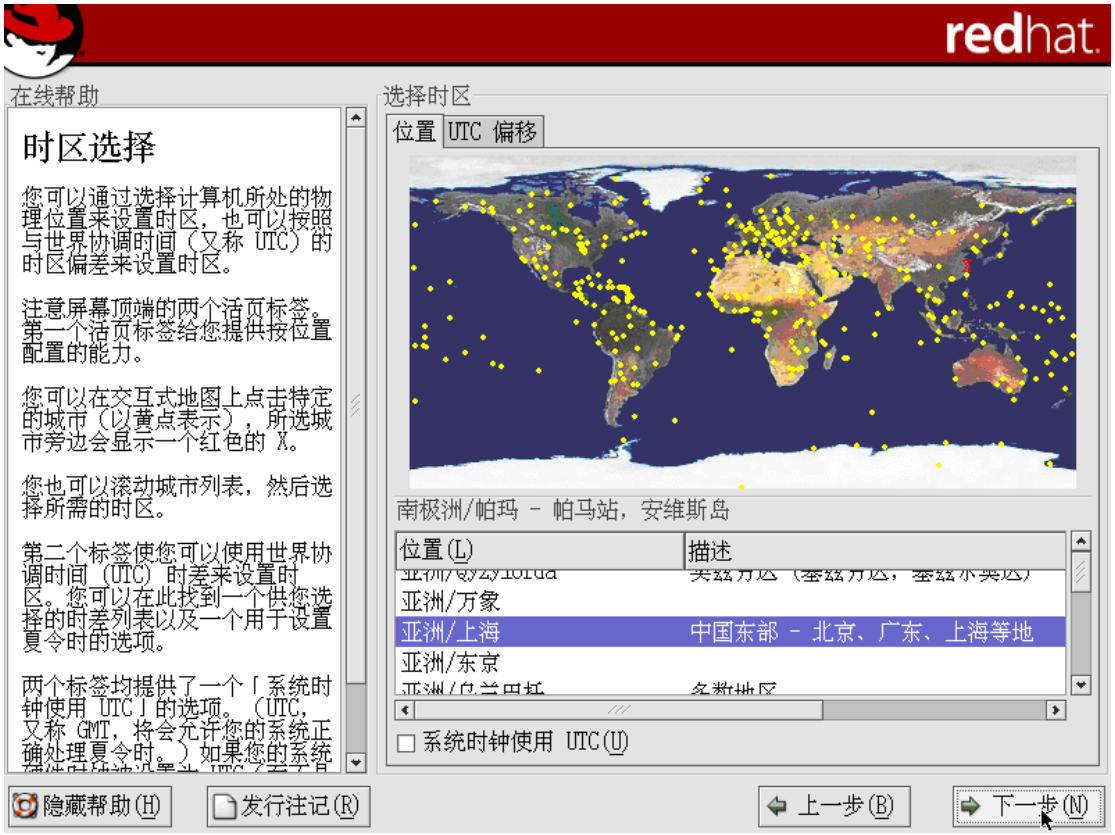
12. 在“主机名”中选“手工配置”，并在文本框中输入服务器名，在“其它设置”中输入该机器的网关和 DNS 信息，然后按“下一步（N）”按钮

13. 选“无防火墙（o）”，并按“下一步（N）”按钮

14. 在“选择系统默认语言”选“Chinese（P.R. of China）”，在“选择您想在该系统上安装

的其它语言（A）”选“English（Great Britain）”，并按“下一步（N）”按钮

15. 在“位置”中选“亚洲/上海”，并按“下一步（N）”按钮



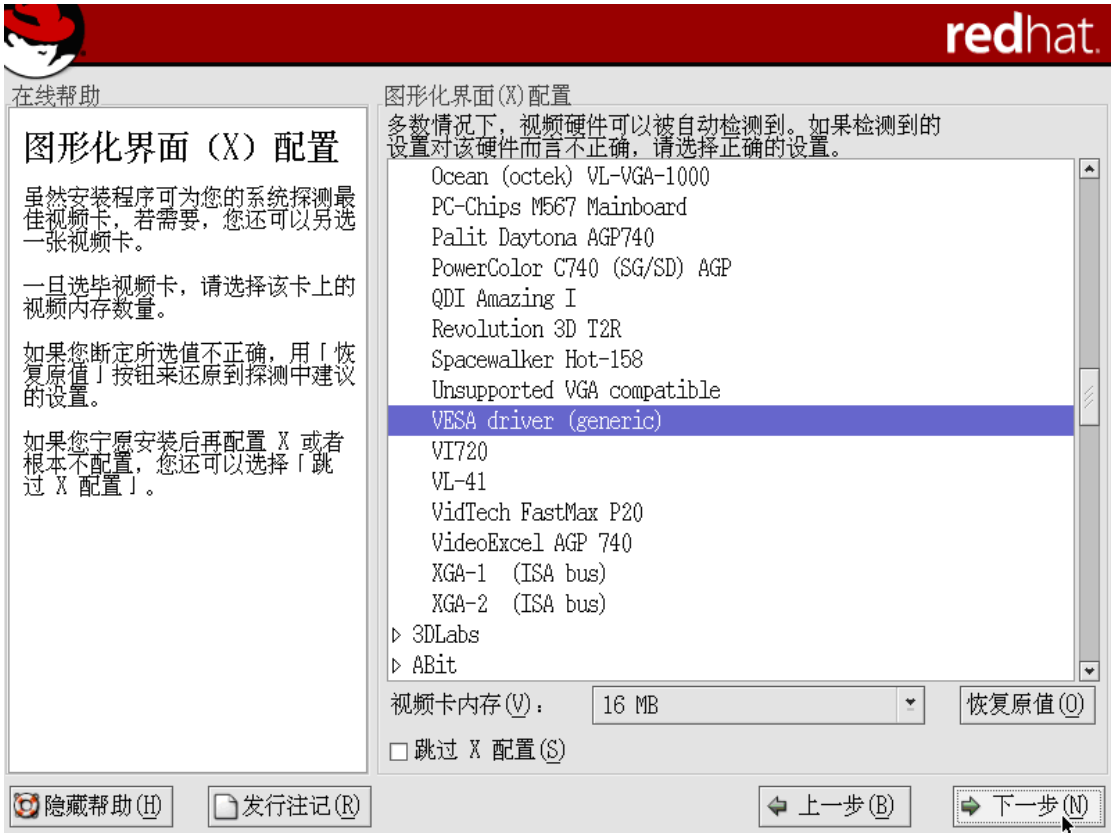
16. 输入 root 用户的口令，并按“下一步（N）”按钮

17. 在“选择软件包组”中，选“全部”，并按“下一步（N）”按钮

18. 按“下一步（N）”按钮

19. 选“否，我不想创建引导盘（D）”，并按“下一步（N）”按钮

20. 按“下一步（N）”按钮



21. 按“下一步（N）”按钮

22. 按“下一步（N）”按钮

23. 按“退出（E）”按钮，安装完成。

2 配置服务器网络部分

安装时没配置网络，或网络参数需要修改
假设机器 IP 地址为 172.21.5.72，网关为 172.21.5.254，DNS 为 172.21.0.1

2.1 配置服务器的 IP

修改/etc/sysconfig/network-scripts/ifcfg-eth0 文件，内容大致如下
DEVICE=eth0
BOOTPROTO=static
BROADCAST=172.21.5.255
IPADDR=172.21.5.72
NETMASK=255.255.255.0
NETWORK=172.21.5.0
ONBOOT=yes

2.2 配置服务器的网关

修改/etc/sysconfig/network 文件，内容大致如下
NETWORKING=yes
HOSTNAME=localhost
GATEWAY=172.21.5.254

2.3 配置服务器的 DNS

修改/etc/resolv.conf 文件，内容大致如下
nameserver 172.21.0.1

2.4 让新的配置起作用

重启机器
或运行以下命令
#service network restart

3 安装与配置 SecureCRT 软件（Windows 下）

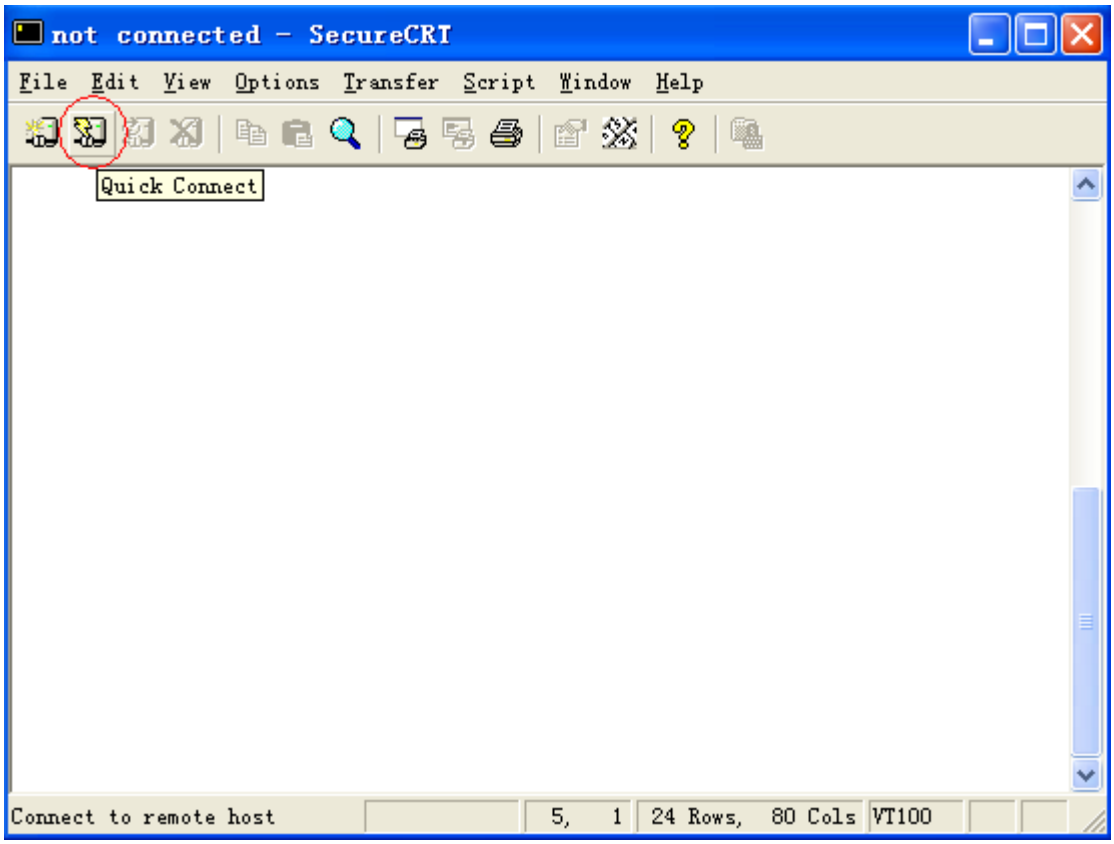
该客户软件允许计算机登录到远程的服务器并运行该服务器上的应用程序。

假设远程的服务器的地址为 172.21.6.229，用户名为 abc，密码为 abc123

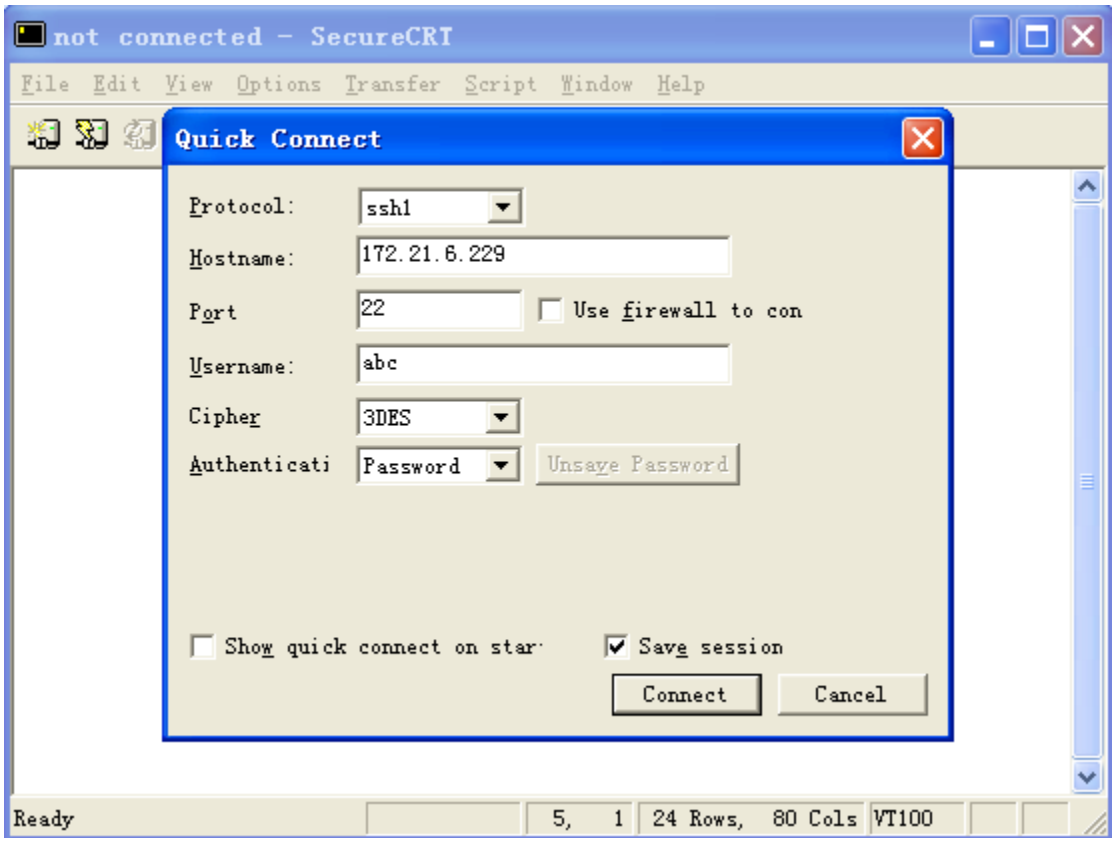
3.1 安装 SecureCRT 软件（略）

3.2 首次登录

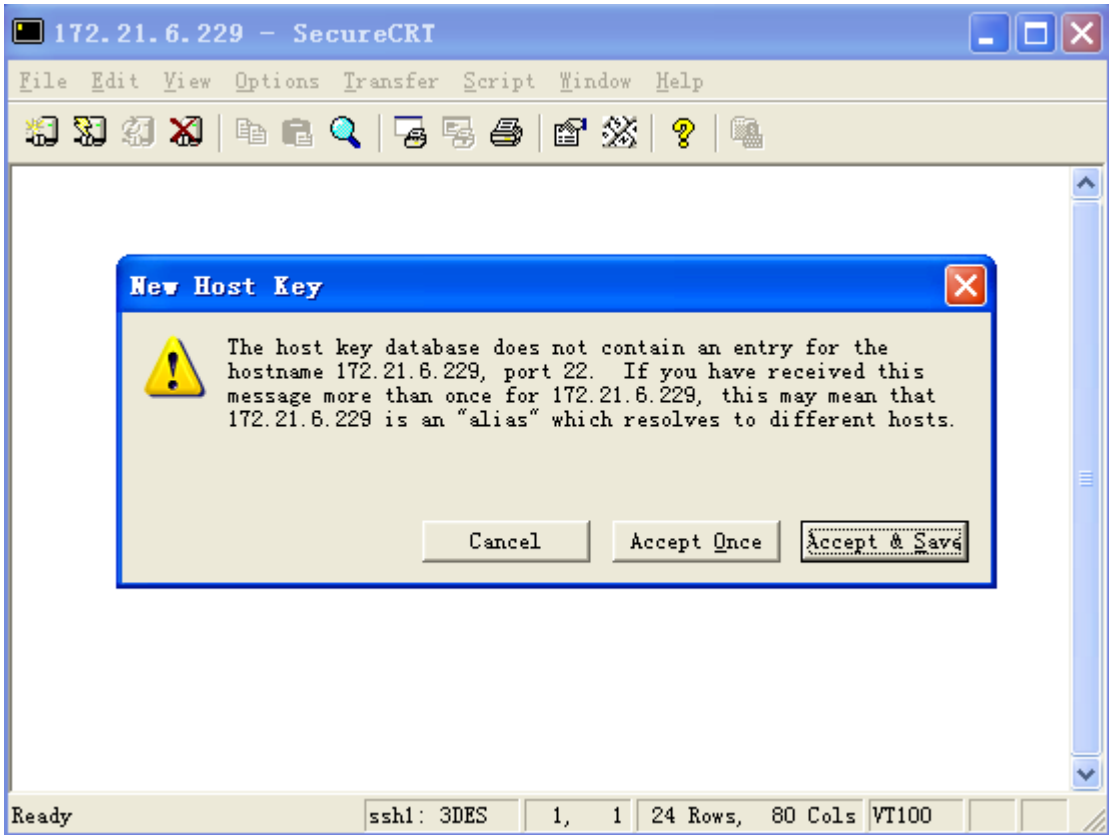
1. 按“Quick Connect”按钮



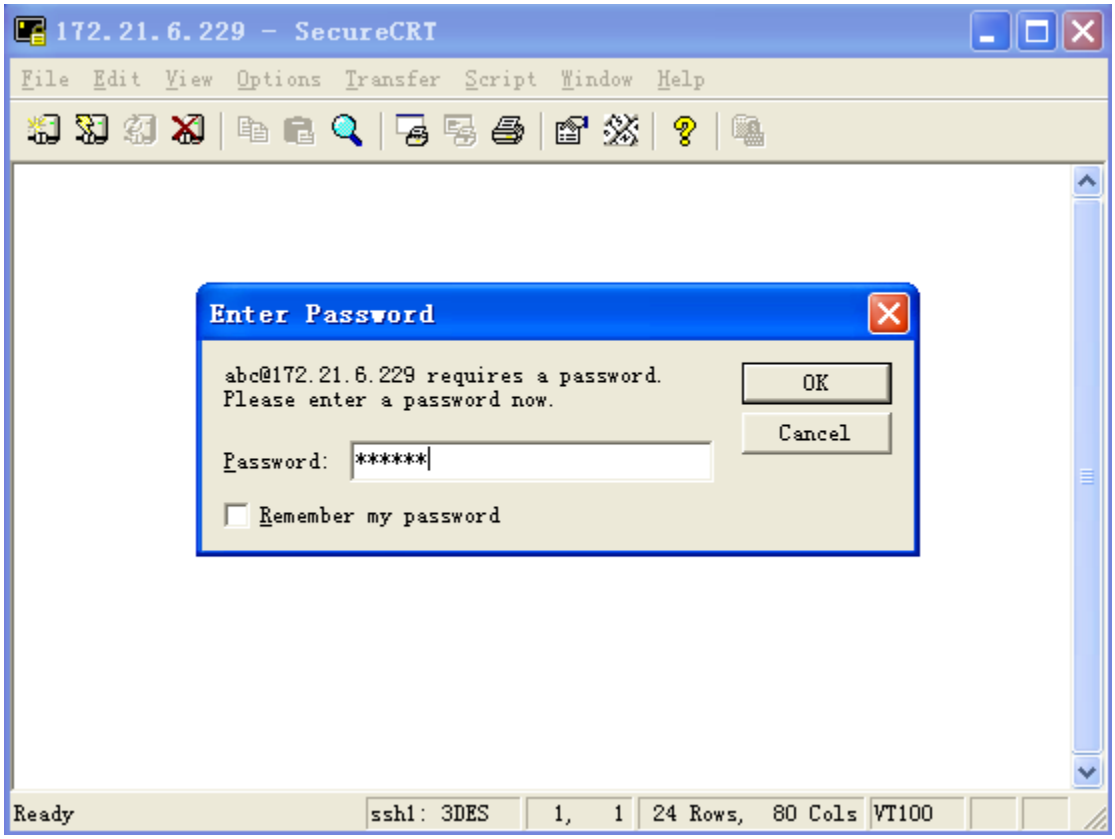
2. 在“hostname”中输入远程服务器的地址，并按“Connect”按钮



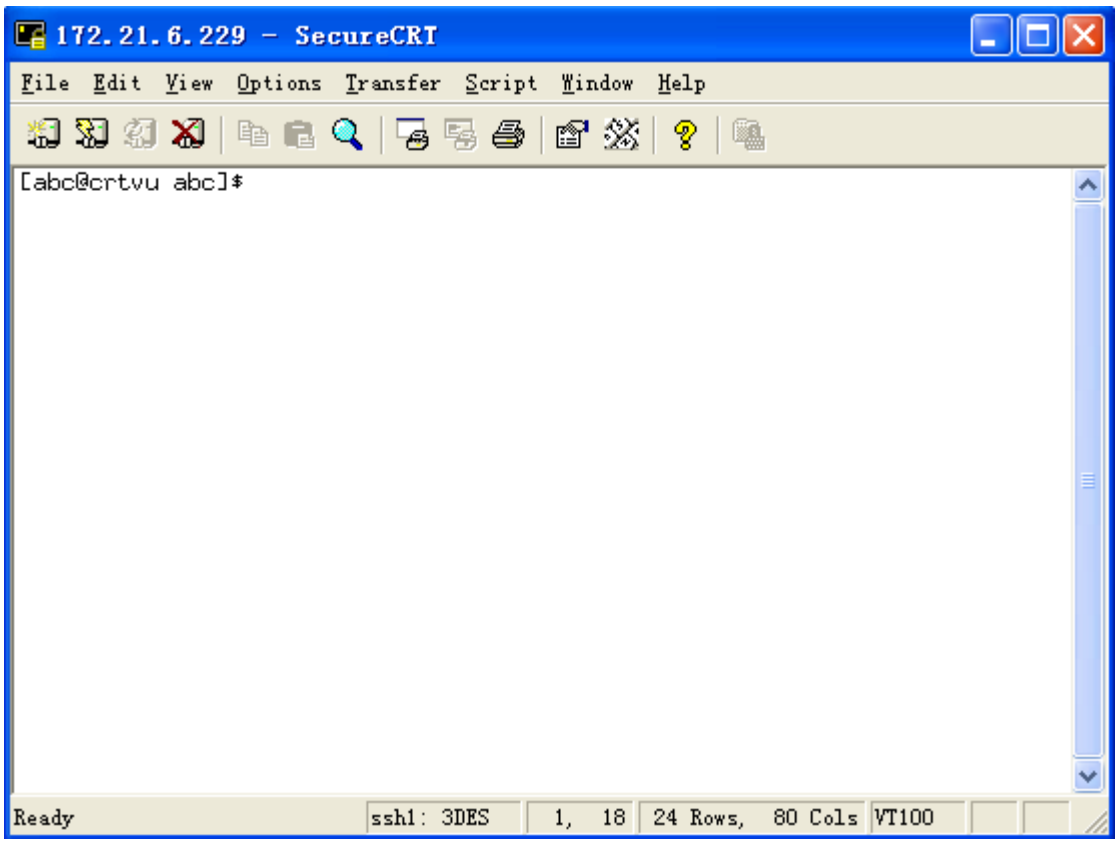
3. 选“Accept Once”



4. 输入用户“abc”的密码“abc123”，并按“OK”按钮

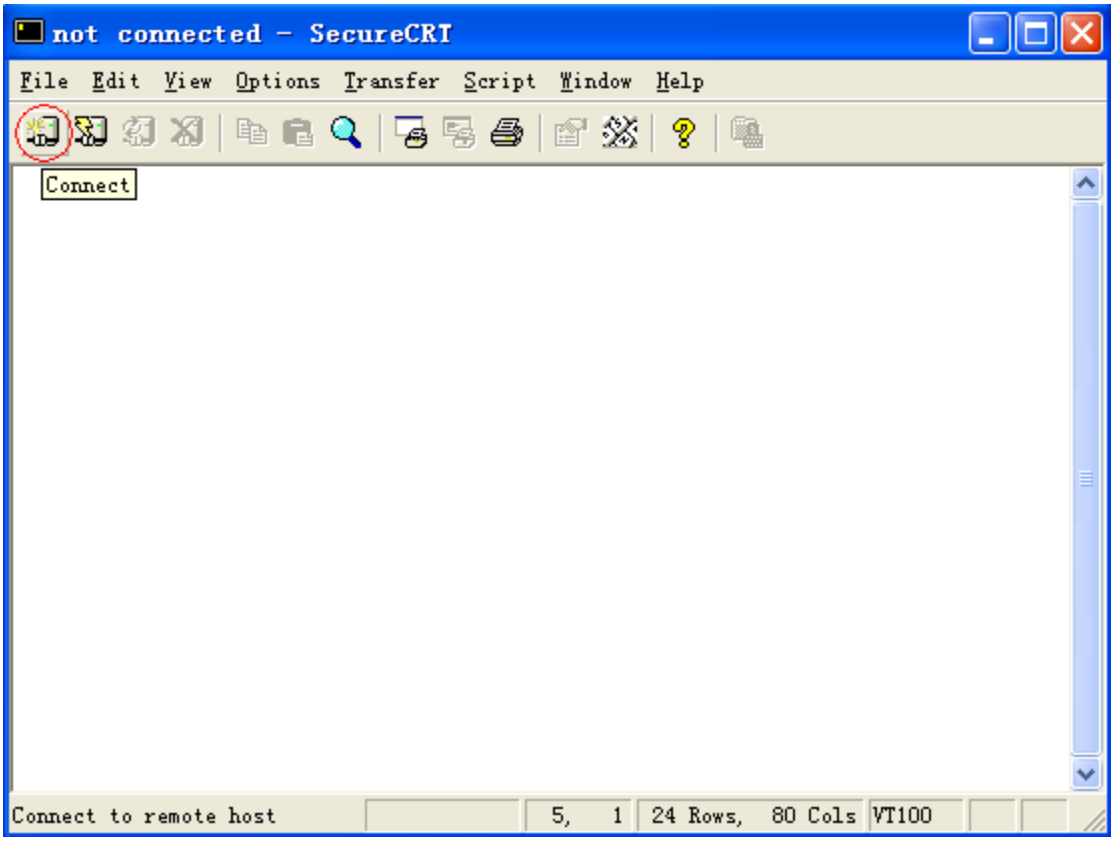


5. 在以下界面可以在远程服务器上运行该普通用户可以运行所有命令

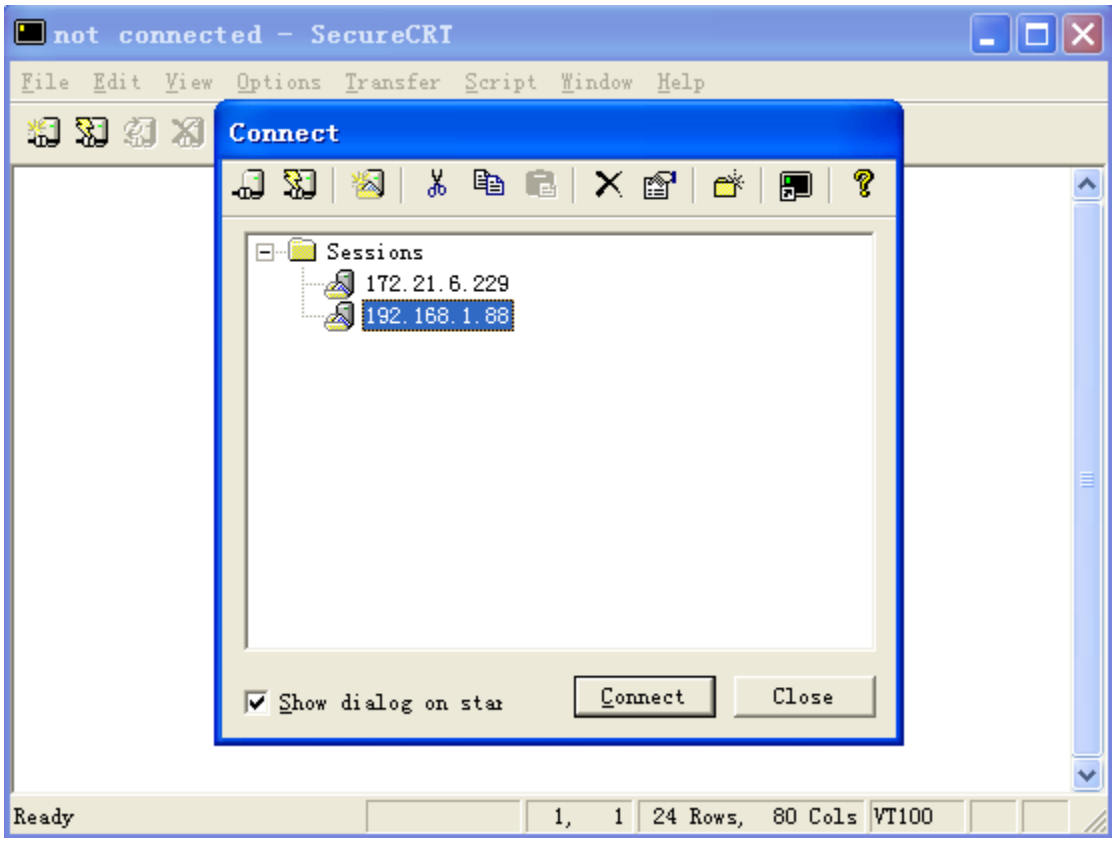


3.3 再次登录

1. 按“Connect”按钮



2. 选择要登录的服务器，并按“Connect”



3~5 同“首次登陆”的 3~5

4 编辑源程序

4.1 用 Linux 下的 Vi 或其他编辑器编辑源程序，或

4.2 先在 Windows 下编辑源程序，然后通过 Ftp 传到 Linux 下

Linux 下重启 Ftp 服务命令：
#service vsftpd restart
注：需要 root 用户权限。#提示符表示在 root 用户下

或设置 vsftpd 自动启动
chkconfig vsftpd on 或
运行 ntsysv 将 vsftpd 选上（用空格键选中或取消服务） 或
echo "/usr/local/sbin/vsftpd &" >>/etc/rc.local

5 编译连接源程序

命令格式：
$gcc –o 要生成的可执行文件名 源文件名
注：普通用户权限即可，$提示符表示在普通用户下

5.1 编译连接服务器程序

$gcc -o tcpserv01 tcpserv01.c
注：-o 后是要输出的可执行文件，tcpserv01.c 是服务器端的程序

5.2 编译连接客户端程序

$ gcc -o tcpcli01 tcpcli01.c
注：-o 后是要输出的可执行文件，tcpcli01.c 是客户端的程序

6 运行源程序

格式：
$./生成的可执行文件 [参数表]

6.1 运行服务器程序

$./tcpserv01 &
注：&表示以后台程序启动

6.2 运行客户端程序

$./tcpcli01 172.21.15.72
注：172.21.15.72 是服务器的地址

7 停止程序

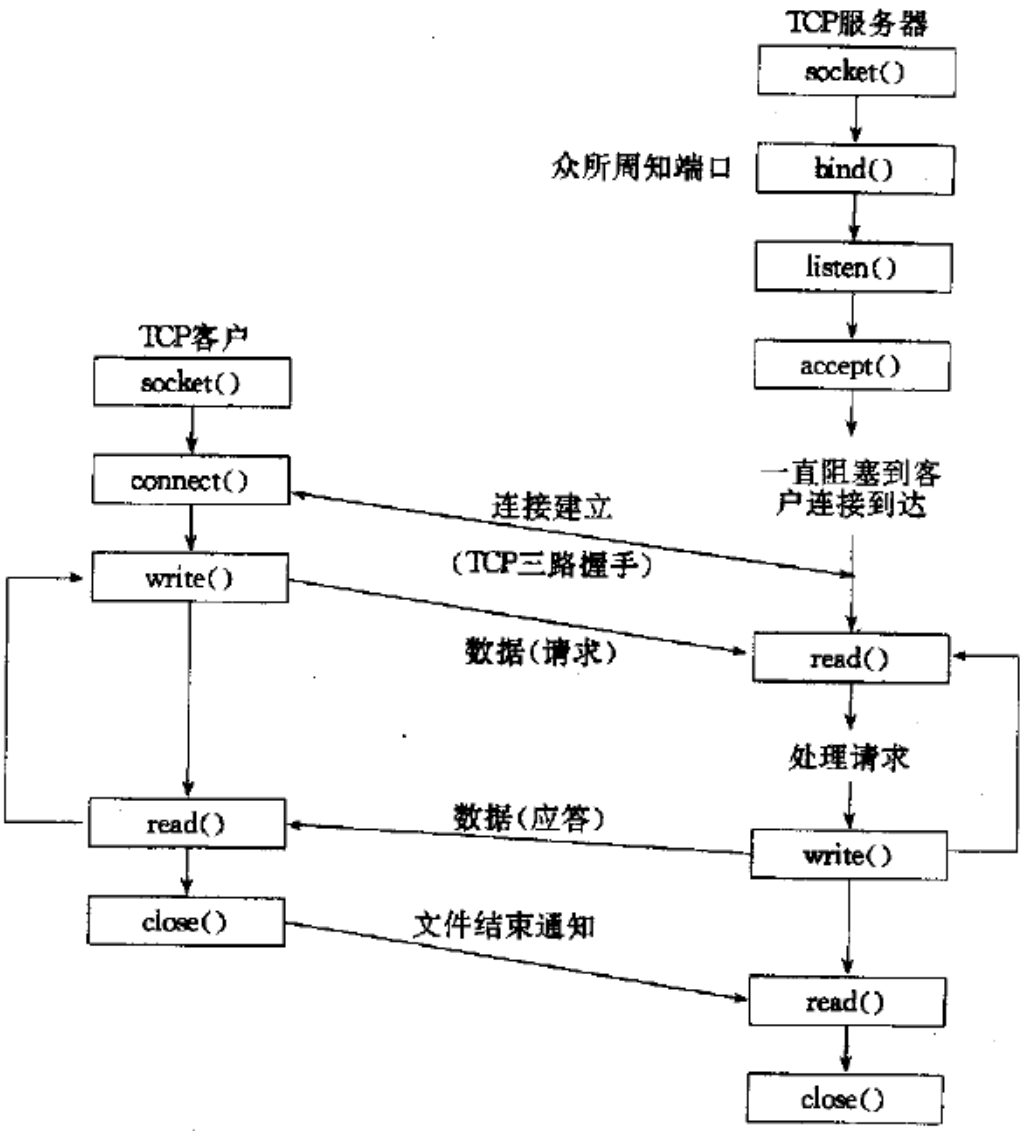
7.1 查看进程命令

$ps aux

7.2 杀死后台进程命令

$kill 进程号

8 基本 TCP 客户-服务器程序的套接口函数



9 启动/停止服务器 daytime 服务

9.1 启动服务器 daytime 服务

1. 修改/etc/xinetd.d/daytime 文件

将其中的“disable = yes”改为“disable = no”

2．启动 xinetd

#/etc/rc.d/init.d/xinetd restart

9.2 停止服务器 daytime 服务

将/etc/xinetd.d/daytime 文件中的“disable = no”改为“disable = yes”，并重启 xinetd 即可

10 附：源程序清单

10.1 取服务器日期时间客户端程序：daytimecli.c

```
#include        <netinet/in.h>
#include        <errno.h>
#include        <stdio.h>
#define MAXLINE                 4096

int
main(int argc, char **argv)
{
        int sockfd, n;
        char recvline[MAXLINE + 1];
        struct sockaddr_in servaddr;
        if (argc != 2) {
                printf("usage: daytimecli <IPaddress>\n");
                exit(1);
        }

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

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(13);          /* daytime server */
        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);
}
```

10.2 字符回显的服务器程序：tcpserv01.c

```
#include        <netinet/in.h>
#include        <errno.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        <sys/socket.h>
#define MAXLINE                 4096
#define LISTENQ                 1024      /* 2nd argument to listen() */
#define SERV_PORT               9877
#define SA      struct sockaddr

void str_echo(int);
ssize_t readline(int, void *, size_t);
static ssize_t my_read(int, char *);

int
main(int argc, char **argv)
{
    int    listenfd, connfd;
    pid_t    childpid;
    socklen_t    clilen;
```

```
        struct sockaddr_in    cliaddr,servaddr;

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

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(SERV_PORT);
        if (bind(listenfd, (SA *)&servaddr, sizeof(servaddr)) < 0) {
                printf("bind error.\n");
                exit(1);
        }

        if (listen(listenfd,LISTENQ) < 0) {
                printf("listen error.\n");
                exit(1);
        }

        for( ; ;) {
                clilen = sizeof(cliaddr);
                if ( (connfd = accept(listenfd, (SA *) &cliaddr, &clilen)) < 0) {
                        printf("accept error.\n");
                        exit(1);
                }

                if((childpid = fork()) == 0) {
                close(listenfd) ;
                        str_echo(connfd);
                        exit(0);
                }
                close(connfd);
        }
}

void
str_echo(int sockfd)
{
                ssize_t                n;
                char                     line[MAXLINE];

                for ( ; ; ) {
```

```
                if ( (n = readline(sockfd, line, MAXLINE)) == 0)
                        return;         /* connection closed by other end */

                write(sockfd, line, n);
        }
}
/*end str_echo*/

ssize_t
readline(int fd, void *vptr, size_t maxlen)
{
        int             n, rc;
        char    c, *ptr;

        ptr = vptr;
        for (n = 1; n < maxlen; n++) {
                if ( (rc = my_read(fd, &c)) == 1) {
                        *ptr++ = c;
                        if (c == '\n')
                                break;    /* newline is stored, like fgets() */
                } else if (rc == 0) {
                        if (n == 1)
                                return(0);          /* EOF, no data read */
                        else
                                break;              /* EOF, some data was read */
                } else
                        return(-1);             /* error, errno set by read() */
        }

        *ptr = 0;           /* null terminate like fgets() */
        return(n);
}
/* end readline */

static ssize_t
my_read(int fd, char *ptr)
{
        static int          read_cnt = 0;
        static char         *read_ptr;
        static char         read_buf[MAXLINE];

        if (read_cnt <= 0) {
again:
                if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
```

```
                        if (errno == EINTR)
                                goto again;
                        return(-1);
                } else if (read_cnt == 0)
                        return(0);
                read_ptr = read_buf;
        }

        read_cnt--;
        *ptr = *read_ptr++;
        return(1);
}
/*end ssize_t*/
```

10.3 字符回显的客户端程序：tcpcli01.c

```
#include        <netinet/in.h>
#include        <errno.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        <sys/socket.h>
#define MAXLINE                 4096
#define LISTENQ                 1024    /* 2nd argument to listen() */
#define SERV_PORT               9877
#define SA      struct sockaddr

void str_cli(FILE *,int);
ssize_t readline(int, void *, size_t);
static ssize_t my_read(int, char *);

int
main(int argc, char **argv)
{
    int sockfd;
    struct sockaddr_in    servaddr;

    if (argc != 2) {
    printf("usage:tcpcli01 <IPaddress>");
        exit(1);
    }
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket error.\n");
```

```
        exit(1);
    }

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    if (connect(sockfd, (SA *)&servaddr, sizeof(servaddr)) < 0) {
        printf("connect error.\n");
        exit(1);
    }

    str_cli(stdin, sockfd);
    exit(0);
}

void
str_cli(FILE *fp,int sockfd)
{
        char sendline[MAXLINE], recvline[MAXLINE];

        while (fgets(sendline,MAXLINE, fp) != NULL) {
                write(sockfd, sendline, strlen(sendline));
        if (readline(sockfd, recvline, MAXLINE) == 0) {
              printf("str_cli:server terminated prematurely.\n");
                        exit(1);
                }
        fputs(recvline, stdout);
        }
}
/*end str_cli */

ssize_t
readline(int fd, void *vptr, size_t maxlen)
{
        int                     n, rc;
        char    c, *ptr;

        ptr = vptr;
        for (n = 1; n < maxlen; n++) {
                if ( (rc = my_read(fd, &c)) == 1) {
                        *ptr++ = c;
                        if (c == '\n')
                                break;    /* newline is stored, like fgets() */
```

```
                } else if (rc == 0) {
                        if (n == 1)
                                return(0);          /* EOF, no data read */
                        else
                                break;                /* EOF, some data was read */
                } else
                        return(-1);                 /* error, errno set by read() */
        }

        *ptr = 0;           /* null terminate like fgets() */
        return(n);
}
/* end readline */

static ssize_t
my_read(int fd, char *ptr)
{
        static int          read_cnt = 0;
        static char         *read_ptr;
        static char         read_buf[MAXLINE];

        if (read_cnt <= 0) {
again:
                if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
                        if (errno == EINTR)
                                goto again;
                        return(-1);
                } else if (read_cnt == 0)
                        return(0);
                read_ptr = read_buf;
        }

        read_cnt--;
        *ptr = *read_ptr++;
        return(1);
}
/*end ssize_t*/
```

附录 B Java 网络编程基础知识

1 Internet 地址操作

网络中的每一台计算机都有一个IP地址来唯一的标识。这些IP地址有4个字节长的IPv4地址，也有 16 个地址长的 IPv6 地址。为了方便记忆这些 IP 地址，人们使用了域名，域名服务使得域名和 IP 地址关联起来。

Java.net.InetAddress 类是对 IP 地址（包括 IPv4 和 IPv6）的抽象和封装，它是后续网络类学习和网络编程的基础。

1. 类的声明

public class InetAddress implements Serializable

该类继承自 Object 对象，并实现了 Serializable 接口。

2. 对象的创建

InetAddress 类提供了四个静态的构造方法，这三个方法如下：

```
public static InetAddress getByAddress(String host, byte[] addr)
        throws UnknownHostException
```

根据给出的IP地址和主机名创建一个InetAddress对象，不需要访问域名服务获得信息。

```
public static InetAddress getByName(String host)
        throws UnknownHostException
```

给出主机的名称，决定主机的IP地址。

```
public static InetAddress getByAddress(byte[] addr)
        throws UnknownHostException
```

给出原始的IP地址，返回一个InetAddress对象。

这三个方法在执行的过程中，其中后面的两个需要连接域名服务器获得相关的信息，并给类中的属性进行赋值操作（如类中的属性String hostName，int address等），访问域名服务器的操作封装在类中的一个私有方法（private static Object getAddressFromNameService (String host)）中。

InetAddress.getByName(String hostName)这个静态构造方法以主机名为参数，使用域名服务查找该主机的IP地址。调用的代码如下：

```
InetAddress addr = InetAddress.getByName("www.sina.com.cn");//在类中已经导入了
                                                    //java.net.InetAddress，也就是类中有这样的代码：
                                                    //import java.net.InetAddress
```

在调用该方法的时候，如果在域名服务中没有找到主机，会抛出UnknownHostException的异常，需要进行捕获处理：

```
...
try
{
    InetAddress addr = InetAddress.getByName("www.sina.com.cn");//根据机器名创建 InetAddress 对象
    ...//进行其他处理操作
}
catch(UnknownHostException ex)//捕获异常
```

```
{
    ex.printStackTrace();//打印异常抛出栈
    ...//进行后续处理
}
...//后续处理
```

InetAddress.getByAddress(String host, byte[] addr)方法在程序员知道IP地址和域名的准确信息时使用，在该方法的执行过程中不需要访问域名服务获得信息，所以在性能上会有一定的提高。

有的时候一个域名会对应多个IP地址，InetAddress.getAllByName(String hostName)会返回一个InetAddress的数组。用法与InetAddress.getByName(String hostName)类似，同样在调用的时候需要捕获UnknownHostException异常。

此外，InetAddress类还提供了一个查找本机IP地址的构造函数，public static InetAddress getLocalHost() throws UnknownHostException，该函数返回一个包含本机IP地址信息的InetAddress对象。

3. 属性读取方法

InetAddress包含了主机名（String hostName）和IP地址（int address）两个关键属性，这两个属性在类创建的过程中完成赋值，并提供了四个属性读取的方法来获得这些属性。

public byte[] getAddress()方法会以网络字节顺序的字节数组的方式获得IP地址，使用该方法的原因就是用来进行IP地址的属性判断，如判断一个IP地址是IPv4地址还是一个IPv6地址，这可以通过字节数组的长度来进行判断，如果长度为4则是IPv4的地址，如果长度为16则为IPv6的地址。通常的用法如下：

```
...//前面的处理，如创建 InetAddress 对象
byte[] addr = inetAddr.getAddress();//inetAddr 是一个 InetAddress 类型的对象
if (addr.length == 4)
    ...//后续处理
```

public String getCanonicalHostName()方法用来返回规范化的域名。通常的用法如下：

```
...//前面的处理，如创建 InetAddress 对象
String canHostName = InetAddr.getCanonicalHostName();//InetAddr 是一个//InetAddress 对象
System.out.println(canHostName);//进行后续处理
...
```

public String getHostAddress()方法返回文本格式的IP地址，如33.23.124.54，用法与上面的方法类似，不再详细介绍。

public String getHostName()方法返回字符串类型的主机名，如果该主机没有主机名，则返回文本格式的IP地址，用法与上面的方法类似，不再详细介绍。

4. 地址类型判断方法

不同的地址具有不同的特征，有的是本地地址，有的是组播地址。Java提供了大量的方法进行地址类型的判断，方便用户针对不同的地址类型进行后续处理工作，如表1-1所示：

表 1-1　　IP 地址类型判断方法

方法声明	说明
public boolean isMulticastAddress()	判断InetAddress是不是组播地址
public boolean isAnyLocalAddress()	判断InetAddress是不是通配地址

public boolean isLoopbackAddress()	判断InetAddress是不是回路地址
public boolean isLinkLocalAddress()	判断InetAddress是不是本地链接地址
public boolean isSiteLocalAddress()	判断InetAddress是不是本地网站地址
public boolean isMCGlobal()	判断InetAddress是不是全球组播地址
public boolean isMCNodeLocal()	判断InetAddress是不是本地接口组播地址
public boolean isMCLinkLocal()	判断InetAddress是不是子网范围组播地址
public boolean isMCSiteLocal()	判断InetAddress是不是网站范围组播地址
public boolean isMCOrgLocal()	判断InetAddress是不是组织范围组播地址
public boolean isReachable(int timeout) throws IOException	判断InetAddress是不是在规定时间内可达

上面的地址类型判断和测试IP地址的可达性主要是针对一些比较专业的应用，开发普通网络应用的程序员不必深究。

2 URL 和 URLConnection

URL(Uniform Resource Locator：统一资源定位符)用来标识 Internet 上资源的位置，典型的 URL 如 http://www.gridsphere.org/gridsphere/gridsphere?cid=grid，其中 http 表示所用的协议，www.gridsphere.org 指明了 URL 的主机名，而 gridsphere/gridsphere 指明了服务器上的路径和文件名，cid=grid 指明了向服务器提供的查询参数。本节主要介绍了 URL 相关的 java.net.URL 类和 java.net.URLConnection 类。

1. URL 类

URL 类是对 URL 的抽象，URL 类根据 URL 的各个组成部分设置了不同的属性，如协议、主机名、端口、路径、文件名等等，这些属性的设置方法只对包内成员可见，但是具有公共的属性读取方法来获得它们的值。

（1）类的声明

```
public final class URL implements java.io.Serializable
```

该类继承子 java.lang.Object 对象，并实现了 java.io.Serializable 接口。它是一个 final 类型的类，也就是不能基于 URL 类派生新的子类。

（2）对象的创建

URL 类提供了六个普通的构造函数，用于创建一个 URL 对象。这六个构造函数的区别在于输入的参数不同，也就是创建 URL 对象时拥有的信息不同。在创建对象时如果创建没有提供支持的协议的时候，会抛出不良 URL 的异常（MalformedURLException）。这六个构造函数的列表和说明如表 2-1 所示。

表 2-1　　URL 对象的构造方法

方法声明	说明
public URL(String protocol,	根据给定的协议、主机、端口、文件和流处理

String host, int port, String file, URLStreamHandler handler) throws MalformedURLException	器创建URL对象。host可以是主机名或者文本类型的IP地址。如果port传入的数值是-1，则表示采用该支持协议的默认端口。每一个URL对象在工作时都有一个URL的流处理器，这里可以制定一个定制的流处理器，而不是采用默认值。给出的参数不合法会抛出MalformedURLException。
public URL(String spec) throws MalformedURLException	给定一个字符串表示的URL构造一个URL对象，这是创建URL对象的最简单的方式。给出的参数不合法会抛出MalformedURLException。
public URL(URL context, String spec) throws MalformedURLException	在上下文URL和相对URL的基础上构造一个新的URL对象。给出的参数不合法会抛出MalformedURLException。
public URL(URL context, String spec, URLStreamHandler handler) throws MalformedURLException	在上下文URL和相对URL的基础上构造一个新的URL对象,并给定定制的流处理器。给出的参数不合法会抛出MalformedURLException
public URL(String protocol, String host, String file) throws MalformedURLException	根据给定的协议、主机和文件来创建URL对象。host可以是主机名或者文本类型的IP地址。给出的参数不合法会抛出MalformedURLException。
public URL(String protocol, String host, int port, String file) throws MalformedURLException	根据给定的协议、主机、端口和文件来创建URL对象。host可以是主机名或者文本类型的IP地址。如果port传入的数值是-1，则表示采用该支持协议的默认端口。给出的参数不合法会抛出MalformedURLException。

通常的用法如下：

```
...
try
{
    ...
    //创建新的 URL 对象
    URL url = new URL(“http”, “http://java.sun.com”, -1, “/j2se/1.5.0/docs/api/”);
    ...
}
catch(MalformedURLException)//捕获异常
{
    ...//进行异常处理
}
...//后续处理
```

（3）属性的读取方法

我们已经知道一个典型的 URL 是由协议、主机名、路径、端口等若干个部分组成的。URL 类把这些 URL 的若干部分实现为类的属性，并提供了读取这些属性的方法，如表 2-2 所示。

表 2-2　　URL 类的属性读取方法

方法声明	说明
public String getQuery()	获得URL的查询字符串部分，如果URL不包含一个查询字符串，则返回null。
public String getPath()	获得URL的路径部分，如果URL中不包含一个路径，则返回null。与getFile()不同的是，它只包含路径，不包含查询字符串。
public String getAuthority()	获得URL的授权部分，通常授权信息包括用户名和口令等用户信息、主机和端口。
public int getPort()	获得URL的端口部分，如果端口没有设置，则返回-1。
public int getDefaultPort()	获得与URL中协议相关的默认端口，如果没有定义默认端口则返回-1。
public String getProtocol()	获得URL中的协议名称部分。
public String getHost()	获得URL中的主机名部分。
public String getFile()	获得URL中的文件名部分，如果URL中没有包含查询字符串，则该方法与getPath()的返回值是一样的；如果URL中包含了查询字符串，则该方法除了getPath()的内容以外，还包含查询字符串。
public String getRef()	获得URL中的片断标识符部分，如果URL中没有片断标识符，则返回null。

通常的用法如下：

```
...
URL url = null;
try
{
    //创建新的 URL 对象
url = new URL(“http://news.sina.com.cn/c/2006-09-26/183110114488s.shtml”);
}
catch(MalformedURLException ex)
{
    …//进行异常处理
}
String host = url.getHost();
…
```

（4）判断方法

public boolean sameFile(URL other)用来比较两个 URL 指向的是不是同一个文件，通常用法如下：

```
...
URL url = null;
try
{
```

```
    //创建新的 URL 对象
url = new URL("http://news.sina.com.cn/c/2006-09-26/183110114488s.shtml");
}
catch(MalformedURLException ex)
{
    …//进行异常处理
}
//判断两个 URL 是不是相同
try
{
     //因为这里也创建了一个新的 URL，所以应该捕获 MalformedURLException
if(url.sameFile(new URL("http://news.sina.com.cn/c/2006-09-26/183110114489s.shtml")))
     …//进行后续处理
}
catch(MalformedURLException ex)
{
    …//进行异常处理
}
…
```

（5）从指定的 URL 中获得数据的方法

仅仅封装 URL 并不是 URL 类的目的，建立 URL 类的目的是从 URL 指向的位置获得数据。URL 提供了 5 个从 URL 指向的位置获取数据的方法。

public URLConnection openConnection() throws IOException

public URLConnection openConnection(Proxy proxy) throws IOException

前面的这两个方法都是返回一个 java.net.URLConnection 对象，关于 URLConnection 对象的使用在后文中会有详细介绍。

public final InputStream openStream() throws IOException

获得该 URL 的一个连接，并返回一个 InputStream 对象从该连接中获得数据。该方法就是 openConnection().getInputStream()的缩写。它的使用方法在 URLConnection 对象时有详细介绍。

public final Object getContent() throws IOException

该方法获得 URL 指向资源的内容，并尝试把它转换一个对象，是 openConnnection().getContent()的缩写。该方法的通常用法如下：

```
...
try
{
     //创建一个新的 URL 对象
    URL url = new URL("http://news.sina.com.cn/2006-09-26/183110114489s.shtml");
try
{
    Object obj = url.getContent();//获得 URL 的内容
         ...//后续处理
}
catch(IOException ioex)
```

```
{
    ...//异常处理
}
}
catch(MalformedURLException ex)
{
    …//进行异常处理
}
…
```

public final Object getContent(Class[] classes) throws IOException

在 getContent()方法的使用过程中，我们会发现因为不知道获得的对象的类别，所以在实际的应用过程中很难进行有针对性的处理。该方法尝试以类数组中的类对象来完成内容的转换，这样就比较有针对性。

★ 请注意 ★

在 Java 中，网络数据的传输都是建立在流（Stream）的基础上，所以发送、接受数据和读、写文件没有分别，使用的都是 Java 的 I/O 机制，这也就是在 URL 对象获得数据的时候抛出都是 IOException 的原因。

2. URLConnection 类

URLConnection 是一个抽象类，它是表示应用程序和一个 URL 之间的通信连接的所有类的基类。该类的实例既可以向 URL 所指向的资源写入数据，也可以从资源读取数据。对于与服务器的交互，URLConnection 提供了更多的控制。URLConnection 提供了更加易于使用、更加高级的网络连接，但是它更加贴近于 HTTP 协议，非常适用于连接 HTTP 服务器。

总的说来，创建一个 URLConnection 可以分为以下几个步骤：

- 通过访问 URL 对象的 openConnection()方法创建 connection 对象；
- 配置启动参数和总的请求属性；
- 使用 connect()方法创建与远程对象的实际连接；
- 创建完连接以后，远程对象就可以访问了，访问的内容包括首部字段和内容。

（1）对象的创建

protected URLConnection(URL url)

URLConnection 类只有一个受保护的构造函数，不能基于该构造函数创建对象，只能通过别的类的 openConnection()方法来创建，本书重点介绍基于前文介绍的 URL 类创建 URLConnection 的方法。创建 URLConnection 对象的通常用法如下：

```
...
try
{
    URL url = new URL(“http://news.sina.com.cn”);//创建 URL 对象
    URLConnection uConn = url.openConnection();//创建 URLConnection 对象
    ...//后续处理
}
catch(MalformedURLException ex)
{
    ...//进行异常处理
}
...
```

URLConnection 作为一个抽象类，它具有一个抽象方法 public abstract void connect()

throws IOException，该方法在派生子类的时候必须得到实现，它负责建立与服务器的连接。

（2）获得头部信息

一个客户与服务器的交互协议可能设置一个头部，头部一般包括一些控制信息。例如 HTTP 的头部包括所请求文档的内容类型、文档长度、编码字符集、时间等信息。URLConnection 提供了读取头部信息的大量的函数，如表 2-3 所示。

表 2-3　　头部信息读取方法

方法声明	说明
public int getContentLength()	获得content-length头部信息，也就是URL所指向的资源的内容的长度，如果长度不可知，则返回-1。
public String getContentType()	获得content-type头部信息，也就是URL所指向的资源的内容类型，如果未知，则返回null。
public String getContentEncoding()	获得content-encoding头部信息，也就是URL所指向的资源的内容编码，如果未知，则返回null。
public long getExpiration()	获得expire头部信息，也就是URL指向的资源的过期日期，如果未知则返回0。
public long getDate()	获得date头部信息，也就是URL指向的资源的发送日期，如果未知则返回0。
public long getLastModified()	获得last-modified头部信息，也就是URL指向的资源的最后修改日期，如果未知则返回0。
public String getHeaderField(String name)	获得指定名称的头部字段，
public int getHeaderFieldInt(String name, int Default)	获得指定名称的头部信息，并尝试把它转换成为一个整数。
public long getHeaderFieldDate(String name, long Default)	获得指定名称的头部信息，并尝试把它转换成为一个日期。
public String getHeaderFieldKey(int n)	获得第n个头部字段的键值。
public String getHeaderField(int n)	获得第n个头部字段的值。

（3）配置启动参数和请求属性

URLConnection 提供了七个属性进行数据传输过程的控制：

protected URL url;

protected boolean doInput = true;

protected boolean doOutput = false;

protected boolean allowUserInteraction = defaultAllowUserInteraction;

protected boolean useCaches = defaultUseCaches;

protected long ifModifiedSince = 0;

protected boolean connected = false;

并提供了设置这些属性和读取这些属性的方法，如 public void setDoInput(boolean doInput)和 public boolean getDoInput()，这些方法一般在连接之前完成设置。

（4）发送和接受数据

读取数据的流程如下：

- 调用 URL 对象的 openConnection()方法，获得 URLConnection 对象；
- 调用 URLConnection 对象的 getInputStream()方法获得一个输入流；
- 使用输入流操作数据

通常的用法如下：

```
...
try
{
    URLConnection uConn = url.openConnection();
    InputStream is = uConn.getInputStream()
    ...//后续处理
}
catch(IOException ex)
{
    ...//进行异常处理
}
...//后续处理
```

发送数据的流程与写入数据的流程基本类似，不同的是，调用的是 URLConnection 对象的获得输出流的方法

3 套接字编程

套接字（Socket）是一种软件形式的抽象，用于表达两台机器间一个连接的两端。针对一个连接，每台机器上都有一个套接字。在 Java 中，我们创建一个套接字，用它建立与其他机器的连接，得到的结果是一个 InputStream 以及 OutputStream，从而将连接作为一个 IO 流对象对待。有两个基于流的套接字类：java.net.ServerSocket 和 java.net.Socket，分别用于服务器端和客户端建立套接字连接。

1. 客户端套接字

使用客户端套接字进行编程的步骤一般包括：

- 构造一个新的套接字对象；
- 使用新创建的套接字对象连接远程服务器；
- 连接成功，从套接字获得输入流和输出流，进而开始发送和接受数据；
- 数据传输完毕，关闭连接。

下面我们开始针对上面的每一个步骤进行说明。

（1）新建一个套接字对象

客户端套接字总共提供了 9 个构造函数，其中一个是受保护类型（protected），只能用于派生类提供自己的套接字实现时使用；两个用于构造 UDP 传输的构造函数已经废弃，采用新的数据报套接字（DatagramSocket）替代；还有两个用于创建未连接的客户端套接字对象的构造函数，在基于客户端套接字类派生新的子类实现一种特殊的套接字类的时候使用；剩余的 4 个构造函数经常用于创建套接字对象，这里给予详细介绍。

- public Socket(String host, int port) throws UnknownHostException, IOException

这个构造函数用于创建一个指定主机上的指定端口的套接字，并且尝试连接远程主机。host 是用字符串表示的主机名，port 是 0～65535 之间的整数值，表示端口。因为在该构造函数不仅仅是创建套接字对象，还尝试连接远程的主机，所以当给定的主机名无法解析时会抛出 UnknownHostException 异常；其他原因导致无法连接时会抛出 IOException 异常。通常

的用法如下：

```
...
try
{
    //创建套接字对象
    Socket s = new Socket("news.sina.com.cn", 80);
    //进行后续处理
    ...
}
catch(UnknownHostException uhex)
{
    //进行异常处理
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
}
...
```

- public Socket(InetAddress address, int port) throws IOException

这个构造函数与上一个构造函数类似，同样是构造一个指定主机上的指定端口的套接字，并且尝试连接远程主机，不同的是传入的参数不同，它使用 InetAddress 对象指定主机，而不是使用字符串形式的主机名。如果在连接主机的过程中出现无法连接主机的错误，会抛出 IOException 异常。通常的用法如下：

```
...
try
{
    //创建 InetAddress 对象
    InetAddress ia = InetAddress.getByName("news.sina.com.cn");
    //创建套接字对象
    Socket s = new Socket(ia, 80);
    //后续处理
    ...
}
catch(UnknownHostException uhex)
{
    //进行异常处理
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
```

```
}
...
```

- public Socket(String host, int port, InetAddress localAddr, int localPort) throws IOException

这个构造函数同样是创建一个指定主机上指定端口的套接字对象，并且尝试进行连接。不同的是，它连接到前面两个参数指定的主机和端口，并且从后面两个参数指定的本地网络地址和端口进行连接。如果在连接主机的过程中出现无法连接主机的错误，会抛出IOException 异常。

- public Socket(InetAddress address, int port, InetAddress localAddr, int localPort) throws IOException

这个构造函数与前面的构造函数功能相同，不同的是传入的参数不同，它使用InetAddress 对象指定主机，而不是使用字符串形式的主机名。如果在连接主机的过程中出现无法连接主机的错误，会抛出 IOException 异常。

（2）接受数据

public InputStream getInputStream() throws IOException 方法返回一个输入流，可以从套接字读取数据。通常的用法如下：

```
...
try
{
    //创建套接字对象
    Socket s = new Socket("news.sina.com.cn", 80);
    //获得输入流对象
    InputStream is = s.getInputStream();
    //后续处理
    ...
}
catch(UnknownHostException uhex)
{
    //进行异常处理
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
}
...
```

（3）发送数据

public OutputStream getOutputStream() throws IOException 返回一个 OutputStream 对象，把数据写入套接字。通常的用法如下：

```
...
try
{
    //创建套接字对象
```

```
        Socket s = new Socket("news.sina.com.cn", 80);
        //获得输入流对象
        OutputStream os = s.getOutputStream();
        //创建经过缓冲的输入流对象
        OutputStream bos = new BufferedOutputStream(os);
        //创建 Writer 对象
        Writer writer = new OutputStreamWriter(bos, "ASCII");
        //写入数据
        writer..write("GET / HTTP 1.0\r\n\r\n");
        //后续处理
        ...
    }
    catch(UnknownHostException uhex)
    {
        //进行异常处理
        ...
    }
    catch(IOException ioex)
    {
        //进行异常处理
        ...
    }
    ...
```

（4）关闭连接

套接字对象在创建、使用完毕以后要最好进行关闭。虽然套接字对象在程序结束的时候如果没有在程序中显式地得到关闭，垃圾收集器会自动回收，但是在套接字使用完毕的时候在程序中显示地关闭是一个良好的编程习惯。套接字类提供了一个方法用来关闭一个套接字对象：public synchronized void close() throws IOException。通常的用法如下：

```
    ...
    Socket s = null;
    try
    {
        //创建套接字对象
        s = new Socket("news.sina.com.cn", 80);
        //进行后续处理
        ...
    }
    catch(UnknownHostException uhex)
    {
        //进行异常处理
        ...
    }
    catch(IOException ioex)
    {
```

```
    //进行异常处理
    ...
}
finally
{
    //关闭套接字对象
    if(s != null) s.close();
}
...
```

2. 服务器端套接字

通过客户端套接字的介绍，我们了解了如何为正在侦听连接的服务器打开一个套接字的过程。但是仅仅有客户端套接字是不够的，而客户端套接字（简称套接字）不能够编写服务器程序。Java 为编写服务器端套接字提供了 java.net.ServerSocket 类，用于编写服务器程序，负责侦听接入的 TCP 连接。

- 在某个端口构造服务器端套接字对象；
- 侦听接入该端口的连接，该过程一直阻塞，直到客户端进行连接，连接成功会返回一个套接字对象；
- 发送和接受数据；
- 客户端或者服务器端关闭连接；
- 服务器返回步骤 2，侦听下一次连接。

下面我们针对上面的每一个步骤进行说明。

（1）创建服务器端套接字对象

服务器端套接字类共提供了 4 个构造方法，用于创建一个服务器端套接字对象。这里主要介绍经常使用的一个构造函数：public ServerSocket(int port) throws IOException。这个构造函数用于在指定的端口创建一个服务器端套接字，如果不能在指定的端口创建套接字，会抛出 IOException 异常。抛出异常的原因可能是因为在指定的端口已经存在另外一个服务器端套接字在运行，这样就可以利用这样的特性来进行端口的检测，详见本章典型问题解答中的问题 2。

该构造函数的典型用法如下：

```
...
try
{
    //创建服务器端套接字对象
    ServerSocket ss = new ServerSocket(90);
    //进行后续处理
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
}
...
```

（2）侦听连接

服务器端套接字通常在不断地接受连接，在每次循环中都会调用一个方法来侦听接入指定端口的连接，该方法一直阻塞，直到连接发生；连接成功以后会返回一个套接字对象。这个方法就是 public Socket accept() throws IOException。

该方法的通常的用法如下：

```
...
ServerSocket ss = null;
try
{
    //创建服务器端套接字对象
    ss = new ServerSocket(90);
    //进行后续处理
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
}
try
{
    //侦听接入的连接
    while(true)
    {
        //侦听方法
        Socket s = ss.accept();
        //进行后续处理
        ...
    }
    ...
}
catch(IOException ioex)
{
    //进行异常处理
    ...
}
...
```

（3）发送和接收数据

在服务器端套接字接受进入的连接并返回一个套接字对象以后，就可以利用上一节介绍的套接字的方法来发送和接收数据了，这里不再详细介绍。

（4）关闭连接

在连接建立并完成数据的发送和接收以后，无论是客户端还是服务器端都可以进行关闭操作，套接字的关闭前面已经介绍，这里不再详细介绍。

（5）重新侦听新的连接

在一个客户端连接接入、完成数据的发送和接受并关闭以后，服务器端套接字可以重新

侦听新的连接，这主要是通过一个循环来实现的，如上面的代码所示。

附录 C

计算机网络课程设计

实验报告

专业班级_____

姓名_____

学号_____

组长姓名学号_____

组长联系方式_____

同组人姓名学号_____

_____

实验日期_____

年　　月　　日

一、实验内容和要求

1、实验一

2、实验二

3、.....

二、实验环境

三、程序的需求分析与逻辑框图

1、实验一

2、实验二

3、.....

四、程序核心功能的实现机制

1、实验一

2、实验二

3、.....

五、程序源代码（核心部分）

1、实验一

2、实验二

3、.....

六、程序扩展功能的需求分析与实现

1、实验一

2、实验二

3、.....

七、实验数据、结果分析

八、总结

九、同组人分工情况

学号	姓名	承担任务

附录 D Red hat linux9.0 上安装 Intel 千兆网卡过程

1 下载文件（本实验中已下载）

e1000-8.0.19.tar.gz

2 把该文件拷到 Linux 系统/usr/local/src/目录中，并解压（本实验中已解压）

#tar zxvf e1000-8.0.19.tar.gz

3 进入到解压后文件夹，执行安装

#cd /usr/local/src/e1000-8.0.19/src

#make

#make install

4 重启机器

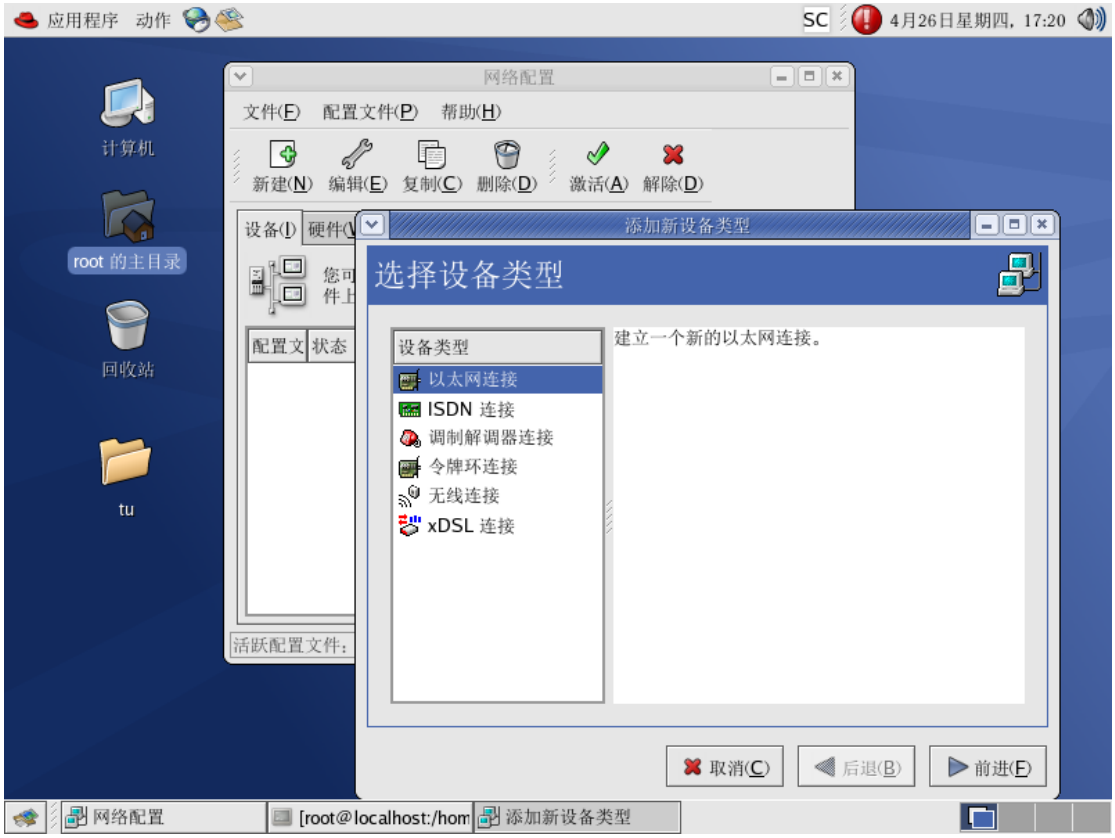
5 检查网卡驱动是否安装

点击“应用程序/系统设置/网络”，在“网络配置/硬件”中检查是否

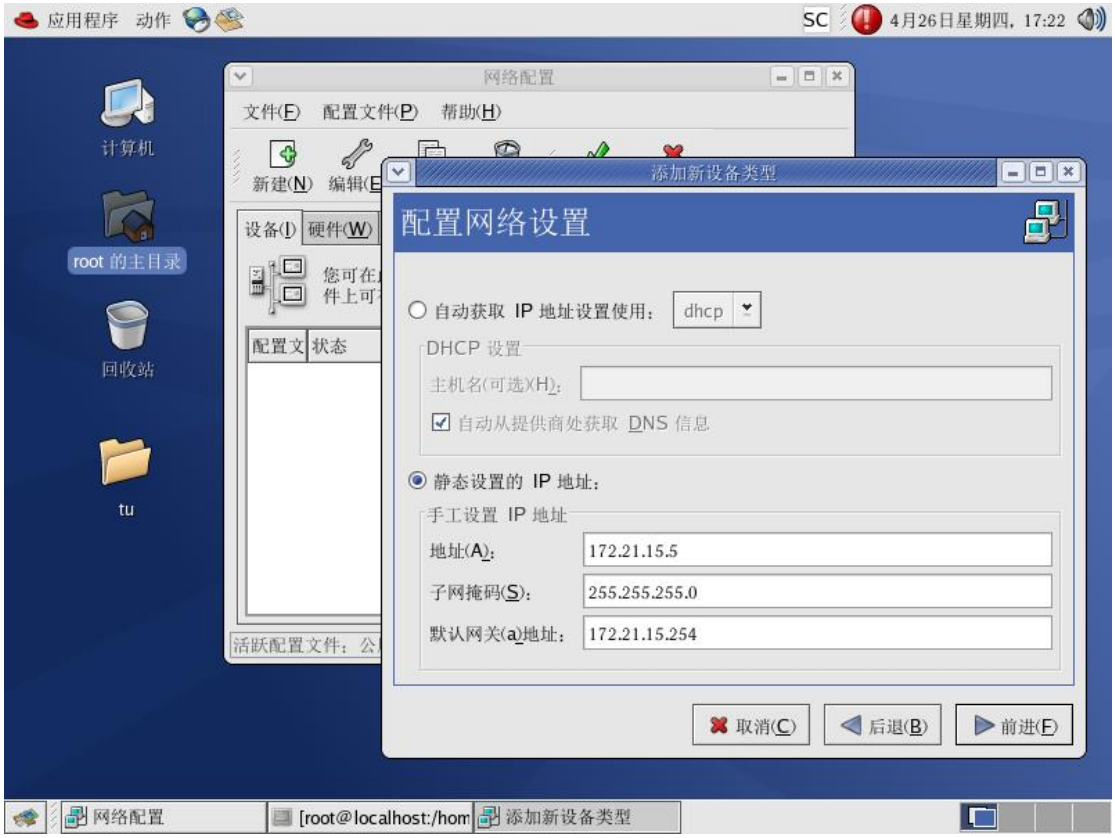
有 Intel® PRO/1000 MT Server Adapter（注：具体名字要看系统安装情况），若有，说明网卡驱动已安装；

6 配置网络接口地址、子网掩码、默认路由等

点击“应用程序/系统设置/网络”，在“网络配置”中选中“设备”，然后点击“新建”，在“设备类型”中选“以太网连接”，点击“前进”；

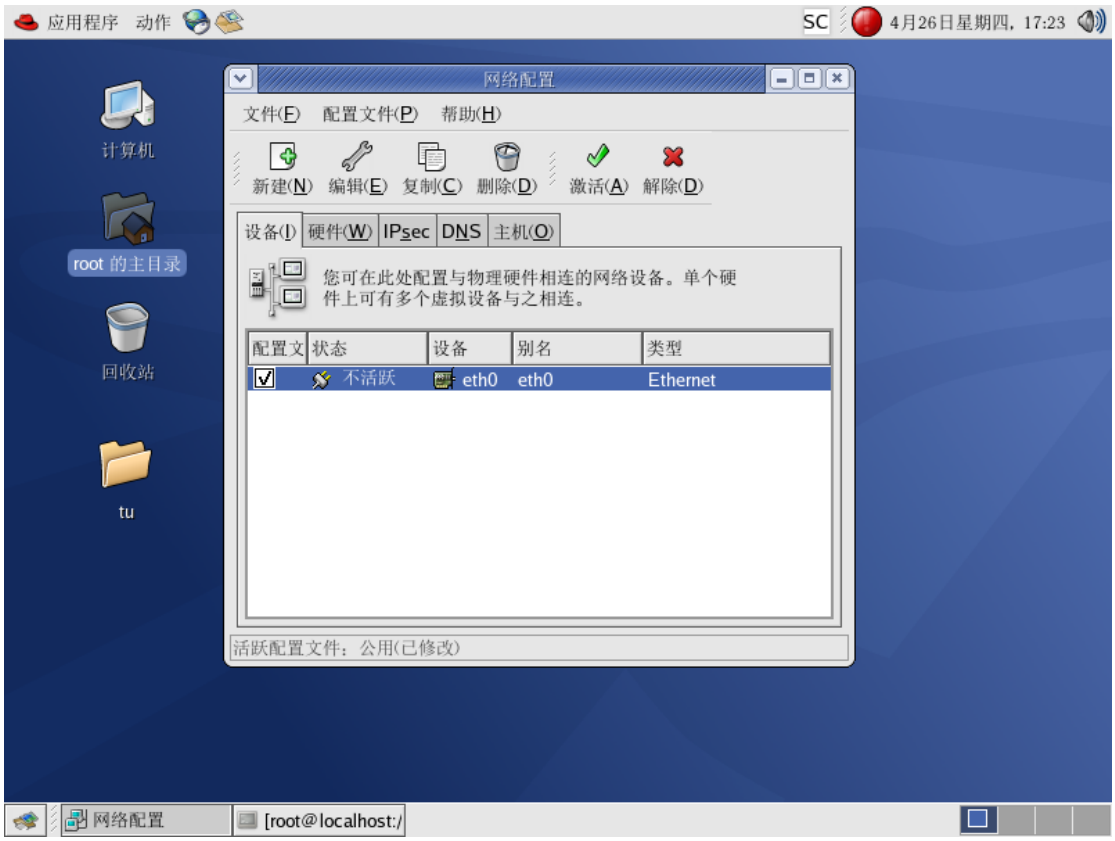


在“选择以太网设备”中选 Intel® PRO/1000 MT Server Adapter（注：具体名字要看系统安装情况），点击“前进”；在“配置网络设置”中，选“静态设置的 IP 地址”，配置“地址、子网掩码、默认网关地址”等信息，

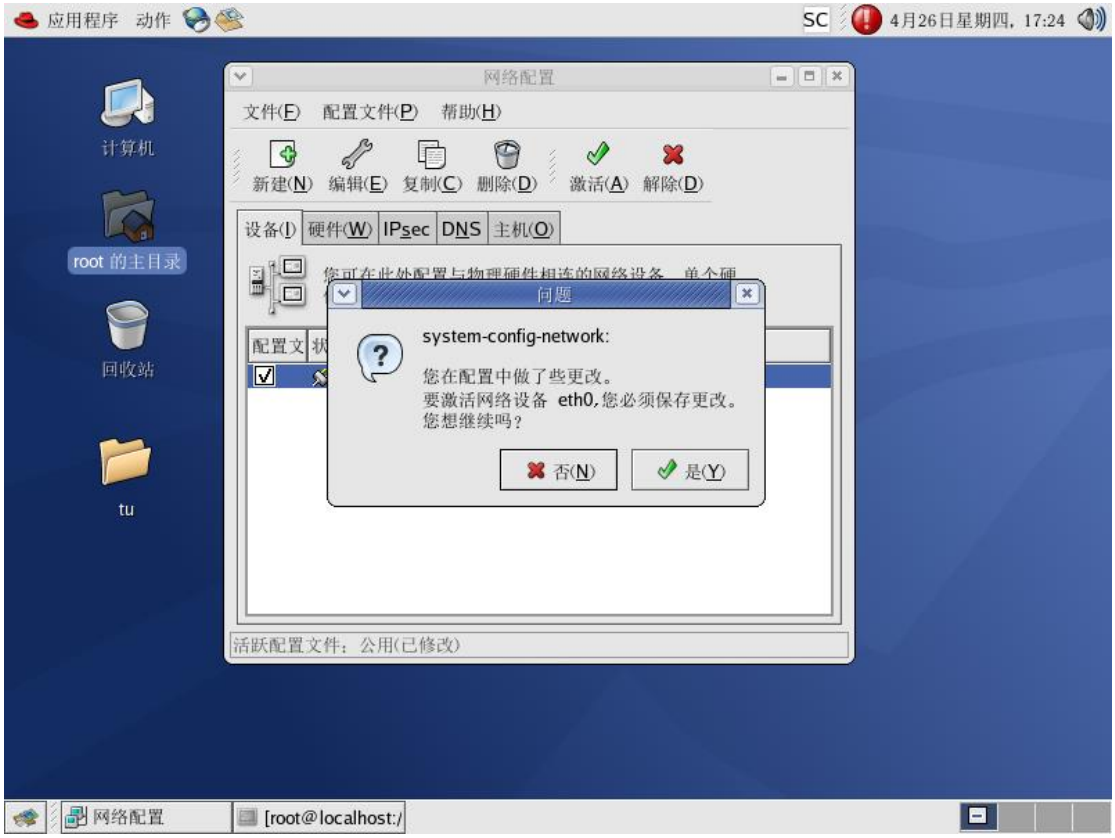

应用程序 动作
SC
4月26日星期四，17:22
计算机
root 的主目录
回收站
tu
网络配置
文件(F) 配置文件(P) 帮助(H)
新建(N) 编辑(E
设备(I) 硬件(W)
配置文
状态
活跃配置文件：公
添加新设备类型
配置网络设置
自动获取 IP 地址设置使用：
dhcp
DHCP 设置
主机名(可选)(H)：
自动从提供商处获取 DNS 信息
静态设置的 IP 地址：
手工设置 IP 地址
地址(A)：
172.21.15.5
子网掩码(S)：
255.255.255.0
默认网关(a)地址：
172.21.15.254
取消(C)
后退(B)
前进(F)
网络配置
[root@localhost:/hom
添加新设备类型

点击“前进”后，点击“应用”

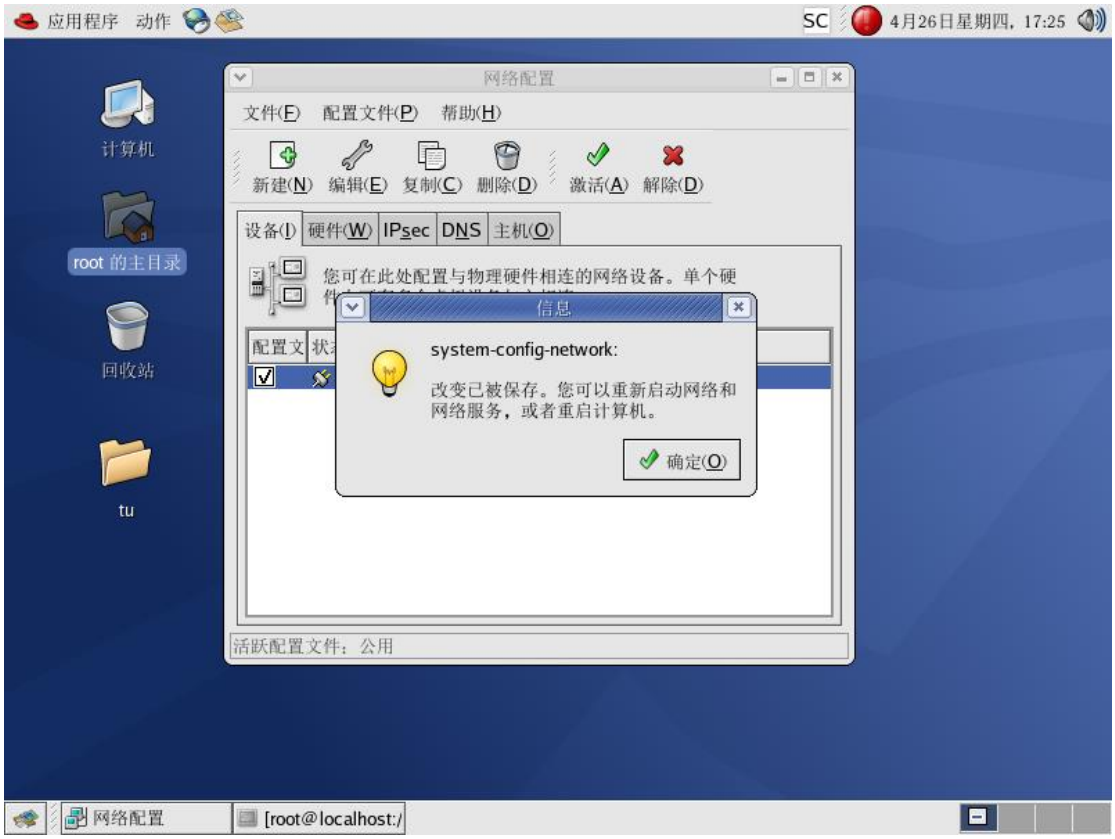
7 激活网卡


应用程序 动作
SC
4月26日星期四，17:23
网络配置
文件(F) 配置文件(P) 帮助(H)
新建(N) 编辑(E) 复制(C) 删除(D) 激活(A) 解除(D)
设备(I) 硬件(W) IPsec DNS 主机(O)
您可在此处配置与物理硬件相连的网络设备。单个硬件上可有多个虚拟设备与之相连。
配置文 状态 设备 别名 类型
不活跃 eth0 eth0 Ethernet
活跃配置文件：公用(已修改)
计算机
root 的主目录
回收站
tu
网络配置
[root@localhost:/

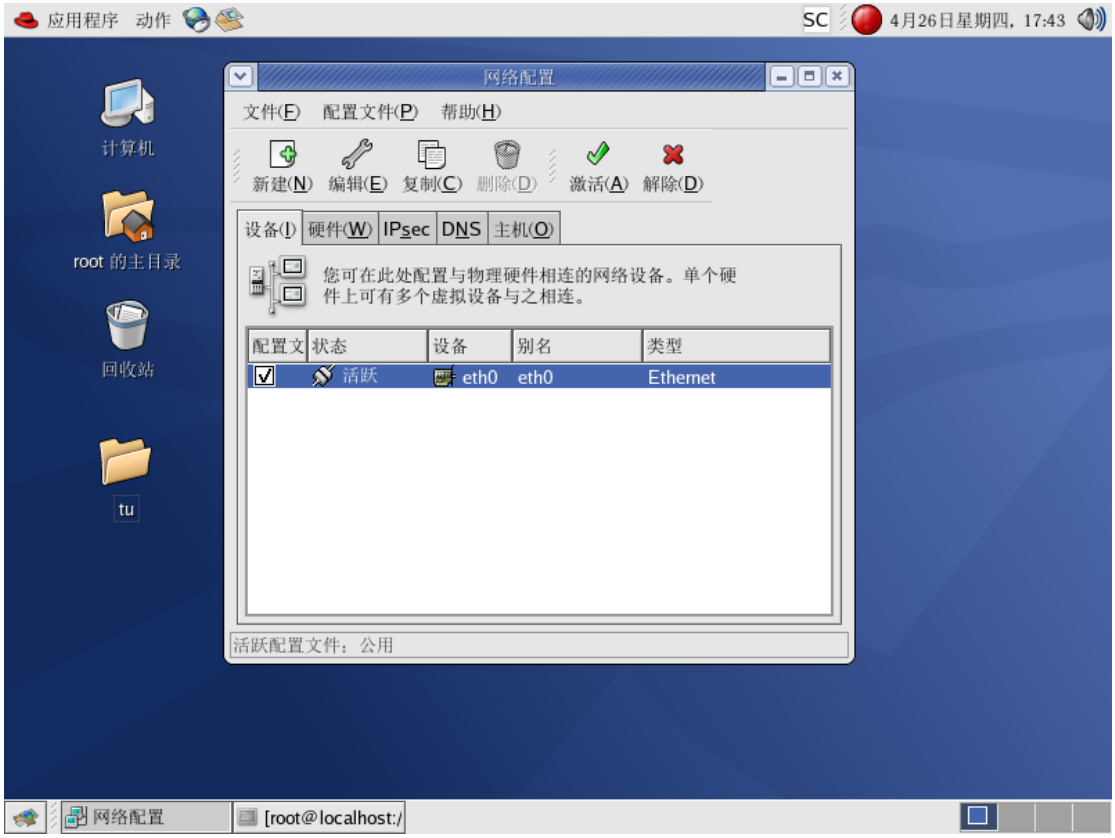
点击“激活”



点击“是”



点击“确定”，状态中显示“活跃”，成功了。



8 遇到问题(实验中不会出现)

注：如果 make 不过去，检查开发工具是否安装上，如果未安装，请先安装。

开发组件安装方法：点击“应用程序/系统设置/添加/删除程序”，选中“开发/开发工具”


应用程序 动作
SC
4月26日星期四，17:57
软件包管理
添加或删除软件包
网络服务器 [0/16]
这些软件包包括基于网络的服务器，例如 DHCP、Kerberos 和 NIS。
遗留网络服务器 [0/9]
这些软件包包括用于老网络协议的服务器，例如 rsh 和 telnet。
开发
开发工具 [57/75] 细节
这些工具包括 automake、gcc、perl、python 和调试器之类的核心开发工具。
X 软件开发 [0/18]
这些软件包允许您开发用于 X 窗口系统的应用程序。
GNOME 软件开发 [0/30]
总安装大小：2,387 MB
退出(Q)
更新(U)
root
4-U4 i386 Di
3
[cdrecorder]
root@localhos
软件包管理