




第四章

TCP/UDP编程

讲师：任继梅

课程目标

掌握计算机网络工作的原理

-  五层协议模型
-  掌握交换机原理
-  掌握路由器原理

掌握网络编程常用API

-  TCP编程 UDP编程

网络高级编程

-  i/o模型

嵌入式

课程安排



第一天

上午：计算机网络概述

下午：计算机网络原理



第二天

上午：网络原理详解

下午：TCP/IP协议栈详解



第三天

上午：Socket编程基础

下午：TCP&UDP编程基础



第四天

上午：高级网络编程

下午：高级网络编程续



第五天

上午：网络编程实战

下午：网络编程实战续

课前提问

1. TCP/IP协议族使用地址结构是什么？分别有哪些重要成员？
2. 字节序是怎么回事？网络字节序采用的是哪种字节序？
3. 如何根据域名得到IP地址？
4. IP地址有几种表示方法？

本章目标

✓ UDP编程



✓ UDP客户端



✓ UDP服务器端



✓ TCP编程



✓ TCP客户端



✓ TCP服务器端



第一节 UDP编程

UDP编程

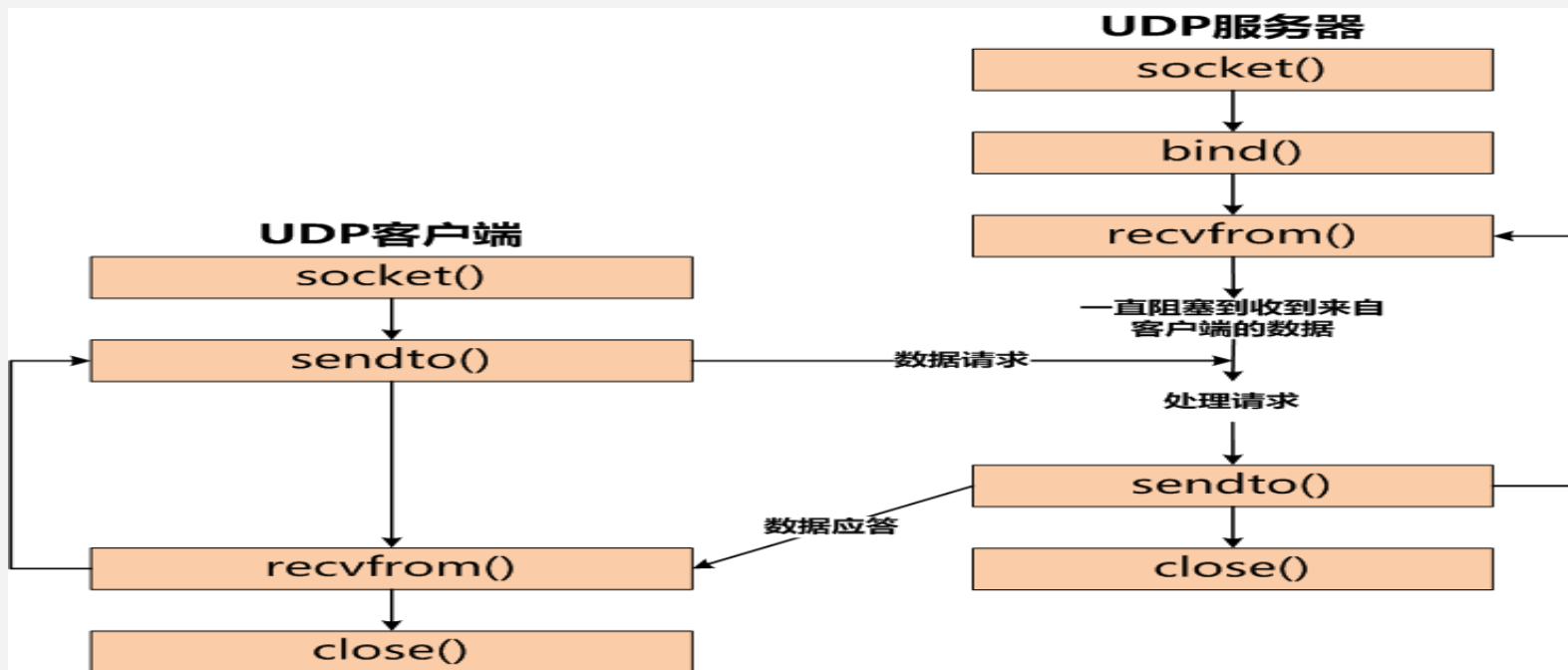


UDP协议

- ③ User Datagram Protocol 用户数据报协议
- ③ 无连接，客户端和服务端之间不必存在长期的关系
- ③ 不可靠，不能保证数据是否到达
- ③ 无序，不保证数据按序到达
- ③ 不重传
- ③ 全双工
- ③ 数据报式，不同于字节流，每个数据报都有一个长度，UDP协议将该长度随数据一起发往对端，不像TCP没有任何记录边界。
- ③ UDP客户端可以通过一个主动套接字给多个UDP服务器发数据
- ③ UDP服务器可以通过一个主动套接字接收多个UDP客户端的数据

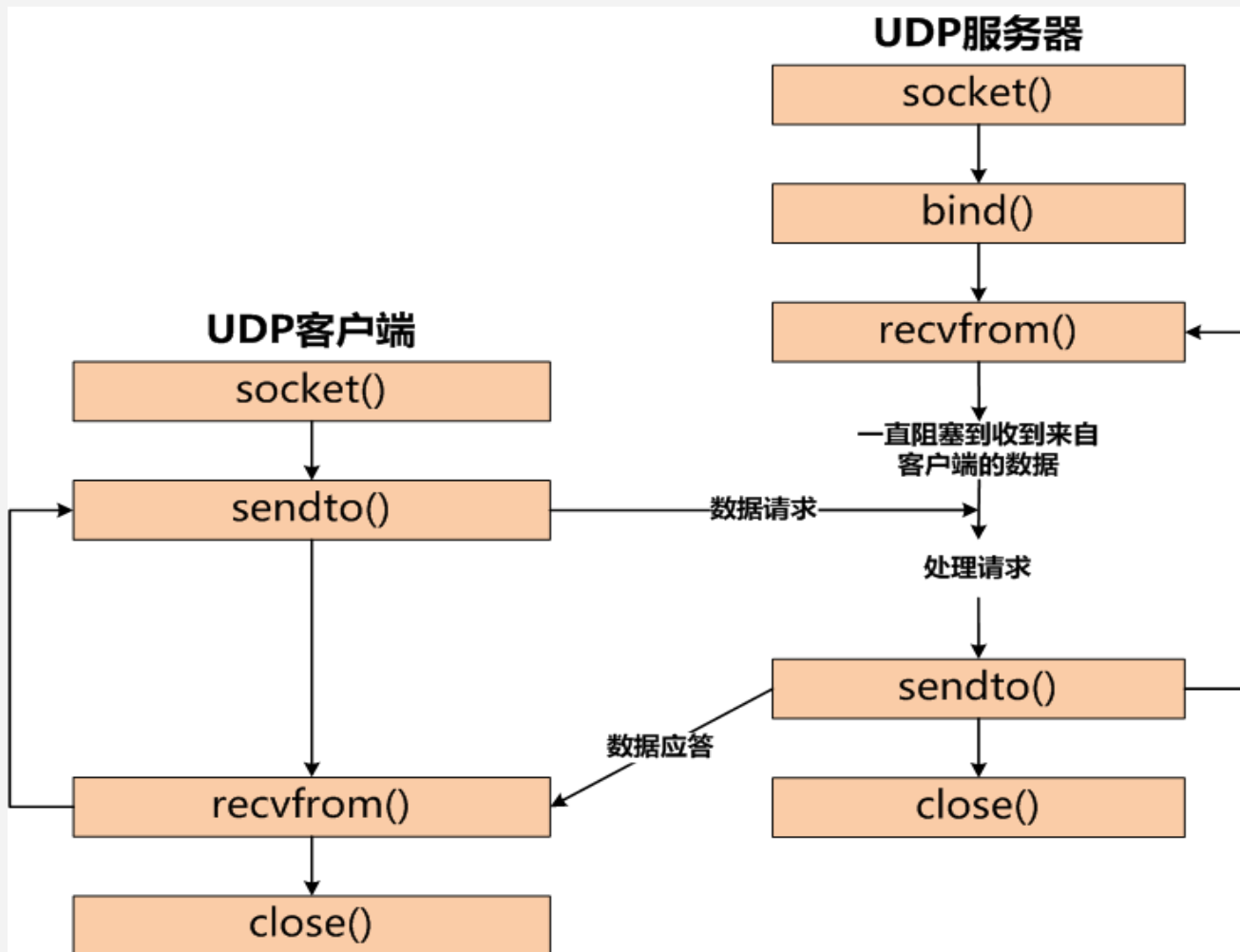
UDP范式

未连接的UDP-Socket模式



UDP范式

UDP通信模式



UDP服务器

● 代码模板

- ◆ `int sockfd;`
- ◆ `struct sockaddr_in servaddr, cliaddr;`
- ◆ `socklen_t cliaddrlen = sizeof(struct sockaddr_in);`
- ◆ `sockfd = socket(AF_INET, SOCK_DGRAM, 0);` // 创建套接字
- ◆ `bzero(&servaddr, sizeof(servaddr));`
- ◆ `servaddr.sin_family = AF_INET;` // 指定协议族
- ◆ `servaddr.sin_port = htons(xxx);` // 指定服务器端口号
- ◆ `inet_aton("a.b.c.d", &servaddr.sin_addr);` // 指定服务器IP地址
- ◆ `bind(sockfd, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));`
- ◆ `while(1){` // 数据传输
- ◆ `recvfrom(sockfd, buff, len, 0, &cliaddr, &cliaddrlen);`
- ◆ `// 数据处理`
- ◆ `sendto(sockfd, buff, len, 0, &cliaddr, cliaddrlen);`
- ◆ `close(sockfd);` // 关闭套接字

TCP服务端

绑定函数bind()

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

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

- 功能：将本地协议地址赋予指定的socket描述符
- 返回值：成功为0，失败 < 0
- 参数：
 - sockfd: socket函数返回的socket描述符
 - myaddr: 指向与协议族相符的地址结构指针
 - addr len: 实际地址结构的大小
- 对于AF_INET协议族，其地址组成为IP地址+端口号，结构为struct sockaddr_in

UDP范式

sendto发送消息

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

- 功能：向指定地址发送期望字节数量的数据
- 返回值：成功返回实际发送的字节数。失败 < 0
- 参数：
- sockfd、buff、nbytes、flags：含义同send函数
- to：指向接收者的协议地址结构指针。（类似于connect对应参数的含义）
- addrlen：to所用的地址结构的字节长度。类似于connect对应参数的含义）

UDP范式

Recvfrom接收消息

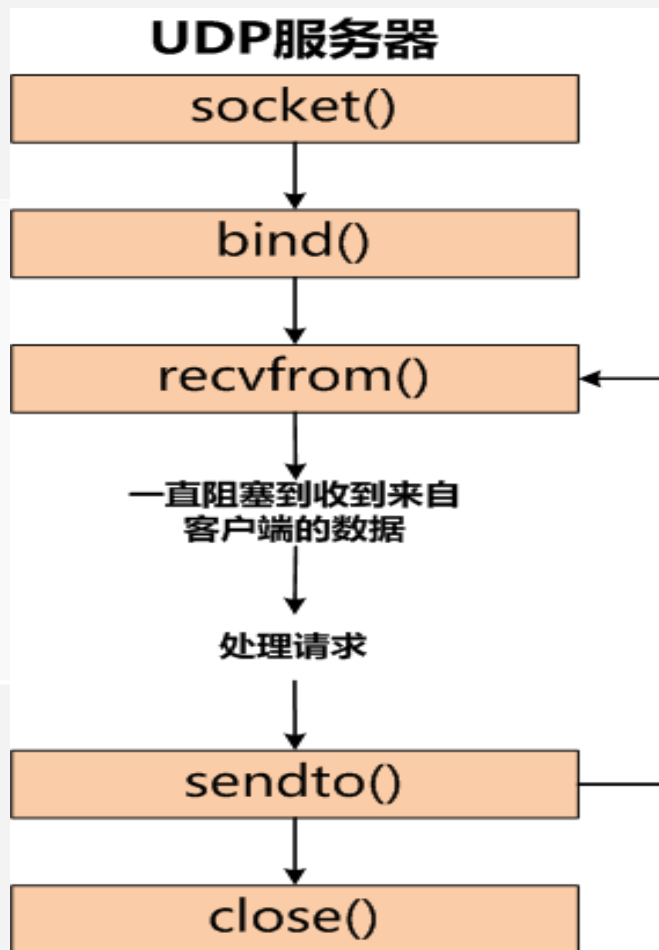
```
#include <sys/types.h>
#include <sys/socket.h>
```

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

- 功能：从指定的UDP-Socket里接收数据，同时填充发送方的协议地址结构
- 返回值：成功返回实际接收到的字节数。失败 < 0
- 参数：
- src_addr：指向发送方的协议地址结构指针，类似于accept对应参数的含义，用于保存发送方地址，以便于通过sendto回应对方相应数据。
- addrlen：src_addr所用的地址结构的字节长度，指出实际使用的地址结构长度。
- 返回0是可接受的正确的结果，并非表示socket关闭，意味着可以发送空的UDP数据报（只有UDP首部而没有数据部分）给远端进程。
- from和addrlen可以同时为NULL，表示不关心发送方地址信息。

UDP服务器

UDP服务器编程步骤：



UDP服务器

例子:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>
#include <strings.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char **argv)
{
    int serverfd;
    socklen_t cliilen;
    struct sockaddr_in cliaddr, servaddr;
    char buf[20] = {'\0'};

    serverfd = socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(servaddr.sin_addr));
    servaddr.sin_port = htons(5678);

    bind(serverfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```


UDP服务器

例子:

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
inet_aton("127.0.0.1", &(servaddr.sin_addr));
servaddr.sin_port = htons(5678);

bind(serverfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

while(1)
{
    clilen = sizeof(cliaddr);
    recvfrom(serverfd, buf, 11, 0, (struct sockaddr *)&cliaddr,
                                                    &clilen);

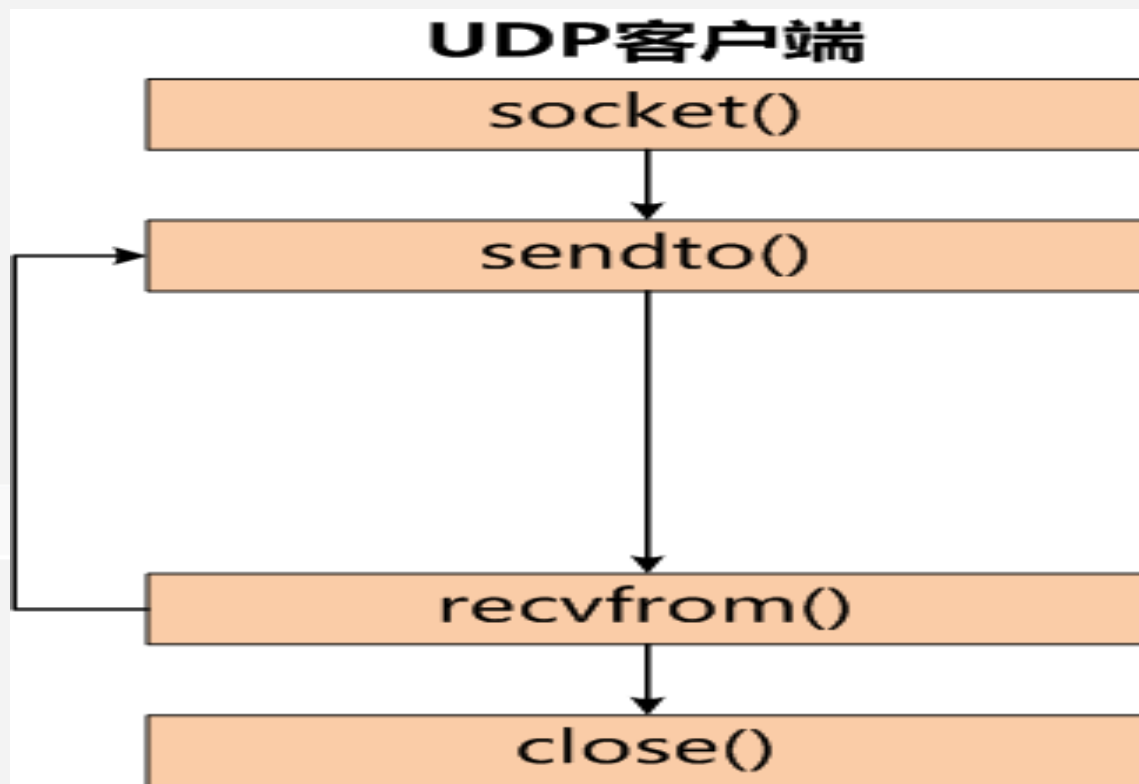
    printf("Server say:%s\n", buf);
    sendto(serverfd, buf, 11, 0, (struct sockaddr *)&cliaddr,
                                                    clilen);
}
close(serverfd);

return 0;
}
```

⚠ Code will never be executed

UDP客户端

● 未连接UDP-Socket编码模式



UDP客户端

● 未连接UDP-Socket代码模板

- ◆ `int sockfd;`
- ◆ `struct sockaddr_in servaddr;`
- ◆ `sockfd = socket(AF_INET, SOCK_DGRAM, 0);` // 创建套接字
- ◆ `bzero(&servaddr, sizeof(servaddr));`
- ◆ `servaddr.sin_family = AF_INET;` // 指定协议族
- ◆ `servaddr.sin_port = htons(xxx);` // 指定服务器端口号
- ◆ `inet_aton("a.b.c.d", &servaddr.sin_addr);` // 指定服务器IP地址
- ◆ `while(1){` //数据传输
- ◆ `//制作数据`
- ◆ `sendto(sockfd, buff, len, 0, (struct sockaddr *)&servaddr, sizeof(servaddr));`
- ◆ `recvfrom(sockfd, buff, len, 0, NULL, NULL);`
- ◆ `//数据处理}`
- ◆ `close(sockfd);` // 关闭套接字

UDP客户端

● 未连接UDP-Socket实例

◆ chapter4/udphello/client1.c

上 嵌

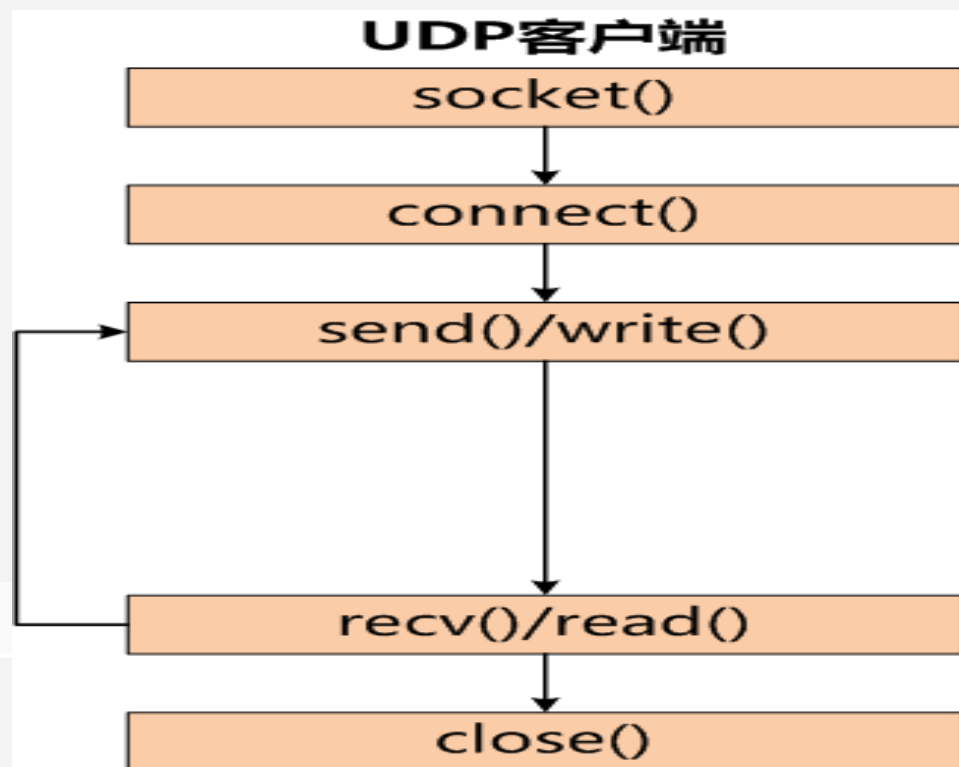
UDP客户端

● 未连接UDP-Socket不足之处

- ◆ 知道客户端临时端口和地址的任何进程都可以向其发送UDP包，客户端需要区分哪些包是自己期望的包（比如可以通过recvfrom的最后两个参数与期望服务器的协议地址比较来区分）
- ◆ 如果服务器进程未启动，客户端将永久阻塞在recvfrom，无法获取相关错误信息。内核在发送UDP数据前有一个使用ARP协议验证服务器地址的过程，这个过程如果发现错误，内核却无法将此错误返回给sendto，除非这个UDP是一个已连接的UDP

UDP客户端

● 已连接UDP-Socket编码模式



UDP客户端

● 已连接UDP-Socket代码模板

- ◆ `int sockfd;`
- ◆ `struct sockaddr_in servaddr;`
- ◆ `sockfd = socket(AF_INET, SOCK_DGRAM, 0);` // 创建套接字
- ◆ `bzero(&servaddr, sizeof(servaddr));`
- ◆ `servaddr.sin_family = AF_INET;` // 指定协议族
- ◆ `servaddr.sin_port = htons(xxx);` // 指定服务器端口号
- ◆ `inet_aton("a.b.c.d", &servaddr.sin_addr);` // 指定服务器IP地址
- ◆ `connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));`
- ◆ `while(1){` // 数据传输
- ◆ `// 制作数据`
- ◆ `send(sockfd, buff, len, 0);`
- ◆ `recv(sockfd, buff, len, 0);`
- ◆ `// 数据处理}`
- ◆ `close(sockfd);` // 关闭套接字

UDP客户端

● 已连接UDP-Socket connect说明

- ◆ 没有三路握手过程
- ◆ 内核只是检查是否存在立即可知的错误（如不可到达的目的地）。
- ◆ 记录对端的IP地址和端口号，立即返回调用进程
- ◆ 三个变化：
 - 不能给发送操作指定目的地址
 - 不必使用recvfrom接收数据，限制能且仅能与一个对端交换数据报
 - 异步错误将返回给调用进程

套接字类型	write或send	不指定目的地址的 sendto	指定目的地址的 sendto
TCP套接字	可以	可以	EISCONN
UDP套接字，已连接	可以	可以	EISCONN
UDP套接字，未连接	EDESTADDRREQ	EDESTADDRREQ	可以

UDP客户端

```
#include <stdio.h>
#include <sys/socket.h>
#include <strings.h>
#include <arpa/inet.h>
int main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in servaddr;
    char buf[20] = {'\0'};

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);




    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(5678);
    inet_aton("127.0.0.1", &servaddr.sin_addr);

    sendto(sockfd, "hello world", 11, 0, (struct sockaddr*)&servaddr,
                                                    sizeof(servaddr));

    recvfrom(sockfd, buf, 11, 0, NULL, NULL);
    printf("Client say:%s\n", buf);
    close(sockfd);
    return 0;
}
```


UDP编程练习:

1. 局域网聊天室

-  1.1 单聊
-  1.2 群聊
-  1.3 用户管理

上
嵌

第二节 TCP编程



TCP协议

- Transmission Control Protocol 传输控制协议
- 可靠的
- 面向连接的
- 字节流式的
- 全双工的
- 已排序的
- 被大多数网络应用程序采用
- 使用步骤：
 - 建立连接
 - 数据传输
 - 终止连接

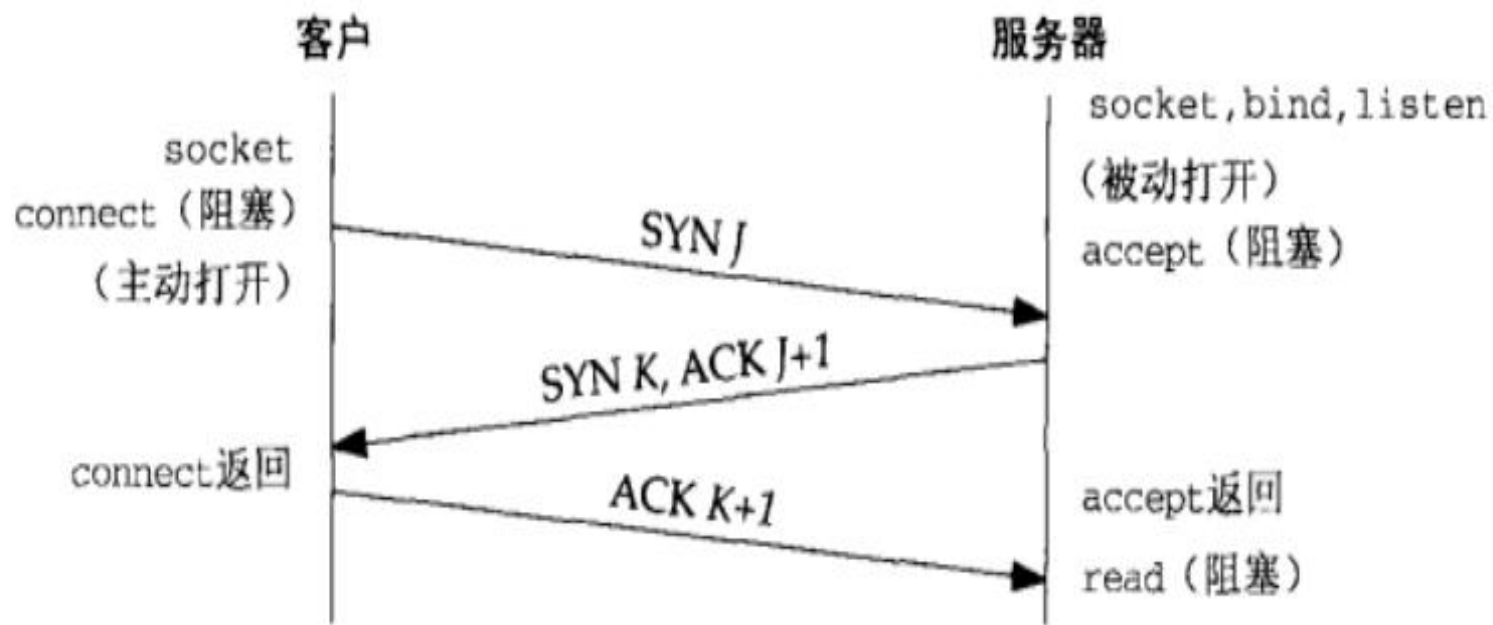
嵌入式

TCP连接建立过程

- 分节：指TCP传递给IP的数据单元
- 三路握手：
 - 服务器必须准备好接受外来连接。这通常通过调用socket、bind和listen这3个函数完成，称之为被动打开（passive open）。
 - 客户通过调用connect发起主动打开（active open）。这导致客户发送SYN（同步）分节，它告诉服务器客户将在（待建立的）连接中发送数据用的初始序列号。
 - 服务器必须确认（ACK）客户的SYN，同时也得发送一个自己的SYN分节，它含有服务器将在同一连接中发送数据的初始序列号。服务器的这个ACK和SYN做进同一个分节中。
 - 客户必须确认服务器的SYN。
 - 由于整个过程需要三个分节，因此称为三路握手（Three-way handshake）

TCP编程

人 TCP连接——三路握手



TCP连接终止过程

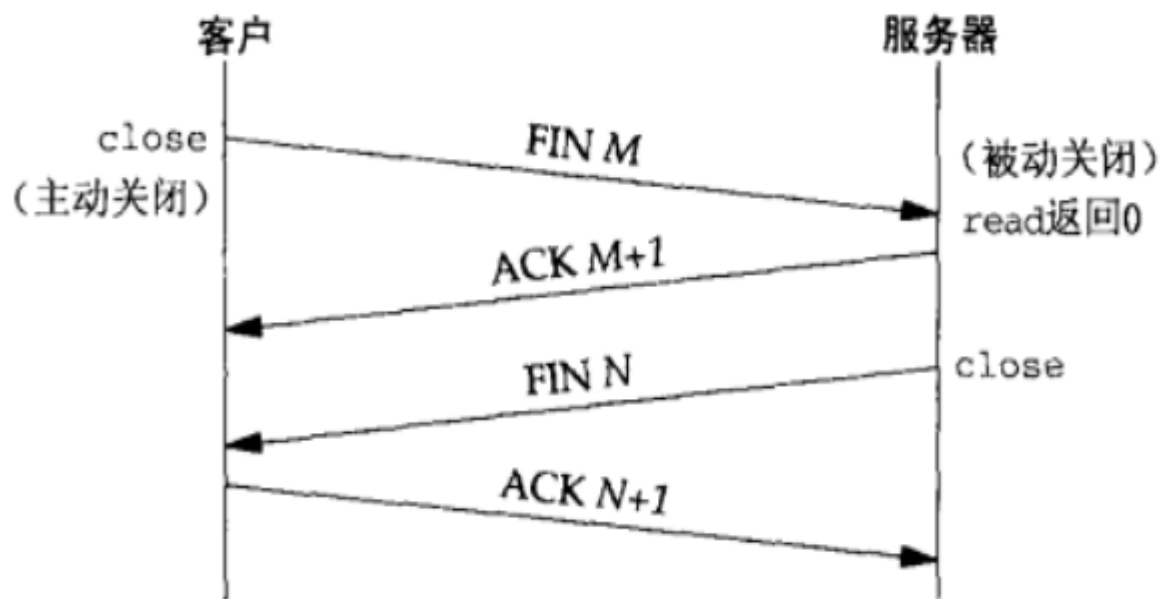
- 4个分节

- 终止过程：

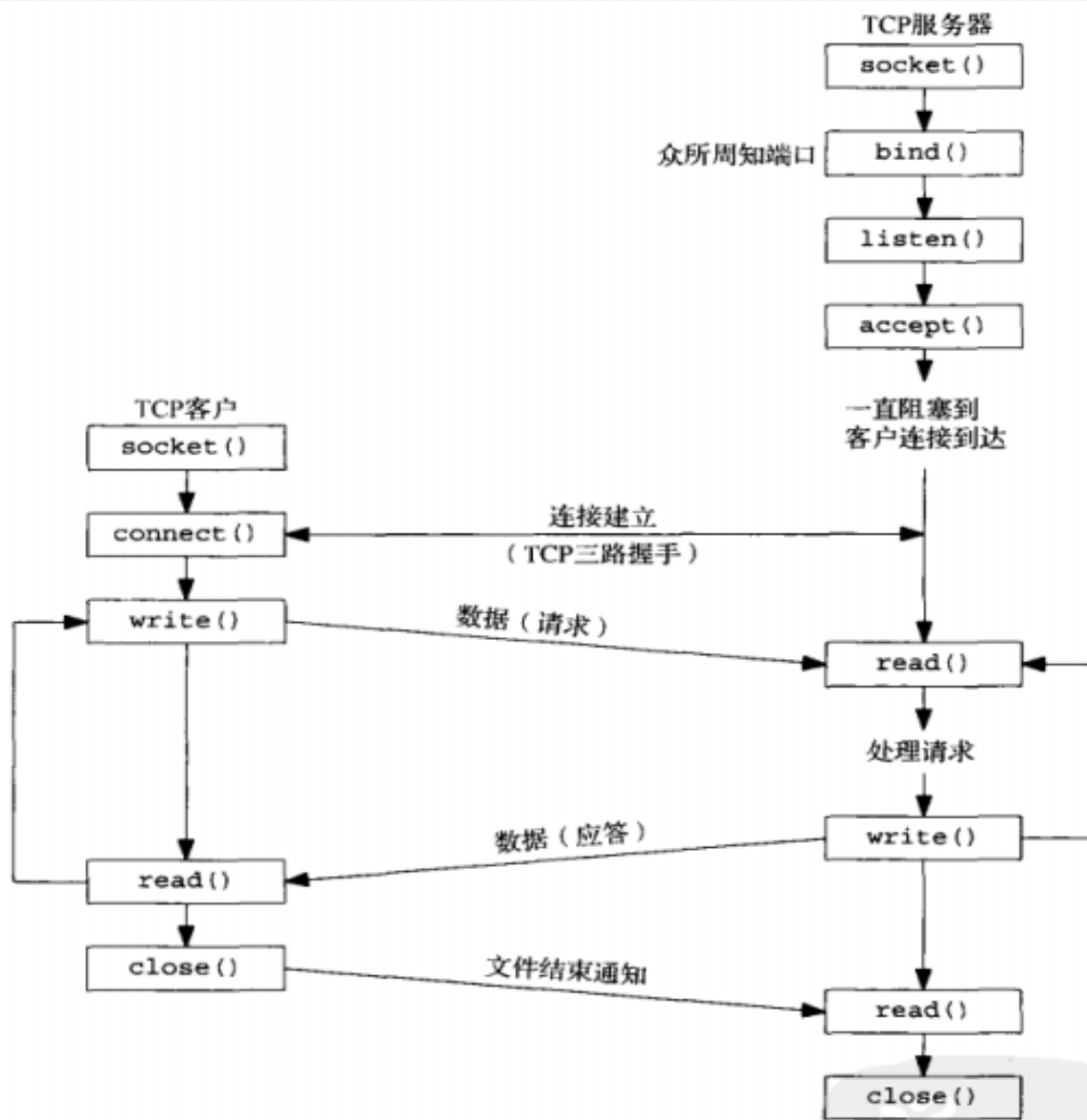
- 某进程调用close，执行主动关闭（active close），发送FIN分节，表示数据发送完毕。
- 对端进程接收到FIN执行被动关闭（passive close），由此可知相应连接上再无数据可接收，发送ACK分节进行确认。
- 对端进程一段时间后调用close，也发送自己的FIN分节到主动关闭端进程。
- 主动关闭端进程发送ACK确认
- 见下页图

TCP编程

📖 TCP连接终止过程

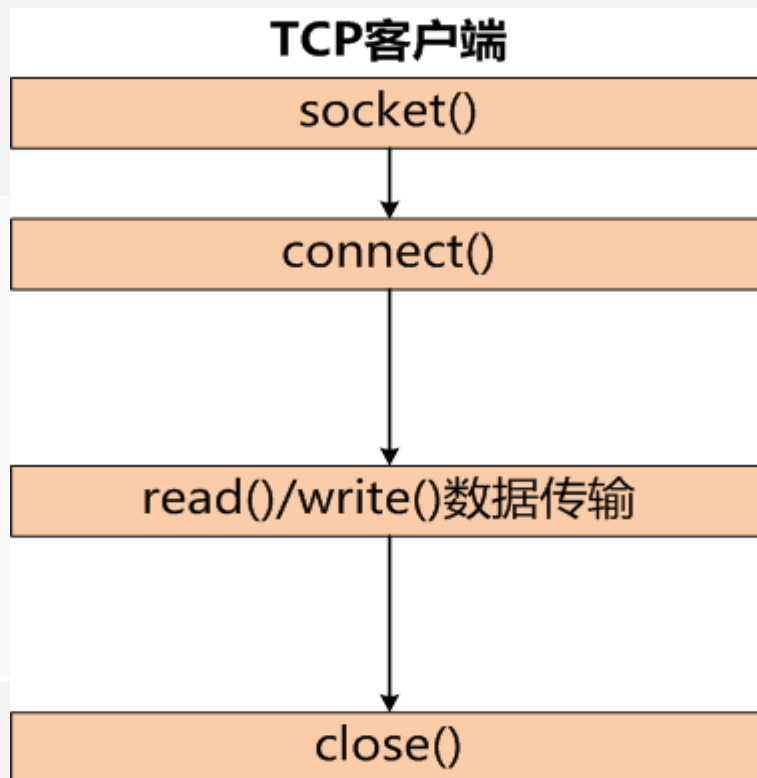


TCP编程



TCP客户端

客户端编程步骤



TCP客户端

连接函数connect ()

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- 功能：通过三路握手建立与TCP服务器的连接
- 返回值：成功为0，失败 < 0
- 参数说明：
 - sockfd：socket函数返回的socket描述符
 - servaddr：指向与协议族相符的地址结构指针
 - addrlen：实际地址结构的大小
- 对于AF_INET协议族，其地址组成为IP地址+端口号，结构为struct sockaddr_in

TCP客户端

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <strings.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in servaddr;
    char buf[20] = {'\0'};
    int ret = 0;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(5678);
    inet_aton("127.0.0.1", &servaddr.sin_addr);
```

TCP客户端

```
ret = connect(sockfd, (struct sockaddr *)&servaddr,  
               sizeof(servaddr));  
if (ret < 0)  
{  
    perror("connect");  
    close(sockfd);  
    return 1;  
}  
  
write(sockfd, "hello world", 11);  
read(sockfd, buf, 11);  
printf("Client say:%s\n", buf);  
close(sockfd);  
return 0;  
}
```

TCP服务端

服务端编程步骤



TCP服务端

监听函数listen()

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

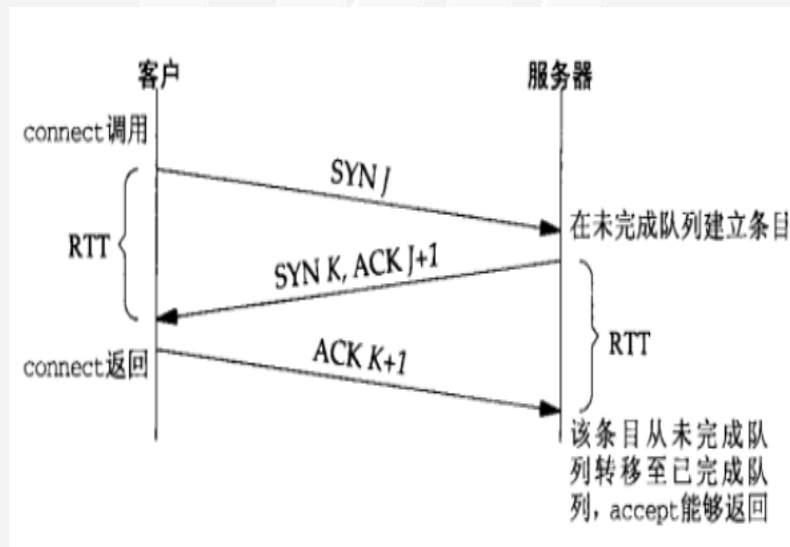
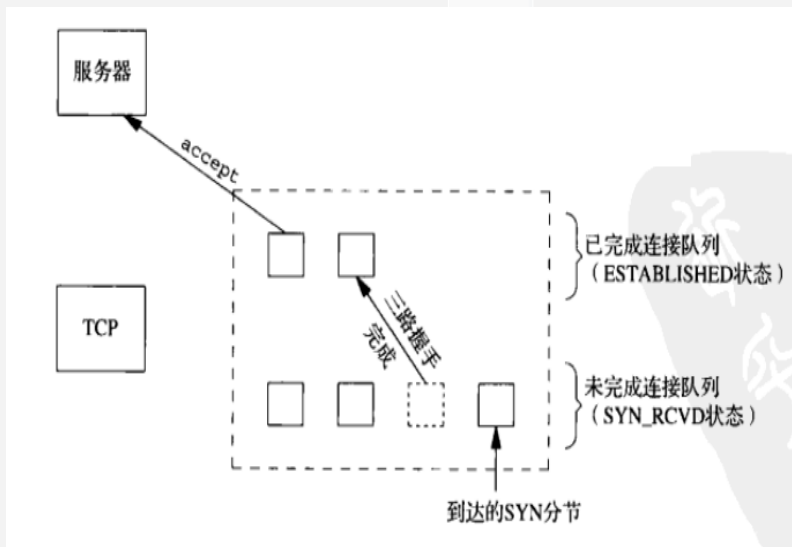
int listen(int sockfd, int backlog);
```

- 功能：将主动套接字改为被动套接字（用来接受指向该套接字的连接请求），并设定套接字排队的最大连接个数。
- 返回值：成功0，失败 < 0
- 参数：
- sockfd：处于未连接的套接字描述符
- backlog：指定管理连接请求的队列长度

TCP服务端

监听函数 `listen()`

- 内核针对每个被动套接字采用两个队列管理连接请求：（见下页图）
- 未完成连接队列，已完成连接队列
- 该函数不会阻塞
- `backlog`实际上是一个基数，两个队列的长度由这个基数采用一定算法计算而得。
- 一般情况下`backlog`经验值为5-20。
- 对于大型服务器（如新浪主机）一般指定为系统能支持的最大数。



TCP服务端

接收函数accept ()

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

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

- 功能：从已完成连接队列的对头返回一个已完成连接，队列空则阻塞等已完成连接
- 返回值：成功返回已完成连接对应的socket描述符，服务器通过这个描述符与请求连接的客户端进行数据传输。失败 < 0
- 参数：
 - sockfd：监听套接字（即服务器用来处理连接请求专用的被动套接字）。
 - cliaddr：返回已连接的对端进程（客户端）的协议地址。
 - addrlen：值-结果参数，调用前指定cliaddr的地址结构长度，返回后保存对应地址结构的确切字节数。

TCP服务器例子:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <strings.h>
#include <arpa/inet.h>

int main(int argc, char **argv)
{
    int listenfd = -1;
    int connfd = -1;
    socklen_t cliilen = 0;
    struct sockaddr_in cliaddr = {0};
    struct sockaddr_in servaddr = {0};
    int ret = 0;
    char buf[20] = {'\0'};

    listenfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(servaddr.sin_addr));
    servaddr.sin_port = htons(5678);
    ret = bind(listenfd, (struct sockaddr *) &servaddr,
               sizeof(servaddr));
```

TCP服务器例子:

```
ret = listen(listenfd, 5);

while(1)
{
    clilen = sizeof(cliaddr);
    connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &
                    clilen);

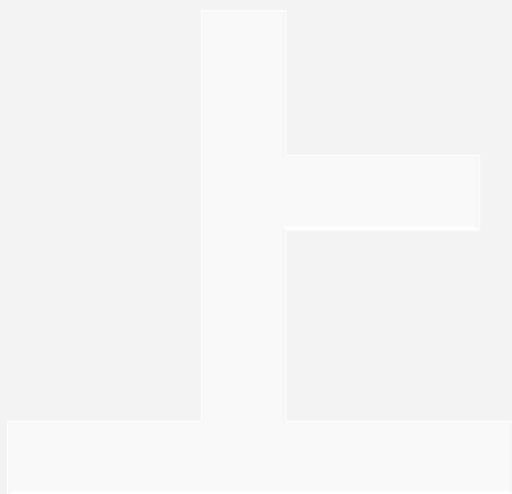
    read(connfd, buf, 11);
    printf("Server say:%s\n", buf);
    write(connfd, buf, 11);
    close(connfd);
}
close(listenfd);
}
```

TCP编程练习:



1. ftp服务器

- 1.1 显示服务器文件列表
- 1.2 下载
- 1.3 上传



联系方式

上 嵌

课程总结



本节课程内容

- TCP范式
- TCP客户端
- TCP服务端
- UDP范式
- UDP服务端
- UDP客户端



下节课

- 高级函数
- 多进程网络编程
- 心跳机制
- 本地Socket

嵌入式