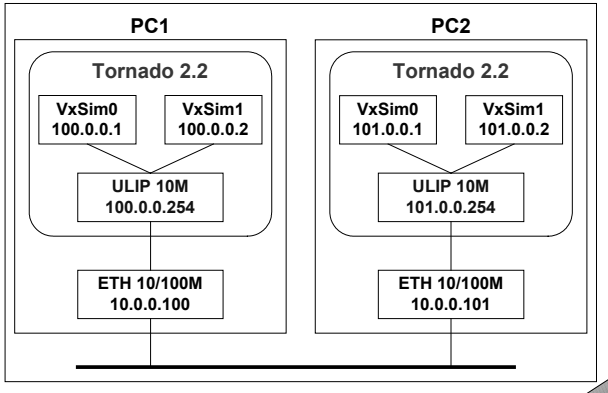
5.7 VxWorks 网络编程

1. VxWorks虚拟网卡应用
2. VxWorks网络概述
3. VxWorks网络程序设计
4. 远程访问服务

1. VxWorks网络应用示例

- VxWorks全仿真
- 安装ULIP虚拟网卡
- VxSim网卡的配置
- TFTP示例演示

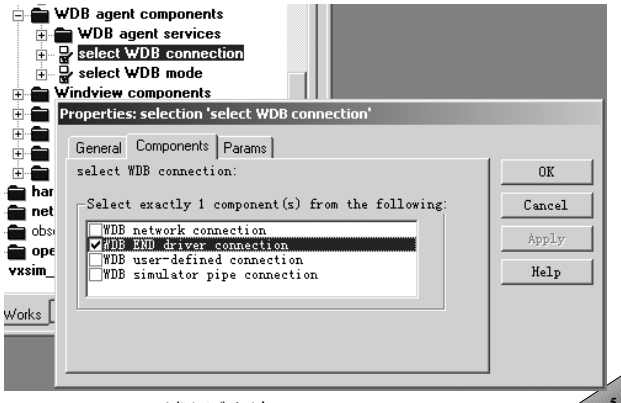
VxSim网络拓扑



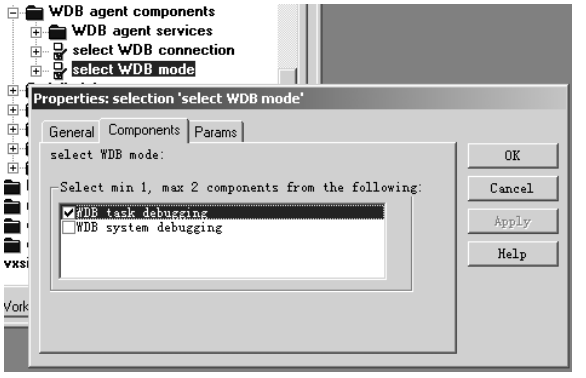
VxWorks全仿真

- 安装具有全仿真功能的Tornado版本，具有虚拟网络仿真功能
- 建立Tornado BSP工程，创建全仿真VxWorks映像
- 配置Target Server连接到目标系统上
- 所有的软件开发与真实的带有网卡的目标环境一样。

VxWorks组件配置

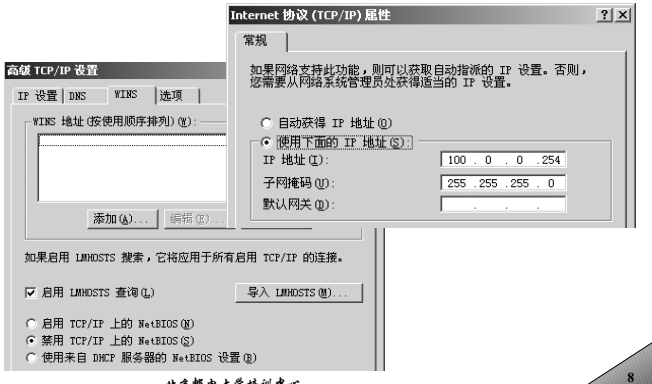


VxWorks组件配置



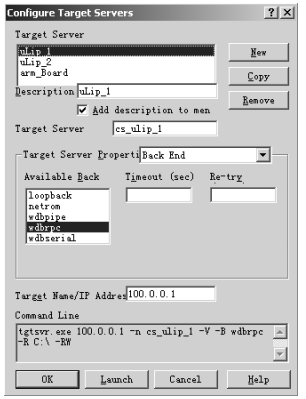
安装ULIP虚拟网卡

VxSim网卡的配置

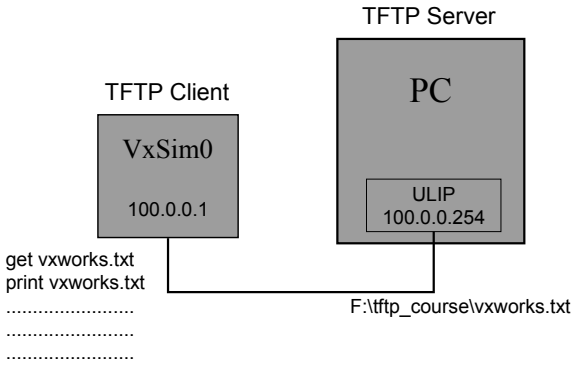


ULIP网卡的路由使能

连接到VxSim目标系统



TFTP示例演示



2. VxWorks网络概述

- 网络上的设备需要以某种标准交互数据
- IP协议套件提供了系统独立的协议
- VxWorks基于BSD4.4 release提供了TCP/IP协议的实现
- 包括以下特性：
  - 增强的可配置和可伸缩性
  - MUX层
  - 支持其他兼容的Internet特性

VxWorks网络功能

- 支持IP Multicase、CIDR和RFC-1323
- 支持IP、TCP、UDP
- 可以作为DHCP服务器、客户端和代理
- 可以作为DNS客户端
- 可以作为SNTP（Simple Network Time Protocol）服务器、客户端
- 支持RIP、OSPF等路由协议
- 支持ICMP、IGMP、ARP
- BSD4.4兼容Socket库
- 可作为BOOTP客户端
- 可作为RPC/NFS服务器和客户端
- 可作为FTP/TFTP服务器和客户端
- 可作为RSH客户端和Telnet服务器
- 集成对MIB-II的支持

VxWorks网络组件

- basic network initialization components：基本的网络初始化组件
  - VxWorks基本网络支持
  - 网络设备的启动
- network devices：网络设备
  - 网络设备的类型
- networking protocols：网络协议
  - TCP/IP组件
  - 应用及路由协议

基本的网络初始化组件

- 基本的网络支持
- 网络缓冲区的初始化
- 网络MUX初始化
- 网络设备名初始化
- DHCP地址初始化
- 初始化网络配置
  - 启动时初始化网络
  - 启动时不初始化网络

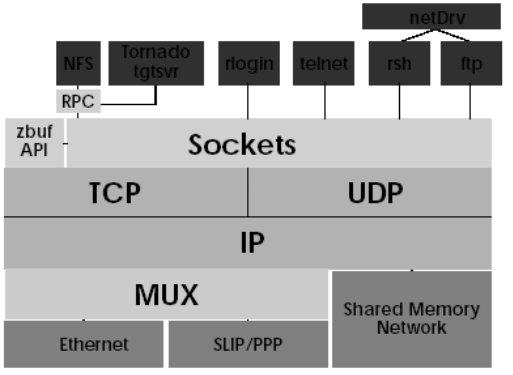
网络设备

- BSD网络接口
- END网络接口
- PPP接口
- SLIP接口
- 共享存储器网络组件

网络协议

- 核心TCP/IP组件
  - SOCKET/ICMP/IGMP/IPv4/UDP/ARP
- 网络应用
  - Filter/DHCP/DNS/PING/RLOGIN/SNTP/TELNET/MIB2/proxy ARP
- 网络调试
  - DHCP/ICMP/IGMP/TCP/UDP show
- 网络文件系统
  - FTP/NFS/TFTP
- 网络路由协议
  - RIP/OSPF（软件包）

VxWorks网络架构



MUX

- MUX的目的是提供一个接口，隔离网络接口驱动和网络业务
- MUX支持两种网络驱动接口类型
  - END: Enhanced Network Driver，一种面向帧结构的驱动
  - NPT: Network Protocol Toolkit，一种面向分组的驱动，所有链路层信息被去除
- WindRiver支持的所有VxWorks网络驱动都是END，NPT是MUX的一种扩展
- MUX位于数据链路层与网络层之间，但不是一个新的层

MUX到网络服务的绑定

```
void * muxBind
    (
    char * pName,               /* interface name, for example, ln, ei,... */
    int    unit,                /* unit number */
    BOOL (* stackRcvRtn) (void* , long, M_BLK_ID, LL_HDR_INFO * , void* ),
                                /* receive function to be called. */
    STATUS (* stackShutdownRtn) (void* , void* ),
                                /* routine to call to shutdown the stack */
    STATUS (* stackTxRestartRtn) (void* , void* ),
                                /* routine to tell the stack it can transmit */
    void (* stackErrorRtn) (END_OBJ* , END_ERR* , void* ),
                                /* routine to call on an error. */
    long   type,                /* protocol type from RFC1700 and many */
                                /* other sources (for example, 0x800 is IP) */
    char * pProtoName,          /* string name for protocol */
    void * pSpare               /* per protocol spare pointer */
    )
```

- 停止网络服务
- 向网络服务发送出错消息
- 向网络服务发送数据包
- 重启网络服务

IP和链路层接口

1. 绑定IP到MUX
2. 为接口设置网络掩码
3. 为接口指定IP地址
4. 为接口指定广播地址

IP到MUX的绑定

- BSD TCP/IP协议栈和MUX之间的接口

```
int ipAttach
    (
    int    unit,            /* Unit number */
    char * pDevice          /* Device name (i.e. ln, ei etc.). */
    )


STATUS ipDetach
    (
    int    unit,            /* Unit number */
    char * pDevice          /* Device name (i.e. ln, ei etc.). */
    )
```


接口IP地址配置

- 设置IP地址

```
STATUS ifAddrSet
    (
    char * interfaceName,     /* name of interface to configure, i.e. ei0 */
    char * interfaceAddress   /* Internet address to assign to interface */
    )
```

ifAddrSet(“nt0”, “100.0.0.1”);

- 设置掩码地址

```
STATUS ifMaskSet
    (
    char * interfaceName,     /* name of interface to set mask for, i.e. ei0 */
    int    netMask            /* subnet mask (e.g. 0xff000000) */
    )
```


接口IP地址配置

- 设置网络标识

```
STATUS ifFlagSet
    (
    char * interfaceName,     /* name of the network interface, i.e. ei0 */
    int    flags              /* network flags */
    )
```

- 设置广播地址

```
STATUS ifBroadcastSet
    (
    char * interfaceName,     /* name of interface to assign, i.e. ei0 */
    char * broadcastAddress   /* broadcast address to assign to interface */
    )
```

手工添加网络接口示例

1. fei2=muxDevLoad(0,fei82557EndLoad,"-1:0x00:0x20:0x20:0x00",1,0)
2. muxDevStart(fei2)
3. ipAttach(0,"fei")
4. ifMaskSet("fei0",0xffffff00)
5. ifAddrSet("fei0","136.12.117.10")
6. hostAdd("woof-route-10","136.12.117.10")
7. muxShow
8. hostShow
9. ifShow
10. mRouteShow

路由概述

- IP网络路由算法：
  - if(目的节点与路由器直接相连) {
  - send_to(des_node); //向目的节点发送数据
  - }
  - else {
  - find_in(route_table); //查找路由表，获得正确路由
  - send_to(next_hop); //向路由表中的下一跳路由器（或默认路由）发送数据
  - }
- 可以通过两种方法建立目标路由表
  - 静态：使用mRouteAdd()或routeAdd()向本地路由表添加路由
  - 动态：主机之间通过路由协议交换信息建立路由

手工添加静态路由

- routeLib
- routeAdd()：添加一条静态路由
- routeNetAdd()：添加一条目的是网络的路由
- routeDelete()：删除一条静态路由
- mRouteAdd()：向目的节点添加多条路由
- mRouteEntryAdd()：添加一条指定协议路由
- mRouteEntryDelete()：删除一条指定协议路由
- mRouteDelete()：从路由表中删除静态路由
- routeShow()：查看路由表

VxWorks动态路由协议

- VxWorks支持两种动态路由协议：
  - RIP：Route Information Protocol（路由信息协议） // ripLib
  - OSPF：Open Shortest Path First（开放式最短路径优先）

主机名

- 将主机名指定给Internet地址
  - STATUS hostAdd(
  - char * hostName, /* host name */
  - char * hostAddr /* host ip address */
  -)
- 显示主机名表中的内容
  - void hostShow (void)

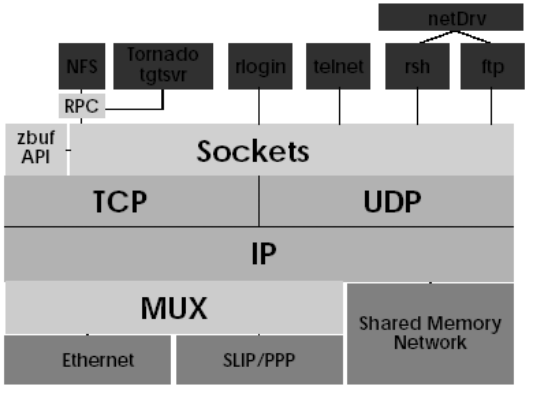
实例

```
-> hostShow
hostname          inet address       aliases
--------          ------------       -------
localhost         127.0.0.1
vxTarget          10.0.0.1
host              10.0.0.254
value = 0 = 0x0
-> hostAdd("victor","192.168.0.88")
value = 0 = 0x0
-> hostShow
hostname          inet address       aliases
--------          ------------       -------
localhost         127.0.0.1
vxTarget          10.0.0.1
host              10.0.0.254
victor            192.168.0.88
value = 0 = 0x0
->
```


3. VxWorks网络程序设计

- VxWorks的网络组件
- TCP/IP基础
- Socket编程
- 客户/服务器编程模型

VxWorks网络组件



网络组件

- 套接字（Sockets）:允许运行在VxWorks或其他主机环境下的任务之间相互通信
- 远程调用（Remote Procedure Calls）：运行一个任务唤醒实际运行在另一台机器上的过程
- 远程文件访问：允许VxWorks任务通过网络文件系统（NFS）、远程shell（RSH）、文件传输协议（FTP）、TFTP访问远程主机上的文件

VxWorks支持的网络连接

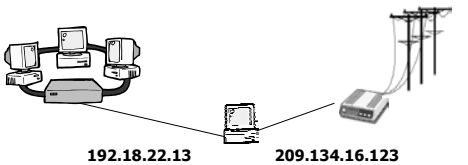
- 以太网（Ethernet）
- 串行线通信协议（SLIP和CSLIP）
- 共享网络内存

Internet Protocol（IP）

- 数据报（Datagram）通信协议
- 是一种尽力而为业务（Best-effort ）
  - 数据丢失（Loss）
  - 重新排序（Reordering）
  - 数据重复（Duplication）
  - 延时（Delay）
- 主机到主机的数据传送

IP地址

- 32-bit的识别符 (IPv4, IPv6=128 bits)
- 点分方式的四个十进制数来表示: 192.118.56.25
- www.mkp.com -> 167.208.101.28
- IP地址识别的是主机的接口（而非主机本身）



传输协议（Transport Protocols）

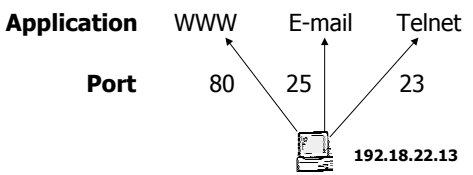
- 用户数据报协议（ User Datagram Protocol (UDP)）
  - 对数据进行校验
  - 仍然是尽力而为的服务
- 传输控制协议（Transmission Control Protocol (TCP)）
  - 对数据进行校验
  - 可靠的字节流传送
  - 流量和拥塞控制

主机名与IP地址的对应

- Internet中能够用两种方式识别主机的位置
  - IP地址
  - 主机名
- 对于小的网络，对hosts文件进行配置，即可实现主机名和IP地址的解析
- 随着网络的增大，使用DNS服务器来实现主机名和IP地址的解析

端口（ Ports）

- 用于识别最终的目的地
- IP地址用于识别主机
- 主机上有很多应用
- Ports (16-bit 识别符) 1-65,535



常用端口名和端口号的对应

- RFC1700规定常用端口号从0到1023，1024到49151是已注册的端口号。
- 常见服务协议的端口
  - telnet（23）
  - ftp（21）
  - SMTP（25）
  - http（80）

网络字节顺序

- Integer
  - 主机上原始的保存方式

Little-Endian	0	0	92	246

23,798 (0x5CF6)

Big-Endian	246	92	0	0

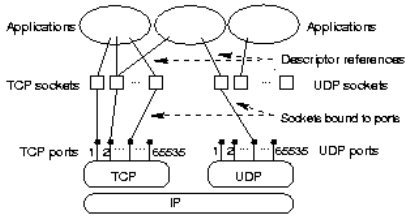
  - 网络字节顺序 (Big-Endian)
    - 用于多字节，二进制数据的交换
    - htonl(), htons(), ntohl(), ntohs()

套接字（sockets）

- Sockets向TCP/IP提供接口
- Sockets也为其他多种协议提供通用接口

套接字（sockets）续

- 由协议和本地/远端地址/端口识别
- 一个应用可以指向多个sockets
- Sockets可以被多个应用操作



VxWorks的套接字

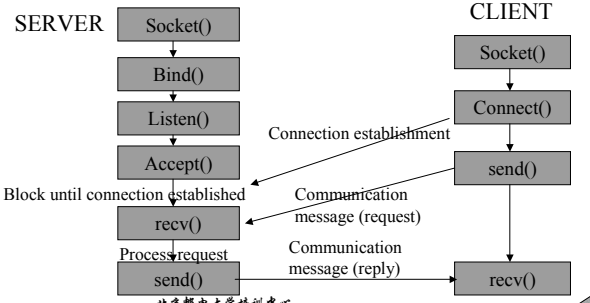
- VxWorks实现了与BSD4.4 TCP/IP兼容的sockets编程接口
- VxWorks目前支持两种套接字
  - 数据报套接字：支持双向的数据流，但不保证数据传送的可靠性、有序性和无重复
  - 流套接字：提供双向的、有序的、无重复并且无记录边界的数据服务

VxWorks套接字描述符

- 与文件操作一样，每个套接字在创建时都生成一个套接字描述符
- 该套接字描述符是在文件描述符表中的索引值
- 该套接字描述符在描述符表中的表项并不指向文件表，而是指向一个于该套接字有关的数据结构

套接字中的客户机/服务器模型

- 使用客户机/服务器模型进行编程时，服务器端有一个任务在指定的端口等待客户来连接，一旦连接上之后，就可以按设计的数据交换方法和格式进行数据传送
- 客户端在需要的时刻向服务器发出连接请求



创建socket

- Int socket (domain, type, protocol)
  - 创建一个socket
  - Socket可以是面向连接的(TCP=SOCK_STREAM), 面向无连接的(UDP=SOCK_DATAGRAM), 和原始数据(SOCK_RAW).
  - 返回一个socket描述符（文件描述符），便于以后使用socket
- 例子：
  - sd = socket (AF_INET, SOCK_STREAM, 0)
  - AF_INET　internet domain
  - SOCK_STREAM　规定SOCK_STREAM使用TCP
  - 0　规定协议类型，一般为0

Connect

STATUS connect (int s, struct sockaddr *name, int namelen)

- 在一个指定的socket上建立一个连接
- S: 指向一个已经打开的socket描述符
- Name：是一个地址结构，指向服务器端的名字
  - u_char sa_len　整个地址结构的大小
  - u_char sa_family　地址族
  - char sa_data[4]　指向服务器端的实际地址

close

- STATUS close (int fd)
  - 当客户端或服务器端使用完一个socket时，可以使用close来关闭该socket
  - fd：socket的文件描述符

Send

- Int send (int s, char *buf, int buflen, int flags)
  - 用于处于连接状态的socket，将一个消息发送到另一个socket
  - S: 一个已连接的套接字描述符
  - Buf：待发送数据的缓冲区
  - Buflen：缓冲区中数据的长度
  - Flags：调用执行方式
  - 如果发送时没有可用的缓存，send就被阻塞

sendto

- Int sendto (int s, caddr_t buf, int buflen, int flags, struct sockaddr * to, int tolen)
  - 用于不需要处于socket处于连接状态的socket，将一个消息发送到另一个socket
  - S: 一个已连接的套接字的描述符
  - Buf：待发送数据的缓冲区
  - Buflen：缓冲区中数据的长度
  - Flags：调用执行的方式
  - To：指向目的套接字的地址
  - Tolen：to所指向的地址的长度

recv

- Int recv (int s, char *buf, int buflen, int flags)
  - 从一个已经连接的socket处接收数据
  - S: 从s所指向的socket接收数据
  - Buf：接收到的数据所存放的缓冲区
  - Buflen：数据缓冲区的长度
  - Flags：调用执行的方式
  - 返回值为实际接收到的数据的长度
  - 如果没有消息到来，recv就被阻塞

recvfrom

- Int recvfrom (int s, char * buf, int buflen, int flags, struct sockaddr * from, int * pFromLen)
  - 从一个socket接收数据，不管该socket是否是面向连接
  - S: 从s所指向的socket接收数据
  - Buf：接收到的数据所存放的缓冲区
  - Buflen：数据缓冲区的长度
  - Flags：调用执行的方式
  - From：发送方地址
  - pFormLen：发送方地址的长度
  - 返回值为实际接收到的数据长度

bind

- STATUS bind (int s, struct sockaddr *name, int namelen)
  - 将一个名字绑定到一个socket上，为一个未命名的socket分配一个名字，该名字中包含本地的地址
  - S：套接字描述符
  - Name：要绑定给套接字的名字
  - Namelen：名字的长度

listen

- STATUS listen (int s, int backlog)
  - 让socket对连接进行侦听
  - S：套接字描述符
  - Backlog：可以排队等待的最大连接数
  - Listen只能用于SOCK_STREAM或SOCK_SEQPACKET类型的socket上

accept

- Int accept (int s, struct sockaddr *addr, int * addrlen)
  - 在指定的socket上接受一个连接
  - S:套接字描述符。该socket是由socket调用创建，由bind调用绑定到一个地址上，再由listen调用的socket
  - Addr：用于存放调用返回的地址信息
  - Addrlen：保存addr的精确长度

select

- Int select (int width, fd_set *pReadFds, fd_set *pWriteFds, fd_set *pExecptFds, struct timeval *pTimeOut)
- 等待一些描述符改变状态。可以由它来通知连接的一个或多个I/O条件是否得到满足。当一个任务处理多个连接时，为了得到某一时刻哪些连接可以进行读写，也需使用select。
- Width：需要测试的宽度，即要遍历多少个描述符
- pReadFds， pWriteFds， pExecptFds：分别指向可读，可写和状态异常的描述符集合指针
- pTimeOut：设置函数调用的超时值
- 函数返回所有描述字集合已准备好的总位数，如果超时，则返回0

其它相关函数

- hostGetByName(): 通过主机名字查询主机地址
- hostGetByAddr(): 根据IP地址查询主机名称
- Sethostname(): 设置主机名
- Gethostname(): 获得主机名

数据传送

- 在连接建立以后
  - Send (mysock_descrip, buf, buflen, flags)
  - Recv (mysock_descrip, buf, buflen, flags)
- 如果发送时没有可用的缓存，send就被阻塞
- 如果没有到来的消息可用，recv就被阻塞

TCP Client/Server通信过程

Server starts by getting ready to receive client connections…

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server通信过程－续

```
/* Create socket for incoming connections */
 if ((sFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror("socket");
```

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server通信过程－续

```
serverAddr.sin_family = AF_INET;                /* Internet address family */
 serverAddr.sin_port = htons(SERVER_PORT_NUM);      /* Local port */
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);/* Any incoming interface */
 if (bind(sFd, (struct sockaddr *) &serverAddr, sockAddrSize)) == ERROR)
perror("bind() ");
```

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server通信过程－续

```
/* Mark the socket so it will listen for incoming connections */
   if (listen(sFd, SERVER_MAX_CONNECTIONS) <
ERROR)
      perror("listen");
```

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server通信过程－续

```
for (;;) /* Run forever */
{
   if ((newFd=accept(sFd,(struct sockaddr *)&clientAddr,
&sockAddrSize)) ==ERROR)
      perror("accept() ");
```

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

Server is now blocked waiting for connection from a client

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

Later, a client decides to talk to the server…

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

```
/* Create a reliable, stream socket using TCP */
if ((sFd = socket(AF_INET, SOCK_STREAM, 0)) ==
                                        ERROR)
    printf("socket() ");
```

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

```
serverAddr.sin_family      = AF_INET;          /* Internet address family */
  serverAddr.sin_addr.s_addr = inet_addr(servIP); /* Server IP address */
  serverAddr.sin_port        = htons(SERVER_PORT_NUM); /* Server port */

  if (connect(sFd, (struct sockaddr *) &serverAddr, sockAddrSize)) < 0)
      printf("connect()");
```

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

```
/* Send the string to the server */
  if (send(sFd, (char *)&myRequest,sizeof (myRequest)) ==ERROR)
     perror("send() ");
```

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

```
if ((newFd=accept(sFd,(struct sockaddr
   *)&clientAddr, &sockAddrSize)) ==ERROR)
      perror("accept() ");
```

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

```
/* Receive message from client */
   if ((recvMsgSize = recv(newFd, echoBuffer,
   RCVBUFSIZE)) < 0)
      perror("recv() failed");
```

Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server 通信过程－续

close(sFd);　　close(newFd)

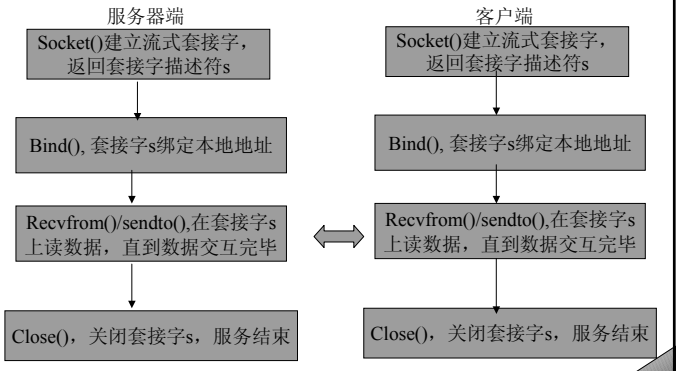
Client
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server
1. Create a TCP socket
2. Assign a port to socket
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

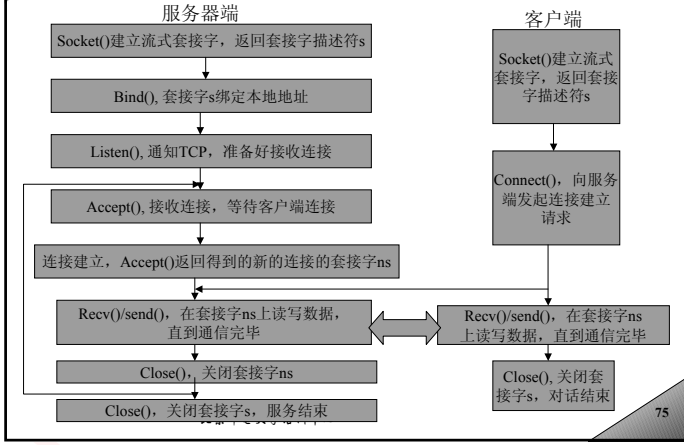
客户/服务器

- 客户：请求服务的一方称为客户
- 服务器：提供某种服务的乙方称为服务器
- Internet的应用中大多数应用都是比较单一的客户/服务器模式
- 网络编程的程序分为客户端程序和服务器端程序

无连接的协议调用



面向连接的协议调用



客户端程序设计

- 客户端程序：发出服务请求的程序。
- 客户端程序要知道服务器端的地址，提供服务所使用的端口号，服务使用的传输层协议
- 客户端寻找服务器端
  - 通过指定服务器的IP地址和端口号来寻找服务器
  - 通过广播的方式来找服务器

TCP客户端编程

- TCP协议是一种保证传输可靠，保证顺序的传输层协议。
- TCP客户端编程结构
  - 创建套接字，domain为PF_INET, type为SOCK_STREAM
  - 通过各种途径获得服务器端的IP地址和服务端口号
  - 使用connect和服务器端进行连接
  - 连接建立好之后，就可以通过send、recv等调用进行通信
  - 通信完毕，使用close关闭连接

UDP客户端编程

- UDP是一种不保证传输可靠性和传输顺序的传输层协议
- UDP客户端编程结构
  - 创建套接字，domain为PF_INET，type为SOCK_DGRAM
  - 通过各种途径获得服务器端的IP地址和服务端口号
  - 发送和接收数据
    - 使用connect的UDP可以通过调用connect来填写有关套接字的数据结构；在后面使用send和recv时就不用每次都填写IP地址和端口号
    - 不使用connect的UDP可以直接使用sendto和recvfrom来发送和接收数据，每次都要指定服务器端的IP地址和端口号
  - 发送和接收数据后使用close关闭套接字

服务器端程序设计

- 服务器端是提供服务的一方，它一直侦听某个端口，等待请求消息的到来
- 程序的一般结构
  - 创建套接字
  - 绑定一个本地端口
  - 在该端口侦听请求消息
  - 收到请求消息，处理消息，返回响应
- 服务器端程序的设计需要考虑对多个客户进行服务的情况，即考虑程序的并发性

面向连接的服务端概念

- 面向连接的服务器使用的传输层协议能够保证数据传送的可靠性、顺序性。因此服务器端程序的设计就不需要考虑数据的可靠传送
- 面向连接的服务器端程序只要接受连接请求，并通过该连接进行通信即可，底层的传输层协议会保证数据传送的可靠性

无连接的服务端概念

- 无连接的服务端使用的传输层协议不保证数据传送的可靠性
- 无连接的服务端程序如果需要向上层提供有保证的数据服务，就必须在程序设计中考虑报文的顺序、错误校验等问题

服务端程序设计的两种基本模式

- 服务端程序设计的两种模式
  - 循环模式
  - 并发模式
- 循环模式
  - 程序总体结构上是一个循环，一次处理一个请求
  - 有多个客户端请求时，请求放入队列，一次等待处理
- 并发模式
  - 程序可以同时处理多个请求，
  - 父进程接受请求，产生子进程，由子进程处理请求

无连接服务端循环模式程序设计

1. 创建套接字，类型为SOCK_DGRAM
2. 使用bind调用将端口和地址绑定到该套接字上
3. 用recvfrom接受请求消息
4. 使用sendto将响应发送回客户端
5. 处理完成后返回第3步，进入下一次循环

无连接服务端并发模式程序设计

1. 创建套接字，类型为SOCK_DGRAM
2. 使用bind调用来指定本机地址和侦听端口
3. 调用recvfrom接收请求消息，没有消息时，该调用被阻塞
4. 一旦recvfrom收到请求消息，则产生一个新任务，新任务负责和客户端通信，原任务返回第3步
5. 新任务处理完客户请求后结束并退出

面向连接服务端循环模式程序设计

1. 创建套接字，类型为SOCK_STREAM
2. 调用bind指定地址和端口
3. 调用listen指定未处理连接请求的队列长度
4. 调用accept等待连接请求（在阻塞方式下），一旦有连接请求，accept返回一个新的套接字描述符
5. 通过该描述符与客户端交互信息，一旦交互完毕，关闭新产生的套接字并返回第4步

面向连接服务端的并发模式

1. 创建套接字，类型为SOCK_STREAM
2. 调用bind指定地址和端口
3. 调用listen指定未处理连接请求的队列长度
4. 调用accept等待连接请求（在阻塞方式下），一旦有连接请求，accept返回一个新的套接字描述符
   1. 产生一个新任务，该任务使用accept产生的新套接字和客户端进行信息交互
   2. 新任务处理完与客户之间的交互则结束并退出
5. 返回第4步等待新的连接请求

面向连接服务端单进程并发模式

1. 创建一个套接字，类型为SOCK_STREAM，将该套接字加入select的监测是否有输入的集合中
2. 使用bind调用，设定本地地址和服务端口
3. 使用listen调用，指定队列长度
4. 使用select等待某个套接字有数据到达
5. 一旦有数据到达最早产生的套接字，则调用accept来接受请求，并产生新的套接字，并将新的套接字描述符加入select的监测输入集合，返回第4步
6. 一旦有数据到达新的套接字描述符，则根据应用层协议处理新的套接字的信息，返回第4步