Socket编程TCP

TCP网络程序

和刚才UDP类似. 实现一个简单的英译汉的功能

TCP socket API 详解

下面介绍程序中用到的socket API,这些函数都在sys/socket.h中。

socket():

```
NAME
       socket - create an endpoint for communication

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

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

- socket()打开一个网络通讯端口,如果成功的话,就像open()一样返回一个文件描述符;
- 应用程序可以像读写文件一样用read/write在网络上收发数据;
- 如果socket()调用出错则返回-1;
- 对于IPv4, family参数指定为AF_INET;
- 对于TCP协议,type参数指定为SOCK_STREAM, 表示面向流的传输协议
- protocol参数的介绍从略,指定为0即可。

bind():

```
NAME
       bind - bind a name to a socket

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

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

- 服务器程序所监听的网络地址和端口号通常是固定不变的,客户端程序得知服务器程序的地址和端口号后就可以向服务器发起连接; 服务器需要调用bind绑定一个固定的网络地址和端口号;
- bind()成功返回0,失败返回-1。
- bind()的作用是将参数sockfd和myaddr绑定在一起, 使sockfd这个用于网络通讯的文件描述符监听myaddr所描述的地址和端口号;

> · 前面讲过,struct sockaddr *是一个通用指针类型,myaddr参数实际上可以接受多种协议的sockaddr 结构体,而它们的长度各不相同,所以需要第三个参数addrlen指定结构体的长度;

我们的程序中对myaddr参数是这样初始化的:

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(
INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
```

1. 将整个结构体清零;
2. 设置地址类型为AF_INET;
3. 网络地址为INADDR_ANY, 这个宏表示本地的任意IP地址,因为服务器可能有多个网卡,每个网卡也可能绑定多个IP 地址, 这样设置可以在所有的IP地址上监听,直到与某个客户端建立了连接时才确定下来到底用哪个IP 地址;
4. 端口号为SERV_PORT, 我们定义为9999;

listen():

```
NAME
       listen - listen for connections on a socket

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

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

> · listen()声明sockfd处于监听状态, 并且最多允许有backlog个客户端处于连接等待状态, 如果接收到更多的连接请求就忽略, 这里设置不会太大(一般是5), 具体细节同学们课后深入研究;
>
> · listen()成功返回0,失败返回-1;

accept():

```
NAME
       accept - accept a connection on a socket

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

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

> · 三次握手完成后, 服务器调用accept()接受连接;
>
> · 如果服务器调用accept()时还没有客户端的连接请求,就阻塞等待直到有客户端连接上来;
>
> · addr是一个传出参数,accept()返回时传出客户端的地址和端口号;
>
> · 如果给addr 参数传NULL,表示不关心客户端的地址; · addrlen参数是一个传入传出参数(value-result argument), 传入的是调用者提供的, 缓冲区addr的长度以避免缓冲区溢出问题, 传出的是客户端地址结构体的实际长度(有可能没有占满调用者提供的缓冲区);

我们的服务器程序结构是这样的:

```
while (1)
{
 cliaddr_len = sizeof(cliaddr);

 connfd = accept(listenfd, (struct
 sockaddr *)&cliaddr, &cliaddr_len);

 n = read(connfd, buf, MAXLINE);

 ...close(connfd);}
```

理解accecpt的返回值: 饭店拉客例子

connect

```
NAME
       connect - initiate a connection on a socket

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

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

> · 客户端需要调用connect()连接服务器;
>
> · connect和bind的参数形式一致, 区别在于bind的参数是自己的地址, 而connect的参数是对方的地址;
>
> · connect()成功返回0,出错返回-1;

V1 - Echo Server

nocopy.hpp

代码块

```
#pragma once
#include <iostream>
class nocopy
{
public:
    nocopy() {}
    nocopy(const nocopy &) = delete;
    const nocopy &operator=(const nocopy &) = delete;
    ~nocopy() {}
};
```

TcpServer.hpp

代码块

```
#pragma once
#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <cstdlib>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "Log.hpp"
#include "nocopy.hpp"
#include "Comm.hpp"
const static int default_backlog = 6; // TODO
class TcpServer : public nocopy
{
public:
    TcpServer(uint16_t port) : _port(port), _isrunning(false)
    {
    }
    // 都是固定套路
    void Init()
```

```
23      {
24          // 1. 创建socket, file fd, 本质是文件
25          _listensock = socket(AF_INET, SOCK_STREAM, 0);
26          if (_listensock < 0)
27          {
28              lg.LogMessage(Fatal, "create socket error, errno code: %d, error
29              string: %s\n", errno, strerror(errno));
30              exit(Fatal);
31          }
32          int opt = 1;
33          setsockopt(_listensock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
34                     &opt, sizeof(opt));
35          lg.LogMessage(Debug, "create socket success, sockfd: %d\n",
36                        _listensock);
37          // 2. 填充本地网络信息并bind
38          struct sockaddr_in local;
39          memset(&local, 0, sizeof(local));
40          local.sin_family = AF_INET;
41          local.sin_port = htons(_port);
42          local.sin_addr.s_addr = htonl(INADDR_ANY);
43          // 2.1 bind
44          if (bind(_listensock, CONV(&local), sizeof(local)) != 0)
45          {
46              lg.LogMessage(Fatal, "bind socket error, errno code: %d, error
47              string: %s\n", errno, strerror(errno));
48              exit(Bind_Err);
49          }
50          lg.LogMessage(Debug, "bind socket success, sockfd: %d\n",
51                        _listensock);
52          // 3. 设置socket为监听状态, tcp特有的
53          if (listen(_listensock, default_backlog) != 0)
54          {
55              lg.LogMessage(Fatal, "listen socket error, errno code: %d, error
56              string: %s\n", errno, strerror(errno));
57              exit(Listen_Err);
58          }
59          lg.LogMessage(Debug, "listen socket success, sockfd: %d\n",
60                        _listensock);
61      }
62      // Tcp 连接全双工通信的.
63      void Service(int sockfd)
64      {
65          char buffer[1024];
66          // 一直进行IO
67          while (true)
68          {
69              ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
```

```
70              if (n > 0)
71              {
72                  buffer[n] = 0;
73                  std::cout << "client say# " << buffer << std::endl;
74                  std::string echo_string = "server echo# ";
75                  echo_string += buffer;
76                  write(sockfd, echo_string.c_str(), echo_string.size());
77              }
78              else if (n == 0) // read如果返回值是0，表示读到了文件结尾(对端关闭了连
    接！)
79              {
80                  lg.LogMessage(Info, "client quit...\n");
81                  break;
82              }
83              else
84              {
85                  lg.LogMessage(Error, "read socket error, errno code: %d,
86                  error string: %s\n", errno, strerror(errno));
87                  break;
88              }
89          }
90      }
91      void Start()
92      {
93          _isrunning = true;
94          while (_isrunning)
95          {
96              // 4. 获取连接
97              struct sockaddr_in peer;
98              socklen_t len = sizeof(peer);
99              int sockfd = accept(_listensock, CONV(&peer), &len);
100             if (sockfd < 0)
101             {
102                 lg.LogMessage(Warning, "accept socket error, errno code: %d,
103                 error string: %s\n", errno, strerror(errno));
104                 continue;
105             }
106             lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n",
107                           sockfd);
108             // 5. 提供服务啊, v1~v4
109             // v1
110             Service(sockfd);
111             close(sockfd);
112         }
113     }
114     ~TcpServer()
115     {
```

```
116         }
117
118     private:
119         uint16_t _port;
120         int _listensock; // TODO
121         bool _isrunning;
122     };
```

Comm.hpp

代码块

```
1   #include <sys/types.h>
2   #include <sys/socket.h>
3   #include <netinet/in.h>
4   #include <arpa/inet.h>
5   enum
6   {
7       Usage_Err = 1,
8       Socket_Err,
9       Bind_Err,
10      Listen_Err
11  };
12  #define CONV(addr_ptr) ((struct sockaddr *)addr_ptr)
```

· Log.hpp就不再单独粘贴了

TcpClient.cc

代码块

```
1   #include <iostream>
2   #include <string>
3   #include <cstring>
4   #include <cstdlib>
5   #include <unistd.h>
6   #include <sys/types.h>
7   #include <sys/socket.h>
8   #include <netinet/in.h>
9   #include <arpa/inet.h>
10  #include "Comm.hpp"
11  using namespace std;
12  void Usage(const std::string &process)
13  {
14      std::cout << "Usage: " << process << " server_ip server_port" << std::endl;
15  }
```

```cpp
// ./tcp_client serverip serverport
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        Usage(argv[0]);
        return 1;
    }
    std::string serverip = argv[1];
    uint16_t serverport = stoi(argv[2]);
    // 1. 创建socket
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        cerr << "socket error" << endl;
        return 1;
    }
    // 2. 要不要bind? 必须要有Ip和Port, 需要bind, 但是不需要用户显示的bind, client系统随机端口
    // 发起连接的时候, client会被OS自动进行本地绑定
    // 2. connect
    struct sockaddr_in server;
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(serverport);
    // p:process(进程), n(网络) -- 不太准确, 但是好记忆
    inet_pton(AF_INET, serverip.c_str(), &server.sin_addr); // 1. 字符串ip->4字节IP 2. 网络序列
    int n = connect(sockfd, CONV(&server), sizeof(server)); // 自动进行bind哦!
    if (n < 0)
    {
        cerr << "connect error" << endl;
        return 2;
    }
    // 并没有向server一样, 产生新的sockfd.未来我们就用connect成功的sockfd进行通信即可.
    while (true)
    {
        string inbuffer;
        cout << "Please Enter# ";
        getline(cin, inbuffer);
        ssize_t n = write(sockfd, inbuffer.c_str(), inbuffer.size());
        if (n > 0)
        {
            char buffer[1024];
            ssize_t m = read(sockfd, buffer, sizeof(buffer) - 1);
            if (m > 0)
            {
```

```
61                  buffer[m] = 0;
62                  cout << "get a echo messsge -> " << buffer << endl;
63              }
64              else if (m == 0 || m < 0)
65              {
66                  break;
67              }
68          }
69          else
70          {
71              break;
72          }
73      }
74      close(sockfd);
75      return 0;
76  }
```

由于客户端不需要固定的端口号,因此不必调用 bind() ,客户端的端口号由内核自动分配.

注意:

> · 客户端不是不允许调用bind(), 只是没有必要显示的调用bind()固定一个端口号. 否则如果在同一台机器上启动多个客户端, 就会出现端口号被占用导致不能正确建立连接;
>
> · 服务器也不是必须调用bind(), 但如果服务器不调用bind(), 内核会自动给服务器分配监听端口, 每次启动服务器时端口号都不一样, 客户端要连接服务器就会遇到麻烦;

测试多个连接的情况

再启动一个客户端, 尝试连接服务器, 发现第二个客户端, 不能正确的和服务器进行通信.

分析原因, 是因为我们 `accecpt` 了一个请求之后, 就在一直 `while` 循环尝试 `read` , 没有继续调用到 `accecpt` , 导致不能接受新的请求.

我们当前的这个 TCP , 只能处理一个连接, 这是不科学的.

V2 - Echo Server多进程版本

通过每个请求, 创建子进程的方式来支持多连接;

InetAddr.hpp

代码块

```
1  #pragma once
2  #include <iostream>
3  #include <string>
4  #include <sys/types.h>
```

```cpp
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
class InetAddr
{
public:
    InetAddr(struct sockaddr_in &addr) : _addr(addr)
    {
        _port = ntohs(_addr.sin_port);
        _ip = inet_ntoa(_addr.sin_addr);
    }
    std::string Ip() { return _ip; }
    uint16_t Port() { return _port; };
    std::string PrintDebug()
    {
        std::string info = _ip;
        info += ":";
        info += std::to_string(_port); // "127.0.0.1:4444"
        return info;
    }
    const struct sockaddr_in &GetAddr()
    {
        return _addr;
    }
    bool operator==(const InetAddr &addr)
    {
        // other code
        return this->_ip == addr._ip && this->_port == addr._port;
    }
    ~InetAddr() {}

private:
    std::string _ip;
    uint16_t _port;
    struct sockaddr_in _addr;
};
```

TcpServer.hpp

代码块

```cpp
#pragma once
#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
```

```cpp
#include <cstdlib>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include "Log.hpp"
#include "nocopy.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
const static int default_backlog = 6; // TODO
class TcpServer : public nocopy
{
public:
    TcpServer(uint16_t port) : _port(port), _isrunning(false)
    {
    }
    // 都是固定套路
    void Init()
    {
        // 1. 创建socket, file fd, 本质是文件
        _listensock = socket(AF_INET, SOCK_STREAM, 0);
        if (_listensock < 0)
        {
            lg.LogMessage(Fatal, "create socket error, errno code: %d, error
            string: %s\n", errno, strerror(errno));
            exit(Fatal);
        }
        int opt = 1;
        setsockopt(_listensock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt));
        lg.LogMessage(Debug, "create socket success, sockfd: %d\n",
                      _listensock);
        // 2. 填充本地网络信息并bind
        struct sockaddr_in local;
        memset(&local, 0, sizeof(local));
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = htonl(INADDR_ANY);
        // 2.1 bind
        if (bind(_listensock, CONV(&local), sizeof(local)) != 0)
        {
            lg.LogMessage(Fatal, "bind socket error, errno code: %d, error
            string: %s\n", errno, strerror(errno));
            exit(Bind_Err);
        }
        lg.LogMessage(Debug, "bind socket success, sockfd: %d\n",
```

```
53                          _listensock);
54          // 3. 设置socket为监听状态，tcp特有的
55          if (listen(_listensock, default_backlog) != 0)
56          {
57              lg.LogMessage(Fatal, "listen socket error, errno code: %d, error
58              string: %s\n", errno, strerror(errno));
59              exit(Listen_Err);
60          }
61          lg.LogMessage(Debug, "listen socket success, sockfd: %d\n",
62                          _listensock);
63      }
64      // Tcp 连接全双工通信的.
65      void Service(int sockfd)
66      {
67          char buffer[1024];
68          // 一直进行IO
69          while (true)
70          {
71              ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
72              if (n > 0)
73              {
74                  buffer[n] = 0;
75                  std::cout << "client say# " << buffer << std::endl;
76                  std::string echo_string = "server echo# ";
77                  echo_string += buffer;
78                  write(sockfd, echo_string.c_str(), echo_string.size());
79              }
80              else if (n == 0) // read如果返回值是0，表示读到了文件结尾(对端关闭了连
    接! )
81              {
82                  lg.LogMessage(Info, "client quit...\n");
83                  break;
84              }
85              else
86              {
87                  lg.LogMessage(Error, "read socket error, errno code: %d,
88                  error string: %s\n", errno, strerror(errno));
89                  break;
90              }
91          }
92      }
93      void ProcessConnection(int sockfd, struct sockaddr_in &peer)
94      {
95          // v2 多进程
96          pid_t id = fork();
97          if (id < 0)
98          {
```

```
 99              close(sockfd);
100              return;
101          }
102          else if (id == 0)
103          {
104              // child
105              close(_listensock);
106              if (fork() > 0)
107                  exit(0);
108              InetAddr addr(peer); // 获取client地址信息
109              lg.LogMessage(Info, "process connection: %s:%d\n",
110                            addr.Ip().c_str(), addr.Port());
111              // 孙子进程，孤儿进程，被系统领养，正常处理
112              Service(sockfd);
113              close(sockfd);
114              exit(0);
115          }
116          else
117          {
118              close(sockfd);
119              pid_t rid = waitpid(id, nullptr, 0);
120              if (rid == id)
121              {
122                  // do nothing
123              }
124          }
125      }
126      void Start()
127      {
128          _isrunning = true;
129          while (_isrunning)
130          {
131              // 4. 获取连接
132              struct sockaddr_in peer;
133              socklen_t len = sizeof(peer);
134              int sockfd = accept(_listensock, CONV(&peer), &len);
135              if (sockfd < 0)
136              {
137                  lg.LogMessage(Warning, "accept socket error, errno code: %d,
138                  error string: %s\n", errno, strerror(errno));
139                  continue;
140              }
141              lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n",
142                            sockfd);
143              ProcessConnection(sockfd, peer);
144          }
145      }
```

```
146         ~TcpServer()
147         {
148         }
149 
150     private:
151         uint16_t _port;
152         int _listensock; // TODO
153         bool _isrunning;
154     };
```

· 引入InetAddr.hpp,方便后面打印消息

V3 - Echo Server多线程版本

Thread.hpp

代码块

```
1   #pragma once
2   #include <iostream>
3   #include <string>
4   #include <cerrno>
5   #include <cstring>
6   #include <cstdlib>
7   #include <sys/types.h>
8   #include <sys/socket.h>
9   #include <netinet/in.h>
10  #include <arpa/inet.h>
11  #include <sys/wait.h>
12  #include <pthread.h>
13  #include "Log.hpp"
14  #include "nocopy.hpp"
15  #include "Comm.hpp"
16  #include "InetAddr.hpp"
17  const static int default_backlog = 6; // TODO
18  class TcpServer : public nocopy
19  {
20  public:
21      TcpServer(uint16_t port) : _port(port), _isrunning(false)
22      {
23      }
24      // 都是固定套路
25      void Init()
26      {
27          // 1. 创建socket, file fd, 本质是文件
```

```
28          _listensock = socket(AF_INET, SOCK_STREAM, 0);
29          if (_listensock < 0)
30          {
31              lg.LogMessage(Fatal, "create socket error, errno code: %d, error
32              string: %s\n", errno, strerror(errno));
33              exit(Fatal);
34          }
35          int opt = 1;
36          setsockopt(_listensock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
37                     &opt, sizeof(opt));
38          lg.LogMessage(Debug, "create socket success, sockfd: %d\n",
39                        _listensock);
40          // 2. 填充本地网络信息并bind
41          struct sockaddr_in local;
42          memset(&local, 0, sizeof(local));
43          local.sin_family = AF_INET;
44          local.sin_port = htons(_port);
45          local.sin_addr.s_addr = htonl(INADDR_ANY);
46          // 2.1 bind
47          if (bind(_listensock, CONV(&local), sizeof(local)) != 0)
48          {
49              lg.LogMessage(Fatal, "bind socket error, errno code: %d, error
50              string: %s\n", errno, strerror(errno));
51              exit(Bind_Err);
52          }
53          lg.LogMessage(Debug, "bind socket success, sockfd: %d\n",
54                        _listensock);
55          // 3. 设置socket为监听状态，tcp特有的
56          if (listen(_listensock, default_backlog) != 0)
57          {
58              lg.LogMessage(Fatal, "listen socket error, errno code: %d, error
59              string: %s\n", errno, strerror(errno));
60              exit(Listen_Err);
61          }
62          lg.LogMessage(Debug, "listen socket success, sockfd: %d\n",
63                        _listensock);
64      }
65      class ThreadData
66      {
67      public:
68          ThreadData(int sockfd, struct sockaddr_in addr)
69              : _sockfd(sockfd), _addr(addr)
70          {
71          }
72          ~ThreadData()
73          {
74          }
```

```
75
76      public:
77          int _sockfd;
78          InetAddr _addr;
79      };
80      // Tcp 连接全双工通信的.
81      // 新增static
82      static void Service(ThreadData &td)
83      {
84          char buffer[1024];
85          // 一直进行IO
86          while (true)
87          {
88              ssize_t n = read(td._sockfd, buffer, sizeof(buffer) - 1);
89              if (n > 0)
90              {
91                  buffer[n] = 0;
92                  std::cout << "client say# " << buffer << std::endl;
93                  std::string echo_string = "server echo# ";
94                  echo_string += buffer;
95                  write(td._sockfd, echo_string.c_str(), echo_string.size());
96              }
97              else if (n == 0) // read如果返回值是0，表示读到了文件结尾(对端关闭了连
    接！)
98              {
99                  lg.LogMessage(Info, "client[%s:%d] quit...\n",
100                                td._addr.Ip().c_str(), td._addr.Port());
101                 break;
102             }
103             else
104             {
105                 lg.LogMessage(Error, "read socket error, errno code: %d,
106                 error string: %s\n", errno, strerror(errno));
107                 break;
108             }
109         }
110     }
111     static void *threadExcute(void *args)
112     {
113         pthread_detach(pthread_self());
114         ThreadData *td = static_cast<ThreadData *>(args);
115         TcpServer::Service(*td);
116         close(td->_sockfd);
117         delete td;
118         return nullptr;
119     }
120     void ProcessConnection(int sockfd, struct sockaddr_in &peer)
```

```
121         {
122             // v3 多线程
123             InetAddr addr(peer);
124             pthread_t tid;
125             ThreadData *td = new ThreadData(sockfd, peer);
126             pthread_create(&tid, nullptr, threadExcute, (void *)td);
127         }
128         void Start()
129         {
130             _isrunning = true;
131             while (_isrunning)
132             {
133                 // 4. 获取连接
134                 struct sockaddr_in peer;
135                 socklen_t len = sizeof(peer);
136                 int sockfd = accept(_listensock, CONV(&peer), &len);
137                 if (sockfd < 0)
138                 {
139                     lg.LogMessage(Warning, "accept socket error, errno code: %d,
140                     error string: %s\n", errno, strerror(errno));
141                     continue;
142                 }
143                 lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n",
144                               sockfd);
145                 ProcessConnection(sockfd, peer);
146             }
147         }
148         ~TcpServer()
149         {
150         }
151 
152     private:
153         uint16_t _port;
154         int _listensock; // TODO
155         bool _isrunning;
156     };
```

· 使用最原始的接口，使用内部ThreadData类

V3-1 - 多线程远程命令执行

Command.hpp

· 命令类，用来执行命令，并获取结果

· 这里暂停，做一个多线程的小业务

代码块

```
#pragma once
#include <iostream>
#include <string>
#include <set>
#include <unistd.h>
class Command
{
private:
public:
    Command() {}
    Command(int sockfd) : _sockfd(sockfd)
    {
        // 课堂上，只允许少量命令的执行，限定一下命令的个数和范围
        _safe_command.insert("ls");
        _safe_command.insert("pwd");
        _safe_command.insert("ls -l");
        _safe_command.insert("ll");
        _safe_command.insert("touch");
        _safe_command.insert("who");
        _safe_command.insert("whoami");
    }
    bool IsSafe(const std::string &command)
    {
        auto iter = _safe_command.find(command);
        if (iter == _safe_command.end())
            return false; // 要执行的命令不set中，不安全
        else
            return true;
    }
    std::string Execute(const std::string &command)
    {
        if (!IsSafe(command))
            return "unsafe";
        FILE *fp = popen(command.c_str(), "r");
        if (fp == nullptr)
            return std::string();
        char buffer[1024];
        std::string result;
        while (fgets(buffer, sizeof(buffer), fp))
        {
            result += buffer;
        }
        pclose(fp);
        return result;
    }
    std::string RecvCommand()
    {
```

```
48          char line[1024];
49          ssize_t n = recv(_sockfd, line, sizeof(line) - 1, 0); // 因为我们尚未学习如何定制协议，所以这里暂时这样写
50          if (n > 0)
51          {
52              line[n] = 0;
53              return line;
54          }
55          else
56          {
57              return std::string();
58          }
59      }
60      void SendCommand(std::string result)
61      {
62          if (result.empty())
63              result = "done"; // 主要是有些命令没有结果，比如touch
64          send(_sockfd, result.c_str(), result.size(), 0);
65      }
66      ~Command()
67      {
68      }
69
70  private:
71      std::set<std::string> _safe_command;
72      int _sockfd;
73      std::string _command;
74  };
```

TcpServer.hpp

代码块

```
1   #pragma once
2   #include <iostream>
3   #include <string>
4   1 2 3 4
5   #include <cerrno>
6   #include <cstring>
7   #include <cstdlib>
8   #include <sys/types.h>
9   #include <sys/socket.h>
10  #include <netinet/in.h>
11  #include <arpa/inet.h>
12  #include <sys/wait.h>
13  #include <pthread.h>
```

```cpp
#include "Log.hpp"
#include "nocopy.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
#include "Command.hpp"                    // 引入命令执行
    const static int default_backlog = 6; // TODO
class TcpServer : public nocopy
{
public:
    TcpServer(uint16_t port) : _port(port), _isrunning(false)
    {
    }
    // 都是固定套路
    void Init()
    {
        // 1. 创建socket, file fd, 本质是文件
        _listensock = socket(AF_INET, SOCK_STREAM, 0);
        if (_listensock < 0)
        {
            lg.LogMessage(Fatal, "create socket error, errno code: %d, error
            string: %s\n", errno, strerror(errno));
            exit(Fatal);
        }
        int opt = 1;
        setsockopt(_listensock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt));
        lg.LogMessage(Debug, "create socket success, sockfd: %d\n",
                      _listensock);
        // 2. 填充本地网络信息并bind
        struct sockaddr_in local;
        memset(&local, 0, sizeof(local));
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = htonl(INADDR_ANY);
        // 2.1 bind
        if (bind(_listensock, CONV(&local), sizeof(local)) != 0)
        {
            lg.LogMessage(Fatal, "bind socket error, errno code: %d, error
            string: %s\n", errno, strerror(errno));
            exit(Bind_Err);
        }
        lg.LogMessage(Debug, "bind socket success, sockfd: %d\n",
                      _listensock);
        // 3. 设置socket为监听状态, tcp特有的
        if (listen(_listensock, default_backlog) != 0)
        {
            lg.LogMessage(Fatal, "listen socket error, errno code: %d, error
```

```
61              string: %s\n", errno, strerror(errno));
62              exit(Listen_Err);
63          }
64          lg.LogMessage(Debug, "listen socket success, sockfd: %d\n",
65                        _listensock);
66      }
67      class ThreadData
68      {
69      public:
70          ThreadData(int sockfd, struct sockaddr_in addr)
71              : _sockfd(sockfd), _addr(addr)
72          {
73          }
74          ~ThreadData()
75          {
76          }
77
78      public:
79          int _sockfd;
80          InetAddr _addr;
81      };
82      // Tcp 连接全双工通信的.
83      // 新增static
84      static void Service(ThreadData &td)
85      {
86          char buffer[1024];
87          // 一直进行IO
88          while (true)
89          {
90              Command command(td._sockfd);
91              std::string commandstr = command.RecvCommand();
92              if (commandstr.empty())
93                  return;
94              std::string result = command.Execute(commandstr);
95              command.SendCommand(result);
96          }
97      }
98      static void *threadExcute(void *args)
99      {
100         pthread_detach(pthread_self());
101         ThreadData *td = static_cast<ThreadData *>(args);
102         TcpServer::Service(*td);
103         close(td->_sockfd);
104         delete td;
105         return nullptr;
106     }
107     void ProcessConnection(int sockfd, struct sockaddr_in &peer)
```

```
108         {
109             // v3 多线程
110             InetAddr addr(peer);
111             pthread_t tid;
112             ThreadData *td = new ThreadData(sockfd, peer);
113             pthread_create(&tid, nullptr, threadExcute, (void *)td);
114         }
115         void Start()
116         {
117             _isrunning = true;
118             while (_isrunning)
119             {
120                 // 4. 获取连接
121                 struct sockaddr_in peer;
122                 socklen_t len = sizeof(peer);
123                 int sockfd = accept(_listensock, CONV(&peer), &len);
124                 if (sockfd < 0)
125                 {
126                     lg.LogMessage(Warning, "accept socket error, errno code: %d,
127                     error string: %s\n", errno, strerror(errno));
128                     continue;
129                 }
130                 lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n",
131                               sockfd);
132                 ProcessConnection(sockfd, peer);
133             }
134         }
135         ~TcpServer()
136         {
137         }
138 
139     private:
140         uint16_t _port;
141         int _listensock; // TODO
142         bool _isrunning;
143     };
```

· 为了保证上课的速度，不拖慢节奏，这里尽量少改动代码

V4 - Echo Server 线程池版本

· 引入系统部分的线程池，进行简单的业务处理

TcpServer.hpp

```cpp
#pragma once
#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <cstdlib>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <pthread.h>
#include <functional>
#include "Log.hpp"
#include "nocopy.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
#include "ThreadPool.hpp"
const static int default_backlog = 6; // TODO
class TcpServer : public nocopy
{
public:
    TcpServer(uint16_t port) : _port(port), _isrunning(false)
    {
    }
    // 都是固定套路
    void Init()
    {
        // 1. 创建socket, file fd, 本质是文件
        _listensock = socket(AF_INET, SOCK_STREAM, 0);
        if (_listensock < 0)
        {
            lg.LogMessage(Fatal, "create socket error, errno code: %d, error
            string: %s\n", errno, strerror(errno));
            exit(Fatal);
        }
        int opt = 1;
        setsockopt(_listensock, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                   &opt, sizeof(opt));
        lg.LogMessage(Debug, "create socket success, sockfd: %d\n",
                      _listensock);
        // 2. 填充本地网络信息并bind
        struct sockaddr_in local;
        memset(&local, 0, sizeof(local));
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
48          // 2.1 bind
49          if (bind(_listensock, CONV(&local), sizeof(local)) != 0)
50          {
51              lg.LogMessage(Fatal, "bind socket error, errno code: %d, error
52              string: %s\n", errno, strerror(errno));
53              exit(Bind_Err);
54          }
55          lg.LogMessage(Debug, "bind socket success, sockfd: %d\n",
56                        _listensock);
57          // 3. 设置socket为监听状态，tcp特有的
58          if (listen(_listensock, default_backlog) != 0)
59          {
60              lg.LogMessage(Fatal, "listen socket error, errno code: %d, error
61              string: %s\n", errno, strerror(errno));
62              exit(Listen_Err);
63          }
64          lg.LogMessage(Debug, "listen socket success, sockfd: %d\n",
65                        _listensock);
66      }
67      // Tcp 连接全双工通信的.
68      void Service(int sockfd, InetAddr addr)
69      {
70          char buffer[1024];
71          // 一直进行IO
72          while (true)
73          {
74              ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
75              if (n > 0)
76              {
77                  buffer[n] = 0;
78                  std::cout << "client say# " << buffer << std::endl;
79                  std::string echo_string = "server echo# ";
80                  echo_string += buffer;
81                  write(sockfd, echo_string.c_str(), echo_string.size());
82              }
83              else if (n == 0) // read如果返回值是0，表示读到了文件结尾(对端关闭了连
    接! )
84              {
85                  lg.LogMessage(Info, "client[%s:%d] quit...\n",
86                                addr.Ip().c_str(), addr.Port());
87                  break;
88              }
89              else
90              {
91                  lg.LogMessage(Error, "read socket error, errno code: %d,
92                  error string: %s\n", errno, strerror(errno));
93                  break;
```

```
94              }
95          }
96      }
97      void ProcessConnection(int sockfd, struct sockaddr_in &peer)
98      {
99          using func_t = std::function<void()>;
100         InetAddr addr(peer);
101         func_t func = std::bind(&TcpServer::Service, this, sockfd, addr); // 这
    里不能auto
102         ThreadPool<func_t>::GetInstance()->Push(func);
103     }
104     void Start()
105     {
106         _isrunning = true;
107         while (_isrunning)
108         {
109             // 4. 获取连接
110             struct sockaddr_in peer;
111             socklen_t len = sizeof(peer);
112             int sockfd = accept(_listensock, CONV(&peer), &len);
113             if (sockfd < 0)
114             {
115                 lg.LogMessage(Warning, "accept socket error, errno code: %d,
116                 error string: %s\n", errno, strerror(errno));
117                 continue;
118             }
119             lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n",
120                           sockfd);
121             ProcessConnection(sockfd, peer);
122         }
123         _isrunning = false;
124     }
125     ~TcpServer()
126     {
127     }
128
129 private:
130     uint16_t _port;
131     int _listensock; // TODO
132     bool _isrunning;
133 };
```