知识点1【字节序】（了解）

1、字节序的概述

案例1：确定主机的大小端

知识点2【主机和网络的字节】（了解）

1、概述

2、主机字节序 转 网络字节序（发送消息）

3、网络字节序 转 主机字节序（收数据）

知识点3【IP地址转换】（了解）

1、IP地址的形式：

2、将点分十进制数串 转成　32位无符号整型数据（默认大端）

3、将32位无符号整型数据（默认大端） 转成　点分十进制数串

知识点4【UDP的编程流程】

1、概述

2、网络通信 需要解决3大问题（应用层）

3、UDP的编程架构

知识点5【UDP的编程API】(重要)

1、socket创建通信的套接字

2、IPv4地址结构

3、通用地址结构（类型转换）

4、两种地址结构使用场合

5、sendto发送数据

6、bind给udp套接字绑定固定的port、ip信息

7、recvfrom接收udp的消息

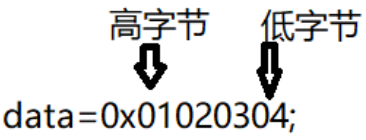
作业：UDP_QQ聊天程序

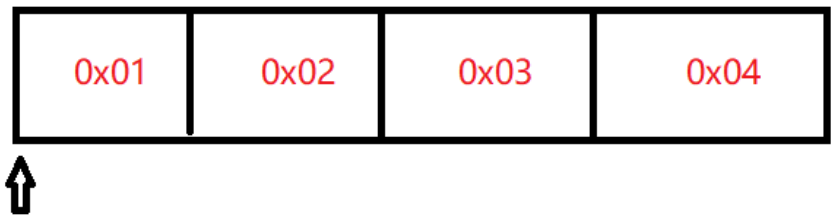
知识点1【字节序】（了解）

1、字节序的概述

是指多字节数据的存储顺序（多字节看成整体）

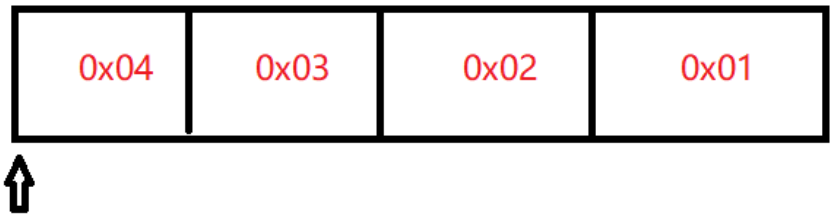
小端格式、大端格式





低地址

大端：将高字节数据存放在低地址。



低地址

小端：将低字节数据存放在低地址

大小端是由系统决定，用户不能确定。

案例1：确定主机的大小端

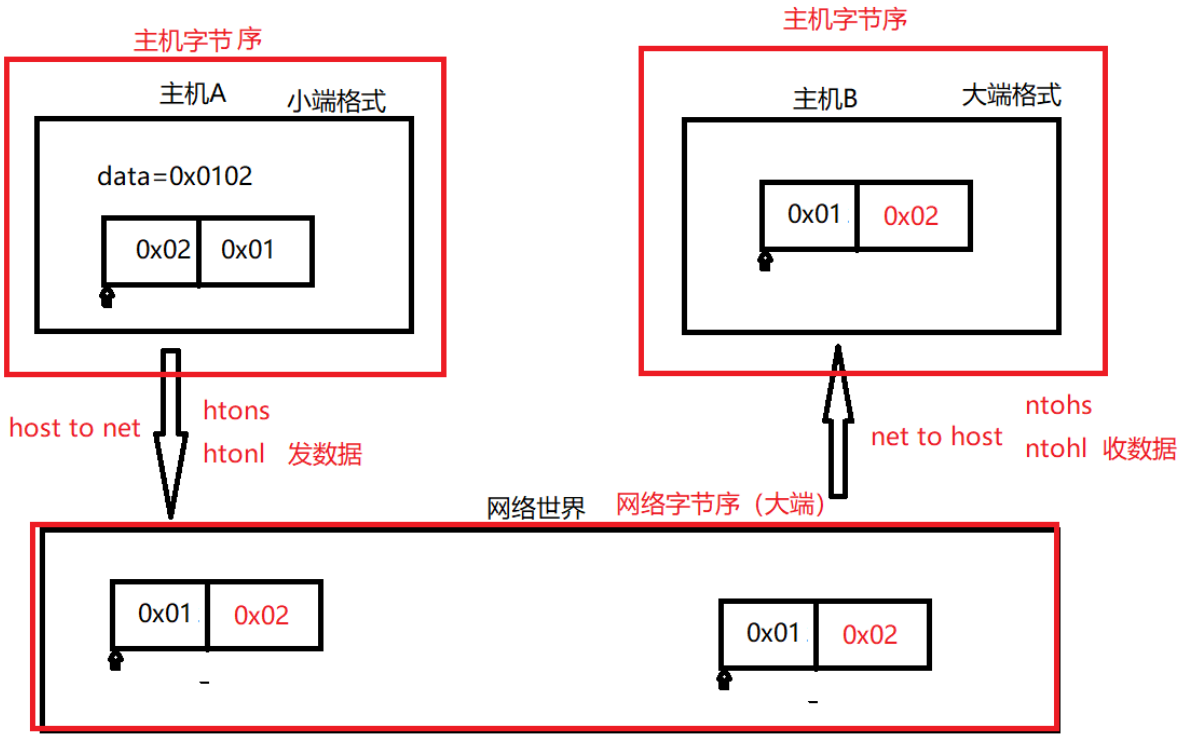
```c
#include <stdio.h>
union DATA
{
    unsigned short data;
    unsigned char a[2];
};

```

```c
int main(int argc, char const *argv[])
{
    union DATA d;
    d.data=0x0102;

    if((d.a[0]==0x01) && (d.a[1]==0x02))
    {
        printf("大端格式\n");
    }
    else if((d.a[0]==0x02) && (d.a[1]==0x01))
    {
        printf("小端格式\n");
    }

    return 0;
}
```

```
edu@edu:~/work/net/day01$ ls
01_test.c
edu@edu:~/work/net/day01$ gcc 01_test.c
edu@edu:~/work/net/day01$ ./a.out
小端格式
edu@edu:~/work/net/day01$
```

知识点2【主机和网络的字节】（了解）

1、概述



2、主机字节序 转 网络字节序（发送消息）

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

htons:将2个字节的主机字节序 转换成 网络字节序

htons:将主机字节序的端口 转换成网络字节序

```
uint16_t htons(uint16_t hostshort);
```

htonl:将主机字节序的IP地址 转换成网络字节序

```
uint32_t htonl(uint32_t hostlong);
```

3、网络字节序 转 主机字节序（收数据）

ntohs:将网络字节序的端口 转换成 主机字节序

```
uint16_t ntohs(uint16_t netshort);
```

ntohl:将网络字节序的IP地址 转换成 主机字节序

```
 uint32_t ntohl(uint32_t netlong);
```

案例1：

```c
#include <stdio.h>
#include <arpa/inet.h>
int main(int argc, char const *argv[])
{
    unsigned short data = 0x0102;
    printf("转换结果:%x\n", htons(data));
    return 0;
}
```

```
edu@edu:~/work/net/day01$ ls
01_test.c  02_test.c  a.out
edu@edu:~/work/net/day01$ gcc 02_test.c
edu@edu:~/work/net/day01$ ./a.out
转换结果:201
edu@edu:~/work/net/day01$
```

知识点3【IP地址转换】（了解）

1、IP地址的形式：

"10.9.21.201":点分十进制数串

网络的IP地址：32位无符号整型数据

2、将点分十进制数串 转成　32位无符号整型数据（默认大端）

```
#include <arpa/inet.h>
int inet_pton(int af, const char *src, void *dst);
```

af:AF_INET　IPv4　　AF_INET6 IPv6

src:点分十进制数串的首元素地址

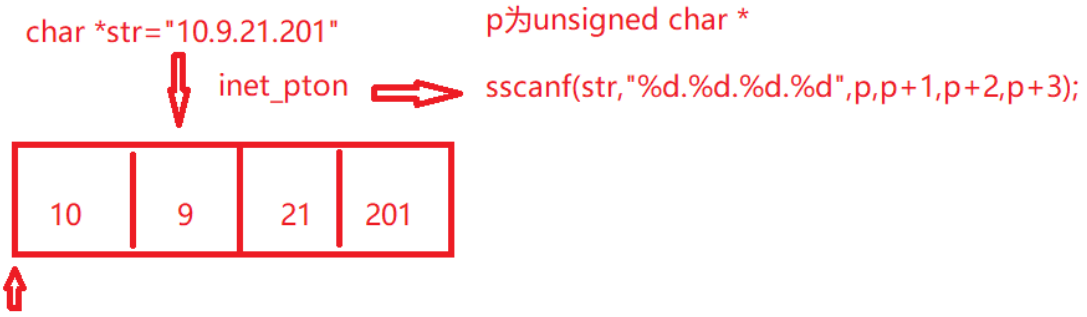
dst：转换的结果

返回值：成功返回 1 、 失败其他

案例1：

```
#include <stdio.h>
#include <arpa/inet.h>
int main(int argc, char const *argv[])
{
    char *str = "10.9.21.201";
    unsigned int addr=0;

    inet_pton(AF_INET, str, (void *)&addr);
    printf("addr = %u\n", addr);

     unsigned char *p = (unsigned char *)&addr;
     printf("%d %d %d %d\n", *p, *(p+1), *(p+2), *(p+3));

     return 0;
}
```

```
edu@edu:~/work/net/day01$ gcc 03_test.c
edu@edu:~/work/net/day01$ ./a.out
addr = 3373598986
10 9 21 201
edu@edu:~/work/net/day01$
```



3、将32位无符号整型数据（默认大端） 转成 点分十进制数串

```
const char *inet_ntop(int family, const void *addrptr,
char *strptr, size_t len);
```

参数：

family：AF_INET IPv4 AF_INET6 IPv6

addrptr：32位无符号整型数据的地址

strptr：点分十进制数串的首元素地址

len：点分十进制数串的最大长度（16字节）

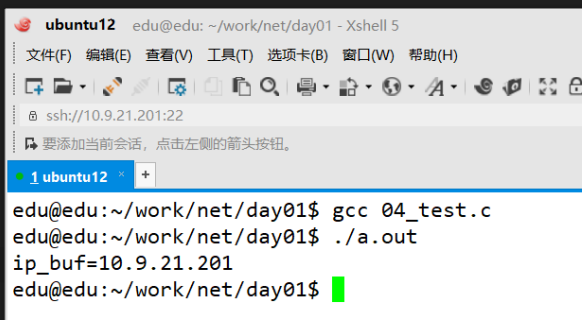
```
len 的宏定义
 #define INET_ADDRSTRLEN 16 //for ipv4
 #define INET6_ADDRSTRLEN 46 //for i
```

返回值：

点分十进制数串的首元素地址

案例1：

```c
#include <stdio.h>
#include <arpa/inet.h>
int main(int argc, char const *argv[])
{
    unsigned char buf[]={10,9,21,201};
    char ip_buf[16]="";

    inet_ntop(AF_INET, buf, ip_buf, 16);
    printf("ip_buf=%s\n", ip_buf);
    return 0;
```



知识点4【UDP的编程流程】

1、概述

UDP 特点

1、相比 TCP 速度稍快些

2、简单的请求/应答应用程序可以使用 UDP

3、对于海量数据传输不应该使用 UDP

4、广播和多播应用必须使用 UDP UDP

应用 DNS(域名解析)、NFS(网络文件系统)、RTP(流媒体)等

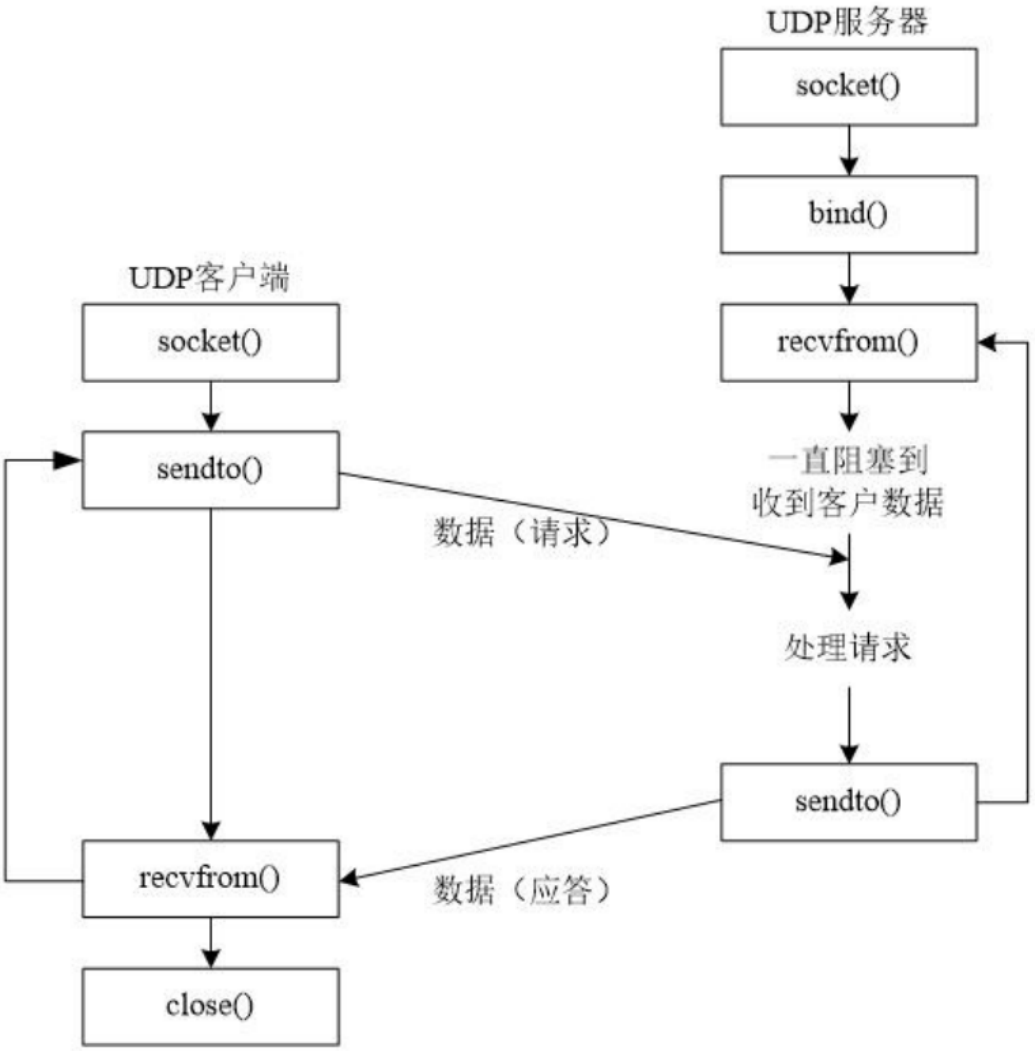
2、网络通信 需要解决3大问题（应用层）

协议、端口（port）、IP地址

socket套接字 是一个特殊的文件描述符，可以使用open write read close进行网络通信

通过socket函数调用得到这个网络通信的文件描述符（套接字）

3、UDP的编程架构



知识点5【UDP的编程API】(重要)

1、socket创建通信的套接字

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

参数:

domain协议族：协议AF_INET IPv4　AF_INET6 IPv6

type类型：SOCK_DGRAM（UDP套接字）、SOCK_STREAM(TCP套接字)，SOCK_RAW(原始套接字)

protocol协议类别：(0、IPPROTO_TCP、IPPROTO_UDP)

返回值：

>0 通信的文件描述符（套接字）

<0 创建失败

案例1：

2、IPv4地址结构

存放IPv4协议通信的所有地址信息

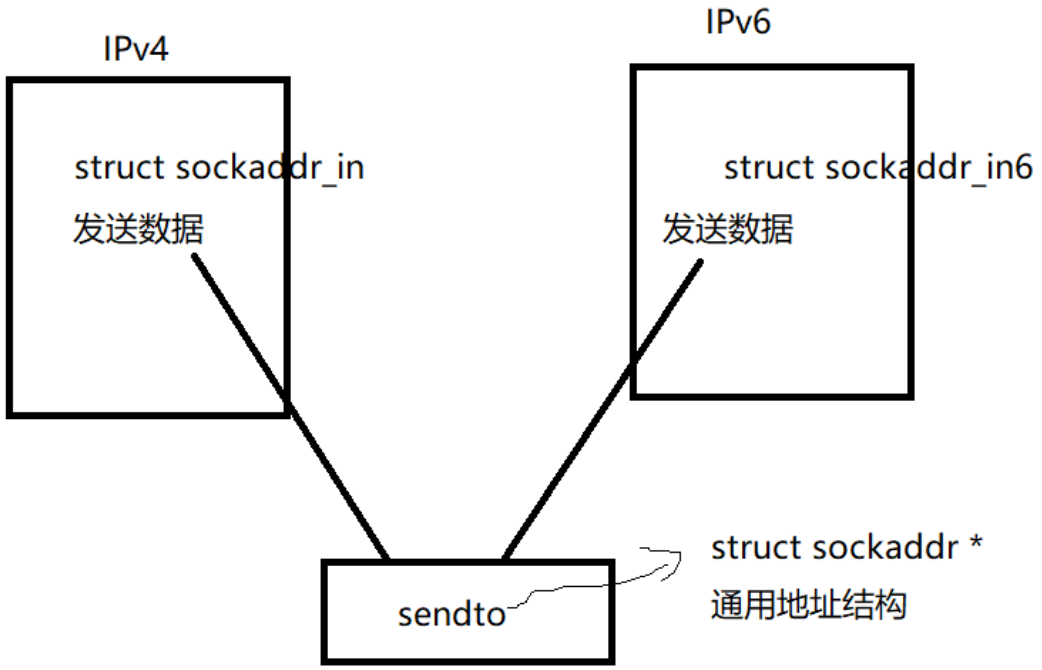
#inclue<netinet/in.h>

```
struct sockaddr_in {
 sa_family_t sin_family;//2 字节 协议AF_INET AF_INET6
 in_port_t sin_port;//2 字节 端口
 struct in_addr sin_addr;//4 字节 IP地址（32位无符号整数）
 char sin_zero[8]//8 字节 全写0
};
```

struct in_addr：

```
struct in_addr {
 in_addr_t s_addr;//4 字节
};
```

3、通用地址结构（类型转换）




```
struct sockaddr {
 sa_family_t sa_family; // 2 字节
 char sa_data[14] //14 字节
```

```
}; 注
```

4、两种地址结构使用场合

```
struct sockaddr_in IPv4地址结构（存放客户端、服务器的地址信息（协议，port,IP））
struct sockaddr 通用地址结构 不是存放数据 socket API 类型转换
```

5、sendto发送数据

```
#include <sys/socket.h>
ssize_t sendto(int socket, const void *message, size_t length,
 int flags, const struct sockaddr *dest_addr,
 socklen_t dest_len);
```

socket：通信套接字

message：需要发送的消息的首元素地址

length：消息的实际长度

flags：0网络默认方式通信

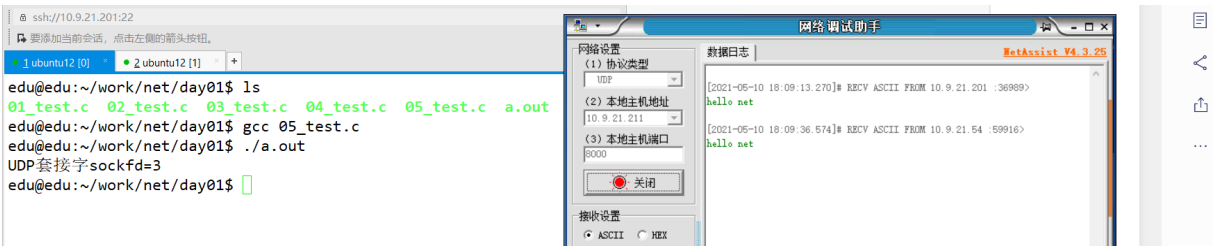
dest_addr：目的主机的地址信息（协议、port、IP地址）

dest_len：地址结构体的长度

返回值：发送的字节数

```
#include <stdio.h>
#include <sys/socket.h>//socket
#include <netinet/in.h>//struct sockaddr_in
#include <string.h>//memset
#include <arpa/inet.h>//htons
int main(int argc, char const *argv[])
{
    //创建通信的UDP的套接字(没有port、ip)
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
     printf("UDP套接字sockfd=%d\n", sockfd);

     //udp客户端 发送消息 给服务器
     //定义一个IPv4地址结构 存放服务器的地址信息（目的主机）
     struct sockaddr_in ser_addr;
     memset(&ser_addr, 0, sizeof(ser_addr));
     ser_addr.sin_family = AF_INET;//IPv4
     ser_addr.sin_port = htons(8000);//服务器的端口
     //服务器的IP地址
     inet_pton(AF_INET, "10.9.21.211", &ser_addr.sin_addr.s_addr);

```

```
    //发送数据
    sendto(sockfd, "hello net", strlen("hello net"), 0,\
    (struct sockaddr *)&ser_addr, sizeof(ser_addr));

     //关闭套接字
    close(sockfd);
    return 0;
}
```



从上面的结果分析：

服务器收到客户端的信息 而且客户端的port是随机的。

如果udp套接字 不适用bind函数 绑定固定端口，那么在第一次调用 sendto系统会自动给套接字分配一个随机端口。后续sendto调用继续使用前一次的端口。

6、bind给udp套接字绑定固定的port、ip信息

```
#include <sys/socket.h>
int bind(int socket, const struct sockaddr *address,
 socklen_t address_len);
```

返回值：

成功为0 失败为-1

bind只能绑定 本地主机的IP

```
//定义IPv4地址结构 存放本机信息
struct sockaddr_in my_addr;
bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(9000);
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
//给udp套接字 bind绑定一个固定的地址信息
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
```

7、recvfrom接收udp的消息

如果udp不发数据前 就要接收消息 必须对udp套接字进行绑定

```
#include <sys/socket.h>
ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
```

```
  int flags, struct sockaddr *restrict address,
  socklen_t *restrict address_len)
```

功能：接收UDP消息（默认没消息 阻塞）

参数：

socket：udp套接字

buffer：用来存放消息的空间起始地址

length：能接受消息的最大字节数

flags：0

address：存放发送者的IPv4地址信息 （不关心发送者信息 NULL）

address_len：地址结构长度（不关心发送者信息 NULL）

返回值

成功：返回收到的实际字节数

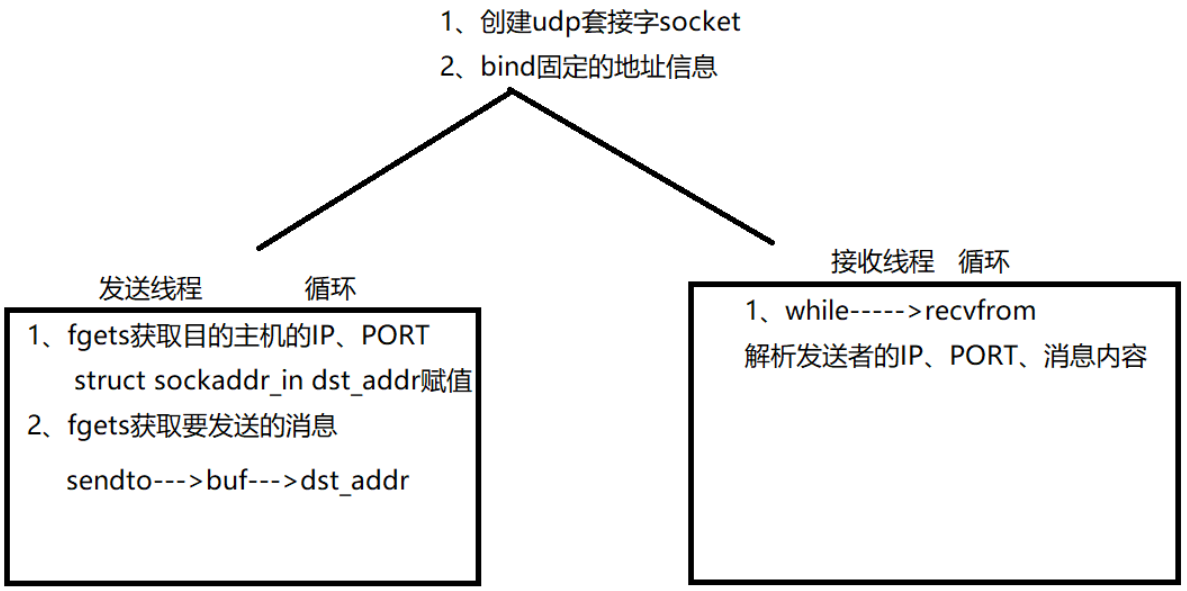
失败：-1

```c
#include <stdio.h>
#include <sys/socket.h>//socket
#include <netinet/in.h>//struct sockaddr_in
#include <string.h>//memset
#include <arpa/inet.h>//htons
#include <unistd.h>//close
int main(int argc, char const *argv[])
{
    //创建通信的UDP的套接字(没有port、ip)
     int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
     printf("UDP套接字sockfd=%d\n", sockfd);

     //定义IPv4地址结构 存放本机信息
     struct sockaddr_in my_addr;
     bzero(&my_addr, sizeof(my_addr));
     my_addr.sin_family = AF_INET;
     my_addr.sin_port = htons(9000);
     my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
     //给udp套接字 bind绑定一个固定的地址信息
     bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));

     //接收udp消息
     while(1)
     {
```

```c
        //定义一个IPv4地址结构 存放发送者的信息
        struct sockaddr_in from_addr;
        socklen_t from_len = sizeof(from_addr);

        unsigned char buf[1500]="";
        int len = recvfrom(sockfd,  buf, sizeof(buf), 0, \
        (struct sockaddr *)&from_addr , &from_len);
        //from_addr存放的就是发送者的信息
        char ip[16]="";
        inet_ntop(AF_INET, &from_addr.sin_addr.s_addr, ip, 16);


        printf("消息来自%s %hu--->", ip, ntohs(from_addr.sin_port));
        printf("len:%d msg:%s\n", len, buf);
    }

     //关闭套接字
    close(sockfd);
    return 0;
}
```

作业：UDP_QQ聊天程序

功能：同时首发数据




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

```c
#include <netinet/in.h>
#include <string.h>
#include <pthread.h>
#include <arpa/inet.h>

void *send_function(void *arg)
{
    //获取套接字
    int sockfd = *(int *)arg;

    //定义目的地址结构
    struct sockaddr_in dst_addr;
    bzero(&dst_addr,sizeof(dst_addr));
    dst_addr.sin_family = AF_INET;

    while (1)
    {
        //获取键盘输入
        char buf[128]="";
        fgets(buf,sizeof( buf), stdin);
        buf[strlen(buf)-1]=0;

        //判断是否是IP port
        //sayto IP  port
        if(strncmp(buf, "sayto", 5) == 0)
        {
            char ip[16]="";
            unsigned short port = 0;
            //sayto 10.9.21.211 8000
            sscanf(buf,"sayto %s %hu", ip, &port);
            dst_addr.sin_port = htons(port);
            inet_pton(AF_INET, ip, &dst_addr.sin_addr.s_addr);
            continue;
        }
        else
        {
            sendto(sockfd, buf, strlen(buf), 0,\
            (struct sockaddr *)&dst_addr, sizeof(dst_addr));

            if(strcmp(buf,"bye") == 0)
```

```
                break;
        }

    }

    return NULL;
}
void *recv_function(void *arg)
{
    int sockfd = *(int *)arg;

    while(1)
    {
        struct sockaddr_in from_addr;
        socklen_t from_len = sizeof(from_addr);
        unsigned char buf[1500]="";
        char ip[16]="";

        int len = recvfrom(sockfd, buf, sizeof(buf), 0,\
        (struct sockaddr *)&from_addr, &from_len);

        printf("%s %hu:%s\n", inet_ntop(AF_INET,&from_addr.sin_addr.s_ad
dr,ip, 16 ) ,\
        ntohs(from_addr.sin_port),buf);
        if(strcmp(buf, "bye")==0)
            break;
    }
    return NULL;
}
int main(int argc, char const *argv[])
{
    //判断参数 ./a.out  8000
    if(argc != 2)
    {
        printf("./a.out 8000\n");
        return 0;
    }

    //创建udp套接字
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

```

```
84      //bind绑定固定的端口、IP
85      struct sockaddr_in  my_addr;
86      bzero(&my_addr, sizeof(my_addr));
87      my_addr.sin_family = AF_INET;
88      my_addr.sin_port = htons(atoi(argv[1]));
89      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
90      bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
91
92      //创建发送线程
93      pthread_t send_tid;
94      pthread_create(&send_tid, NULL, send_function, &sockfd);
95
96      //创建接收线程
97      pthread_t recv_tid;
98      pthread_create(&recv_tid, NULL, recv_function, &sockfd);
99
100      pthread_join(send_tid, NULL);
101      pthread_join(recv_tid, NULL);
102
103      //关闭套接字
104      close(sockfd);
105      return 0;
106  }
```