3 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);
```

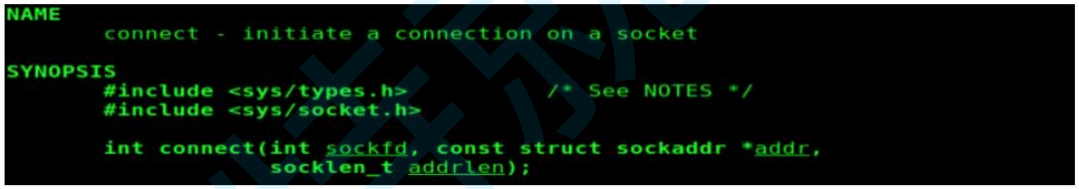
- 三次握手完成后, 服务器调用 accept()接受连接;
- 如果服务器调用 accept()时还没有客户端的连接请求,就阻塞等待直到有客户端连接上来;
- addr 是一个传出参数,accept()返回时传出客户端的地址和端口号;
- 如果给 addr 参数传 NULL,表示不关心客户端的地址;
- addrlen 参数是一个传入传出参数(value-result argument), 传入的是调用者提供的, 缓冲区 addr 的长度以避免缓冲区溢出问题, 传出的是客户端地址结构体的实际长度(有可能没有占满调用者提供的缓冲区);

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



- 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

```
C++
#pragma once
#include <iostream>
```

```
class nocopy
{
public:
    nocopy(){}
    nocopy(const nocopy &) = delete;
    const nocopy& operator = (const nocopy &) = delete;
    ~nocopy(){}
};
```

TcpServer.hpp

```
C++
#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()
    {
        // 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 string: %s\n", errno, strerror(errno));
            exit(Listen_Err);
        }
        lg.LogMessage(Debug, "listen socket success,
sockfd: %d\n", _listensock);
    }
    // Tcp 连接全双工通信的.
    void Service(int sockfd)
    {
        char buffer[1024];
        // 一直进行 IO
        while (true)
        {
            ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
            if (n > 0)
```

```
            {
                buffer[n] = 0;
                std::cout << "client say# " << buffer <<
std::endl;

                std::string echo_string = "server echo# ";
                echo_string += buffer;
                write(sockfd, echo_string.c_str(),
echo_string.size());
            }
            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！)
            {
                lg.LogMessage(Info, "client quit...\n");
                break;
            }
            else
            {
                lg.LogMessage(Error, "read socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                break;
            }
        }
    }
    void Start()
    {
        _isrunning = true;
        while (_isrunning)
        {
            // 4. 获取连接
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = accept(_listensock, CONV(&peer), &len);
            if (sockfd < 0)
            {
                lg.LogMessage(Warning, "accept socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                continue;
            }
            lg.LogMessage(Debug, "accept success, get n new
sockfd: %d\n", sockfd);

            // 5. 提供服务啊, v1~v4
            // v1
```

```
            Service(sockfd);
            close(sockfd);
        }
    }
    ~TcpServer()
    {
    }

private:
    uint16_t _port;
    int _listensock; // TODO
    bool _isrunning;
};
```

Comm.hpp

```
C++
#pragma once
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

enum{
    Usage_Err = 1,
    Socket_Err,
    Bind_Err,
    Listen_Err
};

#define CONV(addr_ptr) ((struct sockaddr *)addr_ptr)
```

- Log.hpp 就不再单独粘贴了

TcpClient.cc

```
C++
#include <iostream>
#include <string>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include "Comm.hpp"

using namespace std;

void Usage(const std::string &process)
{
    std::cout << "Usage: " << process << " server_ip server_port"
<< std::endl;
}

// ./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)
            {
                buffer[m] = 0;
                cout << "get a echo messsge -> " << buffer <<
endl;
            }
            else if(m == 0 || m < 0)
            {
                break;
            }
        }
        else
        {
            break;
        }
    }

    close(sockfd);
    return 0;
}
```

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

```C++
#pragma once
#include <iostream>
#include <string>
#include <sys/types.h>
#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

```
C++
#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 "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",
_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 string: %s\n", errno, strerror(errno));
            exit(Listen_Err);
        }
        lg.LogMessage(Debug, "listen socket success,
sockfd: %d\n", _listensock);
    }
    // Tcp 连接全双工通信的.
    void Service(int sockfd)
    {
        char buffer[1024];
        // 一直进行 IO
        while (true)
        {
            ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
            if (n > 0)
            {
                buffer[n] = 0;
                std::cout << "client say# " << buffer <<
std::endl;

                std::string echo_string = "server echo# ";
                echo_string += buffer;
                write(sockfd, echo_string.c_str(),
echo_string.size());
            }
            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！)
            {
                lg.LogMessage(Info, "client quit...\n");
                break;
            }
            else
            {
                lg.LogMessage(Error, "read socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                break;
            }
        }
    }
    void ProcessConnection(int sockfd, struct sockaddr_in &peer)
    {
        //   v2 多进程
        pid_t id = fork();
        if (id < 0)
```

```
        {
            close(sockfd);
            return;
        }
        else if (id == 0)
        {
            // child
            close(_listensock);
            if (fork() > 0)
                exit(0);

            InetAddr addr(peer); // 获取 client 地址信息
            lg.LogMessage(Info, "process connection: %s:%d\n",
addr.Ip().c_str(), addr.Port());
            // 孙子进程，孤儿进程，被系统领养，正常处理
            Service(sockfd);
            close(sockfd);
            exit(0);
        }
        else
        {
            close(sockfd);
            pid_t rid = waitpid(id, nullptr, 0);
            if (rid == id)
            {
                // do nothing
            }
        }
    }
    void Start()
    {
        _isrunning = true;
        while (_isrunning)
        {
            // 4. 获取连接
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = accept(_listensock, CONV(&peer), &len);
            if (sockfd < 0)
            {
                lg.LogMessage(Warning, "accept socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                continue;
            }
```

```
            lg.LogMessage(Debug, "accept success, get n new
sockfd: %d\n", sockfd);

            ProcessConnection(sockfd, peer);
        }
    }
    ~TcpServer()
    {
    }

private:
    uint16_t _port;
    int _listensock; // TODO
    bool _isrunning;
};
```

- 引入 InetAddr.hpp,方便后面打印消息

V3 - Echo Server 多线程版本

Thread.hpp

```
C++
#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 "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",
_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 string: %s\n", errno, strerror(errno));
            exit(Listen_Err);
        }
        lg.LogMessage(Debug, "listen socket success,
sockfd: %d\n", _listensock);
    }
    class ThreadData
    {
    public:
        ThreadData(int sockfd, struct sockaddr_in addr)
        : _sockfd(sockfd), _addr(addr)
        {}
        ~ThreadData()
        {}
    public:
        int _sockfd;
        InetAddr _addr;
    };
    // Tcp 连接全双工通信的.
    // 新增 static
    static void Service(ThreadData &td)
    {
        char buffer[1024];
        // 一直进行 IO
        while (true)
        {
            ssize_t n = read(td._sockfd, buffer, sizeof(buffer) -
1);
            if (n > 0)
            {
                buffer[n] = 0;
                std::cout << "client say# " << buffer <<
std::endl;

                std::string echo_string = "server echo# ";
                echo_string += buffer;
                write(td._sockfd, echo_string.c_str(),
echo_string.size());
            }
            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！)
            {
                lg.LogMessage(Info, "client[%s:%d] quit...\n",
td._addr.Ip().c_str(), td._addr.Port());
```

```
                break;
            }
            else
            {
                lg.LogMessage(Error, "read socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                break;
            }
        }
    }
    static void *threadExcute(void *args)
    {
        pthread_detach(pthread_self());
        ThreadData *td = static_cast<ThreadData *>(args);
        TcpServer::Service(*td);
        close(td->_sockfd);
        delete td;
        return nullptr;
    }
    void ProcessConnection(int sockfd, struct sockaddr_in &peer)
    {
        //   v3 多线程
        InetAddr addr(peer);
        pthread_t tid;
        ThreadData *td = new ThreadData(sockfd, peer);
        pthread_create(&tid, nullptr, threadExcute, (void*)td);
    }
    void Start()
    {
        _isrunning = true;
        while (_isrunning)
        {
            // 4. 获取连接
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = accept(_listensock, CONV(&peer), &len);
            if (sockfd < 0)
            {
                lg.LogMessage(Warning, "accept socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                continue;
            }
            lg.LogMessage(Debug, "accept success, get n new
sockfd: %d\n", sockfd);
```

```
            ProcessConnection(sockfd, peer);
        }
    }
    ~TcpServer()
    {
    }

private:
    uint16_t _port;
    int _listensock; // TODO
    bool _isrunning;
};
```

- 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

```
C++
#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()
    {
        char line[1024];
        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
        if (n > 0)
        {
            line[n] = 0;
            return line;
        }
        else
        {
            return std::string();
        }
    }
    void SendCommand(std::string result)
    {
        if(result.empty()) result = "done"; // 主要是有些命令没有结
```

```
果，比如 touch
        send(_sockfd, result.c_str(), result.size(), 0);
    }
    ~Command()
    {
    }

private:
    std::set<std::string> _safe_command;
    int _sockfd;
    std::string _command;
};
```

TcpServer.hpp

```
C++
#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 "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 string: %s\n", errno, strerror(errno));
            exit(Listen_Err);
        }
        lg.LogMessage(Debug, "listen socket success,
sockfd: %d\n", _listensock);
    }
    class ThreadData
```

```
    {
    public:
        ThreadData(int sockfd, struct sockaddr_in addr)
            : _sockfd(sockfd), _addr(addr)
        {
        }
        ~ThreadData()
        {
        }

    public:
        int _sockfd;
        InetAddr _addr;
    };
    // Tcp 连接全双工通信的.
    // 新增 static
    static void Service(ThreadData &td)
    {
        char buffer[1024];
        // 一直进行 IO
        while (true)
        {
            Command command(td._sockfd);
            std::string commandstr = command.RecvCommand();
            if (commandstr.empty())
                return;
            std::string result = command.Execute(commandstr);
            command.SendCommand(result);
        }
    }
    static void *threadExcute(void *args)
    {
        pthread_detach(pthread_self());
        ThreadData *td = static_cast<ThreadData *>(args);
        TcpServer::Service(*td);
        close(td->_sockfd);
        delete td;
        return nullptr;
    }
    void ProcessConnection(int sockfd, struct sockaddr_in &peer)
    {
        //   v3 多线程
        InetAddr addr(peer);
        pthread_t tid;
```

```
        ThreadData *td = new ThreadData(sockfd, peer);
        pthread_create(&tid, nullptr, threadExcute, (void *)td);
    }
    void Start()
    {
        _isrunning = true;
        while (_isrunning)
        {
            // 4. 获取连接
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = accept(_listensock, CONV(&peer), &len);
            if (sockfd < 0)
            {
                lg.LogMessage(Warning, "accept socket error, errno
code: %d, error string: %s\n", errno, strerror(errno));
                continue;
            }
            lg.LogMessage(Debug, "accept success, get n new
sockfd: %d\n", sockfd);

            ProcessConnection(sockfd, peer);
        }
    }
    ~TcpServer()
    {
    }

private:
    uint16_t _port;
    int _listensock; // TODO
    bool _isrunning;
};
```

- 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

```
C++
#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);

        // 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 string: %s\n", errno, strerror(errno));
            exit(Listen_Err);
        }
        lg.LogMessage(Debug, "listen socket success,
sockfd: %d\n", _listensock);
    }
    // Tcp 连接全双工通信的.
    void Service(int sockfd, InetAddr addr)
    {
        char buffer[1024];
        // 一直进行 IO
        while (true)
        {
            ssize_t n = read(sockfd, buffer, sizeof(buffer) - 1);
            if (n > 0)
            {
                buffer[n] = 0;
                std::cout << "client say# " << buffer <<
std::endl;

                std::string echo_string = "server echo# ";
                echo_string += buffer;
                write(sockfd, echo_string.c_str(),
echo_string.size());
```

```
            }
            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！)
            {
                lg.LogMessage(Info, "client[%s:%d] quit...\n", addr.Ip().c_str(), addr.Port());
                break;
            }
            else
            {
                lg.LogMessage(Error, "read socket error, errno code: %d, error string: %s\n", errno, strerror(errno));
                break;
            }
        }
    }
    void ProcessConnection(int sockfd, struct sockaddr_in &peer)
    {
        using func_t = std::function<void()>;
        InetAddr addr(peer);
        func_t func = std::bind(&TcpServer::Service, this, sockfd, addr); // 这里不能 auto
        ThreadPool<func_t>::GetInstance()->Push(func);
    }
    void Start()
    {
        _isrunning = true;
        while (_isrunning)
        {
            // 4. 获取连接
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer);
            int sockfd = accept(_listensock, CONV(&peer), &len);
            if (sockfd < 0)
            {
                lg.LogMessage(Warning, "accept socket error, errno code: %d, error string: %s\n", errno, strerror(errno));
                continue;
            }
            lg.LogMessage(Debug, "accept success, get n new sockfd: %d\n", sockfd);

            ProcessConnection(sockfd, peer);
        }
```

```
        _isrunning = false;
    }
    ~TcpServer()
    {
    }

private:
    uint16_t _port;
    int _listensock; // TODO
    bool _isrunning;
};
```

V4-1 - 引入线程池版本翻译(选学)

- 因为之前 UDP 部分已经写过类似的设计方案，此处略
- 后面我们还会涉及 http 相关内容，到时候在引入线程池会更方便，也更合理。

比特就业课