

## 以Internet为例,开始应用层的学习...

### Part1: Internet及其应用开发基础

- 应该了解的Internet知识
  - 4层设计模型
  - Tcp/udp是个什么样的服务?
  - socket套接字接口
- 应用开发基础
- 使用TCP/UDP的Socket 程序设计

### Part2: DNS, C/S应用 (WWW, ...)

### Part3: P2P应用 (BT, ...)



# Computer Networks

## Chapter 7: Internet及其应用开发基础 (1/3)

(Version February 27, 2023)

Xiao Mingzhong

CIST, Beijing Normal University

## Outlines

### □ What is Internet?

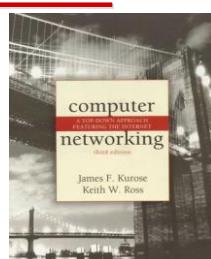
### □ 应用开发基础

### □ Socket程序设计

- 使用TCP

- 使用UDP

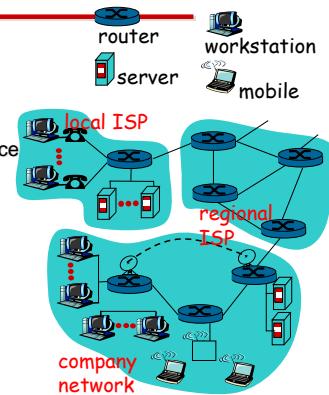
- 例子



## What's the Internet

### □ Communication infrastructure enables distributed applications:

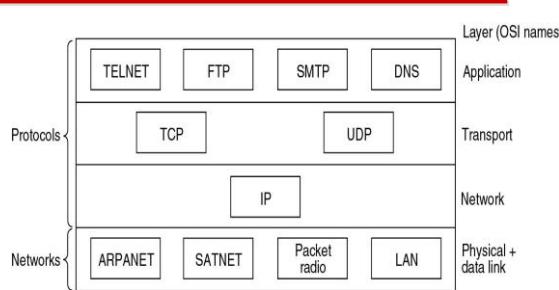
- Web, email, games, e-commerce, file sharing



### □ Communication services provided to apps:

- Connectionless unreliable
- connection-oriented reliable

## Internet的体系结构



出于互联的目的，诞生了IP协议，对等体标识叫IP地址。互联+路由+拥控是问题。



## 无连接的服务

**Goal:** 两端系统间的数据传输

- same as before!

**□ UDP - User Datagram Protocol [RFC 768]:**

- 无连接
- 不可靠数据传输
- 无流控
- 无拥塞控制

**Apps using TCP:**

- HTTP (Web), FTP (file transfer), Telnet (remote login), SMTP (email)

**Apps using UDP:**

- streaming media, teleconferencing, DNS, Internet telephony



## 面向连接的服务

**Goal:** 两端系统间的数据传输

**TCP 服务[RFC 793]**

**□ 握手(handshaking):** 数据传输之前的准备工作

- Hello, hello back human protocol
- set up "state" in two communicating hosts

**□ TCP - Transmission Control Protocol**

- Internet's 面向连接的服务

**□ 可靠的字节序的数据传输**

- 确认和重传机制

**□ 流控flow control:**

- sender won't overwhelm receiver

**□ 拥塞控制congestion control:**

- 当网络拥塞时发送者降低发送速度
- 迷惑?



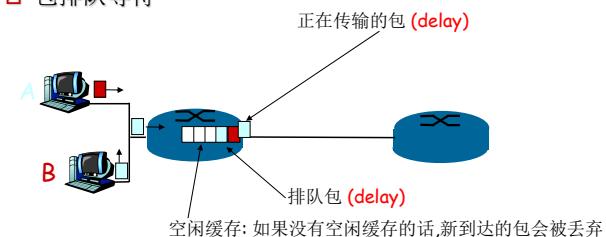
## Internet 传输有延迟和丢包问题

**How do loss and delay occur?**

包在路由器缓存中需要排队

**□ 包的到达速率大于出口速率**

**□ 包排队等待**



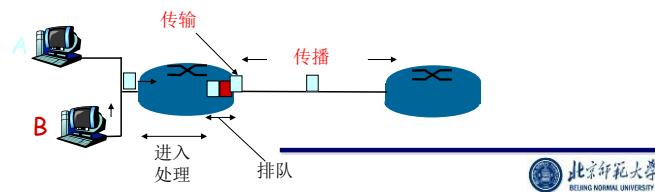
## 包延迟四因素

### 1. 进入处理

- 检错
- 确定输出链路

### 3. 传输延迟:

- $R$ =链路带宽(bps)
- $L$ =包长度 (bits)
- time to send bits into link =  $L/R$



## 一个例子

traceroute: gaia.cs.umass.edu to www.eurecom.fr

三次延迟测量 from  
gaia.cs.umass.edu to cs-gw.cs.umass.edu

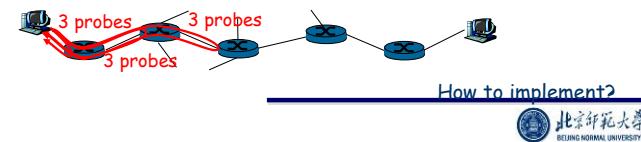
```
1 cs-gw (128.119.240.254) 1 ms 1 ms 2 ms
2 border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145) 1 ms 1 ms 2 ms
3 cht-vbns.gw.umass.edu (128.119.3.130) 6 ms 5 ms 5 ms
4 jn1-at1-0-0-19.wor.vbns.net (204.147.132.129) 16 ms 11 ms 13 ms
5 jn1-so7-0-0.wae.vbns.net (204.147.136.136) 21 ms 18 ms 18 ms
6 abilene-vbns.abilene.ucaid.edu (198.32.11.9) 22 ms 18 ms 22 ms
7 nycm-wash.abilene.ucaid.edu (198.32.8.46) 22 ms 22 ms 22 ms
8 62.40.103.255 (62.40.103.253) 104 ms 109 ms 106 ms 跨海链路
9 de2-1.de1.de.geant.net (62.40.96.129) 109 ms 102 ms 104 ms
10 de.fr1.fr.geant.net (62.40.96.50) 113 ms 121 ms 114 ms
11 renater-gw.fr1.fr.geant.net (62.40.103.54) 112 ms 114 ms 112 ms
12 nio-n2.cssi.renater.fr (193.51.206.13) 111 ms 114 ms 116 ms
13 nice.cssi.renater.fr (195.220.98.102) 123 ms 125 ms 124 ms
14 r3t2-nice.cssi.renater.fr (195.220.98.110) 126 ms 126 ms 124 ms
15 eurecom-valbonne.r3t2.ft.net (193.48.50.54) 135 ms 128 ms 133 ms
16 194.214.211.25 (194.214.211.25) 126 ms 128 ms 126 ms
17 *** 表示无响应 (probe lost, router not replying)
18 *** 
19 fantasia.eurecom.fr (193.55.113.142) 132 ms 128 ms 136 ms
```



## Internet的延迟和路由情况

□ **Traceroute程序:** 提供E2E因特网路径和延迟情况测量的工具. 原理:对于所有路由器  $i$ :

- sends three packets that will reach router  $i$  on path towards destination
- router  $i$  will return packets to sender
- sender times interval between transmission and reply.



## 包丢失

□ 输出链路队列满就会引发路由器丢包, 有可能没满就开始丢包哦.

□ lost packet may be retransmitted by previous node, by source end system, or not retransmitted at all



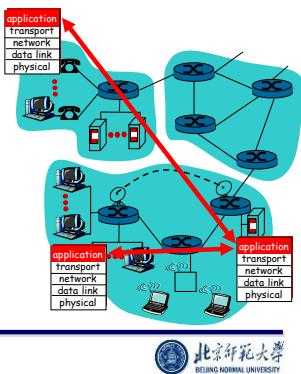
## 网络编程

就是写程序:

- 运行在不同的端系统上,且
- 通过网络通信
- e.g., Web: Web server software communicates with browser software

与子网设备无关:

- 子网设备没有应用代码
- 应用能得以快速开发和使用

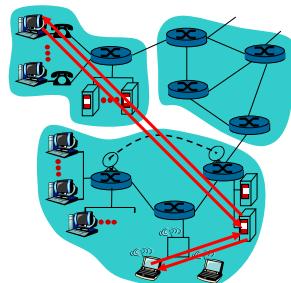


## 应用架构模型

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P



## C/S架构



server:

- 永远在线的主机
- 永久IP地址
- 可以是机群for scaling

clients:

- 跟服务器通信
- 可能是中转通信
- 可以是动态IP地址
- 用户间不通信

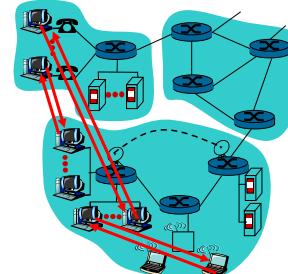


## 纯P2P架构

- 没有总是在线的服务器
- 任意端系统间可以直接通信
- Peers间可以中转通信及不使用固定的IP地址
- example: Gnutella

Highly scalable

But difficult to manage



## 混合型架构

### Napster

- 文件传输是P2P
- 文件发现是集中式的:
  - Peers把自己要共享内容的元数据发送到中央服务器
  - Peers query same central server to locate content

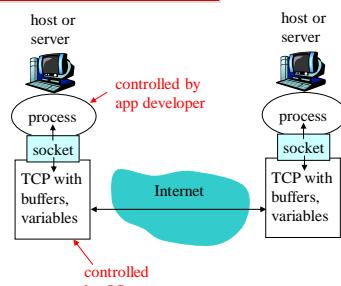
### Instant messaging(IM,即时通信)

- 两个用户之间的聊天是P2P(直接收发, NAT穿透例外)
- 位置信息是集中式的:
  - 用户上线就告诉中心服务器他的IP地址
  - 通过中心服务器获取伙伴的IP地址



## Sockets(套结字接口)

- 进程通过它的socket收发消息 messages
- socket类比门(door)
  - sending process将 message踢出door(它的socket)
  - 网络会把消息送到接收进程的门socket
  - 接口的比喻
- Socket是什么呢?
  - 一个整数 //后述
- API: (1) choice of transport protocol; (2) ability to fix a few parameters (*lots more on this later*)



## 进程间通信

**Process:** 程序的一次运行.

- 同一主机进程间通信使用 OS提供的 **inter-process communication** 技术

- 不同主机间进程通信通过 交换消息(**exchanging messages**) //按协议来

**Client process:** 发起通信的进程.

**Server process:** 等待联系的进程.

- Note: P2P应用主机同时拥有 **client processes & server processes**



## 进程编址 Addressing processes

- **For a process to receive messages, it must have an identifier**

进程Id包括 **IP address and port numbers**

- 一台主机有一个特有的 32位IP地址

Example port numbers:

- HTTP server: 80
- Mail server: 25

More on this later

- Q: 主机的IP地址可以用 来标识一个通信进程么?

- Answer: No, 一台主机上 可以有多个进程.



## 什么是应用层协议？

定义:

- 交换消息的类型 e.g.,  
request & response  
messages
- 消息的语法: 消息中有哪些  
字段及其用途
- 字段值的语义 i.e., meaning  
of information in fields
- 进程何时及如何发送和应答  
消息的规则

开放协议:

- defined in RFCs
- e.g., HTTP, SMTP

私有协议:

- e.g., KaZaA



## 应用需要什么样的传输服务?

数据丢失容忍度

- 某些应用可忍受一些包的丢  
失 (e.g., audio)
- 有些要求100% 的可靠传输  
(e.g., file transfer, telnet)

带宽

- 有些应用对带宽有较高需  
求 (e.g., multimedia)
- 有些无所谓 ("elastic  
apps")

时间敏感性

- 某些应用要求低时延  
(e.g., IP电话, 交互式游  
戏)



## 常见应用对传输服务的需求

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no



## Internet: 应用、应用层协议和传输层协议

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage, Dialpad)	typically UDP



# 小结

- Internet is a communication infrastructure
  - TCP & UDP
    - Connection-oriented, reliable, byte stream transfer service
    - Connectionless, unreliable, datagram transfer service
  - Socket
    - process sends/receives messages to/from its **socket**
    - the IP address and **port numbers** associates with the process
  - Lost & Delay
- 应用开发基础
  - APP vs APP Pro.
    - **Types**, e.g., request & response messages
    - **Syntax**: what fields in messages & how fields are delineated
    - **Semantics**, i.e., meaning of information in fields
    - **Rules** for when and how processes send & respond to messages
  - Choice of TS & Arch for App Pro.
    - Timing, bandwidth & data loss → TCP/UDP
    - C/S,P2P,Hybrid



# Socket 程序设计 (Unix/linux)

- 1 套接字编程的基础
- 2 基于TCP协议的套接字编程
- 3 基于UDP协议的套接字编程
- 4 raw socket\*
- 5 Libpcap\*
- 6 libnet\*



## 1. 套接字编程的基础

- 套接字，也称Berkeley套接字，因为它源自Berkeley Unix。网络程序通过**socket**和**其它几个函数**的调用完成网络的连接以及数据的交换。
- 套接字是一个整型数，是一个用于通信的文件描述符，可以象普通的文件描述符一样来操作，向该描述符的读写操作实现网络之间的数据交换。
- 网络编程所需的知识和概念在很大程度上并不依赖于操作系统，他们都基于TCP/IP协议族，因此绝大多数函数在Windows等操作系统上仍可使用



## 套接字地址结构(应用层协议对等体标识)

```
typedef uint32_t in_addr_t; //32位无符号整数，用于表示网络地址
struct in_addr{
    in_addr_t addr; //32位IPv4地址
}
typedef uint16_t in_port_t; //16位无符号整数，用于表示端口号
struct sockaddr_in{
    uint_8 sin_len; //本结构类型长度，uint_8是8位的无符号整数类型
    sa_family_t sin_family; //套接字地址族，对于IP来说，其值=AF_INET
    int_port_t sin_port; //16位的TCP或UDP端口号
    struct in_addr sin_addr; //32位的IPv4地址
    char sin_zero[8]; //暂不用，总置为0
}
```



## 套接字地址的使用

- 处理socket地址的函数必须能够支持不同的协议族的套接字地址

- 通用套接字地址结构如下：

```
struct sockaddr{  
    uint8_t sa_len;  
    sa_family_t sa_family;  
    char sa_data[14]; //协议相关的地址, why 14?  
}
```

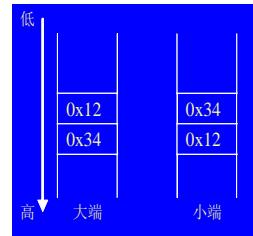
- 当把socket地址作为参数传递给函数时，都是通过结构指针来传递的，而指针则指向通用套接字地址结构。所以，编程时一般先用sockaddr\_in结构建立所需地址信息，当调用函数时再将其转化为指向通用套接字结构的指针。



## 字节序

- 对一个16位整数，它由2个字节组成，内存中存储这两个字节有两种方法：

- 小端(little-endian)字节序：将低序字节存储在起始地址；
- 大端(big-endian)字节序：将高序字节存储在起始地址。



- 把主机系统所用的字节序称为主机字节序(host byte order)

- 如：主机中存储时存在大端小端问题

- 网络协议所用的字节序称为网络字节序(network byte order)

- 网络传输时，先发送哪个字节的问题



## 函数支持

- Unix系统提供了一组函数，用于完成主机字节序和网络字节序之间的转换：

```
#include <netinet/in.h>  
■ uint16_t htons(uint16_t h)  
■ uint32_t htonl(uint32_t h)  
■ uint32_t ntohs(uint16_t n)  
■ uint32_t ntohl(uint32_t n)  
    • h:host; n:network; s:short; l:long
```

- 当使用这些函数时，不关心主机字节序和网络字节序的真实值。所做的只是调用适当的函数来对给定的值进行主机字节序与网络字节序间的转换，而系统的函数实现会根据系统的具体情况完成转换。

- 实际上，在主机字节序和网络字节序相同的系统中，这四个函数的实现是空的宏



## 地址转换函数

- 人们熟悉的是点分十进制IP地址（如：“192.168.1.1”），而上面的套接字地址结构中使用无符号整数存储IP地址。

- Unix系统提供了一组函数，用于在字符串类型的点分十进制地址与32位无符号整数间进行地址转换。

```
#include <arpa/inet.h>  
int inet_aton(const char *strptr, struct in_addr *addrptr); //点分地址到套接字地址  
in_addr_t inet_addr(const char *strptr); //功能同上  
char *inet_ntoa(struct in_addr inaddr); //功能与上相反
```

```
int inet_pton(int family, const char *strptr, void *addrptr);  
const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);  
//扩展支持地址族（如：IPv4, IPv6等），p：点分表示，n：套接字地址的32位表示
```



## 2. 基于TCP协议的网络编程 (1/2)

TCP客户和服务器编程所需要的基本套接字函数：

- **创建套接字函数**
  - int socket(int family, int type, int protocol)
    - family: 说明网络程序使用的通信协议族, 如: AF\_INET;
    - type: 与传输协议匹配的套接字类型, 如: SOCK\_STREAM(TCP套接字), SOCK\_DGRAM(UDP套接字), SOCK\_RAW(原始套接字);
    - protocol=0;
- **绑定函数 //给套接字分配一个本地协议地址**
  - int bind(int sockfd, struct sockaddr \*my\_addr, int addrlen)
    - my\_addr: 使用时, 需要将指向特定协议地址结构 (如: sockaddr\_in) 的指针转换为指向sockaddr的指针, 再传递给函数
- **监听函数 //仅被TCP服务器调用, 将sockfd设为监听套接字**
  - int listen(int sockfd, int backlog) //backlog:请求队列的最大长度
- **接受函数 //仅被TCP服务器调用**
  - int accept(int sockfd, struct sockaddr \*cliaddr, int \*addrlen)
    - 返回的全新描述符, 代表与客户的TCP连接 (服务器同时服务多用户的需要)

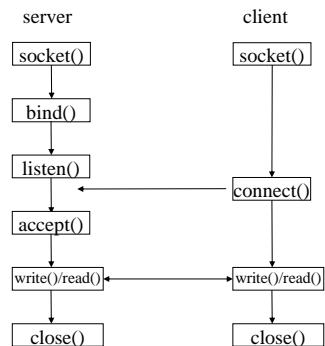


## 基于TCP协议的网络编程 (2/2)

- **连接函数connect //TCP客户用来建立与服务器的连接**
  - int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen)
    - TCP的三次握手过程被激发, 仅在连接成功或失败该函数才返回0或-1。
    - 成功, sockfd代表的是客户与服务器的连接。
    - 在该函数前, 不必调用bind, 内核自动选择本机IP地址和端口号。
- **连接中止函数close //关闭套接字, 中止TCP连接**
  - int close(int sockfd)
- **连接关闭函数shutdown //少用**
  - int shutdown(int sockfd, int howto)
- **写函数write**
  - ssize\_t write(int fd, const void \*buf, size\_t nbytes)
    - 将buf中的n字节写入fd, 返回成功写的字节数, 失败返回-1
- **读函数read**
  - ssize\_t read(int fd, void \*buf, size\_t nbytes)
    - 从fd读n字节到buf, 返回实际读的数量, 为0表示读到文件尾, 失败返回-1



## 基于TCP协议的网络程序结构



TCP server.c //在portnumber监听, 客户连接成功, 返回给客户欢迎信息。

```
(1) 创建套接字  
int sockfd, new_fd; //为什么要定义两个?  
struct sockaddr_in server_addr;  
struct sockaddr_in client_addr;  
sockfd=socket(AF_INET,SOCK_STREAM,0);  
  
(2) 套接字绑定特定地址和端口, 进入监听状态  
/* 服务器端设置 sockaddr 结构 */  
bzero(&server_addr, sizeof(struct sockaddr_in));  
server_addr.sin_family=AF_INET;  
server_addr.sin_addr.s_addr=htonl(INADDR_ANY);  
server_addr.sin_port=htons(portnumber);  
  
/* 绑定sockfd描述符 */  
bind(sockfd, (struct sockaddr *)(&server_addr), sizeof(struct sockaddr));  
listen(sockfd, 5);
```



续...

(3)循环等待并接收客户端连接，向客户发送欢迎信息

```
while(1)
{
    /* 服务器阻塞，直到客户程序建立连接 */
    sin_size=sizeof(struct sockaddr_in);
    new_fd=accept(sockfd, (struct sockaddr *)&client_addr,
                  &sin_size));
    write(new_fd, "hello",strlen("hello")); //发送信息
    close(new_fd); //该连接上的通信结束,关闭连接套接字
    /* 循环接收下一个连接 */
}
close(sockfd); //关闭整个服务器, 关闭监听套接字.
```



## Tcp client.c

```
struct sockaddr_in server_addr;
int portnumber,nbytes;
inet_aton(argv[1],&server_addr.sin_addr) //命令行读参数
portnumber=atoi(argv[2]))
sockfd=socket(AF_INET,SOCK_STREAM,0)) //建立套接字
server_addr.sin_family=AF_INET; //填充服务端的资料
server_addr.sin_port=htons(portnumber);
/*发起连接*/
connect(sockfd,(struct sockaddr*)&server_addr,
         sizeof(struct sockaddr))
/* 连接成功 */
nbytes=read(sockfd, buffer, 1024); //接收数据
/* 结束通信 */
close(sockfd);
```



## 说明

### □ 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
#include <stdio.h>
#include <errno.h>
```

### □ 编译

```
gcc -o tcpserver tcpserver.c
gcc -o tcpclient tcpclient.c
```

### □ 运行

```
./tcpserver 666
./tcpclient 192.168.1.111 666
```



## 3. 基于 UDP 协议的网络编程

同 TCP 程序相比, UDP 程序要简单得多, 客户端不需要先同服务器建立连接就可以直接发送数据, 所以 UDP 程序最常用的是数据收发函数。

### □ 常用的收发函数

### □ 基于 UDP 协议的网络程序结构

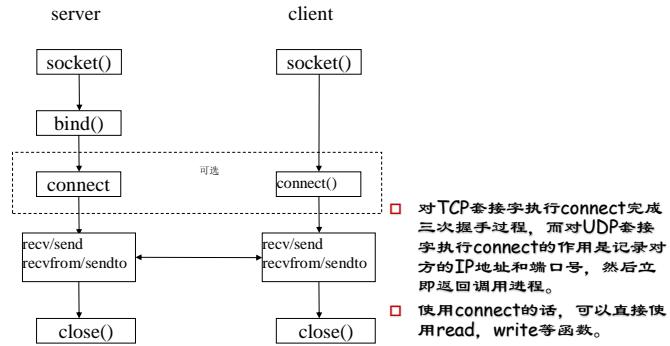
### □ UDP 程序示例



## 常用的收发函数

- int recvfrom(int sockfd, void \*buf, int len, unsigned int flags, struct sockaddr \* from, int \*fromlen)
- int sendto(int sockfd, const void \*msg, int len, unsigned int flags, struct sockaddr \*to, int tolen)

## 基于 UDP 协议的网络程序结构



## 例子：

- 设计实现一个 UDP server 和 UDP Client，服务器在端口 SERVER\_PORT 监听，客户端从控制台输入信息，发送到服务器，服务器返回给客户自己已经收到的信息。

- 服务器端：

(1) 创建套接字  
int sockfd;  
struct sockaddr\_in addr;  
sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

(2) 绑定在特定端口  
bzero(&addr,sizeof(struct sockaddr\_in));  
addr.sin\_family=AF\_INET;  
addr.sin\_addr.s\_addr=htonl(INADDR\_ANY);  
addr.sin\_port=htons(SERVER\_PORT);  
bind(sockfd,(struct sockaddr \*)&addr,sizeof(struct sockaddr\_in))

## 续...

- (3) 循环接收信息，返回应答信息

```
while(1)  
{ /* 从网络上读，写到网络上面去 */  
n=recvfrom(sockfd,msg,MAX_MSG_SIZE,0,(struct sockaddr*)&addr,&addrlen);  
msg[n]='\0';  
/* 返回给客户说明已经收到了信息 */  
fprintf(stdout,"I have received: %s",msg);  
sendto(sockfd, msg,n,0,(struct sockaddr*)&addr,addrlen);  
}
```

- (4) 关闭套接字

```
close(sockfd);
```



续...

□ UDP client

```
(1) 创建套接字
int sockfd, port;
struct sockaddr_in addr;
sockfd=socket(AF_INET, SOCK_DGRAM, 0);

(2) 设置服务器地址结构，循环发送和接收信息
bzero(&addr, sizeof(struct sockaddr_in));
addr.sin_family=AF_INET;
addr.sin_port=htons(port);
inet_aton(argv[1], &addr.sin_addr);
while(1){
    fgets(buffer, MAX_BUF_SIZE, stdin);
    sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr*)&addr, len);
    bzero(buffer, MAX_BUF_SIZE);
    n=recvfrom(sockfd, buffer, MAX_BUF_SIZE, 0, NULL, NULL);
    buffer[n]=\0;
    fputs(buffer, stdout);
}
close(sockfd);
```



## 说明

□ 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <strings.h>
#include <stdio.h>
#include <errno.h>
```

□ 编译

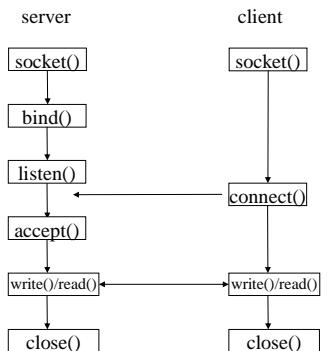
```
Gcc -o udpserver udpserver.c
Gcc -o udpclient udpclient.c
```

□ 运行

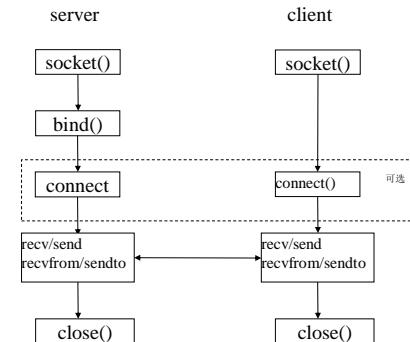
```
./udpserver 666
./udpclient 192.168.1.111 666
```



## 小结: 基于 TCP 协议的网络程序结构



## 小结: 基于 UDP 协议的网络程序结构



## 4. raw socket

### □ raw socket的作用：

- 通过raw socket来读写ICMP、IGMP协议包
  - 如: ping程序
- 通过rawsocket读写内核TCP/IP协议栈不能够处理的IP包
  - 因为OS不一定会实现完整的协议栈
- 使用rawsocket来发送一些自己指定IP头的特殊作用的IP包。
  - 例如: 对某台机器进行拒绝服务攻击, 但又不想让对方知道IP包的真正来源, 可用原始套接字发送伪造源地址的IP包。



## 5. libpcap

### □ libpcap是unix/linux平台下的网络数据包捕获函数包, 大多数**网络监控软件**都以它为基础。Libpcap可以在绝大多数类unix平台下工作

- libpcap提供了系统独立的用户级网络数据包捕获接口, 并充分考虑了应用程序的可移植性。libpcap可以在绝大多数类Unix平台下工作。在Windows平台下, 一个与libpcap很类似的函数包winpcap提供捕获功能



## 6. 网络数据包构造函数库 libnet

- libnet提供一系列的接口函数, 实现和封装了数据包的构造和发送过程。利用它可以自行构造从应用层到链路层的各层协议的数据包头, 并将这些包头与有效数据有序地组合在一起发送出去。
- libnet的开发目的是: 建立一个简单统一的网络编程接口以屏蔽不同操作系统底层网络编程的差别, 使得程序员精力集中在解决关键问题上。
  - 高层接口: libnet主要用C语言写成;
  - 可移植性: libnet目前可以在Linux、FreeBSD、Solaris、Windows NT等操作系统上运行, 并且提供了统一的接口;
  - 数据包构造: libnet提供了一系列的TCP/IP数据包的构造函数以方便用户使用;
  - 数据包的处理: libnet提供了一系列的辅助函数, 利用这些辅助函数, 帮助用户简化那些烦琐的事务性的编程工作;
  - 数据包发送: libnet允许用户在两种不同的数据报发送方法中选择。



## 小结

### □ 介绍了Unix/Linux平台上如何编写网络程序

- 首先介绍了如何使用STREAM和DGRAM套接字编写基于TCP/UDP协议的网络程序。
- 然后介绍几种**编写攻击防护程序**使用的编程技术。
- 原始套接字可以实现构造数据包和接收各种数据包的功能。
- libpcap和libnet是两个函数库, 分别用于实现数据包捕获和数据包的构造。



## 编程任务（选作）

---

### 1.socket接口

- 目的:掌握基于tcp/udp协议的socket程序设计模型
- 建议:参考课堂例子,自行实践。

### 2.http协议

- 目的:通过实现http c&s, 理解协议概念
- 建议:参考课堂例子,自行实践。

### 3.自己找个开源应用源码玩玩

---

