网络通信

是进程间通信的方式

1. 协议

一种规则，是数据传输和数据解释的规则

2. 网络体系结构

1.1 OSI & ISO 模型

网络采用分而治之的方法设计，将网络的功能划分为不同的模块，以分层的形式有机组合在一起。

每层实现不同的功能，其内部实现方法对外部其他层次来说是透明的。每层向上层提供服务，同时使用下层提供的服务

网络体系结构即指网络的层次结构和每层所使用协议的集合

不同层的用途：

应用层 提供用户接口

表示层 加密 解密 编码 解码

会话层 保证不同应用间的数据区分

传输层 提供可靠或者不可靠的数据传输 及数据重传前的错误纠正

网络层 主要负责数据包的格式和地址的格式

数据链路层 规定或者说控制访问下面的物理层 标识链接到介质上的网络设备 在介质上发送数据之前 如何成帧

物理层

不同层的典型协议

传输层 常见协议有TCP/UDP协议。

应用层 常见的协议有HTTP协议，FTP协议。

网络层 常见协议有IP协议、ICMP协议（ping）、IGMP协议。

网络接口层 常见协议有ARP协议、RARP协议。

传输层：

TCP传输控制协议（Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议。

UDP用户数据报协议（User Datagram Protocol）是OSI参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。

应用层：

HTTP超文本传输协议（Hyper Text Transfer Protocol）是互联网上应用最为广泛的一种网络协议

FTP文件传输协议（File Transfer Protocol）

网络层：

IP协议是因特网互联协议（Internet Protocol）

ICMP协议是Internet控制报文协议（Internet Control Message Protocol）它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。

IGMP—分组管理协议

数据链路层

ARP RARP

ARP：地址解析协议 根据ip地址查询到MAC地址

RARP:逆地址解析协议根据MAC地址找到对应ip地址

1.2 TCP/IP 模型

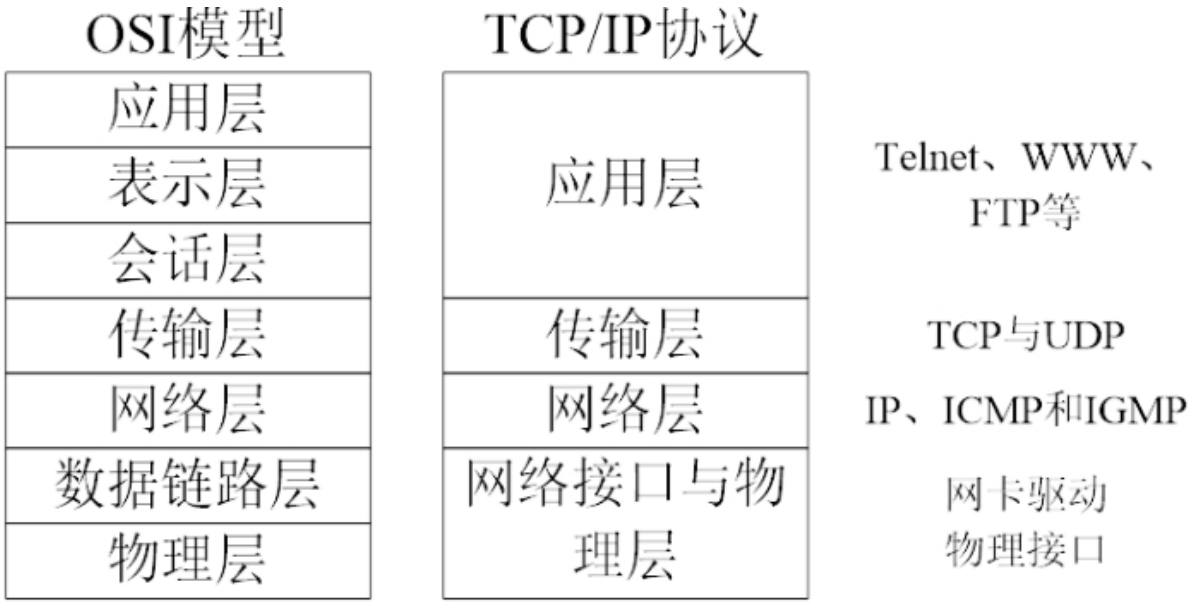
4层：

应用层

传输层

网络层

网络接口层



1.3 数据包的封装

3. 提前量

3.1 通信五要素：

协议，本地地址，本地端口，远程地址，远程端口

3.2 套接字socket

一个应用层和传输层之间的**抽象层**，是特殊的IO接口。

Linux环境下，用于表示进程间网络通信的**特殊文件类型** 通过函数获取： int sockfd = socket();

每个 socket 都可用网络地址结构来表示。随后各种操作都是通过socket 描述符来实现的

3.2.1 网络地址结构

网络地址结构：

```
{
  协议、本地地址、本地端口
}
```

3.2.2 套接字类型

标准套接字

```
SOCK_STREAM      流式套接字 TCP

SOCK_DGRAM      数据报套接字 UDP
```

原始套接字

```
SOCK_RAW       原始套接字
```

4. UDP

User Datagram Protocol 用户数据报协议

4.1 概念

用户数据报协议，是不可靠的无连接的协议。

在数据发送前，因为不需要进行连接，所以可以进行高效率的数据传输。

4.2 适用情况

无需创建连接，所以UDP开销较小，数据传输速度快，实时性较强。多用于对实时性要求较高的通信场合，如视频会议、电话会议等

4.3 通信模式(C/S, B/S)

C/S模式** **client server**

传统的网络应用设计模式，客户机(client)/服务器(server)模式。需要在**通讯两端各自部署客户机和服务器**来完成数据通信。

B/S模式**

浏览器()/服务器(server)模式。只需在一端部署服务器，另外一端使用每台PC都默认配置的浏览器即可完成数据的传输。

优缺点

优点：

1）对于C/S模式来说，客户端位于目标主机上可以保证性能，将数据缓存至客户端本地，从而提高数据传输效率。

客户端和服务器程序由一个开发团队创作，所以他们之间所采用的协议相对灵活。

传统的网络应用程序及较大型的网络应用程序都首选C/S模式进行开发

缺点：

1）由于客户端和服务器都需要有一个开发团队来完成开发。工作量将成倍提升，开发周期较长。

2）从用户角度出发，需要将客户端安插至用户主机上，对用户主机的安全性构成威胁。这也是很多用户不愿使用C/S模式应用程序的重要原因。

B/S模式

1）由于没有独立的客户端，使用标准浏览器作为客户端，其工作开发量较小。只需开发服务器端即可。

2）由于其采用浏览器显示数据，因此移植性非常好，不受平台限制

缺点：

1）由于使用第三方浏览器，因此网络应用支持受限。

2）没有客户端放到对方主机上，缓存数据不尽如人意，从而传输数据量受到限制。

3）必须与浏览器一样，采用标准http协议进行通信，协议选择不灵活。

因此在开发过程中，模式的选择由上述各自的特点决定。根据实际需求选择应用程序设计模式。

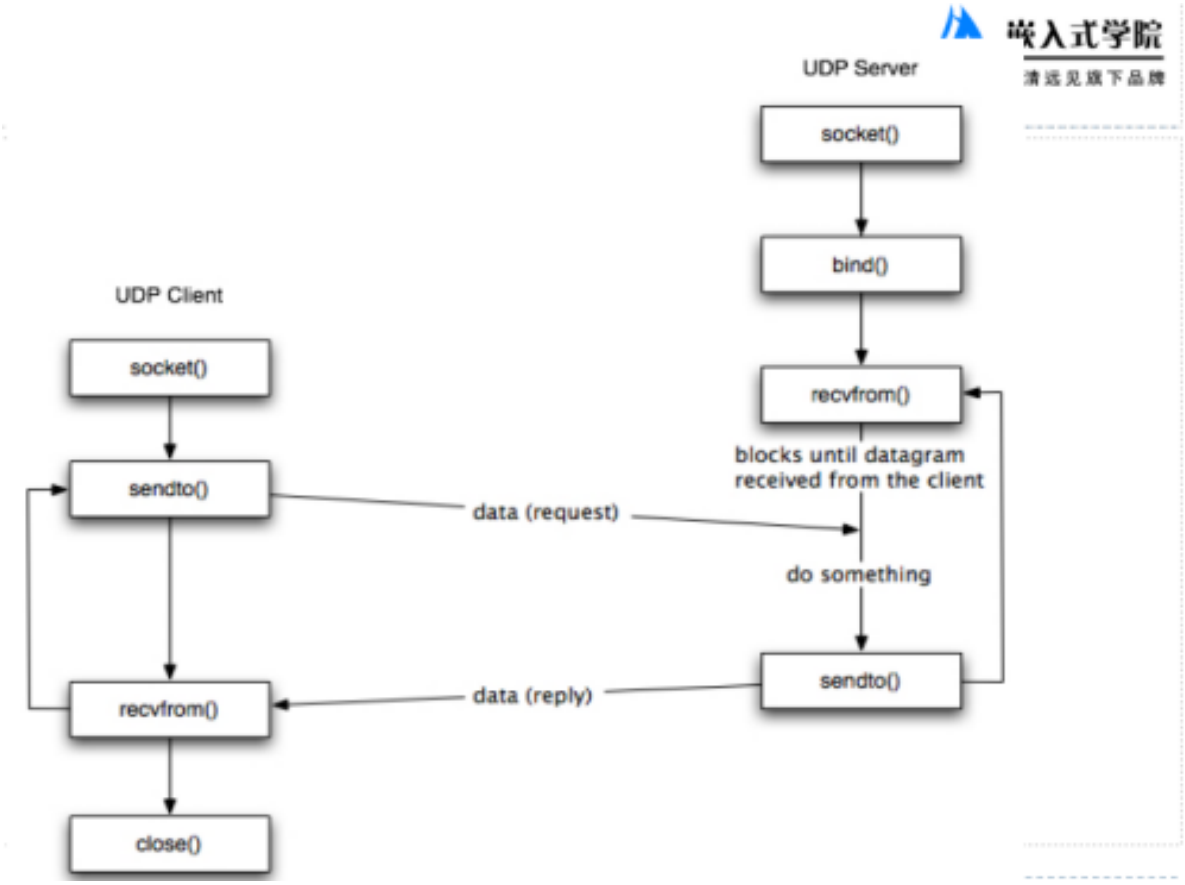
C/S + UDP流程

server:

1. 创建socket
2. bind绑定(本机ip+端口+套接字id)
3. 接受数据recvfrom()
4. 数据处理
5. 关闭套接字close()

client: (一般主动发送)

1. 创建socket
2. 发送数据sendto()
3. 关闭套接字close()



4.4 函数

4.4.1 创建套接字: socket()

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

int socket(int domain, int type, int protocol);

//参数
domain:域，地址族，网际协议
    AF_INET：IPV4 互联网协议第四版
    AF_INET6:IPV6 互联网协议第六版

type：套接字类型
    SOCK_STREAM 流式套接字 使用tcp传输协议
    SOCK_DGRAM  数据报套接字 使用udp传输协议
    SOCK_RAW    原始套接字

protocol：协议号
    0  表示系统自动根据参数2 找到协议号
    IPPROTO_TCP  TCP协议号
    IPPROTO_UDP  UDP协议号

//返回值
成功返回套接字描述符 失败-1
```

例子：IPv4

```
int sockfd = socket(AF_INET,SOCK_DGRAM,0);
if(sockfd<0)
{
    perror("socket");
    exit(-1);
}
```

4.4.2 端口及地址

端口（号）是一个无符号短整型，范围在0~65535之间

```
    1 - 1023   系统占用
    1024 - 49151   被登记过
    49152 - 65535   动态的 可以用的
```

TCP 端口号 和 UDP 端口号独立，互不影响。

地址：ip: “192.168.31.163”

注意是个字符串，有两种格式

网络格式

主机格式

我们需要把主机格式转化成网络格式

4.4.3 网络地址结构体: struct sockaddr_in

协议+本地地址+本地端口封装在一起

- 通用地址结构体（一般不用）

```
struct sockaddr {

    sa_family_t sa_family;//地址族
    char     sa_data[14];//ip+port
}
```

- 只用于IPv4

```
struct sockaddr_in {

    sa_family_t   sin_family;//AF_INET: ipv4地址结构类型
    in_port_t    sin_port; //端口号(网络格式)
    struct in_addr sin_addr; //ip(网络格式)
};

其中struct in_addr为网络格式地址结构体
typedef uint32_t in_addr_t;      //所以in_addr_t的本质是短整型
struct in_addr {
    in_addr_t s_addr;//网络格式地址
};
```

如何填充：

struct sockaddr_in s;

s.sin_family = AF_INET;

s.sin_port = 网络格式的端口（4567）

s.sin_addr.s_addr = 网络格式地址

4.4.4 把端口和ip地址转化成网络格式: htons/htonl()

由于sockaddr_in结构体中，端口（sin_port）和ip地址（sin_addr.s_addr）都需要网络格式，所以：

1. 对于端口：htons

```
myself_addr.sin_port = htons(5678); //5678自己定义的端口号
```

2. 对于IP地址：htonl

```
myself_addr.sin_addr.s_add = htonl(INADDR_ANY);
//INADDR_ANY表示网卡任意一个ip地址
```

函数原型：

```
    #include <arpa/inet.h>

•    uint16_t htons(uint16_t hostshort);//h host n net s short

•       功能：将主机格式的short类型数据转为网络格式short类型数据

•       返回值：成功返回转换后的结果  失败-1
•
•       #include <arpa/inet.h>

•    uint32_t htonl(uint32_t hostlong);
```

- 功能:将主机格式的long 类型转为 网络格式long类型
- 返回值：成功返回转换后的结果 失败-1

```
struct sockaddr_int myself_addr;
myself_addr.sin_family = AF_INET;
myself_addr.sin_port =  htons(5678);
myself_addr.sin_addr.s_add = htonl(INADDR_ANY);
```

4.4.5 绑定地址信息: bind()

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

功能：绑定

参数1：socket返回值

参数2： 地址信息

参数3：参数2的大小

返回值：成功0 失败-1
```

注意1：

第二个参数是地址结构体指针，也就是需要传进一个地址结构体的地址。

且，如果我们刚刚定义的地址结构体是只用于IPv4的（struct sockaddr_in），那么我们需要强转成通用地址结构体（struct sockaddr）

```
bind(sockfd, (struct sockaddr*) (&myself_addr),sizeof(myself_addr));
//注意(struct sockaddr*)
```

注意2:

一定要做错误处理，确保端口不会被占用

```
int ret = bind(sockfd, (struct sockaddr*) (&myself_addr),sizeof(myself_addr));

if(ret < 0)
{
    perror("bind");
    close(sockfd);
    exit(-1);
}
```

4.4.6 接收消息: recvfrom()

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

功能：接受数据

参数1：socket返回值

参数2：存放收到的数据

参数3：参数2的大小

参数4：如果没有数据到来 阻塞等待还是不等待。0表示阻塞，MSG_DONTWAIT 不等待

参数5：发送方的地址信息，不关心则用NULL

参数6：发送方地址信息长度，不关心则用NULL **注意：传的实参必须初始化**

返回值：成功返回收到的字节数 失败-1  ssize_t  有符号整型
```

注意：必须做错误处理

```
char buf[100] = "\0";
ssize_t ret_recv = recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);
if(ret_recv < 0)
{
    perror("recvfrom");
    close(sockfd);
    exit(-1);
}
```

4.4.6.1 网络格式地址转为字符串格式地址: inet_ntoa()

主要用于recevfrom（接收消息）后，关心发送方信息

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr in);

功能：将网络格式地址转为字符串格式地址
```

```
char buf[100] = "\0";
struct sockaddr_in snd_addr;
socklen_t len = sizeof(struct sockaddr_in);

ssize_t ret_rcv = recvfrom(sockfd, buf, sizeof(buf),0, (struct
sockaddr*)&snd_addr, &len);
```

```
if(ret_recv < 0)
{
    perror("recvfrom");
    close(sockfd);
    exit(-1);
}

printf("%s\n",inet_ntoa(snd_addr.sin_addr));
printf("say: %s\n",buf);
```

4.4.7 关闭套接字

```
close(sockfd);
```

4.4.8 终端模拟client: nc -u

1. 查看ip

```
ifconfig
```

2. 链接

```
// nc -u ip port

nc -u 192.168.xx.xx 5678
```

4.4.9 发送数据: sendto()

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

功能：发送数据给对方

参数1：socket返回值

参数2：存放要发送的数据

参数3：参数2的大小

参数4：套接字缓存满 阻塞还是不阻塞 0表示阻塞 MSG_DONTWAIT 不阻塞

参数5：接收方的地址信息

参数6：参数5的大小

返回值：成功返回发送的字节数 失败-1  s
```

4.4.10 字符串格式IP转化成网络格式: inet_addr()

一般用于client的ip

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);

功能：将字符串格式地址转成网络格式地址
```

4.5 C/S UDP整体代码

client写入“hello world”发送给server

server.c

```
#include "apue.h"

int main()
{
    int sockfd = socket(AF_INET, SOCK_DGRAM,0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }
    printf("sockfd %d\n",sckfd);

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);    //根据不同server地址需要修改

    int my_bind = bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr));
    if(my_bind < 0)
    {
        perror("bind");
        close(sockfd);
        exit(-1);
    }

    ssize_t ret;
    char buf[100] = "\0";

    ret = recvfrom(sockfd,buf,sizeof(buf),0,NULL,NULL);

    if(ret < 0)
    {
```

```
        perror("recvfrom");
        close(sockfd);
        exit(-1);
    }
    close(sockfd);
}
```

client.c

```
#include "apue.h"

int main()
{
    int sockfd = socket(AF_INET, SOCK_DGRAM,0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = inet_addr("192.168.31.24");   //根据不同server地
址需要修改

    ssize_t ret;
    char buf[] = "hello world";

    ret = sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr*)&server_addr,
sizeof(server_addr));

    if(ret < 0)
    {
        perror("sendto");
        close(sockfd);
        exit(-1);
    }
    close(sockfd);
}
```

运行时要先执行server

4.6 C/S UDP 回射服务器

服务器从客户端收到什么，就发回给客户端

一些函数：

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

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

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

server.c

```
"apue.h"

int main()
{
    //创建套接字
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }

    //bind绑定
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    if(ret < 0)
    {
        perror("bind");
        close(sockfd);
        exit(-1);
    }

    //从client接收
    struct sockaddr_in client_addr;
    socklen_t len = sizeof(struct sockaddr_in);
    ssize_t ret_rcv;    //client的大小
    char buf[100] = "\0";

    //发回给client
    ssize_t ret_send;

    while(1)
    {
```

```
        ret_rcv = recvfrom(sockfd, buf, sizeof(buf),0,(struct
sockaddr*)&client_addr, &len);
        if(ret_rcv < 0)
        {
            perror("recvfrom");
            close(sockfd);
            exit(-1);
        }

        printf("recv: %s\n",buf);

        ret_send = sendto(sockfd, buf, ret_rcv, 0, (struct
sockaddr*)&client_addr, sizeof(client_addr));
        if(ret_send < 0)
        {
            perror("sendto");
            close(sockfd);
            exit(-1);
        }

        bzero(buf, sizeof(buf));
    }
    close(sockfd);
}
```

client.c

```
#include "apue.h"

int main()
{
    //创建套接字
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }



    //send to server
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = inet_addr("192.168.126.130");

    ssize_t ret_send;   //client的大小
    char buf[100] = "\0";

    //receive from server
    struct sockaddr_in client_addr;
    socklen_t len = sizeof(struct sockaddr_in);
    ssize_t ret_rcv;

    while(1)
```

```
    {
        scanf("%s",buf);
        ret_send = sendto(sockfd, buf, strlen(buf)+1, 0, (struct
sockaddr*)&server_addr, sizeof(server_addr));
        if(ret_send < 0)
        {
            perror("sendto");
            close(sockfd);
            exit(-1);
        }
        bzero(buf, sizeof(buf));
        ret_rcv = recvfrom(sockfd, buf, sizeof(buf),0,(struct
sockaddr*)&client_addr, &len);    //后两个可以直接写NULL
        if(ret_rcv < 0)
        {
            perror("recvfrom");
            close(sockfd);
            exit(-1);
        }
        printf("%s\n",buf);
        bzero(buf, sizeof(buf));
    }
    close(sockfd);
}
```

5. TCP

Transmission Control Protocol)传输控制协议

5.1 概念

传输层主要应用的协议模型有两种，一种是TCP协议，另外一种则是UDP协议。TCP协议在网络通信中占主导地位，绝大多数的网络通信借助TCP协议完成数据传输。但UDP也是网络通信中不可或缺的重要通信手段。

5.1.1 特点

面向连接 传输层协议

能提供可靠的 数据无丢失 数据无失序 数据无重发到达

适用于传输质量要求高以及传输大量数据的情况

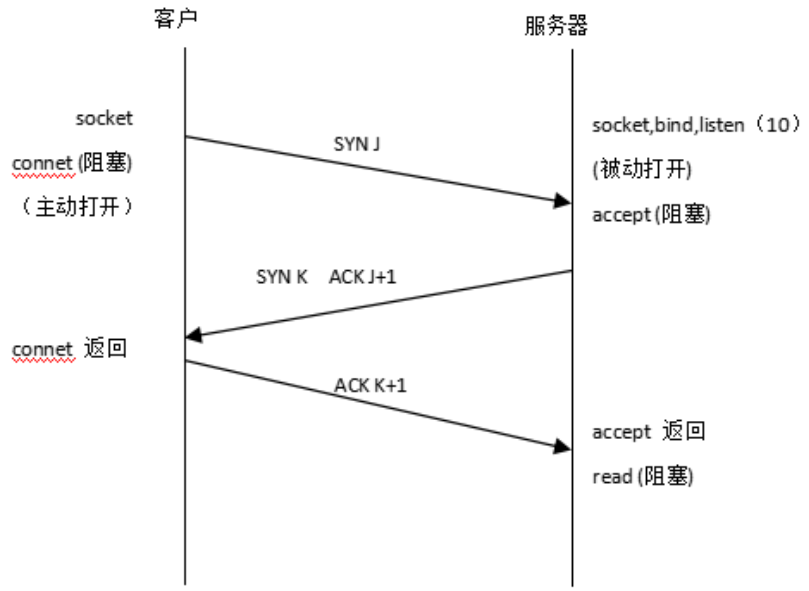
5.1.2 TCP和UDP区别

UDP 面向无连接 不可靠 基于数据报流 快

TCP 面向连接 可靠 基于字节流 慢

5.1.3 三次握手与四次挥手

在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。



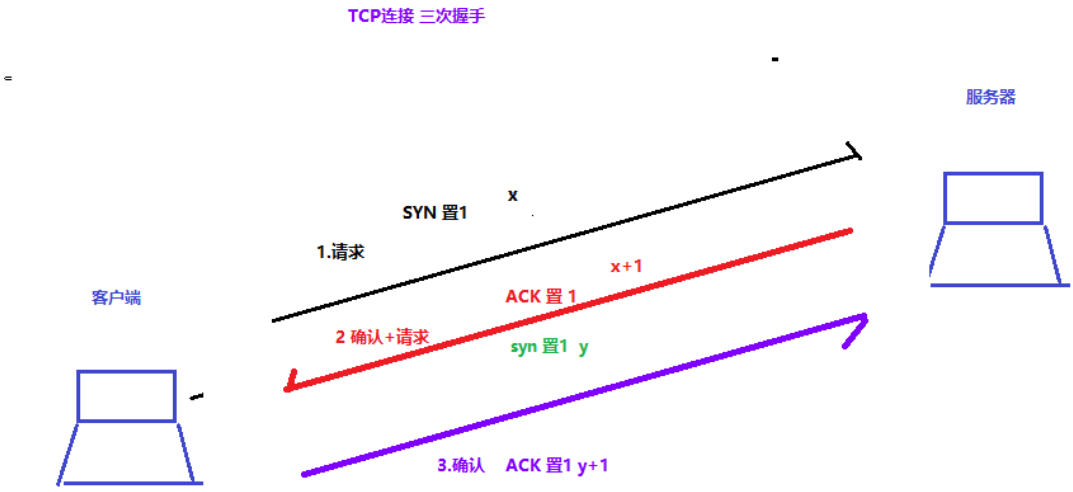
服务器必须准备好接受外来的连接。这通过调用socket、 bind和listen函数来完成，称为被动打开(passive open)。

第一次握手：客户通过调用connect进行主动打开。这引起客户TCP发送一个SYN标志位（表示请求连接）（SYN=J）并进入**SYN_SENT**状态，等待服务器的确认。

第二次握手：服务器必须确认客户的SYN，同时自己也得发送一个SYN请求

服务器向客户发送SYN请求和对客户SYN的ACK（表示确认/应答），此时服务器进入**SYN_RCVD**状态。

第三次握手：客户收到服务器的SYN+ACK。向服务器发送确认ACK 发送完毕，客户服务器进入**ESTABLISHED**状态，完成三次握手。



5.2 C/S + TCP通信流程

server：

1. 创建套接字 socket() : 注意因为是TCP，所以要使用流式套接字SOCK_STREAM
2. 绑定 bind
3. 监听 listen()

4. 阻塞等待三次握手连接 accept()
5. 读写数据 read/write
6. 关闭套接字 close

client

1. 创建套接字 socket
2. 发起三次握手连接 connect()
3. 读写数据 write/read
4. 关闭套接字 close

5.3 函数

5.3.1 监听: listen()

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

int listen(int sockfd, int backlog);

//功能：监听 设置同时允许多少个连接请求等待处理

//参数：
•    sockfd:套接字 socket函数返回值

•    backlog：等待队列长度 (最多可以等待多少个连接请求)

//返回值：成功0 失败-1
```

5.3.2 阻塞等待客户端连接: accept()

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);//recvfrom =
accept+read

•    功能：阻塞等待客户端连接 并取得对方地址信息（只能监听不能收数据）

参数：
•      sockfd:监听套接字:监听sockfd这部手机收到的电话

•      addr:对方的地址信息（来电显示） 不关心可以传递NULL

•      addlen:对方地址信息长度  传递的参数必须初始化 不关心可以传递NULL

•    **返回值：成功会返回一个与客户端连接的新的套接字**  失败-1
```

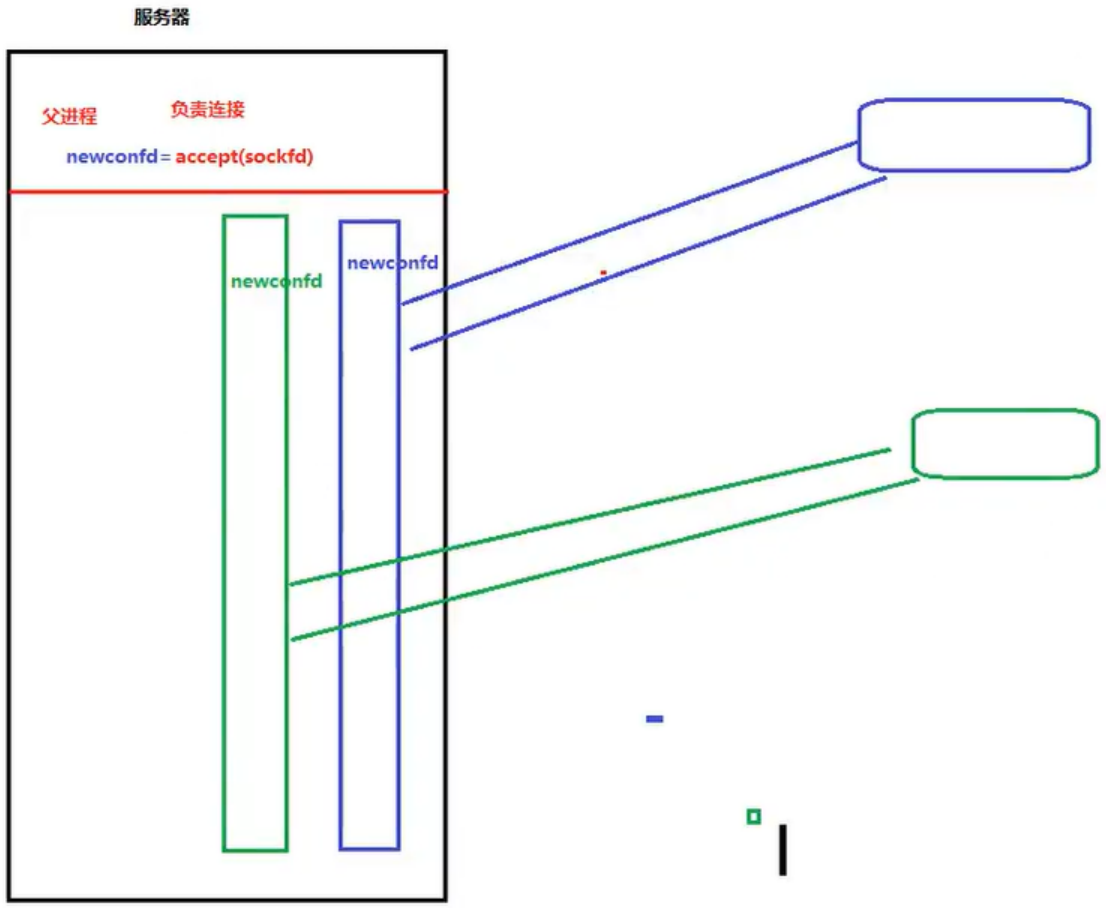
注意：返回值是一个与客户端连接的新的套接字（类似主机转播到分机），之后与客户端通信必须用新的套接字

成功返回表示三次握手的完成！！！

为什么返回新的套接字

为并发服务器做准备：

sockfd每接收到一个客户的连接，就创建一个子进程newconfd处理连接



5.3.3 读/写数据：read() write()

```
char buf[100] = "\0";
read(newconfd, buf, sizeof(buf));
printf("read is: %s\n",buf);
```

```
char buf[100] = "\0";
scanf("%s",buf);
write(sockfd, buf, sizeof(buf));
```

5.3.4 发起三次握手：connect()

客户端连接服务器，发起请求连接

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


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

- 功能：客户端连接服务器(发起连接请求 完成三次握手)
- 参数：
- sockfd:套接字 socket函数返回值
- addr:服务器端地址信息
- addrlen:服务器端地址信息长度
- 返回值：成功0 失败-1

5.3.4 关闭套接字: close

```
close(listenfd);
close(newconfd);
```

注意两个都要关闭

5.4 C/S TCP整体代码

部分函数:

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);//recvfrom = accept+read

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

server.c

```
#include "apue.h"

int main()
{
    //创建套接字
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if(listenfd < 0)
```

```
    {
        perror("socket");
        exit(-1);
    }

    //绑定
    struct sockaddr_in self_addr;
    self_addr.sin_family = AF_INET;
    self_addr.sin_port = htons(5678);
    self_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    int ret = bind(listenfd, (struct sockaddr*)&self_addr, sizeof(self_addr));
    if(ret < 0)
    {
        perror("bind");
        close(listenfd);
        exit(-1);
    }

    //监听，设置最大连接值
    int ret_listen = listen(listenfd, 5);
    if(ret_listen < 0)
    {
        perror("listen");
        close(listenfd);
        exit(-1);
    }

    //阻塞等待连接
    int newconfd = accept(listenfd, NULL, NULL);
    if(newconfd<0)
    {
        perror("accept");
        close(listenfd);
        exit(-1);
    }

    //读写数据
    char buf[100] = "\0";
    read(newconfd, buf, sizeof(buf));
    printf("read: %s\n",buf);

    //关闭套接字
    close(listenfd);
    close(newconfd);

}
```

client.c

```
#include "apue.h"

int main()
{
    //创建套接字
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }

    //发起三次握手连接

    //服务器端地址信息
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = inet_addr("192.168.126.136");

    int ret = connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    if(ret < 0)
    {
        perror("connect");
        close(sockfd);
        exit(-1);
    }

    //读写数据
    char buf[100] = "\0";
    scanf("%s",buf);
    write(sockfd, buf, sizeof(buf));

    //关闭套接字
    close(sockfd);

}
```

5.5 C/S TCP回射服务器

server.c

```
#include "apue.h"

int main()
{
    //创建套接字
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if(listenfd < 0)
```

```
    {
        perror("socket");
        exit(-1);
    }

    //绑定
    struct sockaddr_in self_addr;
    self_addr.sin_family = AF_INET;
    self_addr.sin_port = htons(5678);
    self_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    int ret = bind(listenfd, (struct sockaddr*)&self_addr, sizeof(self_addr));
    if(ret < 0)
    {
        perror("bind");
        close(listenfd);
        exit(-1);
    }

    //监听，设置最大连接值
    int ret_listen = listen(listenfd, 5);
    if(ret_listen < 0)
    {
        perror("listen");
        close(listenfd);
        exit(-1);
    }

    //阻塞等待连接
    int newconfd = accept(listenfd, NULL, NULL);
    if(newconfd<0)
    {
        perror("accept");
        close(listenfd);
        exit(-1);
    }

    //读写数据
    char buf[100] = "\0";
    while(1)
    {
        read(newconfd, buf, sizeof(buf));
        printf("read: %s\n",buf);
        write(newconfd, buf, sizeof(buf));
        bzero(buf,sizeof(buf));
    }

    //关闭套接字
    close(listenfd);
    close(newconfd);

}
```

client.c

```
#include "apue.h"

int main()
{
    //创建套接字
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }

    //发起三次握手连接

    //服务器端地址信息
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = inet_addr("192.168.126.136");

    int ret = connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr));
    if(ret < 0)
    {
        perror("connect");
        close(sockfd);
        exit(-1);
    }

    //读写数据
    char buf[100] = "\0";
    while(1)
    {
        scanf("%s",buf);
        write(sockfd, buf, sizeof(buf));
        bzero(buf,sizeof(buf));
        read(sockfd, buf, sizeof(buf));
        puts(buf);
    }

    //关闭套接字
    close(sockfd);

}
```

6. 多进程TCP实现并发

e.g. QQ聊天中，发送数据的同时可以接收数据

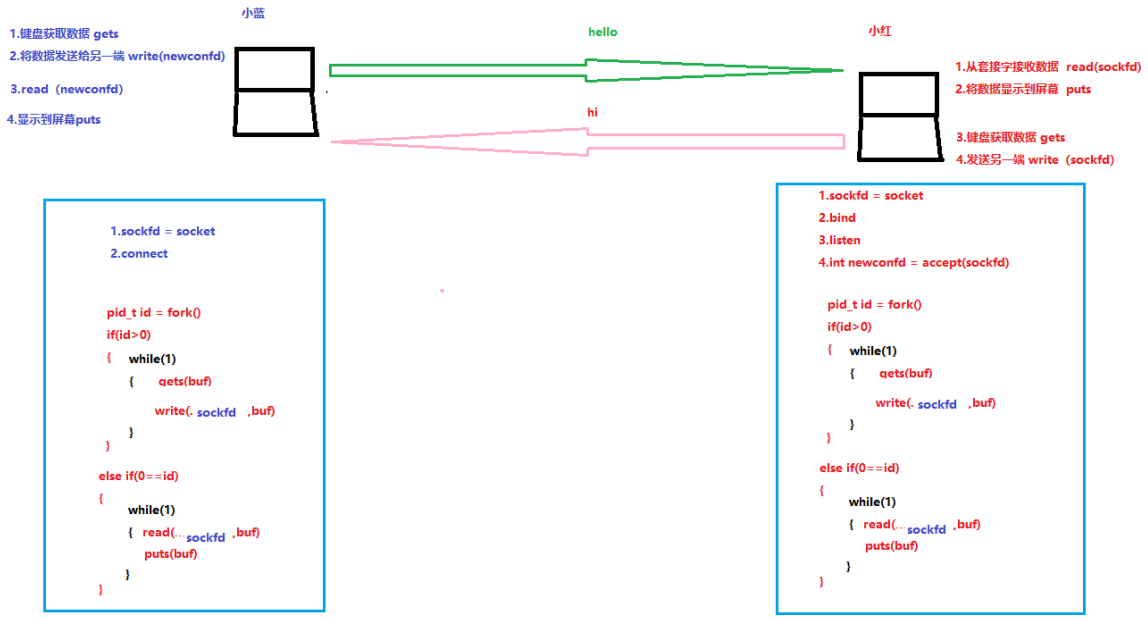
6.1 流程

server.c

1. sockfd = socket()
2. bind
3. listen
4. int newconfd = accept()
5. pid_t pid = fork()
6. if (pid == 0) gets(buf) write(newconfd, buf, strlen(buf)+1)
7. if(pid > 0) read(newconfd, buf, sizeof(buf)), puts(buf)

client.c

1. sockfd = socket()
2. connect()
3. pid_t pid = fork()
4. if (pid == 0) gets(buf) write(newconfd, buf, strlen(buf)+1)
5. if(pid > 0) read(newconfd, buf, sizeof(buf)), puts(buf)



6.2 整体代码

server.c

```
#include "apue.h"

int main()
{
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
```

```
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    int ret = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    if(ret < 0)
    {
        perror("bind");
        close(sockfd);
        exit(-1);
    }

    listen(sockfd, 10);

    struct sockaddr_in client_addr;
    socklen_t len = sizeof(client_addr);
    int newconfd = accept(sockfd, (struct sockaddr *)&client_addr, &len);
    if(newconfd < 0)
    {
        perror("accept");
        close(sockfd);
        close(newconfd);
        exit(-1);
    }

    char buf[100] = "\0";
    int fd = fork();
    if(0 == fd)
    {
        while(1)
        {
            scanf("%s",buf);
            write(newconfd, buf, strlen(buf)+1);
            bzero(buf,sizeof(buf));
        }
    }
    else if(fd > 0)
    {
        while(1)
        {
            read(newconfd, buf, sizeof(buf));
            puts(buf);
            bzero(buf,sizeof(buf));
        }
    }
    else
    {
        perror("fork");
        close(sockfd);
        close(newconfd);
        exit(-1);
    }
```

```
}
```

client.c

```
#include "apue.h"

int main()
{
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5678);
    server_addr.sin_addr.s_addr = inet_addr("192.168.126.136");

    socklen_t len = sizeof(server_addr);
    int ret =  connect(sockfd, (struct sockaddr *)&server_addr, len);
    if(ret < 0)
    {
        perror("connect");
        close(sockfd);
        exit(-1);
    }

    char buf[100] = "\0";
    int fd = fork();
    if(0 == fd)
    {
        while(1)
        {
            scanf("%s",buf);
            write(sockfd, buf, strlen(buf)+1);
            bzero(buf,sizeof(buf));
        }
    }
    else if(fd > 0)
    {
        while(1)
        {
            read(sockfd, buf, sizeof(buf));
            puts(buf);
            bzero(buf,sizeof(buf));
        }
    }
    else
    {
        perror("fork");
        close(sockfd);
```

```
        exit(-1);
    }
}
```

bind错误处理：setsockopt()

bind: Address already in use

我们刚才的程序中，可能会报上面的错，原因有两种：

1. 同时运行两个 server

   解决：ps -aux杀掉 ./s

2. 先退出了server，之后再运行server会造成地址未解绑

   解决：添加一个地址复用（立即解绑）的函数

   ```
   setsockopt 设置套接字选项功能。

   函数原型：
   int setsockopt(int sockfd, int level, int optname,const void optval,
   socklen_t optlen);

   //设置 套接字的 选项功能

   参数1： 套接字描述符。
   参数2： 选项所属协议层。SOL_SOCKET:

   参数3： 选项名称。 SO_REUSERADDR 地址复用 SO_BROADCAST 允许发送广播 int
   参数4： 保存选项值。
   参数5： 选项值的长度。
   返回值： 成功： 0
   失败： -1
   ```

   实现方法：在bind之前添加以下代码：

   ```
   int on = 1;
   setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
   // SO_REUSERADDR可以使地址端口马上重用
   ```

代码简化（父子进程读写）

上面的代码会发现，子进程：从标准输入中读，写到newconfd/sockfd中；父进程：从sockfd中读，写到标准输出中；所以可以创建一个新的函数

```
void rd_op(int fd_rd, int fd_rw)
//参数1：读的id，参数2：写的id
{
    char buf[100] = "\0";
    ssize_t ret = read(fd_rd, buf, sizeof(buf));

    write(fd_rw, buf, sizeof(buf));
}
```

例如client中：

```
if(fd == 0)
{
    while(1)
    {
        rd_op(0, sockfd);
    }
}
else if(fd > 0)
{
    while(1)
    {
        rd_op(sockfd, 1);
    }
}
```

server只需要把sockfd改成newconfd