计 算 机 网 络

实验指示书

清华大学计算机科学与技术系

2000 年 10 月

前　言

《计算机网络》课程是计算机科学与技术专业的重要专业课程之一。随着计算机网络技术的迅速发展和在当今信息社会中的广泛应用，给《计算机网络》课程的教学提出了新的更高的要求。

由于计算机网络是一门实践性较强的技术，课堂教学应该与实践环节紧密结合。将《计算机网络》课程建设成世界一流的课程，是近期《计算机网络》课程努力的方向。

我校自 1984 年开始为本科生同学开设了《计算机网络》课程，当时由于设备紧张，基本没有实践环节。1991 年，在系领导和教研室的大力支持下，我们筹建了《计算机网络》的教学实验环境。这个实验环境为同学提供了基本的网络操作和功能，在当时达到了国外高等院校和科研机构的计算机网络基本水平，这个实验环境为我系的《计算机网络》课程的教学作出了巨大的贡献。1998 年，我们由重新改建了实验环境。目前，本实验室拥有 30 台 SUN 公司的 ULTRA 5 工作站和一台 ULTRA 10 作为服务器。现在的网络实验环境不逊于世界各知名学府，在这样良好的硬件环境下，我们就在软件上也力争达到世界先进水平。所以，我们重新编写了实验指示书，调整了实验安排，加大了实践力度。希望同学们能够充分利用实验条件，认真完成实验，从实验得到应有的锻炼和的培养。

希望同学们在使用本实验指示书及进行实验的过程中，能够帮助我们不断地发现问题，并提出建议，使《计算机网络》成为具有世界一流水平的课程。本学期授课教师为吴建平，徐明伟，参与网络课教学环节的同学还有赵邑新、陈东洛、王彬、盛利杰、鱼小丽、韩博、吴茜、林云峰、吴英华；此外周树云等老师和章淼等同学也给予了热情的支持和帮助。

目　录

实验要求

计算机网络是现代信息社会最重要的基础设施之一。在过去十几年里得到了迅速的发展和应用。《计算机网络原理》课程实验的目的是为了使学生在课程学习的同时，通过在一个计算机网络环境中的实际操作，对现代计算机网络的基本功能有一个初步的了解；通过实现一个数据链路层协议，掌握计算机网络通信协议的基本实现技术；通过一个简单文件传送协议的设计和实现，了解计算机网络高层协议设计实现的环境和方法；通过实现一个简单的协议状态机了解、掌握协议实现中这一重要技术；还提供了一些选做实验以供有余力有兴趣的同学进一步提高。总之，通过上述实验环节，使学生加深了解和更好地掌握《计算机网络原理》课程教学大纲要求的内容。

在《计算机网络原理》的课程实验过程中，要求学生做到：

（1）预习实验指示书有关部分，认真做好实验内容的准备，就实验可能出现的情况提前作出思考和分析。

（2）仔细观察上机和上网操作时出现的各种现象，记录主要情况，作出必要说明和分析。

（3）认真书写实验报告。实验报告包括实验目的和要求，实验情况及其分析。对需编程的实验，写出程序设计说明，给出源程序框图和清单。

（4）遵守机房纪律，服从辅导教师指挥，爱护实验设备。

（5）实验课程不迟到，不早退。如有事不能出席，需要向辅导教师请假，所缺实验一般不补。

实验的验收将分为两个部分。第一部分是上机操作，包括检查程序运行和即时提问。第二部分是提交书面的实验报告。此外，针对以前教学中出现的问题，网络实验将采用**阶段检查**方式，每个实验都将应当在规定的时间内完成，过期视为未完成该实验，以避免期末集中检查产生的诸多不良问题，希望同学们抓紧时间，合理安排，认真完成。

实验准备及简要上机指示

网络课实验上机地点安排在东主楼9区414室,环境为约30台Sun Ultra－5 工作站和一台 Sun Ultra－10 工作站、软件为 Solaris 2.5.1，Openwin 所构成的网络环境。并通过 NIS+ 软件将全部用户的帐号和文件管理在一台 Sun Ultra－10 工作站上，该 Ultra－10 充当 NIS+ 服务器，其余 Ultra－5 为 NIS+客户机。

作为基础,同学应当了解 UNIX 系统上的基本操作,并学习掌握其他有关知识,本部分将提供三个方面的基础知识以方便同学使用系统进行实验：上机基本操作，介绍开机，关机，登录，修改密码等操作；网络基本操作，telnet，ftp；工程文件，书写 makefile 的基础知识等。更详细的内容及深入知识同学可根据个人需要及兴趣查阅有关技术资料。

1 上机基本操作

1.1 开机与关机

由于实验环境提供给每位同学的是一台完整主机（不是终端），系统启动是在本机硬盘上，只用用户帐号和密码管理在服务器上，编译程序时也使用本机资源，由于操作系统 UNIX / Solaris 的文件系统管理组织方式,对于文件系统的完整性有很高的要求，**非正常关机会造成文件系统不完整，至使系统下次启动故障，严重时需要重新安装系统!** 因此要求同学严格按照操作规程进行。如遇特殊情况，可以请辅导教师或系统管理员协助解决。

开机与关机：

开机：打开主机后部的开关，并且打开显示器开关（显示器正面最右边按钮）

关机：首先退出用户自己的帐号，回到系统提示登录状态

Login: halt

Password : halthalt　（连续输入两个 halt 作为密码）

出现 ok 后，同时按住 Shift 键和键盘最右上角的键可以关闭系统，**并关闭主机后部开关**（否则第二天加电后系统就会自行启动），并且关闭显示器。

修改密码:

系统管理员为网络课开放了 ne0000 到 ne0199 的帐号，对于学号在 971209 到 971380 之间的同学，帐号为 ne0 ＋ （学号后三位 － 200，并补 0 补足三位），比如 971209 的帐号为 ne0009，971232 的帐号为 ne0032，971315 的帐号为 ne0115。其他同学可向辅导教师申请一个帐号。所有同学选定帐号后，统一登记，以便于实验及检查。

系统管理员给每个帐号设定的初始密码为 net123，**请同学们首次登录后立刻修改各自的密码!**

由于 NIS 软件的存在，修改密码是则可能有两种情况，假设当前密码为 net123，想把密码修改为 net345：

第一种情况

```
ne0008@net56_/export/home/student/ne0008 >passwd
passwd:    Changing password for ne0008
Enter login(NIS+) password: net123
New password:net345
```

```
Re-enter new password:net345
   NIS+ password information changed for ne0008
   NIS+ credential information changed for ne0008
```

第二种情况，这种情况比较常见

```
ne0025@net56_/export/home/student/ne0025 >passwd
passwd:    Changing password for ne0025
Enter login(NIS+) password: net123
   The password you entered differs from your secure
   RPC password. To reencrypt your credentials with
   the New login password, please enter your
old Secure RPC password: nisplus
New password:net345
Re-enter new password:net345
   NIS+ password information changed for ne0025
   NIS+ credential information changed for ne0025
```

以后再此修改密码时，一般都是第一种情况了。

2 网络基本操作

telnet 和 ftp 是最简单也是非常有用的操作，学会使用 telnet 的基本操作命令，能够从一台计算机登录到另一台计算机，并进行相应的操作；学会使用 ftp 的基本命令，进行两台计算机之间的相互文件传送。

2.1 telnet

telnet 的基本步骤如下：

(1) 利用所分配的帐号登录系统

(2) 使用 telnet 和命令登录到其它主机。

假设你在 net33 上登录，你可以通过下述操作登录到 net34 上

```
net33$ telnet net34                    ( telnet 后为远程主机名)
Trying...                              (以下是系统显示，这里只是作一说明，
Connected to net34.                    个别信息可能不同)
Escape character is ′^]′.
UNIX(r) System V Release 4.0 (net34)
Login: username                        (输入在远程机器上的帐号，请求登录)
Password:
… …
exit                                   退回本机
```

直接输入 telnet 并回车进入

telnet> 状态，再输入？回车可看到该 telnet 支持的全部命令集

2.2 ftp

使用 ftp 命令进行两台主机间的文件传送步骤如下：
假设你处于 net33，想与 net34 进行文件传送。

```
net33$ ftp net34                                    ( 直接输入远程主机名 )
Connected to net34                                  ( 以下是系统显示 )
220 net34 FTP sever (UNIX(r) System V Release
4.0)ready.
Name   (rhost:root): username
Password:                                           ( 进入 ftp 状态 )
ftp>
.....
                                                    退出 ftp 程序
ftp> quit
```

在 ftpt> 状态下输入？回车可看到该 ftp 支持的全部命令集

3 书写 makefile 工程文件

很多同学在 Unix 环境下编程都采用过类似下面的命令：

```
cc myprogram.c -o myprogram
```

这种方式对于简单的文件编译还能够发挥作用，当工程项目比较大，文件个数比较多，继续采用这种方式恐怕非常麻烦，还容易出错；如果写成一个类似批处理的 Shell 脚本，似乎方便、安全，但是如果只是修改了其中一个文件，为了得到新的版本，就要重新编译所有源文件，有时候在时间上是不允许的，诸如此类还有很多问题；而实际上，Unix 系统下都提供了工程维护机制，其中一个应用程序 make 就可以很好地解决上述问题，VC、Delphi 之类中的项目文件的作用大抵都来源于此。

make 程序帮助程序员完成程序编译，执行 make 命令，make 程序缺省去寻找 makefile 或 Makefile 文件进行工作，如果要使用其他文件则需要通过 make 的 -f 选项制定。首先看一个 makefile 的简单例子以了解其功能：

```
# sample makefile
targerfile: target.c target.h
    gcc   target.c -o targetfile
```

其中以 # 开头的行为注释，将被 make 程序忽略。makefile 由一系列“规则”构成，每个规则又部分

目标：依赖关系

动作

注意在 gcc（是一种 C 编译器的）之前**是一个 Tab（制表符），不是一系列的空格**。上面的例子描述了 targetfile 依赖于 target.c 文件和 target.h 文件，如果这两个文件任何一个更新了，那么将重新编译 targetfile，具体如何得到则通过下面一行“动作”指明：运行 gcc 编译 target.c，得到 targetfile。

再来看一个例子：

```
# sample makefile
targerfile: target01.o target02.o target.h
```

```
    gcc   target01.o target02.o   -o targetfile
target01.o: target01.c target01.h
    gcc   -c target01.c -o target01.o
target02.o: target02.c target02.h
    gcc   -c target02.c -o target02.o
```

在这个例子中，最终的目的是为了得到 targetfile，它依赖于 target01.o target02.o target.h 三个文件，如果这三个文件都有且时间早于 targetfile（如果已经存在的话），则不需做任何事情，只是显示 targetfile is up to date 之类的信息。现在假设 target01.o 文件不存在，则 make 继续查找如何得到 target01.o，根据如何得到 target01.o 的规则进行处理，完成后返回前一规则，最终得到 targetfile，如果过程中发现任何错误，则程序员需要根据错误信息进行修改。借助一些 Shell 脚本的知识，可以把 makefile 写得更加实用，比如：

```
#
#makefile for exp01 written by somebody 2000/10/15
#
CFLAGS = -g -c
LIB = -lsocket -lnsl
CC = gcc
TARGET = targetfile
${TARGET}: target01.o target02.o target.h
    $(CC) $(LIB)   target01.o target02.o   -o $(TARGET)

target01.o: target01.c target01.h
    $(CC) $(CFLAGS) target01.c -o target01.o

target02.o: target02.c target02.h
    $(CC) $(CFLAGS) target02.c -o target02.o
```

上面的例子中用到了一些 UNIX Shell 知识，比如变量赋值和宏替换。makefile 的功能可以非常丰富，非常强大强大，当然编写起来也更为复杂，有兴趣的同学可以查阅有关资料。

推荐同学们在做实验室，学写学用 makefile 和 make 程序。

4 编译器

实验环境中提供的 C 编译器为 gcc，编译选项和方法与 cc 类似。

5 注意事项

帐号和密码是用户在系统中的标识和使用凭证，关系到用户及系统的安全，要求同学维护好自己的帐号和密码。由于网络课提供的是一个实验环境，系统对于同学的使用限制较为宽松，希望同学们不要进行危害系统安全和性能的活动。如果在学习和实验中确有特殊要求，请及时与系统管理员和辅导教师联系，将尽力为同学提供条件和方便。

实验一 数据链路层协议的设计与实现

1 实验目的

计算机网络的数据链路层协议保证通信双方在有差错的通信线路上进行无差错的数据传输，是计算机网络各层协议中通信控制功能最典型的一种协议。

本实验实现一个数据链路层协议的数据传送部分，目的在于使学生更好地理解数据链路层协议中的“滑动窗口”技术的基本工作原理，掌握计算机网络协议的基本实现技术。

2 实验内容

在一个数据链路层的模拟实现环境中，用 C 语言实现下面两个数据链路层协议。

（1）“退回到 N 重发”的滑动窗口协议（参考文献[1]第四章的协议 5）；

（2）“选择重发”的滑动滑动窗口协议（参考文献[1]第四章的协议 6）；

3 模拟实现环境及其使用

数据链路层协议位于物理层之上，网络层之下。它使用物理层提供的服务，并且向网络层的分组数据传输提供可靠的服务。因此，实现一个数据链路层协议必须要有一个模拟实现环境。

这个模拟系统由以下几部分组成：

- 两个代码文件 sim.c 和 worker.c；
- 一个通用的头文件 common.h；
- 协议文件使用的头文件 potocol.h。

模拟系统使用三个进程：

- main: 控制整个模拟系统
- MO: 协议 2 和协议 3 的发送方（machine 0）
- M1: 协议 2 和协议 3 的接收方（machine 1）

文件 sim.c 中包含着主程序，它首先分析命令行并且将参数保存起来，接着创建六个管道使得三个进程之间能够进行通信，所创建的文件描述字以如下方式命名：

MO 和 M1 的通信；

w1、r1：MO 到 M1 的帧传递

w2、r2：M1 到 MO 的帧传递

Main 和 MO 的通信：

W3、r3：main 通知 MO go-ahead

w4、r4：MO 通知 main，MO 已准备好

Main 和 M1 通信：

w5、r5: main 通知 M1-go-ahead

w6、r6：M1 通知 main，M1 已准备好。

在创建管道之后，主程序 Main 创建两个子进程 MO 和 M1。MO 和 M1 将分别

调用所选中的协议子程序。所有的协议都被编译成二进制码，无需执行 exec。

每个协议都执行它自己的初始化并运行，实际上它是调用一个 wait_for_event（）函数来完成这项工作的，这个函数以及 protocol.h 中预定义的所有函数都在文件 worker.c 中。Wait_for_event（）函数设置一些计数器，接着读入从 MO 或 M1 传来的帧，此时将帧从管道中取出以防止管道阻塞。读入的帧被保存在数组 quene[]中，在需要的时候也可将其在 guene[]中删除。指针 inp 和 outp 分别指向 qneme[]中第一个空位置和下一个将被删除的帧。Nframes 保存 quene[]中帧的数目。

当输入管道一旦被“吸干”，wait_for_event()向 main 发送一个 4 个字节的消息来说明它已准备执行一个事件，此时它等待 main 发给它一个 go-ahead。

Main 选择一个 worker 来运行，并将当前时间通过文件描述字 w3 或 w5w 传给它做为 go-ahead 信号。Worker 根据从管道传来的值设置自己的时间，从而使两个 worker 保持时间上的同步。接着 worker 调用 pick_event()来决定返回哪个事件。对不同的协议来说，可能发生的事件表是不同的。pick_event()将检查哪些事件是可行的，并做出选择。例如，如果没有运行定时器，或者被模拟的协议不存在定时器，那么超时事件是不能被返回的；如果没有帧保存于 quene[]中，那以帧到达事件是不能被返回的。

一旦一个事件被返回，wait_for_event()将返回其调用者，即协议例程。这些例程可以调用 worker.c 中所有的库函数，它们完成管理时钟，向管道中写帧等操作。Worker.c 中的代码很简单并且有很多注释。

主程序也很简单。它选择进程并且向通信管道中写一个 4 个字节整数的时间值做为 go-ahead 信号。被选中的进程检查是否能运行。如果可以，那么返回代码 OK；如果现在不能运行且没有未关掉的时钟，那么返回代码 NOTHING；如果两个进程都返回 NOTHING，那么表示发生死锁。死锁被设定为超时间隔的三倍，可能过于保守，但能够消降错误的死锁事件。

模拟系统以命令行的方式执行，它包括整数类型的六个模拟参数：

sim protocol events timeout pct_loss pct_cksum debug_flags

其中

protocol 表示要运行哪个协议，例如 5；

events 表示模拟系统要运行多长时间；

timeout 给出超时间隔；

pct_loss 给帧的平均丢失率（0-99）；

pct_cksum 给出到达帧的平均出错率（0-99）；

debug_flags 给出几种调试方式。

模拟实现环境的源程序放置在相应的目录下，请同学们首先把这些文件拷贝到自己的用户目录下。

4 实验步骤和注意事项

实验按下述步骤进行：

（1）熟悉已给出的数据链路层协议模拟实现环境的功能；

（2）编写两个数据链路层协议的实现程序；

（3）在模拟实现环境下调试并运行自己编写的协议实现程序；

（4）了解协议的工作轨迹，如出现异常情况，在实验报告中写出原因分析；

（5）保留你实现的娄据链路层协议在你的用户目录下，以备辅导教师检查。

实验二 文件传输协议的设计与实现

1 实验目的

文件传送是各种计算机网络都实现的基本功能，文件传送协议是一种最基本的应用层协议。本实验环境已在 TCP/IP 的基础上提供了一个文件传送实现 FTP，希望通过本实验，同学能够了解它的具体实现细节。

本实验的目的是，学会利用已有网络环境设计并实现简单应用层协议，掌握 TCP/IP 网络应用程序基本的设计方法和实现技巧。

2 实验内容和要求

我们的计算机网络实验环境建立在 TCP/IP 网络体系结构之上。各计算机除了安装 TCP/IP 软件外，还安装了 TCP/IP 开发系统，使是各计算机具备了 4.3 BSD UNIX 中进程通信套接字 socket 的编程接口功能，可为用户提供全网范围的进程通信功能。本实验要求学生利用这些功能，设计和实现一个简单的 文件传送协议。具体要求是：

用 socket 编程接口编写两个程序，分别为客户程序(client.c)和服务器程序（server.c），服务器程序在后台运行后，运行客户程序，应能实现下述命令功能：

get 取远方的一个文件
put 传给远方一个文件
pwd 显示远主当前目录
dir 列出远方当前目录
cd 改变远方当前目录
? 显示你提供的命令
quit 退出返回

这此命令的具体工作方式（指给出结果的形式）可以参照 FTP 的相应命令，有余力的同学可以多实现几个命令。

最后，写出实验报告。

3 实验说明

SCO TCP/IP 的 socket 编程接口实现了 Berkeley 4.3 BSD Socket 机构。用户通过库程序 libsocket.a 中提供的功能来编写其进程通信程序。库程序 libsocket.a 包括系统调用的库子程序两部分，形成了面向用户的编程接口。

（1）系统调用

UNIX 连网的系统调用通过对套接定 Socket 的一系列操作来实现进程间的通信。这些系统调用直接涉及 UNIX 系统原语，包括：

UNIX Socket 连网的系统调用

Call	Description
accept(SSC)	accept a connection on a socket
bind(SSC)	bind a name to a socket
connect(SSC)	initiate a connection on a socket
getpeername(SSC)	get name of connected peer
getsockname(SSC)	get socket name

getsockopt(SSC)	getoptions on sockets
setsockopt(SSC)	set options on sockets
listen(SSC)	listen for connections on a socket
recv(SSC)	receive a message from a socket
recvfrom(SSC)	receive a message from a socket
send(SSC)	send a message to a socket
sendto(SSC)	send a message to a socket
shutdown(SSC)	shut down part of a full-duplex connection
socket(SSC)	create an endpoint for communication
select(SLIB)	await event on a socket

一个典型的使用套接字 Socket 完成进程间通信的工作过程如下所示。这时，客户进程（Client Process）是通过 Socket 进行通信的主动方，服务器进程（Server Process）是通过 Socket 进行通信的被动方。

Serving Process	Client Process
socket()	socket()
bind()	bind()
listen()	
accept()	connect()
read()	write()
write()	read()
close()	close()

（2）库子程序

UNIX 连网的库子程序是其系统调用的补充，主要用于网络地址映射和操作，包括：

a 主机名到网络地址的映射；

b 网络名到网络号的映射；

c 协议名到协议号的映射；

d 服务名到端口号以及相应协议的映射。

4 实验帮助

有几个用 socket 编写的程序例子可供参考，存放在相应的 exp3/目录下。

不同机器之间两个应用程序的通讯，可通过给出主机地址及端口号，利用 socket 系统调用建立一个套接字并进行联接，然后就可用“读——写”方式进行两得间的通讯。其中主机地址可用库了程序转换，端口号可由用户指定。

常用的 socket 系统调用

（1）socket 创建套接字

一般的调用形式：

```
int sockfd:   /*套接字描述符*/
sockft=socket(AF_INET, SOCK_STREAM, 0)
```

（2）close 关闭套接字

```
close(sockfd);
```

（3）bind 确定本地地址并告知系统，一般只在服务程序中调用

```
bind（int sockfd, struct sockaddr *addr, int addrlen）;
```

其中第二个参数是指向一协议专用的地址指针，第三个参数是此地址结构的大

小；

（4）connect 联接一个套接字，只在客户程序中调用。

connect(int sockfd, struct sockaddr *servaddr, int addrlen)；

（5）listen　确定服务器队列长度，只在服务器程序使用。

listen（int sockfd, struct sockaddr *addr, int *addrlen）

（6）read　从套接字读数据

read（int sockfd, char *buf, int buflen）；

（7）write 向套接字写数据

write（int sockfd, char *buf, int buflen）；

（8）　accept 等待并接收连接，只在服务器程序中调用。

accept (int sockfd, struct sockaddr *addr, int　*addrlen)

常用的库子程序

本实验用的是主机名到网络地址的映射，实现的函数为：

struct hostent *hp；　/*特定的地址结构*/

hp=gethostbyname（char *hostname）；

其中 hostname 为主机名，如 rhost 等。

地址结构 sockaddr 有三个要赋值的域，一般可用以下语句：

#define SERV_PORT 2001 /*设定端口号*/

struct sockaddr *addr；

……

addr.sin_family=AF_INET

addr.sin_port=htons(SERV_PORT)；

对于客户程序：

bzero((char *)&addr, sizeof(addr))

bcopy(hp->h_addr, (char.*)&addr. sin_addr, hp->length)；

其中 hp 是上述调用 gethostbyname 的返回结果

对于服务程序：

addr.sin_addr.s_addr=htonl(INADDR_ANY)；

其中 INADDR_ANY 表示可接受来自任何主机客户的请求。

5 注意事项

(1) 关于端口号（假设用 SERV_PORT 来表示）的设定，原则上 2000 至 5000 都可用，为避免冲突，建议取你学号后三位数加上 2000，比如学号为 971234，则可定义：　#define SERV_PORT 2234

(2) 客户和服务程序中要有相应的 include 文件（参考所给例子程序），并要求在同一台机器测试。

(3) 编译程序需连接 socket 库（-lsocket）。

(4) 有些同学的 server 方程序支持多连接，为了不占用更多的系统资源，连接数限制在 3 个以内。

实验三 协议状态机的简单实现

1 实验目的

状态机是协议描述的重要方式之一，也是协议实现中的重要依据，简单说来，协议的状态机由一组变迁构成，每个变迁的形式为：

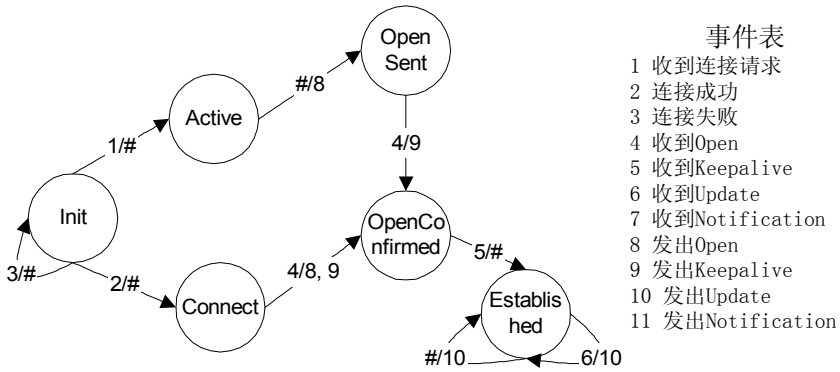
当前状态 ＋ 输入 －－> 输出 ＋ 下一个状态

其含义为，在“当前状态”下，收到了某个特定的“输入”，比如收到了一个数据帧，或发生了一个特定事件（比如超时），则系统产生一个输出，比如发出一个数据帧，同时状态进入“下一个状态”。完善的状态机的描述是高质量的协议描述的必要条件，同样，完善的状态机实现是保证高质量协议实现的必要条件。在本实验中将要求同学实现一个简单的协议状态机，从而加深对协议状态机描述的认识，同时对于实现协议状态机有一个初步的认识。同时，实验内容取材于一种重要的路由协议，通过这个实验，同学们也可以增进对于路由协议的理解。

2 实验内容和要求

2.1 协议行为简述

这是基于实际路由协议[1]，根据网络课实验的需求进行简化处理之后得到的一个协议。主要过程是通信双方经过建立连接，进行协商，最终进入稳定状态交互信息。协议实现（Protocol Implementation 简称 PI）最初处于初始状态，由于它要具有建立连接的能力，所以它要具有主动和对方发起连接请求的能力，又要具有等待、接收对方发出连接请求的能力。以后一种情况为例，PI 收到对方的连接请求之后，发出一个 Open 帧（帧格式定义在稍后给出），并等待对方发一个 Open 帧，收到后，发出一个 Keepalive 帧，完成协商，然后开始用 Update 帧交互信息。在这一过程中如果发生了任何错误，则进行相应的错误处理。这样的描述显然不够精确，因此，协议行为常常用状态机来进行说明。实验中要求实现的协议状态机如下图所示：



协议状态图

图中有 Init，Active，OpenSent，Connect，OpenConfirmed，Established 六个状态，状态之间的箭头表示了状态之间的变迁。箭头上"a / b"表示的形式表示了该变迁发生的输入和相应的输出。以 OpenSent 到 OpenConfirmed 之间的变迁为例，在 OpenSent 状态下，当发生事件 4（收到 Open）帧，则发出 Keepalive 帧（事件 9），进入状态 OpenConfirmed。应当注意的是，输入输出可以是多个事件，比如 Connect

到 OpenConfirmed 的变迁中 4/8,9 表示输入为 4，输出 8，再输出 9，状态进入下一状态；也可能为空用＃表示，比如 1/#，表示输入为 1，没有输出，＃/10 表示没有输入，产生输出＃，这通常是协议实现要“主动”地进行某些操作。

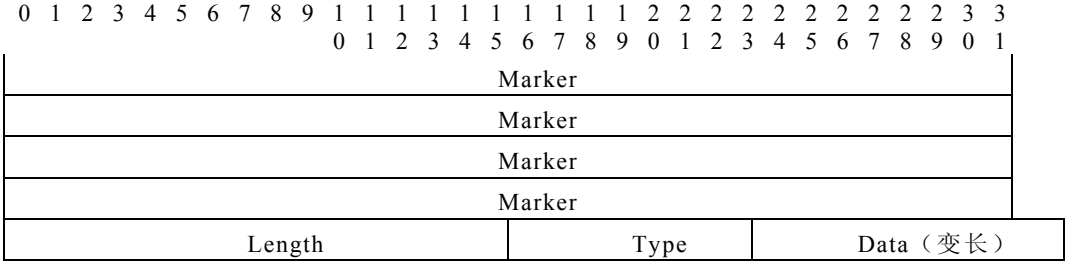
需要说明的是，为了清晰起见，很多有关错误处理变迁在图中没有画出，这些变迁规定如下：

在 Init，Active， Connect 状态下，如果发生错误，则返回 Init 状态，这里通常都是底层通信、建连时发生的错误。

在其他状态下，如果发生错误，比如收到错误的帧，则应当发出一个 Notification 帧之后，关闭通信连接，返回 Init 状态。如果在这些状态收到 Notification 帧，则关闭连接，返回 Init 状态。应当注意的时，“错误的帧”可能有两类，一是帧本身有错：类型错误、长度错误和数据错误等，也可能是“不适时”的帧，所谓不适时的帧就是该帧本身没有错，但是“来的不是时候”，在本不应当出现的状态下出现了。对于这两类错误都应当考虑。实际上，在协议实现中，往往对错误情况的处理要比对正确情况的处理复杂的多，这也是保证协议实现正确性、鲁棒性的保证，通过这个实验，同学们在这方面也会有所体会。

2.2 帧结构定义

实验中定义并使用 4 种帧，OPEN、KEEPALIVE、UPDATE、NOTIFICATION。为了便于实现，帧格式定义如下：



其中第一行为比特数，0－31，下面四行是 16 个 FF（16 进制）字节，Length 域为 2 字节无符号整数，是长度域，记录整个帧的长度。Type 域为 1 字节无符号整数，说明帧的类型，其值共有四种：

值	类型
1	OPEN
2	UPDATE
3	NOTIFICATION
4	KEEPALIVE

Data 是变长数据域，可以是 0 个字节或多个字节。因此，Length 域最小为 19（十进制），此时 Data 域为空。

2.3 实验要求

1 使用 TCP 服务作为底层通信支持。

2 程序在 ServerPort（比如学号后三位＋2000）上监听连接请求，如果有连接请求到来则按照相应的状态变迁进行会话，至 Established 状态，并交互信息。

3 程序还要能够主动和指定的目的地址试图建立连接，如果连接请求成功，则按照相应的状态变迁进行会话，至 Established 状态，并交互信息。

4 程序能够进行重复连接检测和消除。即在一对通信实体间，2 和 3 所建立的连接只能保留其中一个，如果发生重复连接，程序应当能够发现并关闭其中一个。

5 程序可以处理四种帧，能够发现错误并发出 Notification 帧，Notificaiton 帧中的 Data 域应当指明错误原因，其他帧的 Data 域填充收到的前一个对方发送帧的 Data 域。

6 监听端口和对方地址应当可以作为参数制定。

3 实验说明

本实验的背景来自一个重要的路由协议 BGP－4[1]的状态机部分，在实验里进行了大量的简化。比如事件处理上，忽略了很多有关时钟的操作，在帧格式上，忽略了各个帧具体的内容，在 BGP-4 协议中，帧结构是非常复杂的，对于帧的解析需要大量处理。应当说明的是，对于路由协议而言，协议行为和特性的很大一部分是在协议进入 Established 状态之后进行的，路由器之间会交互、处理大量的路由信息，而在状态机上只表现为从 Established 状态到本身的自环。因此，协议的状态机描述对于某些协议来说并不充分，需要探索、使用新的描述方法；路由协议的特点是有大量的信息处理。这两点希望同学们也有所认识。

4 实验帮助

如何实现状态机可能需要一些认真的考虑，可以通过大量的条件判断实现：

```
 if ((current_state ==OPEN) && (input_event == RECV_OPEN))
{
    send_out(UPDATE);
    current_state = OPEN_SENT
}
else
  {
… …
   }
```

也可以采用一种表式的结构：

输入事件 / 当前状态	事件 1	事件 2	事件 3	……
状态 1	输出 11，下一状态 11	输出 12，下一状态 12	……	……
状态 2	输出 21，下一状态 21	输出 22，下一状态 22	……	……
……	……	……	……	……

实际上，这也是协议的另外一种描述方式，即所谓状态表。在实现上，通过一个类似二维数组的方式就可以将该表组织起来。

重复连接检测和消除可能也需要一些考虑。由于程序要同时具有被动相应连接请求的能力和主动发起连接的能力，所以程序中要有两类的 Socket：Server Socket 和 Client Socket，那么这两个 Socket 是在一个进程中实现还是在两个进程中实现呢？如果在一个进程中实现，则要考虑，在一般情况下 Server Socket 在 listen 调用之后，就进入阻塞状态，在收到连接请求之前，程序不会继续往下执行，所以必须采用一些方法（比如设定 Socket 的参数、选项，或时钟）保证在一段时间之后如果没有连接请求则退出等待，进行下面的工作，比如试图主动和对方发起连接请求（使用 Client Socket）。如果在两个进程中实现，则需要解决进程间通信、确

定使用哪个 Socket 进行会话。

5 参考文献

[1] Rekhter Y, Li T. A Border Gateway Protocol 4 (BGP-4). RFC 1771, 1995

选做实验一 传输控制协议的简单实现

1 实验目的

TCP/IP 协议是目前在 Internet 上主要采用的传输技术。TCP 协议在 TCP/IP 协议族中是比较复杂的。它为两个任意处理速率的、使用不可靠 IP 连接机制的机器之间的通信提供可靠的、具有流量控制的、端到端的数据流服务，在整个的网络体系结构中占有非常重要的地位。TCP 使用 IP 来携带数据。每一个 TCP 消息封装在一个 IP 数据报后通过互联网。当数据报到达目的主机，IP 将数据报的内容传给 TCP。

从应用的角度来看，TCP 提供的服务有七个主要特征：

（1）面向连接（Connection Orientation）。

TCP 提供的是面向连接的服务，一个应用程序必须首先请求一个到目的地的连接，然后使用这一连接来传输数据。

（2）点对点通信（Point-To-Point Communication）。

每一个 TCP 连接有两个端点。

（3）完全可靠（Complete Reliability）。

TCP 确保通过一个连接发送的数据按发送是时一样正确地传递，且不会发生数据丢失或乱序。

（4）全双工通信（Full Duplex Communication）。

一个 TCP 连接允许数据在任何一个方向上流动，并允许任何一个应用程序在任何时刻发送数据。

（5）流接口（Stream Interface）。

TCP 提供了一个流接口，一个应用利用它可以发送一个连续的字节流穿过连接。TCP 不确保数据传递到接收端时会与发送端有同样尺寸的段。

（6）可靠的连接建立（Reliable Connection Startup）。

TCP 要求当两个应用创建一个连接时，两端必须遵从新的连接，前一次连接所用的重复的包是非法的，不会影响新的连接。

（7）完美的连接终止（Graceful Connection Shutdown）。

一个应用程序能打开一个连接，发送任意数量的数据，然后请求终止连接。TCP 确保在关闭连接之前传递的所有数据的可靠性。

本实验的目的是让学生了解 TCP 协议的机制及其实现结构，加深对于 TCP 中“三次握手”、“滑动窗口”等概念的理解。

2 实验要求

在一个网络层（IP）的模拟实现环境下，用 C 语言实现 TCP 协议（参见 RFC793）规定的基本功能。

如果实现一个完全的 TCP，需要的代码量还是很大的。为了减轻同学的负担，现将对于实现功能的简化说明如下：（以下使用的一些 TCP 术语请参见 RFC793）

1) 不要求对于 TCP 的复用。实现的 TCP 只要支持一个连接即可。
2) 不要求对于 TCP 选项（Option）的处理。这里主要避免了“最大段长”（MSS：Maximum Segment Size）的协商问题。MSS 将由宏定义 TCP_MSS 给出。（See

Page 18 in RFC793）

3) 在控制位（Control Bits）中，只要支持 SYN、FIN、ACK，不用考虑 RST、URG、PSH。（See Page 16 in RFC793）
4) 不考虑 TCP 报文失序到达的问题，确认机制使用“否定性确认”机制（NAK：Negative Acknowledgement）。
5) 只考虑滑动窗口的控制而不考虑拥塞控制，即不实现 Slow Start 算法。
6) 超时重传时间的时间使用常数，不用动态计算。

3 模拟实现环境及其使用

在网络体系结构中，TCP 位于传输层。它使用网络层提供的接口，为上一层的应用程序提供服务。所以，所实现的 TCP 将使用本实验环境中提供的 IP 层接口，并向上提供一定的接口，提供服务。

3.1 向下的接口

在本实验环境中，提供了一个模拟的 IP 接口。主要是两个接口函数：ipsend()和 iprecv()。

(1) ipsend()的调用格式为：

```
int ipsend(faddr, data, datalen)
IPaddr  faddr;  /* 目的地的 IP 地址 */
char    *data;      /* 所要发送 ip 数据的内存首地址 */
int  datalen;       /* 所要发送 ip 数据的长度  */
```

ipsend()是非阻塞的，即调用后就立即返回。

(2)iprecv()的调用格式为：

```
int iprecv(faddr, data, datalen)
IPaddr  faddr;  /* 要接收的 IP 地址 */
char    *data;      /* 用于存放接收到的 ip 数据的内存首地址 */
int  *datalen;  /* 接收到的 ip 数据的长度  */
```

iprecv()是阻塞的，如果 IP 没有数据要交给 TCP，则进程会阻塞在这里。

3.2 向上的接口

这里定义的接口仅仅供参考。同学可以自己定义接口，只要能够实现以下测试所需的功能即可。可以参考 UNIX 中 Socket 接口的定义。

TCP 实现向上提供的接口格式为：

（1）int tcp_write(sock, pch, len)

```
int sock;   /* 连接的 socket 号，这里没有实际意义 */
char pch;   /* 所要发送数据的内存首地址          */
int len;    /* 所要发送数据的长度              */
```

通过 TCP 发送数据。返回实际发送的数据长度。

（2）int tcp_read(sock, pch, len)

```
int sock;   /* 连接的 socket 号，这里没有实际意义 */
char   *pch;  /* 保存接收数据的内存首地址          */
int len;    /* 准备读取的数据长度              */
```

从 TCP 读取数据。返回实际读取的数据长度。

（3）int tcp_close(sock)

```
int sock;   /* 连接的 socket 号，这里没有实际意义 */
```

关闭连接。

（4）int tcp_bind(sock, name, namelen)

```
int     sock;   /* 连接的 socket 号，这里没有实际意义 */
struct sockaddr*name;
int     namelen;
```

绑定本机地址。

（5）int tcp_listen(sock)

```
int sock;
```

监听功能。

（6）int tcp_accept(sock,name,namelen)

```
int     sock;
struct sockaddr*name;
int     namelen;
```

等待并接收连接。

（7）int tcp_connect(sock,name,namelen)

```
int     sock;
struct sockaddr*name;
int     namelen;
```

联结一个套接字。

3.3 测试

为了检验 TCP 实现的正确性，需要利用 TCP 向上提供的接口编制两个应用程序。一个作为 Server，一个作为 Client。

测试的过程为：

（1）启动 Server，使 Server 进入 listen 状态。

（2）启动 Client，与 Server 建立连接。

（3）Client 向 Server 传输一定数量的数据。

（4）Client 主动关闭连接。

（5）Server 关闭连接。

以上过程实际上是一个标准的 Client-Server 的通信过程。这里涉及了 TCP 的主动打开（Active Open）、被动打开（Passive Open）、连接的建立（三次握手）、数据的传输（包括滑动窗口的处理）、连接的关闭（三次握手）等内容。由于我们规定的向上的接口与一般的 Socket 接口很类似，所以大家可以象编写一般的 Socket 程序那样来编写测试程序。

请注意，这里的测试是很初步的，实际的 TCP 实现作为系统软件的一部分，对其健壮性有相当高的要求，需要考虑许多特殊的情况。这里为减少本次实验的工作量，忽略这些特殊情况。

测试程序的编写可以参见查阅有关 Socket 编程的资料。

3.4 编译和运行

在 UNIX 系统下编制程序，没有很好的集成环境，对程序编译和链接需要自己书写 Makefile 文件。本实验中使用的编译工具是 gcc。关于 Makefile 的书写和 gcc 的使用可以参见提供的范例。

选做实验－附录 TCP 协议的实现范例

为便于同学学习和研究，提供在 Xinu 操作系统上的 TCP 实现的 C 源码，以便参考。

如果有条件，可以参阅 Douglas E. Comer 所著的《Internetworking With TCP/IP》Vol II：Design,Implementation,and Internals(Second Edition)（中文名称为：《用 TCP/IP 进行网际互连》 第 2 卷：设计、实现和内部构成（第 2 版））中有关 TCP 实现的部分，相信对于完成本次实验会有相当大的帮助。

1 实现结构

TCP 的实现结构一般为 3 个进程：

- ✧ Input 进程：负责处理到来的 TCP 报文。
- ✧ Output 进程：负责发送 TCP 报文。
- ✧ Timer 进程：负责管理 TCP 的时钟。

这 3 个进程间有一些共享数据（如 TCB：Tcp Control Block），所以不能直接采用 UNIX 系统提供的进程。建议用 UNIX 下提供的线程来实现以上的三个进程(线程调用的接口请参见附录）。当然，如果有兴趣的话，也可以研究利用 UNIX 下进程的通信机制来实现数据的共享。

在测试时，用户的测试程序（Client 和 Server）也应当作为线程，与以上的三个 TCP 线程共同运行于一个 UNIX 进程中。

2 数据结构

在 TCP 实现中会使用到一些数据结构，可以参见 h/tcp.h。对于其中两个比较重要的数据结构下面作一些说明，仅作为实现时的参考。

TCB（Tcp Control Block）

见 h/tcp.h 中 struct tcb 的定义。这里 tcb 的定义主要参考 Xinu 下 TCP 的实现，根据本次实验的需要，作了一些简化。如果有兴趣，也可以直接参考 Xinu 的源代码（位于 xinu/h/tcb.h 中）。tcb 中各个数据项的意义在注释中已经有了比较详细的说明，这里不再重复。

tcp 报文

见/ /tcp.h 中 struct tcp 的定义。这里 tcp 的定义与 Xinu 中的定义基本相同。只有数据项 tcp_data 与 Xinu 中的定义不同。

3 推荐参考文献

1． RFC793 文档

见提供的文件 document/RFC0793.txt。

2． UNIX 系统下关于线程的调用

UNIX 系统下使用线程主要用两个系统调用。一个是 pthread_create（），功能是在进程中创建一个新的线程。另一个是 pthread_join（），功能与 UNIX 系统调用 wait()类似。

使用这两个系统调用需要在程序中加入：

```
#include <pthread.h>
```

具体的使用方法，请参见所提供的范例（/paradigm/thread.c）。

3．UNIX 系统下关于信号量的使用

在实现 TCP 时，不可避免的会出现多个进程（注：这里的进程概念与 UNIX 的进程概念不同）对某个数据单元的共享访问。为避免发生访问冲突，必须使用某种机制以实现互斥访问。这里我们推荐使用信号量。在 UNIX 下，进程的信号量和线程的信号量使用不同的系统调用。因为我们假定实现是基于线程的，所以这里只提供关于线程信号量的说明。关于进程信号量的系统调用，请查阅相关的文档。

UNIX 下提供有关线程的系统调用包括：sema_post（）、sema_wait（）、sema_init（）、sema_destroy（）等。有关这些系统调用的具体说明，请查阅 UNIX 的联机手册。

为了便于同学使用，我们使用以上系统调用，重新提供了一套接口：s_init()、s_create()、s_signal()、s_wait()、s_count()、s_reset()。这些接口的实现，请查看文件 /experiment/sem/sem.c 。关于接口的说明，请查看文件 /experiment/sem/sem_doc.txt。

4．Xinu 系统中 TCP 的实现源码

xin/目录下提供了 Xinu 系统部分关于网络底层支持的源代码，其中目录 xinu/net/tcp 和 xinu/net/tcpd 下是有关 TCP 实现的，同学实现 TCP 时可以参考。有关的头文件在 xinu/h 下。

5．Makefile 和 gcc 的书写

参见 paradigm/Makefile 及实验准备