知识点1【TCP的概述】（了解）

1、TCP编程流程

知识点2【TCP的客户端】（重要）

1、创建TCP套接字

2、connect连接服务器

3、send发送请求

4、recv接收应答 默认带阻塞

5、close

6、TCP客戶端实例

知识点3【TCP的服务器】（重要）

1、socket创建tcp套接字 （监听套接字）

2、bind给服务器的绑定固定的port、IP地址信息

3、listen监听 并创建连接队列

4、accept提取客户端的连接（阻塞）

5、send、recv发送或接收客户端的消息

tcp中recv的注意点：

tcp中send的注意点：

6、close关闭所有套接字

7、tcp服务器

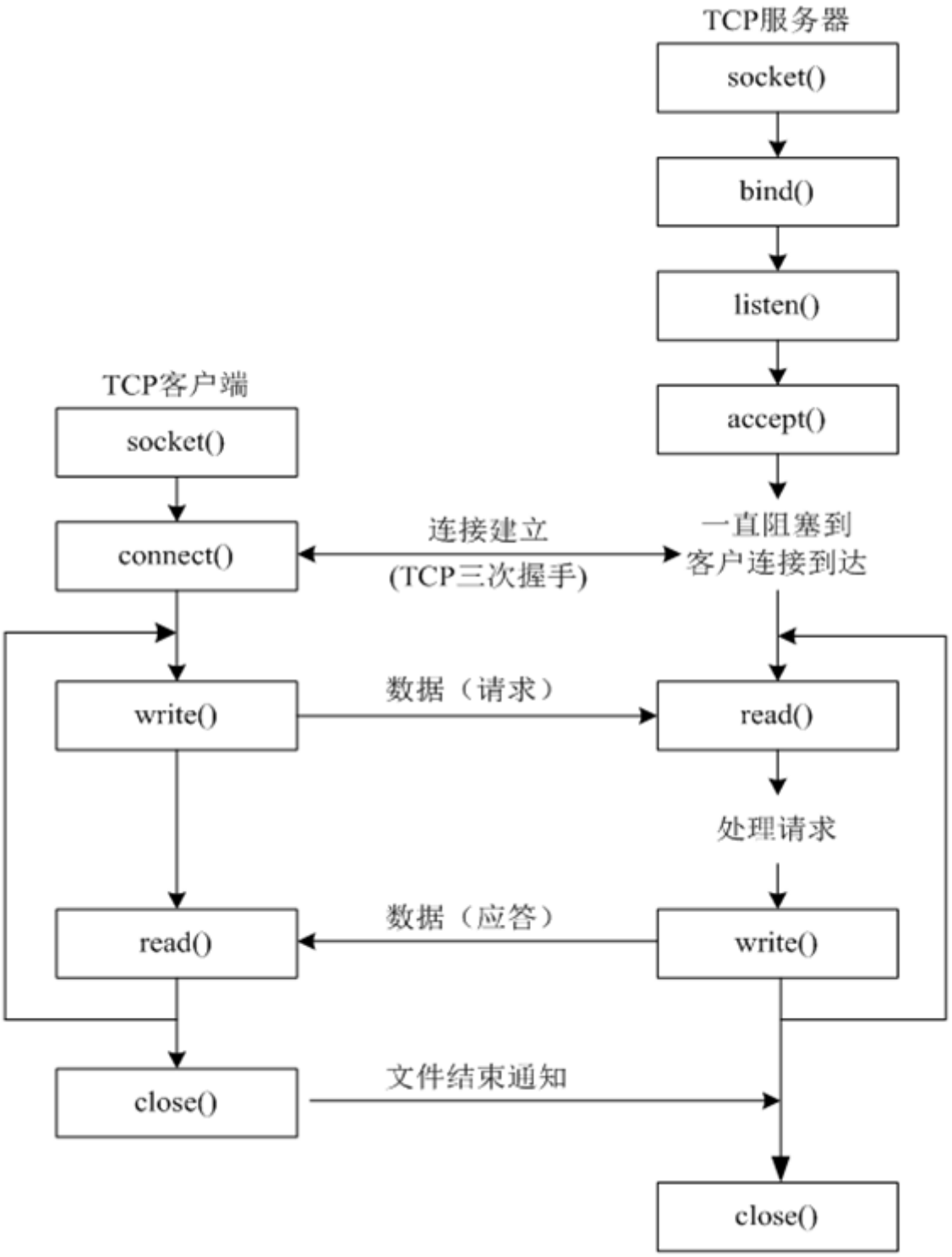
知识点4【三次握手】（重要）

知识点5【四次挥手】（重要）

知识点6【TCP状态转换】（了解）

知识点1【TCP的概述】（了解）

1、TCP编程流程



知识点2【TCP的客户端】（重要）

1、创建TCP套接字

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

socket创建的套接字：没有端口、主动连接别人

2、connect连接服务器

如果sockfd 没有被bind绑定，第一调用connect系统自动分配随机端口，后续使用该端口

```
int connect(int socket, const struct sockaddr *address,
```

```
socklen_t address_len)
```

参数：

socket：发起连接到的套接字

address：服务器的地址

address_len：服务器的地址长度

返回值：成功为0 失败-1

如果客户端和服务器通信， 必须使用connect事先建立连接。

3、send发送请求

```
ssize_t send(int socket, const void *buffer, size_t length, int flags)
```

参数：

socket：客户端套接字

buffer：发送的消息

length：消息长度

flags：0

返回值：

成功：返回发送的字节数 失败：-1

4、recv接收应答 默认带阻塞

```
ssize_t recv(int socket, void *buffer, size_t length, int flags);
```

参数：

socket：客户端套接字

buffer：接收的消息

length：能接收的最大长度

flags：0

返回值：

成功：成功接收的字节数 失败：-1

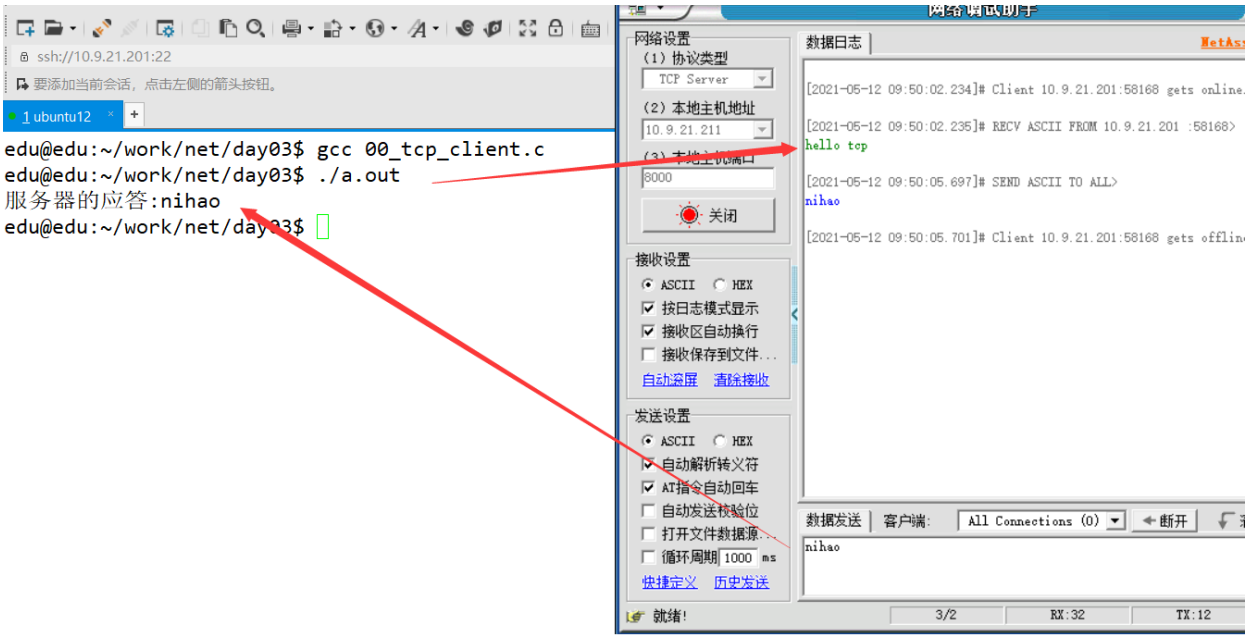
5、close

```
#include <unistd.h>
int close(int fildes);
```

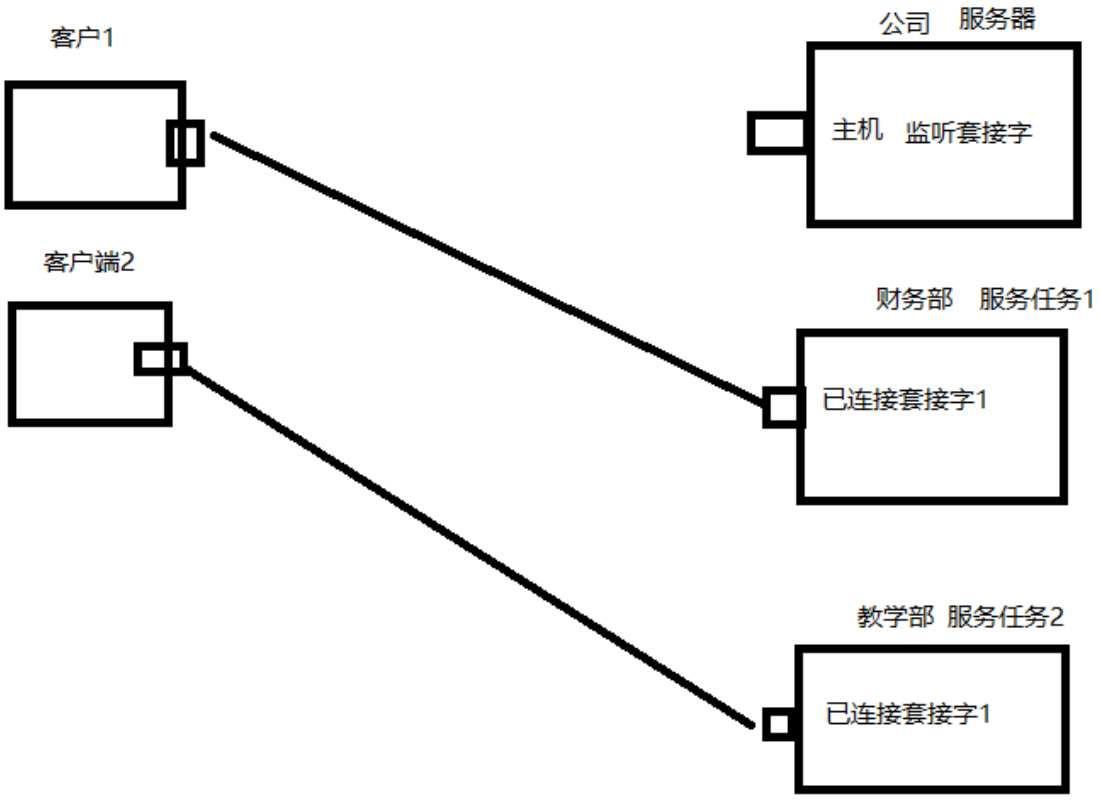
6、TCP客戶端实例

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

```c
#include <string.h>
#include <arpa/inet.h>
int main(int argc, char const *argv[])
{
    //创建tcp套接字 SOCK_STREAM
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);

     //connect连接服务器 （知道服务器地址信息）
     struct sockaddr_in ser_addr;
     bzero(&ser_addr,sizeof(ser_addr));
     ser_addr.sin_family = AF_INET;
     ser_addr.sin_port = htons(8000);
     ser_addr.sin_addr.s_addr = inet_addr("10.9.21.211");
     connect(sockfd, (struct sockaddr *)&ser_addr, sizeof(ser_addr));

     //客户端发送请求
     send(sockfd, "hello tcp", strlen("hello tcp"), 0);

     //客户端接收服务器的应答
     unsigned char buf[1500]="";
     int len = recv(sockfd, buf, sizeof(buf), 0);
     printf("服务器的应答:%s\n", buf);

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



知识点3【TCP的服务器】（重要）



1、socket创建tcp套接字 （监听套接字）

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

2、bind给服务器的绑定固定的port、IP地址信息

```
//bind绑定固定的port、ip地址信息
```

```
struct sockaddr_in my_addr;
bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(8000);
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
```

3、listen监听 并创建连接队列

listen监听：等待客户端的连接到来，经过三次握手（底层自动），将客户端放入连接队列

```
#include <sys/socket.h>
int listen(int socket, int backlog);
```

功能：

1、将监听套接字 由主动 变 被动 。

2、为该套接字 创建连接队列。

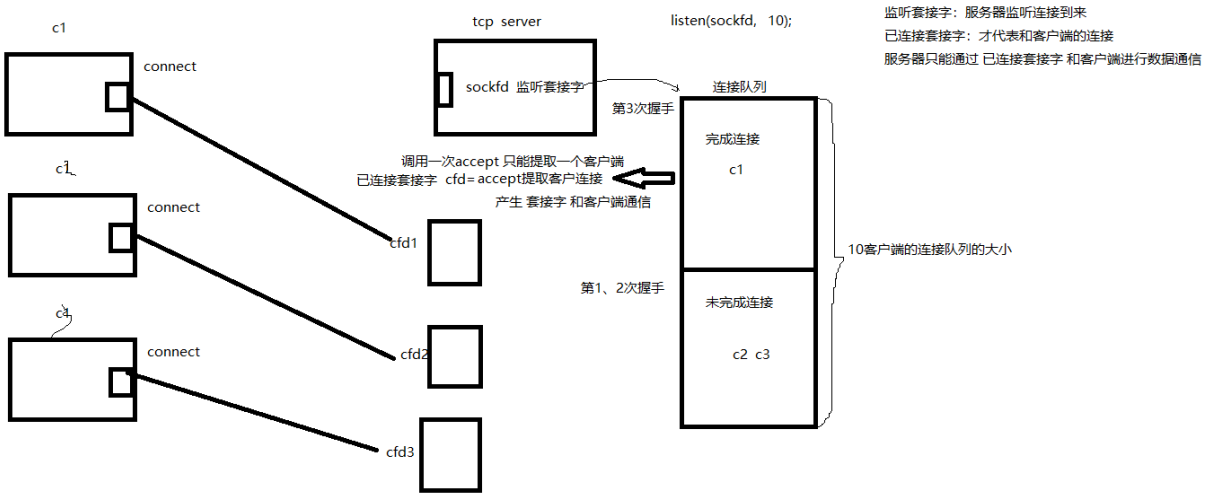
参数：

socket：变被动的套接字

backlog：连接队列的大小

返回值：成功为0， 失败：-1

4、accept提取客户端的连接（阻塞）




```
int accept(int socket, struct sockaddr *restrict address,
 socklen_t *restrict address_len);
```

功能：

从完成连接队里中 提取一个客户端的连接

如果已完成连接队列中没有客户端默认阻塞e

参数：

socket：监听套接字

address：客户端的地址结构

address_len：地址结构长度

返回值：

成功：返回已连接套接字（和客户端通信的套接字）

失败：-1，

5、send、recv发送或接收客户端的消息

tcp中recv的注意点：

```
int len = recv(已连接套接字, buf, sizeof(buf), 0);
如果len>0表示recv收到的实际字节数
如果len为0，表示客户端已经断开连接
如果len为-1，表示客recv读取数据出错

```

tcp中send的注意点：

```
int len = send(cfd, buf, buf_len, 0);
如果len>0 表示发送成功（实际发送的字节数）
如果len为-1表示发送是失败
注意：
 tcp不允许 send发送0长度报文
 udp允许 sendto发送0长度报文
```

6、close关闭所有套接字

```
close(套接字)会导致对方recv收到0长度报文
```

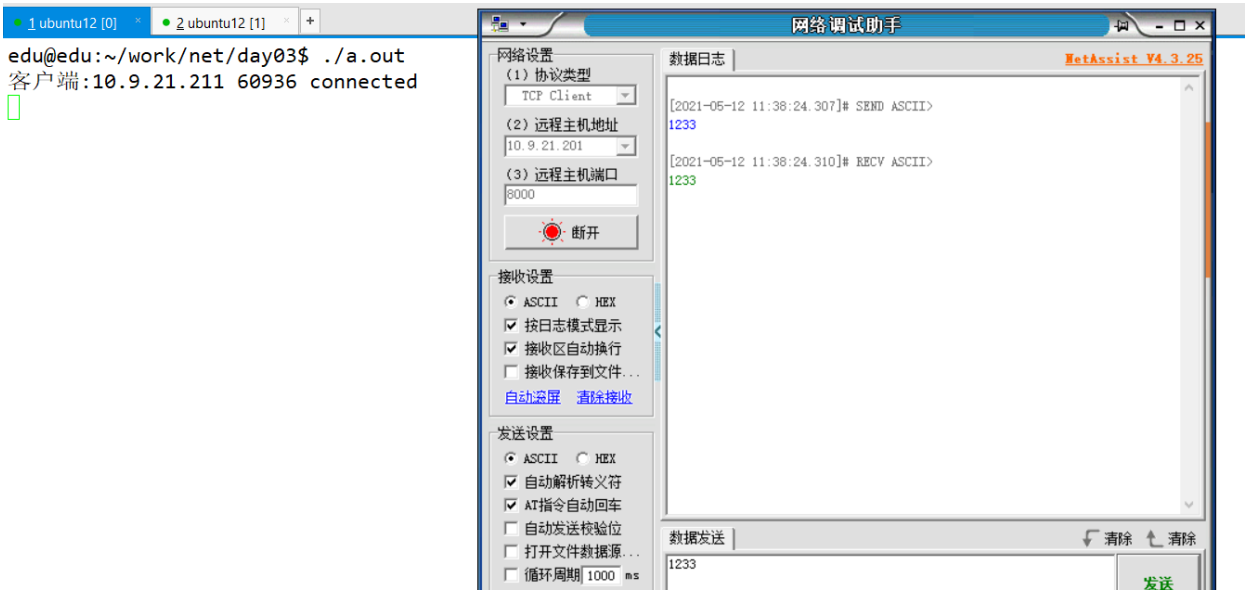
7、tcp服务器

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
int main(int argc, char const *argv[])
{
    //创建tcp 服务器的监听套接字
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if(sockfd < 0)
     {
```

```
12          perror("socket");
13          return 0;
14      }
15
16      //bind绑定固定的port、ip地址信息
17      struct sockaddr_in my_addr;
18      bzero(&my_addr, sizeof(my_addr));
19      my_addr.sin_family = AF_INET;
20      my_addr.sin_port = htons(8000);
21      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
22      int ret = bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr))
23      if(ret == -1)
24      {
25          perror("bind");
26          return 0;
27      }
28
29      //监听套接字 创建连接队列
30      ret = listen(sockfd, 10);
31      if(ret == -1)
32      {
33          perror("listen");
34          return 0;
35      }
36
37      //提取客户端的连接
38      while(1)
39      {
40          //一次只能提取一個客戶端
41          struct sockaddr_in cli_addr;
42          socklen_t cli_len = sizeof(cli_addr);
43          int cfd = accept(sockfd, (struct sockaddr *)&cli_addr, &cli_len)
44          if(cfd < 0)//提取失败
45          {
46              perror("accept\n");
47              break;
48          }
49          else
50          {
51              char ip[16]="";
```

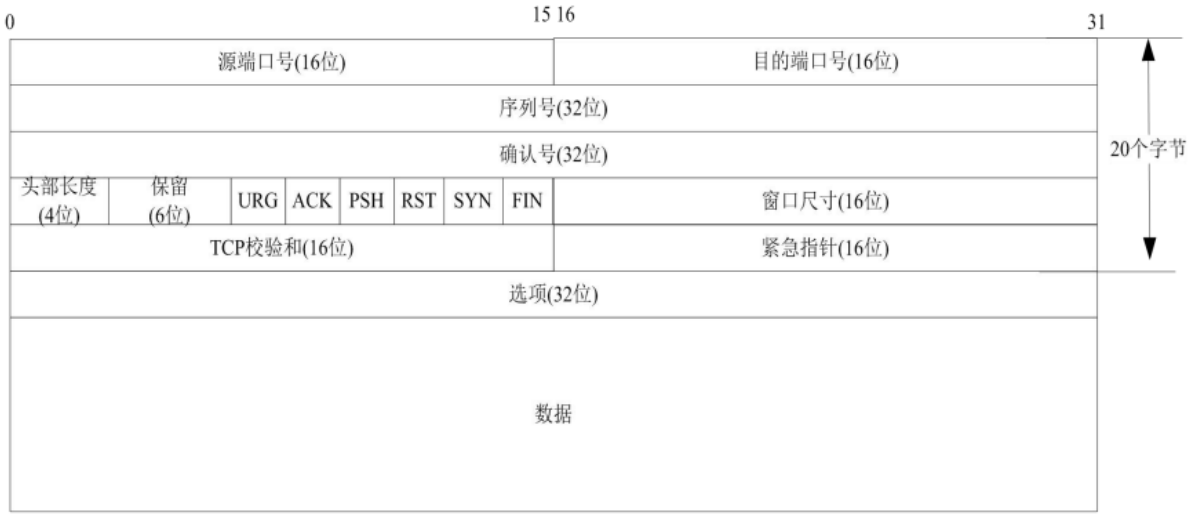
```
52              unsigned short port=0;
53              inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip, 16);
54              port = ntohs(cli_addr.sin_port);
55              //打印客户端的信息
56              printf("客户端:%s %d connected\n", ip, port);
57
58              while(1)
59              {
60                  //获取客户端的请求
61                  unsigned char buf[1500]="";
62                  int len = recv(cfd, buf, sizeof(buf), 0);
63                  if(len == 0)//客户端已经关闭
64                  {
65                      //关闭与客户端连接的套接字
66                      close(cfd);
67                      break;
68                  }
69
70                  //应答客户端
71                  send(cfd, buf, len, 0);
72              }
73
74          }
75      }
76
77
78      //关闭监听套接字
79      close(sockfd);
80      return 0;
81  }
```

运行结果：



知识点4【三次握手】（重要）

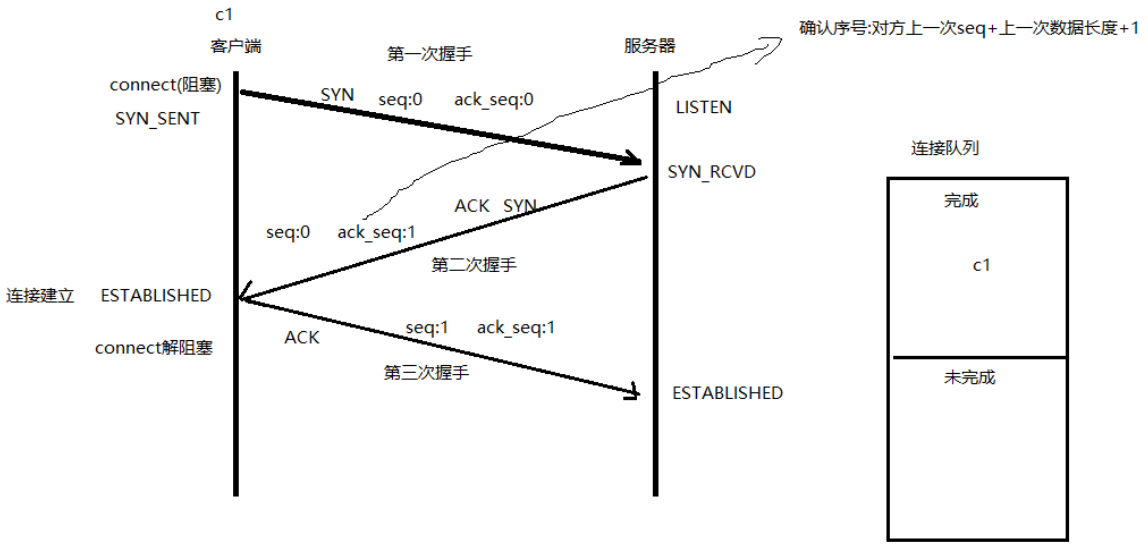
当客户端调用connect连接服务器时，底层会发生“三次握手”，握手成功，连接建立，connnect解阻塞继续执行。



SYN：建立连接、FIN断开连接、ACK回应、RST复位、URG紧急指针、PSH推送

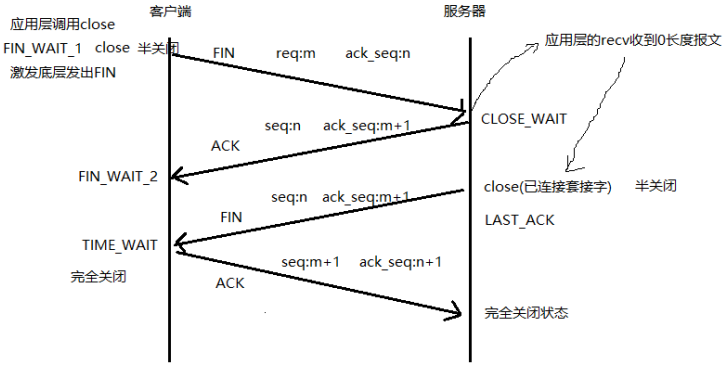
序号：当前报文的标号 seq

确认序号：指希望对方下次发送数据的序号。ack_seq



客户端发出SYN请求，处于SYN_SENT状态完成第一次握手
服务器收到客户端的SYN请求，处于SYN_RCVD状态，并发出SYN以及ACK请求，完成第二次握手
客户端收到服务器的SYN以及ACK处于ESTABLISHED(连接状态)，并发出ACK请求 完成第三次握手

知识点5【四次挥手】（重要）



第一个问题：为啥是四次挥手

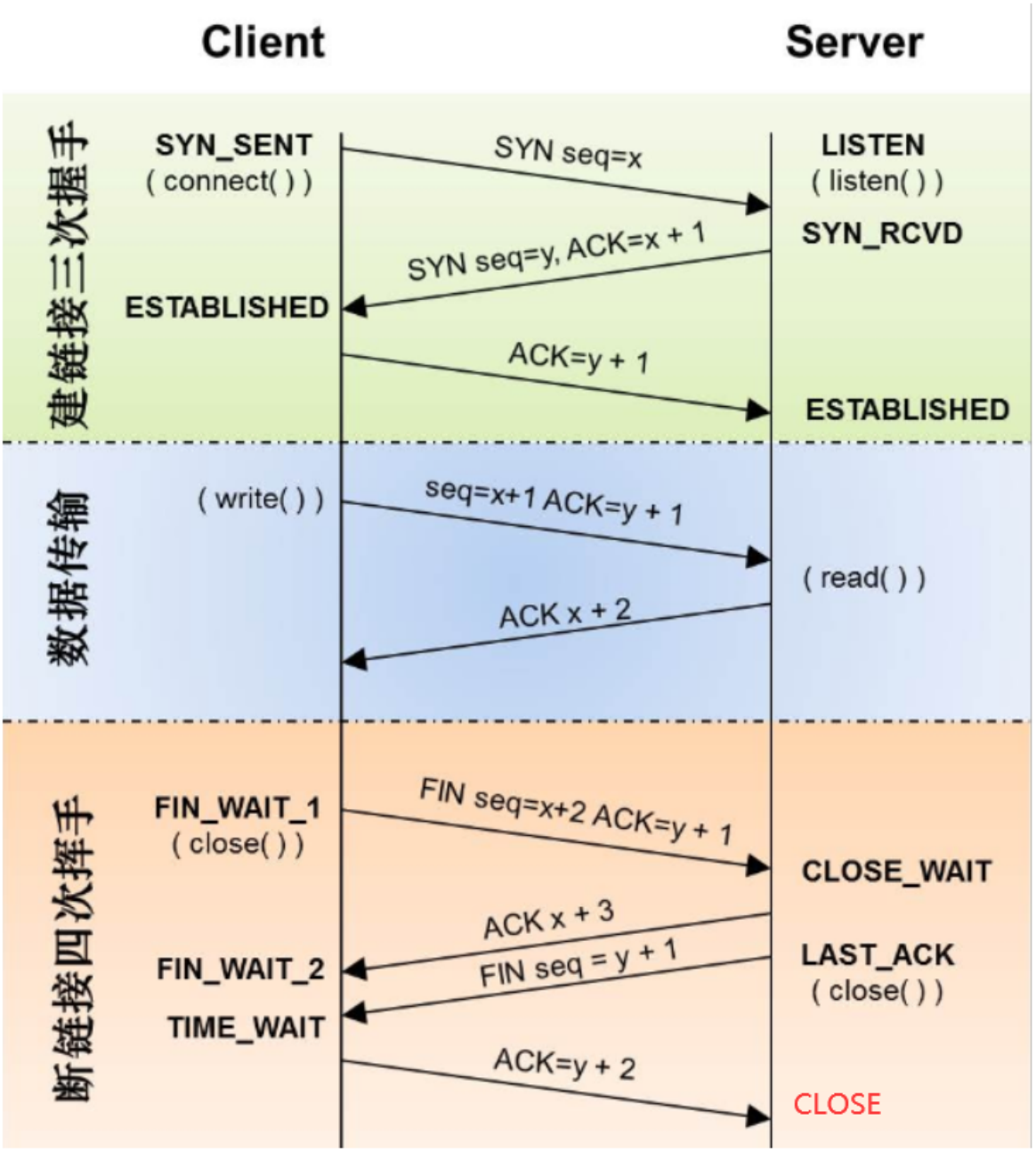
第二次、第三次报文存在时间差异，不能组成一个报文发送

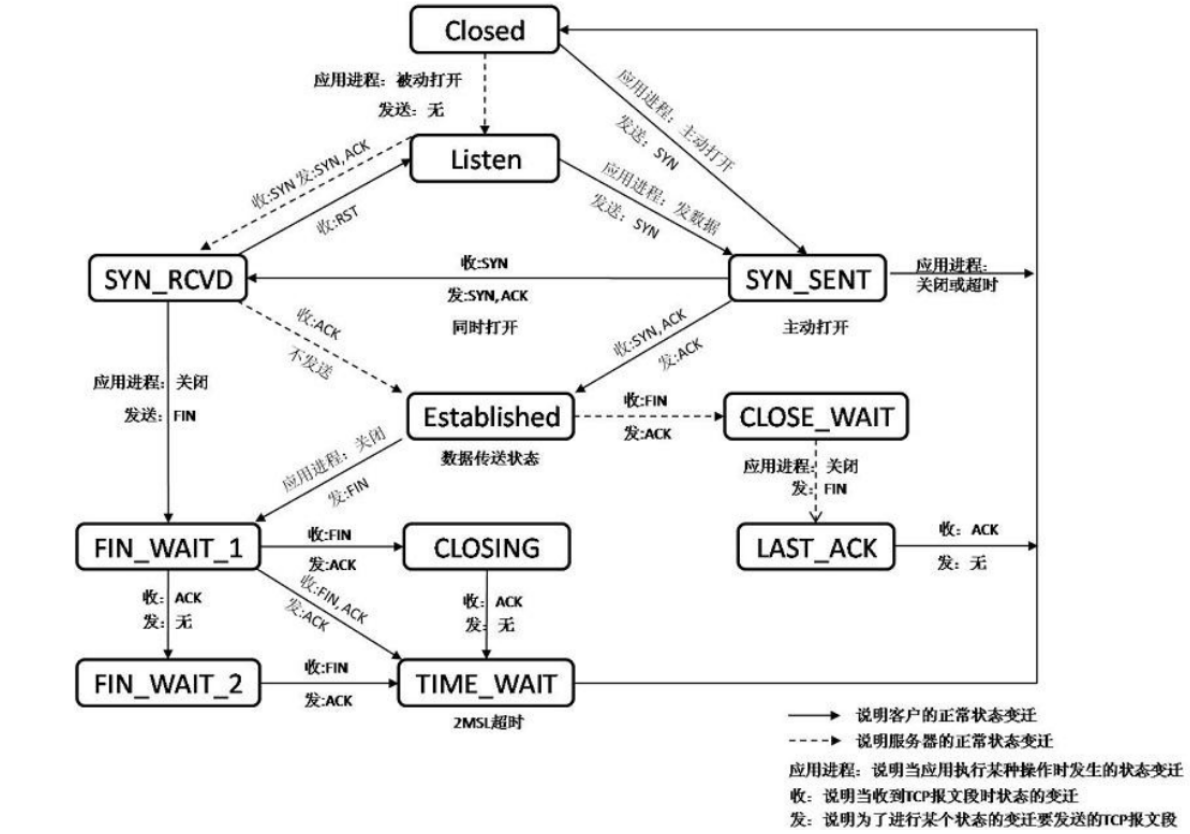
第二个问题：客户端调用close,为啥能收到服务器的FIN以及ACK?

客户端调用close只是处于半关闭状态（应用层不能收发，但底层可以）

当客户端调用close,激发底层发出FIN请求，完成第一次挥手
服务器收到客户端的FIN，立马会议ACK报文完成第二次挥手
服务器应用层调用close，激发底层发出FIN请求，完成第三次挥手
客户端收到服务器的FIN请求，发出ACK应答完成第四次挥手

知识点6【TCP状态转换】（了解）

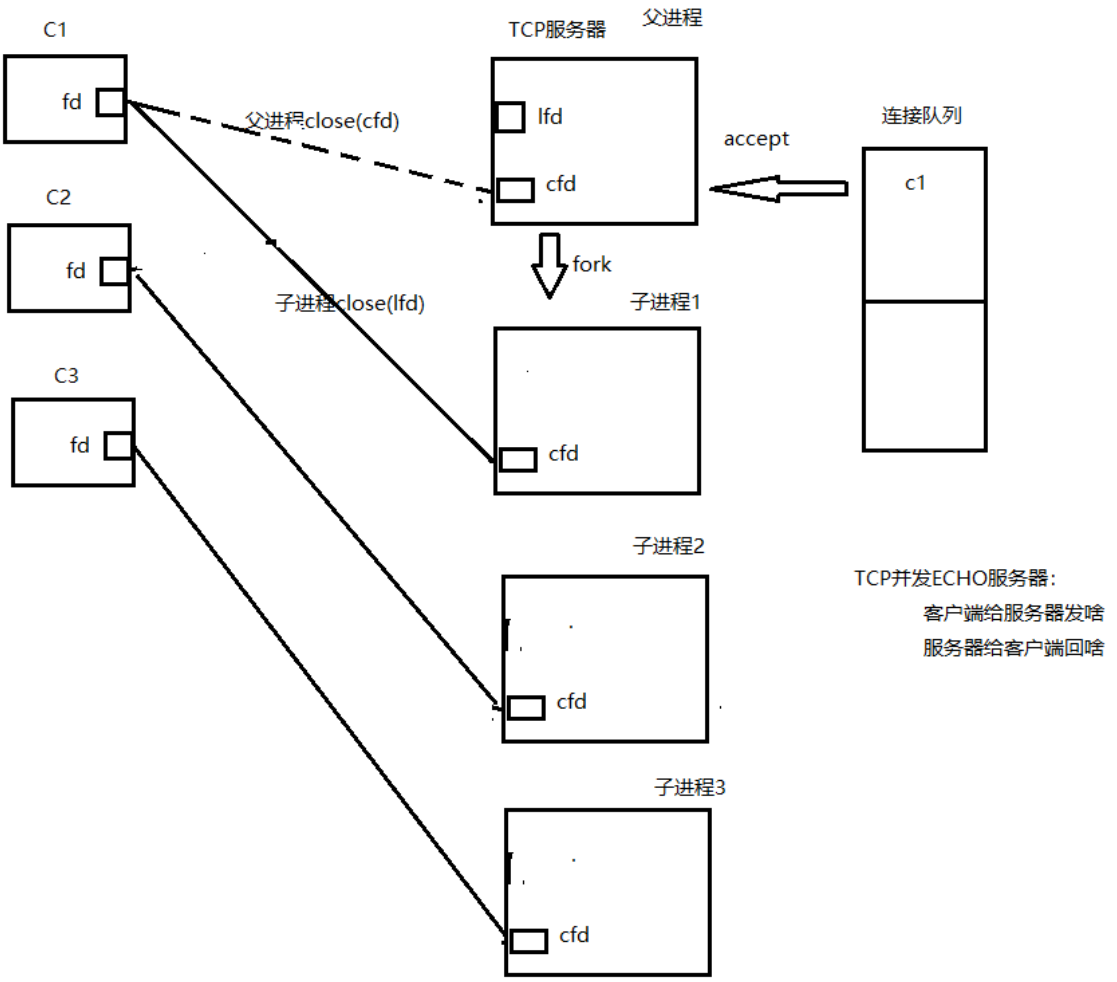

Client
Server
建链接三次握手
SYN_SENT
(connect( ))
SYN seq=x
LISTEN
(listen( ))
SYN_RCVD
SYN seq=y, ACK=x + 1
ESTABLISHED
ACK=y + 1
ESTABLISHED
数据传输
(write( ))
seq=x+1 ACK=y + 1
(read( ))
ACK x + 2
断链接四次挥手
FIN_WAIT_1
(close( ))
FIN seq=x+2 ACK=y + 1
CLOSE_WAIT
ACK x + 3
FIN_WAIT_2
FIN seq = y + 1
LAST_ACK
(close( ))
TIME_WAIT
ACK=y + 2
CLOSE



知识点7【TCP并发服务器--多进程】（重点）

必须TCP服务器、并发（同时服务器多个客户端）

1、TCP并发ECHO服务器（进程版）



流程:

```
//1、创建tcp监听套接字
//2、bind给服务器的监听套接字 绑定固定的IP、port
//3、listen将服务器的套接字主动变被动  创建连接队列 进行监听
//4、while-->accept提取客户端
//5、一个客户端 创建一个进程（父进程close(cfd),  子进程close(lfd)）
//6、子进程 服务器客户端
```

```
#include <stdio.h>
#include <sys/socket.h>//socket
#include <netinet/in.h>//struct sockaddr_in
#include <arpa/inet.h>//inet_pton  inet_addr
#include <string.h>//bzero
#include <stdlib.h>//_exit
#include <unistd.h>//fork
int main(int argc, char const *argv[])
{
    if(argc != 2)
    {
```

```
12          printf("./a.out 8000\n");
13          _exit(-1);
14      }
15
16      //1、创建tcp监听套接字
17      int lfd = socket(AF_INET, SOCK_STREAM, 0);
18      if(lfd < 0)
19      {
20          perror("socket");
21          _exit(-1);
22      }
23
24      //设置端口复用
25      int opt = 1;
26      setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
27
28      //2、bind给服务器的监听套接字 绑定固定的IP、port
29      struct sockaddr_in my_addr;
30      bzero(&my_addr, sizeof(my_addr));
31      my_addr.sin_family = AF_INET;
32      my_addr.sin_port = htons(atoi(argv[1]));
33      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
34      int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
35      if(ret < 0)
36      {
37          perror("bind");
38          _exit(-1);
39      }
40
41      //3、listen将服务器的套接字主动变被动  创建连接队列 进行监听
42      ret = listen(lfd, 128);
43      if(ret < 0)
44      {
45          perror("listen");
46          _exit(-1);
47      }
48
49      //4、while-->accept提取客户端
50      while(1)
51      {
```

```
52          struct sockaddr_in cli_addr;
53          socklen_t cli_len = sizeof(cli_addr);
54
55          int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli_len);
56          char ip[16]="";
57          unsigned short port = 0;
58          inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip, 16);
59          port = ntohs(cli_addr.sin_port);
60          printf("%s %hu connected\n", ip, port);
61
62          //5、一个客户端 创建一个进程（父进程close(cfd),  子进程close(lfd)）
63          pid_t pid = fork();
64          if(pid == 0)//子进程
65          {
66              //子进程close(lfd)
67              close(lfd);
68
69              //6、子进程 服务客户端
70              while(1)
71              {
72                  //获取客户端的请求
73                  unsigned char buf[1500]="";
74                  int len = recv(cfd, buf, sizeof(buf), 0);
75                  if(len <= 0)
76                  {
77                      printf("%s %hu 退出了\n", ip, port);
78                      break;
79                  }
80                  else
81                  {
82                      printf("%s\n", buf);
83                      //将数据 发给客户端
84                      send(cfd, buf, len, 0);
85                  }
86              }
87
88              //关闭cfd
89              close(cfd);
90              _exit(-1);//进程退出
91
```

```
        }
        else if(pid > 0)//父进程
        {
            //父进程close(cfd)
            close(cfd);
        }
    }

    //关闭监听套接字
    close(lfd);
    return 0;
}
```

回收子进程版本

```
#include <stdio.h>
#include <sys/socket.h>//socket
#include <netinet/in.h>//struct sockaddr_in
#include <arpa/inet.h>//inet_pton  inet_addr
#include <string.h>//bzero
#include <stdlib.h>//_exit
#include <unistd.h>//fork
#include <signal.h>
#include <errno.h>
void free_child(int sig)
{
    //回收子进程的资源
    while(1)
    {
        pid_t pid = waitpid(-1, NULL, WNOHANG);
        if(pid <= 0)//没有子进程退出
        {
            break;
        }
        else if(pid > 0)
        {
            printf("子进程%d退出了\n", pid);
        }
    }

    return;
```

```
}
int main(int argc, char const *argv[])
{
    if(argc != 2)
    {
        printf("./a.out 8000\n");
        _exit(-1);
    }

    //1、创建tcp监听套接字
    int lfd = socket(AF_INET, SOCK_STREAM, 0);
    if(lfd < 0)
    {
        perror("socket");
        _exit(-1);
    }

    //设置端口复用
    int opt = 1;
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    //2、bind给服务器的监听套接字 绑定固定的IP、port
    struct sockaddr_in my_addr;
    bzero(&my_addr, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(atoi(argv[1]));
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if(ret < 0)
    {
        perror("bind");
        _exit(-1);
    }

    //3、listen将服务器的套接字主动变被动  创建连接队列 进行监听
    ret = listen(lfd, 128);
    if(ret < 0)
    {
        perror("listen");
        _exit(-1);
```

```
67      }
68
69      //将SIGCHLD放入阻塞集
70      sigset_t set;
71
72      //清空集合
73      sigemptyset(&set);
74      //将SIGCHLD放入集合中
75      sigaddset(&set, SIGCHLD);
76      sigprocmask( SIG_BLOCK,  &set, NULL);
77
78      //4、while-->accept提取客户端
79      while(1)
80      {
81          struct sockaddr_in cli_addr;
82          socklen_t cli_len = sizeof(cli_addr);
83
84          int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli_len);
85          if(cfd < 0)
86          {
87              if(errno == ECONNABORTED || errno==EINTR)
88                  continue;
89              break;
90          }
91          char ip[16]="";
92          unsigned short port = 0;
93          inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip, 16);
94          port = ntohs(cli_addr.sin_port);
95          printf("%s %hu connected\n", ip, port);
96
97          //5、一个客户端 创建一个进程（父进程close(cfd),  子进程close(lfd)）
98          pid_t pid = fork();
99          if(pid == 0)//子进程
100          {
101              //子进程close(lfd)
102              close(lfd);
103
104              //6、子进程 服务客户端
105              while(1)
106              {
```

```
107                 //获取客户端的请求
108                 unsigned char buf[1500]="";
109                 int len = recv(cfd, buf, sizeof(buf), 0);
110                 if(len <= 0)
111                 {
112                     printf("%s %hu 退出了\n", ip, port);
113                     break;
114                 }
115                 else
116                 {
117                     printf("%s\n", buf);
118                     //将数据 发给客户端
119                     send(cfd, buf, len, 0);
120                 }
121             }
122 
123             //关闭cfd
124             close(cfd);
125             _exit(-1);//进程退出 发出SIGCHLD
126 
127         }
128         else if(pid > 0)//父进程
129         {
130             //父进程close(cfd)
131             close(cfd);
132 
133             //注册SIGCHLD处理函数
134             signal(SIGCHLD, free_child);
135             //将SIGCHLD从阻塞集中解除
136             sigprocmask( SIG_UNBLOCK,  &set, NULL);
137         }
138     }
139 
140     //关闭监听套接字
141     close(lfd);
142     return 0;
143 }
144 
```

知识点8【端口复用】（了解）

1、端口复用概述

默认的情况下，如果一个网络应用程序的一个套接字 绑定了一个端口(占用了 8000)，这时候，别的套接字就无法使用这个端口(8000)

端口复用：允许在一个应用程序可以把 n 个套接字绑在一个端口上而不出错

SO_REUSEADDR可以用在以下四种情况下。 （摘自《Unix网络编程》卷一，即UNPv1）

1、当有一个有相同本地地址和端口的socket1处于TIME_WAIT状态时，而你启动的程序的socket2要占用该地址和端口，你的程序就要用到该选项。

2、SO_REUSEADDR允许同一port上启动同一服务器的多个实例(多个进程)。但每个实例绑定的IP地址是不能相同的。在有多块网卡或用IP Alias技术的机器可以测试这种情况。

3、SO_REUSEADDR允许单个进程绑定相同的端口到多个socket上，但每个socket绑定的ip地址不同。这和2很相似，区别请看UNPv1。

4、SO_REUSEADDR允许完全相同的地址和端口的重复绑定。但这只用于UDP的多播，不用于TCP

注意：置端口复用函数要在绑定之前调用，而且只要绑定到同一个端口的所有套接字都得设置复用

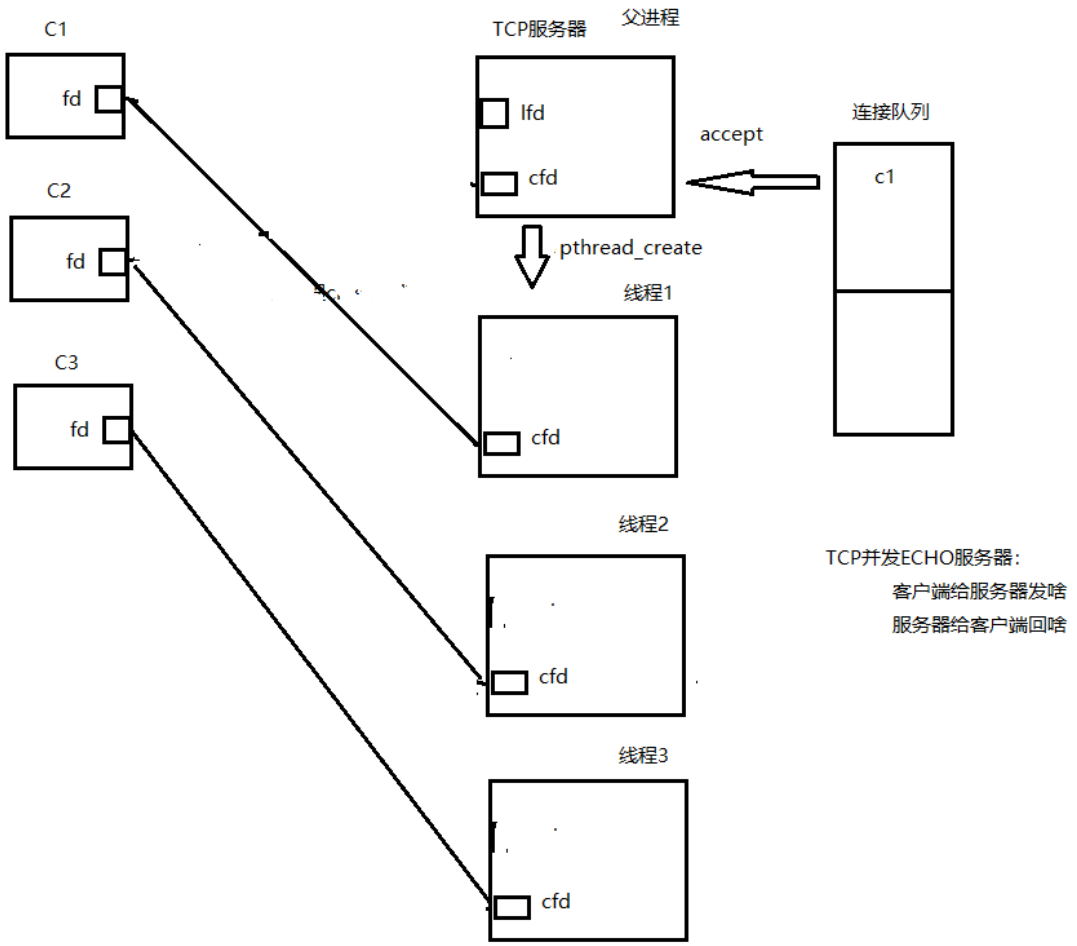
2、设置套接字端口复用

```
int opt = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
```

上面的sockfd为需要使用同一端口复用的的套接字

知识点8【TCP并发服务器--多线程】（重点）

1、TCP并发ECHO服务器（线程版）



2、多线程并发服务器的步骤

```c
#include <stdio.h>
#include <sys/socket.h>//socket
#include <netinet/in.h>//struct sockaddr_in
#include <arpa/inet.h>//inet_pton  inet_addr
#include <string.h>//bzero
#include <stdlib.h>//_exit
#include <pthread.h>//线程相关函数
#include <unistd.h>
typedef struct {
    int cfd;//存放已连接套接字
    struct sockaddr_in addr;//存放客户端的信息
}CLIENT_MSG;
void *deal_client_fun(void *arg)
{
    CLIENT_MSG *p = (CLIENT_MSG *)arg;

    //打印客户端的信息
    char ip[16]="";
    unsigned short port = 0;
```

```c
    inet_ntop(AF_INET, &p->addr.sin_addr.s_addr, ip, 16);
    port = ntohs(p->addr.sin_port);
    printf("%s %hu connected\n", ip, port);

    //while获取客户端的请求 并回应
    while(1)
    {
        unsigned char buf[1500]="";
        int len = recv(p->cfd, buf, sizeof(buf), 0);
        if(len <= 0)
        {
            printf("%s %hu 退出了\n", ip, port);
            close(p->cfd);
            break;
        }
        else
        {
            printf("%s %d:%s\n", ip, port, buf);
            send(p->cfd, buf, len, 0);
        }
    }

    //释放堆区空间
    if(p != NULL)
    {
        free(p);
        p=NULL;
    }

    //线程结束
    pthread_exit(NULL);

    return NULL;
}
int main(int argc, char const *argv[])
{
    if(argc != 2)
    {
        printf("./a.out 8000\n");
        _exit(-1);
```

```
60      }
61
62      //1、创建tcp监听套接字
63      int lfd = socket(AF_INET, SOCK_STREAM, 0);
64      if(lfd < 0)
65      {
66          perror("socket");
67          _exit(-1);
68      }
69
70      //设置端口复用
71      int opt = 1;
72      setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
73
74      //2、bind给服务器的监听套接字 绑定固定的IP、port
75      struct sockaddr_in my_addr;
76      bzero(&my_addr, sizeof(my_addr));
77      my_addr.sin_family = AF_INET;
78      my_addr.sin_port = htons(atoi(argv[1]));
79      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
80      int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
81      if(ret < 0)
82      {
83          perror("bind");
84          _exit(-1);
85      }
86
87      //3、listen将服务器的套接字主动变被动  创建连接队列 进行监听
88      ret = listen(lfd, 128);
89      if(ret < 0)
90      {
91          perror("listen");
92          _exit(-1);
93      }
94
95      //4、while-->accept提取客户端
96      while(1)
97      {
98          struct sockaddr_in cli_addr;
99          socklen_t cli_len = sizeof(cli_addr);
```

```
        int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli_len);

        CLIENT_MSG *p = (CLIENT_MSG *)calloc(1,sizeof(CLIENT_MSG));
        p->cfd = cfd;
        p->addr = cli_addr;

        //5、一个客户端 创建一个线程
        pthread_t tid;
        pthread_create(&tid, NULL, deal_client_fun,  (void *)p);
        //线程分离
        pthread_detach(tid);
    }

    //关闭监听套接字
    close(lfd);
    return 0;
}
```

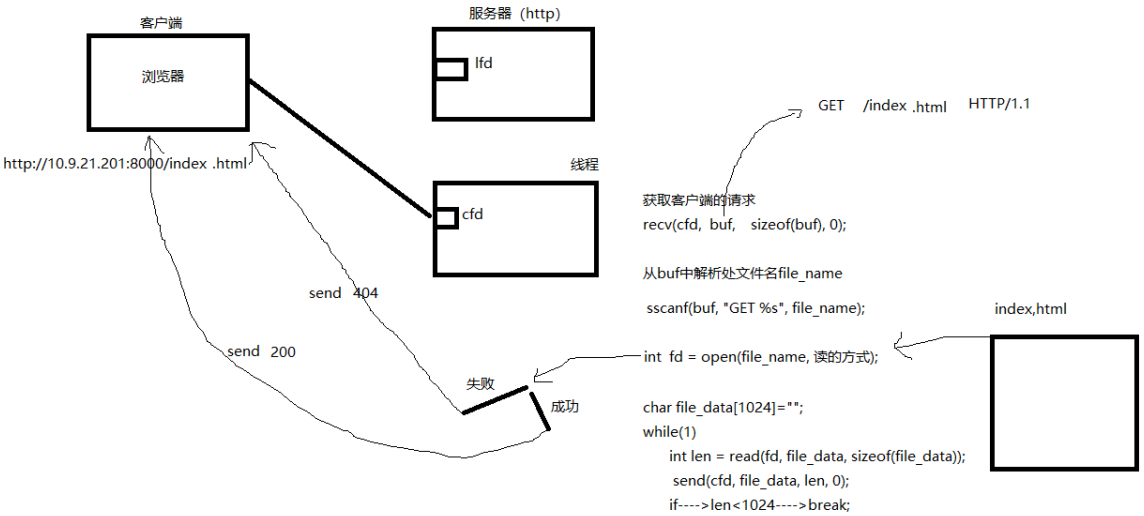
知识点9【基于tcp的http服务器】（了解）

1、http协议的概述

基于tcp协议的超文本传送协议。它是浏览器与服务器之间的通信协议。

一个连接 只能处理一个请求。

2、http服务器的流程



3、webserver实现

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

```c
#include <netinet/in.h>//struct sockaddr_in
#include <arpa/inet.h>//inet_pton  inet_addr
#include <string.h>//bzero
#include <stdlib.h>//_exit
#include <pthread.h>//线程相关函数
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
char head[] = "HTTP/1.1 200 OK\r\n"                    \
                  "Content-Type: text/html\r\n"             \
                  "\r\n";
char err[]= "HTTP/1.1 404 Not Found\r\n"            \
                    "Content-Type: text/html\r\n"           \
                    "\r\n"                                  \
                    "<HTML><BODY>File not found</BODY></HTML>";
typedef struct {
    int cfd;//存放已连接套接字
    struct sockaddr_in addr;//存放客户端的信息
}CLIENT_MSG;
void *deal_client_fun(void *arg)
{
    CLIENT_MSG *p = (CLIENT_MSG *)arg;

    //打印客户端的信息
    char ip[16]="";
    unsigned short port = 0;
    inet_ntop(AF_INET, &p->addr.sin_addr.s_addr, ip, 16);
    port = ntohs(p->addr.sin_port);
    printf("%s %hu connected\n", ip, port);

    //获取浏览器的请求
    unsigned char buf[1500]="";
    recv(p->cfd, buf, sizeof(buf), 0);

    //提取请求中的文件名
    char file_name[512]="./html";
    sscanf(buf, "GET %s", file_name+6);
    printf("##%s##\n", file_name);
```

```
    if(file_name[7] == '\0')//没有提取到文件名
    {
        strcpy(file_name,"./html/index.html");
    }

    //从本地打开file_name文件
    int fd = open(file_name, O_RDONLY);
    if(fd < 0)//本地没有改文件
    {
        //send 404
        send(p->cfd, err, strlen(err), 0);

        //退出线程
        close(p->cfd);
        if(p != NULL)
        {
            free(p);
            p=NULL;
        }
        return NULL;
    }
    //本地文件打开成功
    //send 200
    send(p->cfd, head, strlen(head), 0);

    //循环读取本地文件 发送给浏览器
    while(1)
    {
        unsigned char file_data[1024]="";
        //读取文本文件数据
        int len = read(fd, file_data, sizeof(file_data));
        //将file_data发送给浏览器
        send(p->cfd, file_data, len, 0);
        if(len < 1024)
            break;
    }

    //释放堆区空间
    close(p->cfd);
    if(p != NULL)
```

```
    {
        free(p);
        p=NULL;
    }

    //线程结束
    pthread_exit(NULL);

    return NULL;
}
int main(int argc, char const *argv[])
{
    if(argc != 2)
    {
        printf("./a.out 8000\n");
        _exit(-1);
    }

     //1、创建tcp监听套接字
     int lfd = socket(AF_INET, SOCK_STREAM, 0);
     if(lfd < 0)
     {
         perror("socket");
         _exit(-1);
     }

     //设置端口复用
     int opt = 1;
     setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

     //2、bind给服务器的监听套接字 绑定固定的IP、port
     struct sockaddr_in my_addr;
     bzero(&my_addr, sizeof(my_addr));
     my_addr.sin_family = AF_INET;
     my_addr.sin_port = htons(atoi(argv[1]));
     my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
     int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
     if(ret < 0)
     {
         perror("bind");
```

```
123         _exit(-1);
124     }
125 
126     //3、listen将服务器的套接字主动变被动   创建连接队列 进行监听
127     ret = listen(lfd, 128);
128     if(ret < 0)
129     {
130         perror("listen");
131         _exit(-1);
132     }
133 
134     //4、while-->accept提取客户端
135     while(1)
136     {
137         struct sockaddr_in cli_addr;
138         socklen_t cli_len = sizeof(cli_addr);
139         int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli_len);
140 
141         CLIENT_MSG *p = (CLIENT_MSG *)calloc(1,sizeof(CLIENT_MSG));
142         p->cfd = cfd;
143         p->addr = cli_addr;
144 
145         //5、一个客户端 创建一个线程
146         pthread_t tid;
147         pthread_create(&tid, NULL, deal_client_fun,  (void *)p);
148         //线程分离
149         pthread_detach(tid);
150     }
151 
152     //关闭监听套接字
153     close(lfd);
154     return 0;
155 }
```


Qt 4.3: Qtopia Core Reference D
班级管理 - 查看班级
班级管理 - 座位模式
10.9.21.201:8000/index.html
导入书签...
火狐官方站点
新手上路
常用网址
京东商城
京东商城
火狐主页
Home · All Classes · Main Classes · Grouped Classes · Modules · Functions
服务器的IP
服务器的port
请求的文件
Qtopia Core Reference
Qt for Embedde
Getting Started
General
What's New in Qt 4.3
About Qt