高等学校计算机科学与技术教材

TCP/IP 网络编程技术基础

COMPUTER Science and Technology

□ 王雷 编著

- 原理与技术的完美结合
- 教学与科研的最新成果
- 语言精练，实例丰富
- 可操作性强，实用性突出

清华大学出版社 ● 北京交通大学出版社

高等学校计算机科学与技术教材

TCP/IP 网络编程技术基础

王 雷 编著

清华大学出版社
北京交通大学出版社
·北京·

内 容 简 介

本书是一本基于TCP/IP协议进行计算机网络编程的教科书。全书通过原理介绍与例程剖析的形式，系统介绍了LINUX环境下如何使用C语言基于TCP/IP协议进行网络编程的详细步骤与过程。

与国内外出版的同类教材相比，本书主要的特点为：在注重阐述TCP/IP网络通信原理与套接字API编程原理的基础上，通过对例程的深入剖析，深入浅出地介绍服务器与客户软件的编程技巧；同时，在章节的编排上更加富有衔接性。本书第1章和第2章主要介绍TCP/IP网络通信原理与套接字API编程原理，第3章和第4章主要介绍循环服务器软件的设计方法，第5章介绍服务器的并发机制，第6章到第8章则主要介绍并发服务器的设计方法，第9章主要介绍服务器并发性的统一与高效管理技术，第10章主要介绍客户进程中的并发机制，第11章主要介绍客户－服务器系统中的死锁问题，第12章则介绍了GCC编译器的安装与使用方法，整个12章按照“原理→循环服务器软件设计→并发服务器软件设计→并发客户软件设计→客户－服务器系统中的死锁问题→客户－服务器软件编译环境”的顺序，通过C语言例程剖析，由浅入深地介绍了基于TCP/IP协议进行计算机网络编程的方法。通过以上连贯的章节编排，使得读者能够更加简洁、系统地掌握网络编程技术。

本书可供计算机与通信专业的本科生、从事计算机网络编程的技术人与网络编程爱好者使用，同时，也可供其他专业的学生、计算机网络技术的爱好者，以及计算机应用技术相关的工程技术人员参考。

图书在版编目(CIP)数据

TCP/IP网络编程技术基础/王雷编著．—北京：清华大学出版社；北京交通大学出版社，2012.3
（高等学校计算机科学与技术教材）
ISBN 978－7－5121－0903－2

Ⅰ．①T…　Ⅱ．①王…　Ⅲ．①计算机网络－通信协议－高等学校－教材　②计算机网络－网络编程－高等学校－教材　Ⅳ．①TN915.04　②TP393.09

中国版本图书馆CIP数据核字（2012）第018322号

责任编辑：谭文芳　　特邀编辑：胡花蕾
出版发行：清华大学出版社　　邮编：100084　　电话：010－62776969
　　北京交通大学出版社　　邮编：100044　　电话：010－51686414
印 刷 者：北京交大印刷厂
经　　销：全国新华书店
开　　本：185×260　　印张：12.25　　字数：313千字
版　　次：2012年3月第1版　　2012年3月第1次印刷
书　　号：ISBN 978－7－5121－0903－2/TN·82
印　　数：1～4 000册　　定价：23.00元

前　言

背景动机

现代社会是信息社会，随着 Internet 在全球范围内的迅速普及，网络对人们的学习、工作、生活以及对社会的影响越来越大。计算机网络技术被誉为是“近代最深刻的技术革命”，人们用“网络时代”、“网络经济”等术语来描述计算机网络对社会信息化与经济发展的影响。

目前，国内各主要高校的计算机应用技术与软件工程专业本科生均开设了“TCP/IP 网络编程技术基础”这一门课程，但传统的教材缺乏对所给例程的深入剖析，从而导致初学者在采用这些教材进行学习时难以轻松掌握所学内容。为此，本书在作者多年讲授 TCP/IP 网络编程技术课程的基础上，首先在传统教材所介绍的 Socket 网络编程相关概念与技术的基础之上，进行了大幅度的内容增减与结构调整；其次，为了使不同层次的读者均能够更加方便地掌握所学内容，本书还在各章节中对所给出的例程新增了全面深入的剖析；最后，本书还新增了对 GCC 编译器的有关介绍，使得全书内容更加完整。

上述这些原因构成了编著本书的一个主要背景动机。

目标读者

本书的目标读者包括计算机相关专业的本科生与研究生、计算机网络编程技术与 C 语言的爱好者，以及计算机应用技术相关的工程技术人员。

组织结构

考虑到读者在阅读本书之前对计算机网络编程技术的了解程度不尽相同，特将本书分为以下四大部分，其中：

① 第 1 章和第 2 章为第一部分，主要介绍 TCP/IP 网络通信原理与套接字 API 编程原理；

② 第 3 章至第 9 章为第二部分，其中，第 3 章主要介绍循环服务器软件的设计方法，第 4 章介绍服务器的并发机制，第 5 章到第 8 章则主要介绍并发服务器的设计方法，第 9 章主要介绍服务器并发性的统一与高效管理技术；

③ 第 10 章至第 11 章为第三部分，其中，第 10 章主要介绍客户进程中的并发机制，第 11 章主要介绍客户 - 服务器系统中的死锁问题；

④ 第 12 章为第四部分，主要介绍了 GCC 编译器的安装与使用方法。

编　者

2011 年 12 月

目　录

第 1 章　TCP/IP 网络通信原理

TCP/IP 协议是实现网络通信的基础，本章将在简要介绍 TCP/IP 协议及 TCP/IP 参考模型的基础上，深入介绍 TCP/IP 网络通信的基本原理与 TCP/IP 网络通信中所采用的客户 - 服务器通信模型，然后在此基础上，进一步系统介绍客户 - 服务器通信模型中的客户软件与服务器软件的设计流程。

1.1　TCP/IP 协议概述

1.1.1　TCP/IP 参考模型

TCP/IP（Transmission Control Protocol/Internet Protocol），即传输控制协议/因特网协议，是一个由多种协议组成的协议族（Protocol Family），定义了计算机通过网络互相通信及协议族各层次之间通信的规范。

TCP/IP 参考模型是一个抽象的分层模型，这个模型中，属于 TCP/IP 协议族的所有网络协议都被归类到了以下四个抽象的“层”之中。

1. 主机 - 网络层（Host to Network Layer）

主机 - 网络层是 TCP/IP 参考模型的最低层，也称为网络接口层，它主要负责接收从互联网络层交来的 IP 数据报并将其通过低层物理网络发送出去，或者从低层物理网络上接收物理帧并从中抽出 IP 数据报交给互联网络层。其中，网络接口主要有以下两种类型：第一种是设备驱动程序，如局域网的网络接口；第二种是含自身数据链路协议的复杂子系统。在 TCP/IP 参考模型中未定义数据链路层，这主要是因为在 TCP/IP 最初的设计中已经使其可以使用各种典型的数据链路层协议。

2. 互联网层（Internet Layer）

也称为网际互联层或 IP 层，主要负责将源主机的报文分组发送到目的主机，源主机与目的主机可以在一个网络上，也可以在不同的网络上。由于 TCP/IP 参考模型中网络层协议是 IP 协议，因此互联网络层也称为 IP 层。其中，IP 协议是一种不可靠、无连接的数据报传送服务的协议，它提供的是一种“尽力而为（Best Effort）”的服务。IP 协议的协议数据单元是 IP 分组，由于在 IP 层提供数据报服务，因此，也常将 IP 分组称为 IP 数据报。

3. 传输层（Transport Layer）

传输层主要负责在互联网中源主机与目的主机的对等进程实体之间提供可靠的端到端的数据传输。在 TCP/IP 参考模型的传输层中定义了以下这两种协议。

（1）TCP 协议。TCP 协议是一种可靠的面向连接的协议，它允许将一台主机的字节流（Byte Stream）无差错地传送到目的主机。TCP 协议将应用层的字节流分成多个字节段

(Byte Segment)，然后将一个个的字节段传送到互联网络层，并最终发送到目的主机。当互联网络层将接收到的字节段传送给传输层时，传输层再将多个字节段还原成原始的字节流，并传送到应用层。TCP 协议同时要完成流量控制功能，协调收发双方的发送与接收速度，以达到正确传输的目的。

(2) UDP 协议 (User Datagram Protocol，用户数据报协议)。UDP 协议是一种不可靠的无连接协议，它主要用于不要求分组顺序到达的传输服务之中，在基于 UDP 协议的传输服务中，分组的传输顺序检查与排序由应用层完成。UDP 协议主要面向请求 - 应答式的交易型应用，一次交易往往只有一来一回两次报文交换，假如为此而建立和撤销连接将导致网络开销过大，因此，在这种情况下使用 UDP 就非常有效。另外，UDP 协议也常用于那些对可靠性要求不高，但要求网络的延迟较小的场合，如语音和视频数据的传送等。

4. 应用层 (Application Layer)

应用层包括了所有的高层协议，目前 TCP/IP 参考模型中的应用层协议主要包括以下几种：

(1) 网络终端协议 Telnet；
(2) 文件传输协议 (File Transfer Protocol，FTP)；
(3) 简单邮件传输协议 (Simple Mail Transfer Protocol，SMTP)；
(4) 域名系统 (Domain Name System，DNS)；
(5) 简单网络管理协议 (Simple Network Management Protocol，SNMP)；
(6) 超文本传输协议 (Hyper Text Transfer Protocol，HTTP)。

1.1.2 TCP/IP 参考模型的通信原理

TCP/IP 参考模型的通信原理如图 1.1 所示，其中，第一至二层为串联的，而第三至四层则是端到端 (End to End) 的。

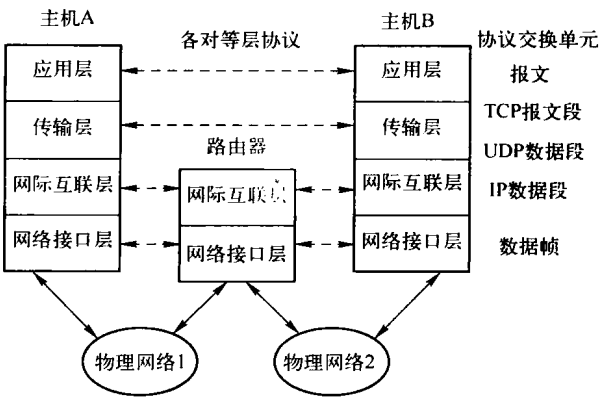

图 1.1 TCP/IP 参考模型的通信原理

由图 1.1 可知，网际互联层与网络接口层实现了计算机网络中处于不同位置的主机之间的数据通信，但是数据通信不是计算机网络的最终目的，计算机网络最本质的活动是实现分布在不同地理位置的主机之间的进程通信，以实现各种网络服务功能。而设置传输层的主要目的就是要实现上述这种分布式进程之间的通信功能。

在单机系统中，由于每个进程都在自己的地址范围内运行，为保证两个相互通信的进程之间既互不干扰又协调一致工作，LINUX 操作系统为进程之间的通信提供了相应的设施，如管道（Pipe）、命名管道（Named Pipe）、软中断信号（Signal）和信号量（Semaphore）等，但上述这些设施都仅限于用在本机进程之间的通信。而网间分布式进程通信要解决的是不同主机进程间的相互通信问题（同机进程通信只是其中的一个特例）。为此，传输层需要解决在网络环境中分布式进程通信所需解决的以下两个方面的问题。

（1）进程命名与寻址。在同一台计算机中，每一个进程均被分配了一个唯一的端口（Port）与端口号（Port Number，也称为进程标识），其中，端口是一个信息缓存区，用于保留 Socket 中的输入/输出信息，端口号是一个 16 位无符号整数，范围是 0 ~ 65 535，以区别主机上的每一个应用程序进程（如果将端口类比为房间，则端口号就是房间号。端口号一般有以下两种基本的分配方式：第一种为全局分配，这是一种集中分配方式，由一个公认权威的中央机构根据用户的需要进行统一分配，并将结果公布于众，通过这种方式分配的端口号也称为熟知端口号；第二种为本地分配，又称动态连接，是当进程需要访问传输层服务时，向本地操作系统提出申请，再由操作系统返回本地唯一的端口号）。由于同一台机器上的不同进程所分配到的端口号不同，因此，同一台机器上的不同进程就可以用端口号来唯一标识；但在网络环境中，显然，若要标识一个完整的进程，除了端口号之外，还需使用到本地主机的 IP 地址（本地地址）来唯一标识进程所在的本地主机（这是由于不同机器上的进程可以拥有相同的端口号）。

（2）多重协议的识别。由于 LINUX 操作系统支持的网络协议众多，不同协议的工作方式与地址格式均不相同。因此在网络环境中，一个应用程序进程最终需要使用一个三元组 < 协议，本地地址，本地端口号 > 来唯一地标识；另外，在 TCP/IP 网络环境中，一个完整的网间通信需要由两个进程组成，并且这两个进程之间只能使用相同的传输层协议才能进行通信，也就是说，不可能通信的一端使用 TCP 协议，而另一端使用 UDP 协议。因此，一个完整的网间通信需要用一个五元组 < 协议，本地地址，本地端口号，远程地址，远程端口号 > 才能唯一地标识。其中，二元组 < 本地地址，本地端口号 > 称为网间进程通信中的本地端点地址（End-point Address），二元组 < 远程地址，远程端口号 > 称为网间进程通信中的远程端点地址，而三元组 < 协议，本地地址，本地端口号 > 称为一个半相关（Half-Association），五元组 < 协议，本地地址，本地端口号，远程地址，远程端口号 > 则称为一个相关（Association）。

1.1.3　LINUX 系统实现网络通信的基本原理

1. LINUX 中提供的基本 I/O 功能

操作系统是一个用来和计算机硬件打交道并为用户程序提供一个有限服务集的低级支撑软件。一个计算机系统是一个由硬件和软件组成的共生体，它们互相依赖，不可分割。但是硬件若没有软件来操作和控制，它们自身是不能工作的，而完成上述控制工作的软件就称为操作系统（Operation System）。

LINUX 操作系统是最受欢迎的计算机操作系统之一，它是一个用 C 语言写成并符合 POSIX 标准的类 UNIX 操作系统。LINUX 操作系统最早是 1991 年由芬兰黑客 Linus Torvalds 为尝试在英特尔 x86 架构上提供自由免费的类 UNIX 操作系统而开发的。为了让 LINUX 系统支持网络通信功能，TCP/IP 协议被集成到了 LINUX 操作系统的内核之中。

在 LINUX 操作系统中，所有的设备都看作是一种具体的文件，LINUX 操作系统提供了以下一组（共包括六个）基本的系统函数，用来对本地设备或文件进行 I/O 操作.

(1) open( ) 函数

该函数用于打开一个文件（设备），其返回值为一个整型变量，若返回值为 -1，则表示调用 open 函数打开文件时出错；否则，则表示打开文件成功，此时该返回值也称为所打开文件的文件描述符（File Descriptor）。其函数原型如下：

```
#include < sys/types. h >      /* 基本数据类型头文件,含有基本系统数据类型的定义 */
#include < sys/stat. h >       /* 文件状态头文件,含有文件或文件系统状态结构和常量的定义 */
#include < fcntl. h >          /* 文件控制头文件,含有文件及其描述符的操作控制常数符号的
                                定义 */
int open( const char * pathname, int flags, mode_t mode);
```

在上述 open( ) 函数的原型中，各参数的含义如下。

pathname：指向欲打开的文件路径字符串。

flags：文件打开方式的标志，主要包括以下几种。

- O_RDONLY：以只读方式打开文件。
- O_WRONLY：以只写方式打开文件 。
- O_RDWR：以可读写方式打开文件。

上述三种文件打开方式标志是互斥的，不可以同时使用，但可与下列文件打开方式标志利用 OR(|)运算符进行组合。

- O_CREAT：若欲打开的文件不存在则自动建立该文件。
- O_APPEND：当写文件时会从文件尾开始移动，也就是所写入的数据会以附加的方式加入到文件末尾。
- O_TRUNC：若文件存在并且以可写的方式打开时，此标志会令文件长度清 0。

mode：被打开文件的存取权限，只有在建立新文件时才会生效，即只有当 flags 取值中有 O_CREAT 时才有效。

(2) close( ) 函数

该函数用于关闭 open( ) 函数所打开一个文件（设备），若调用成功时将返回 0；若调用出错则返回 -1。其函数原型如下：

```
#include < unistd. h > /* LINUX 标准头文件,包含了各种 LINUX 系统服务函数原型、符号常数和
                        类型的定义 */
int close( int fd);
```

在上述 close( ) 函数的原型中，主要参数的含义如下。

fd：欲关闭文件的文件描述符。

(3) read( ) 函数

该函数用于从指定的文件（由文件描述符参数 fd 指定）中读取指定的字节数据（由参数 count 指定）存放到指定的缓存区（由参数 buf 指定）之中。调用成功时返回读取的字节数；若返回 0 表示已到达文件尾；若返回 -1 则表示调用出错。其函数原型如下：

```
#include < unistd. h >
```

```
ssize_t read(int fd, void * buf, size_t count);
```

在上述 read( ) 函数的原型中，各参数的含义如下。

fd：指定文件的文件描述符。

buf：指定用来存储所读取的数据的缓冲区。

count：指定要读取的字节数。在读取普通文件时，若读到要求的字节数之前已达到文件尾部，则返回的字节数将会小于希望读取的字节数 count。

注：*数据类型 ssize_t 是 signed size_t，而数据类型 size_t 是标准 C 库中定义的，表示 unsigned int（无符号整形）。*

(4) write( ) 函数

该函数用于向指定的文件（由文件描述符参数 fd 指定）中写入指定的字节数据（由参数 count 指定）存放到指定的缓存区（由参数 buf 指定）之中。调用成功时返回实际写入的字节数；若调用出错则返回 -1。其函数原型如下：

```
#include < unistd. h >
ssize_t write(int fd, const void * buf, size_t count);
```

在上述 write( ) 函数的原型中，各参数的含义如下。

fd：指定文件的文件描述符。

buf：指定存储写入数据的缓冲区。

count：指定写入的字节数。

(5) lseek( ) 函数

该函数用于移动文件的读写指针，更改打开文件的偏移量，实现在文件内部的定位。调用成功时返回所设置的新的偏移量；若调用出错则返回 -1。其函数原型如下：

```
#include < unistd. h >
#include < sys/types. h >
off_t lseek(int fd, off_t offset, int whence);
```

在上述 lseek( ) 函数的原型中，各参数的含义如下。

fd：文件描述符。

offset：偏移量，每一次读写操作所需移动的距离，单位是字节的数量，可正可负（为正时表示从当前基点位置向前移动，为负时表示从当前基点位置向后移动）。

whence：表示当前基点位置，取值如下。

- SEEK_SET：当前基点位置为文件的开头，新位置为偏移量的大小。
- SEEK_CUR：当前基点位置为文件指针位置，新位置为当前位置加上偏移量。
- SEEK_END：当前基点位置为文件的结尾，新位置为文件的大小加上偏移量。

(6) ioctl( ) 函数

该函数是设备驱动程序中对设备的 I/O 通道进行管理的函数。所谓对设备的 I/O 通道进行管理，就是对设备的一些特性进行控制。尽管多数情况下硬件设备的操作可以通过文件的 read、write、lseek 等操作实现，但总有一些特例，如弹出光盘、让磁带机倒带、设置声卡的采样率等，当需要对设备的上述这类特性进行控制时，就需要用到 ioctl( ) 函数。若调用成

功时将返回0；若调用出错则返回-1。其函数原型如下：

```
#include <sys/ioctl.h> //该头文件中包含了I/O控制函数的定义
int ioctl(int fd, int cmd, …);
```

在上述ioctl()函数的原型中，各参数的含义如下。

fd：文件（设备）的文件描述符。

cmd：用户程序对设备的控制命令。

注：ioctl是一个可变长参数的函数。可变长参数函数的原型为type VAFunction（type arg1, type arg2, …）；参数可以分为个数确定的固定参数和个数可变的可选参数两部分。函数至少需要一个固定参数，固定参数的声明和普通函数一样；可选参数由于个数不确定、可有可无，需要根据实际情况而定，声明时用“…”表示。固定参数和可选参数公同构成一个函数的参数列表。可变长参数的函数的定义方法如下例1-1所示。

例1-1：求任意个自然数的平方和。

```
int SqSum(int n1, …){
    va_list arg_ptr; //定义一个指向个数可变的参数列表指针
    int nSqSum=0, n=n1;
    va_start(arg_ptr, n1); /*va_start(arg_ptr, argN):使参数列表指针arg_ptr指向函数参数列表
                           中的第一个可选参数,说明:argN是位于第一个可选参数之前的那个
                           固定参数,(即最后一个固定参数;亦即"…"之前的那个参数),函数
                           参数列表中参数在内存中的顺序与函数声明时的顺序是一致的。如
                           果有一个va函数的声明是void va_test(char a, char b, char c, …),
                           则它的固定参数依次是a,b,c,最后一个固定参数argN为c,因此就
                           是va_start(arg_ptr, c)*/
    while(n>0){
        nSqSum += (n*n);
        n=va_arg(arg_ptr, int); /*va_arg(arg_ptr, type):返回参数列表中指针arg_ptr所指的参
                                数,其返回类型为type,并使得指针arg_ptr指向参数列表中的
                                下一个参数*/
    }
     va_end(arg_ptr); /*va_end(arg_ptr):清空参数列表,并置参数指针arg_ptr无效*/
    return nSqSum;
}
```

针对上述定义的可变长参数的函数SqSum，可通过如下方法进行调用：

```
int totalsum;
int totalsum=SqSum(7,2,7,11,-1); /*调用SqSum(7,2,7,11,-1)将计算7*7+2*2+7*7+
                                 11*11之和,当n=-1是将跳出while(n>0)循环*/
```

2. LINUX中基本I/O系统函数的用法示例

(1) open()、read()、write()、lseek()、close()函数的用法示例

```
#include <unistd.h>        //调用close()、read()、write()、lseek()函数所需的头文件
```

```
#include <sys/types.h>      //调用 open()、lseek()函数所需的头文件
#include <sys/stat.h>       //调用 open()函数所需的头文件
#include <fcntl.h>          //调用 open()函数所需的头文件
#include <stdlib.h>         /*C 标准库头文件,包含了 C 语言标准库函数(如 exit()、atoi()等)原型
                              的定义*/
#include <stdio.h>          /*标准输入输出头文件,包含了标准输入输出函数(如 perror()和 printf
                              ()等)的定义*/
#include <string.h>         /*字符串头文件,包含了字符串操作函数(如 strlen()等)的定义*/
int main(void){
    char *buf="Hello! I'm writing to this file!"; /*定义了一个指向包含有 10 个字符的字符常
                                                   量"Hello! I'm writing to this file!"的字符型
                                                   指针*/
    char buf_r[11]; /*定义了一个长度为 11 个字符的字符型数组变量,字符在计算机中以其
                      ASCII 码方式表示,其长度为 1 个字节,因此,计算机在编译时将留出连
                      续 11 个字节的空间,即 buf_r[0]到 buf_r[10]共 10 个字符变量,但只有
                      前 10 个变量可供用户使用,第 11 个字符变量 buf_r[10]将用来存放字
                      符串终止符 NULL 即"\0",其中,终止符是编编译程序自动加上的*/
```

注：字符型指针与字符型数组变量是不同的，当两者在用 sizeof 取长度时，在 32 位机器下，字符型指针的长度为 4 个字节，如 sizeof(buf)的长度为 4；而字符型数组变量的长度为其中缓存的字符串的长度，如 sizeof(buf_r)的长度为 11。

```
    int fd, size, len;
    len=strlen(buf);          /* strlen 函数的原型为 int strlen(char *s);在头文件<string.h>中定
                                 义,其功能为计算字符型指针 s 所指向的字符串的长度*/
    if((fd=open("/tmp/hello.c", O_CREAT | O_TRUNC | O_RDWR,0666))<0){
    //调用 open 函数打开 hello.c 文件,并指定打开方式与相应的操作权限
        perror("open failed:");       /* perror 函数的原型为 void perror(const char *s);打印 s
                                         所指的字符串和标准出错信息,相当于 printf("%s: %s
                                         \n", *s, strerror(errno));调用该函数需要包括头文
                                         件<stdlib.h>和 <errno.h>*/
        exit(1);              /* exit 的函数原型为 void exit(int status);其功能是中止程序执行并
                                 返回一个状态值 status 给操作系统,status 为 0 时表示正常终止,为
                                 -1 表示异常终止*/
    } else
        printf("open and create file:hello.c with file descriptor %d OK\n",fd);
                              /* printf 函数的原型为 int printf( const char *format, … );在头文件
                                 <stdio.h>中定义,是一种可变长参数的函数,参数 format 表示如
                                 何来格式字符串的指令*/
    if((size=write(fd, buf, len))<0){          /*调用 write 函数,将 buf 中的内容写入到打开的
                                                 文件中*/
        perror("write:");
        exit(1);
    } else
```

```
        printf("Write:%s OK\n",buf);
    lseek(fd, 0, SEEK_SET ); //定位到文件的开头位置
    if((size=read( fd, buf_r, 10))<0){      /*调用 read 函数从文件中读取 10 个字节到缓存区
                                              buf_r 中*/
        perror("read failed:");
        exit(1);
    } else
        printf("read form file:%s OK\n",buf_r);
    if(close(fd)<0 ){
        perror("close failed:");
        exit(1);
    } else printf("Close hello.c OK\n");
        exit(0);
    }
```

(2) ioctl()函数的用法示例

声卡是普通个人计算机的标准设备。在 LINUX 下，有几种设备文件用来控制声卡的功能。一种是声音混合设备/dev/mixer，用来控制各个声道的音量；另一种设备是声音的采集和播放设备，包括/dev /audio、/dev/dsp 等，以下例程介绍了如何通过对/dev/audio 编程来对声卡的采样频率进行设置。

声卡编程必须包含以下头文件，因为这些头文件中包含了必须的函数声明和变量说明：

```
#include <sys/ioctl.h> //提供对 I/O 控制的函数原型的定义
#include <unistd.h>
#include <sys/soundcard.h>      /*提供对声卡支持的所有采样格式和采样频率等的定义*/
```

对声卡采样频率进行设置：

```
int audio_fd;
int format;
audio_fd=open("/dev/audio", O_WRONLY, 0); //打开声音采集/播放设备
format=AFMT_S16_LE;                       //设置 16 位采样频率 AFMT_S16_LE
ioctl(audio_fd, SNDCTL_DSP_SETFMT, &format);
            // SNDCTL_DSP_SETFMT 命令用于设置声卡的采样频率
```

3. 将 LINUX 中的基本 I/O 功能用于 TCP/IP 网络通信

为了访问 TCP/IP 协议，LINUX 操作系统对上述传统的基本 I/O 调用进行了扩展。首先，LINUX 操作系统扩展了文件描述符集，使得应用程序进程既可以创建用于本机设备访问的文件描述符，又可以创建能被 TCP/IP 网络通信所使用的套接字描述符（Socket Dsecriptor）。其次，LINUX 操作系统还扩展了 read 和 write 两个系统调用，使得其既可以同套接字描述符一起使用，又可以同普通的文件描述符一起使用。这样一来，当应用程序进程需要通过 TCP/IP 网络收发数据时，就可以创建相应的套接字描述符，然后再使用 read 和 write 系统调用来通过 TCP/IP 网络进行数据收发了。

其中，所谓的套接字描述符，就是一个整数类型的值，与文件描述符是用于标识一个活