

# 华东师范大学软件工程学院实验报告

实验课程:	计算机网络	年 级:	2023 级
实验编号:	Lab 07	实验名称:	socket 编程
姓 名:	王海生	学 号:	10235101559

## 目录

1 实验目的	2
2 实验环境	2
3 软件需求分析	3
4 软件设计	4
4.1 架构分层	4
4.2 模块划分	4
4.2.1 服务器端	4
4.2.2 客户端	5
4.3 数据结构	5
4.4 接口设计	5
4.5 异常处理	5
5 软件实现	5
5.1 create socket	5
5.2 bind	6
5.3 listen	6
5.4 accept	7
5.5 connect	7
5.6 send	8
5.7 recv	8
5.8 close	9
6 软件测试	9
6.1 Server	9
6.1.1 能在标准输出打印客户端发送的消息	9
6.1.2 支持 5 个以上客户端同时发送消息并逐一打印	10

6.1.3	绑定至错误的端口号时提示出错信息	11
6.2	Client	12
6.2.1	能从标准输入或文件接收信息	12
6.2.2	标准输入信息以两次回车作为结束标志	13
6.2.3	连接至错误的 IP 地址/端口号时能提示错误信息	14
6.3	整体	16
6.3.1	支持在 <code>localhost</code> 及两台不同机器上运行	16
6.3.2	支持长文本消息（不少于 20KB）	17
6.4	容错性好，无闪退	18
6.5	加分项	18
6.5.1	支持双工通信	18
7	实验结果总结	19
8	附录	20
8.1	代码结构	20
8.2	源代码	20
8.2.1	Makefile	20
8.2.2	socket.hpp	21
8.2.3	socket_client.hpp	24
8.2.4	socket_server.hpp	26
8.2.5	client.cpp	29
8.2.6	server.cpp	31

## 1 实验目的

- 1) 掌握 `Socket` 编程接口；
- 2) 掌握基于 `Socket` 编程接口编写网络通信程序的方法和典型程序构成；
- 3) 根据指定的通信需求，使用 `Socket` 编程接口编写网络通信程序的服务器端程序和客户端程序。

## 2 实验环境

Windows 下：

- Windows 11 家庭中文版 23H2 22631.2715
- gcc version 13.2.0 (MinGW-W64 x86\_64-ucrt-mcf-seh, built by Brecht Sanders)
- GNU Make 3.81

Linux 下：

- Ubuntu 22.04.3 LTS on Windows 10 x86\_64
- gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1 22.04)
- GNU Make 4.3

### 3 软件需求分析

实验要求实现 Client 和 Server 的通信，并满足以下要求：

Server			Client			整体			加分项
能在标准输出打印客户端发送的消息	支持 5 个以上客户端同时发送消息并逐一打印	绑定至错误的端口号时提示出错信息	能从标准输入或文件接收信息	标准输入信息以两次回车作为结束标志	连接至错误的 IP 地址/端口号时能提示错误信息	支持在 local host 及两台不同机器上运行	支持长文本消息（不少于 20KB），有缓冲区管理	容错性好，无闪退	支持双工通信

具体需求分析如下：

#### 1. 服务器端（Server）需求：

- **消息打印：**能够在标准输出上准确打印出客户端发送的消息，确保消息的完整性和准确性，以便服务器端对客户端的请求或信息进行及时处理和反馈。
- **多客户端支持：**具备支持 5 个以上客户端同时发送消息的能力，服务器应能够对每个客户端的连接进行有效管理，并逐一打印来自不同客户端的消息，避免消息混淆和丢失，保证通信的并发处理能力。
- **错误处理：**当绑定至错误的端口号时，服务器应具备错误检测机制，能够及时提示出错信息，明确告知用户端口号无效的原因，如端口号超出范围（小于 0 或大于 65535）或端口已被其他程序占用等情况，以便用户进行相应的调整和处理。

#### 2. 客户端（Client）需求：

- **信息接收：**客户端需要能够从标准输入或指定文件中接收信息，提供灵活的信息获取方式，满足不同用户的使用场景和需求。从标准输入读取信息时，应确保输入的便捷性和流畅性；从文件读取信息时，应支持常见的文本文件格式，准确读取文件中的内容并进行发送。
- **输入结束标志：**客户端从标准输入的信息应以两次回车作为结束标志，这一特定的输入结束条件有助于服务器端和客户端之间明确数据传输的完整性和准确性，避免数据的截断或错误解析，确保双方对数据传输的一致性理解。

- **错误处理:** 在连接至错误的 IP 地址或端口号时, 客户端应具备强大的错误检测和提示能力, 能够及时准确地告知用户连接失败的原因, 例如 IP 地址格式不正确、IP 地址不可达、端口号无效等情况, 帮助用户快速定位和解决连接问题, 提高程序的易用性和稳定性。

### 3. 整体系统需求:

- **跨平台运行:** 支持在 localhost 及两台不同机器上运行, 确保程序在不同的网络环境和操作系统下具有良好的兼容性和可移植性。无论是在同一台主机上的不同进程间通信, 还是在不同物理设备之间的网络通信, 程序都应能够稳定运行, 正确建立连接并进行高效的数据传输。
- **长文本消息支持:** 系统应支持长文本消息的传输, 消息长度不少于 20KB。在处理长文本消息时, 需要合理设计缓冲区管理机制, 确保数据的可靠传输, 避免因消息过长导致的数据丢失、传输中断或内存溢出等问题, 保证大数据量信息的完整性和准确性。
- **容错性:** 整个网络通信系统应具备良好的容错性, 在面对各种异常情况时, 如网络波动、服务器或客户端程序异常退出、连接超时等, 能够保持稳定运行, 不出现闪退等严重影响用户体验的问题。系统应能够自动进行错误恢复或提供友好的错误提示信息, 引导用户采取相应的措施解决问题, 确保通信的连续性和可靠性。
- **加分项: 双工通信支持:** 若要实现加分项, 系统应支持双工通信, 即服务器和客户端之间能够实现双向的信息交互。服务器不仅能够接收客户端发送的消息并进行处理和响应, 还应能够主动向客户端发送消息, 客户端同样能够接收和处理来自服务器的消息, 这种双向通信能力能够大大增强系统的交互性和实用性, 满足更多复杂的网络通信场景需求。

通过满足以上各项需求, 将实现一个功能完善、稳定可靠、兼容性强且具有一定扩展性的基于 Socket 编程的网络通信系统, 能够为用户提供高效、准确的网络通信服务, 适用于多种不同的应用场景, 如即时通讯、文件传输、远程控制等领域。

## 4 软件设计

本项目基于 Socket 编程实现客户端与服务器的网络通信, 采用经典的 C/S 架构。

### 4.1 架构分层

整体分为客户端与服务器端两层: - **客户端:** 负责发起连接请求, 向服务器发送数据, 并接收服务器的反馈。  
- **服务器端:** 监听指定端口, 接收客户端连接, 处理客户端请求并回传数据。

### 4.2 模块划分

#### 4.2.1 服务器端

1. **初始化模块:** 负责初始化 Socket, 在 Windows 系统下还需额外执行相关准备工作, 以此创建通信端点, 并确定通信协议。

2. **绑定监听模块**：先将 Socket 与特定端口绑定，期间严格核验端口有效性，绑定成功后开启监听，设置最大连接数。
3. **连接处理模块**：接收客户端连接请求，成功则记录连接信息，后续利用多线程技术，为每个客户端连接分配独立线程，并行处理通信。
4. **数据交互模块**：承担向客户端发送与接收数据的职责，对收发失败等异常状况妥善处理。

#### 4.2.2 客户端

1. **连接建立模块**：校验 IP 地址与端口号合法性，依此尝试与服务器建立连接。
2. **数据传输模块**：从标准输入、文件获取数据，发送给服务器，同时接收服务器传来的数据，处理收发异常。
3. **输入控制模块**：针对标准输入设计专门的结束规则，例如以两次回车为结束标志，保障输入数据的完整性与准确性。

### 4.3 数据结构

两端都借助 `sockaddr_in` 结构体存储网络地址相关信息。服务器端还用向量存储已连接客户端的套接字描述符，方便管理连接；客户端则增添部分成员变量，辅助记录发送状态。

### 4.4 接口设计

服务器与客户端各自向外暴露关键接口，像 `bind`、`listen`、`send` 等，上层业务代码调用这些接口即可实现网络通信，底层复杂操作被封装隐藏。

### 4.5 异常处理

关键操作执行出错时，抛出特定异常并附加详细错误提示，调用处捕获异常做针对性处理，提升程序容错能力，维持运行稳定。

## 5 软件实现

### 5.1 create socket

在 Linux 下，使用 `sys/socket.h` 头文件（Windows 下是 `winsock2.h`）中的 `socket` 函数创建一个套接字。（在 Windows 下需要先调用 `WSAStartup` 进行准备）

创建套接字

```
1  socket() : m_descriptor(-1) {
2  #ifdef _WIN32
3      WSADATA wsa_data;
4      if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0) {
```

```
5         throw std::runtime_error("WSAStartup_failed");
6     }
7 #endif
8     m_descriptor = ::socket(AF_INET, SOCK_STREAM, 0);
9     if (m_descriptor < 0) {
10         throw std::runtime_error("create_socket_failed");
11     }
12 }
```

其中，AF\_INET 表示使用 IPv4 协议，SOCK\_STREAM 表示使用 TCP 协议。

## 5.2 bind

使用 bind 函数将套接字与端口号绑定。

bind

```
1 void bind(int port) {
2     if (port < 0 || port > 65535) {
3         m_close_socket();
4         throw std::runtime_error("invalid_port");
5     }
6
7     sockaddr_in server_addr;
8     server_addr.sin_family = AF_INET;
9     server_addr.sin_addr.s_addr = INADDR_ANY;
10    server_addr.sin_port = htons(port);
11
12    if (::bind(m_descriptor, (sockaddr *)&server_addr,
13            sizeof(server_addr)) == -1) {
14        m_close_socket();
15        throw std::runtime_error(
16            "bind_to_port_" + std::to_string(port) +
17            "_failed\nmaybe_the_port_is_already_in_use");
18    } else {
19        m_info("bind_to_port_" + std::to_string(port));
20    }
21 }
```

## 5.3 listen

使用 listen 函数监听端口。

listen

```
1 void listen(int n = 10) {
2     if (::listen(m_descriptor, n) == -1) {
3         m_close_socket();
4         throw std::runtime_error("listen_socket_failed");
5     } else {
6         m_info("listening");
7     }
8 }
```

其中，n 表示最大连接数。

## 5.4 accept

使用 accept 函数接受客户端的连接。

accept

```
1 int accept() {
2     sockaddr_in client_addr;
3     socklen_t client_addr_len = sizeof(client_addr);
4     int connected =
5         ::accept(m_descriptor, (sockaddr *)&client_addr, &client_addr_len);
6     if (connected == -1) {
7         m_close_socket();
8         throw std::runtime_error("accept_socket_failed");
9     } else {
10        m_info("accept_socket_id_" + std::to_string(connected) + "_from_" +
11            std::string(inet_ntoa(client_addr.sin_addr)) + ":" +
12            std::to_string(ntohs(client_addr.sin_port)));
13    }
14    return connected;
15 }
```

## 5.5 connect

使用 connect 函数主动与服务器建立连接。

connect

```
1 int connect(const char *ip, int port) {
2     sockaddr_in server_addr;
3     server_addr.sin_family = AF_INET;
4     server_addr.sin_addr.s_addr = inet_addr(ip);
5     server_addr.sin_port = htons(port);
6 }
```

```

7     int connected = ::socket(AF_INET, SOCK_STREAM, 0);
8
9     if (::connect(connected, (sockaddr *)&server_addr,
10                sizeof(server_addr)) == -1) {
11         m_close_socket();
12         throw std::runtime_error("connect_to_ " + std::string(ip) + ":" +
13                                std::to_string(port) + "_failed");
14     } else {
15         m_info("connect_to_ " + std::string(ip) + ":" +
16              std::to_string(port));
17     }
18     return connected;
19 }
20
21 int connect(std::string_view ip, int port) {
22     return connect(ip.data(), port);
23 }

```

## 5.6 send

使用 send 函数发送数据。

send

```

1 void send(int to, const char *data, int len) {
2     if (::send(to, data, len, 0) == -1) {
3         m_error("send_data_to_ " + std::to_string(to) + "_failed");
4     }
5 }
6
7 void send(int to, std::string_view data) {
8     send(to, data.data(), data.size());
9 }

```

## 5.7 recv

使用 recv 函数接收数据。

recv

```

1 bool recv(int from, char *data, int len) {
2     int read_num = ::recv(from, data, len, 0);
3     if (read_num == -1) {
4         m_error("recv_data_from_ " + std::to_string(from) + "_failed");

```



```
5         return false;
6     }
7     if (read_num == 0) {
8         return false;
9     }
10    return true;
11 }
```

## 5.8 close

使用 `close` 函数关闭套接字。

close

```
1 void m_close_socket() {
2     if (m_descriptor != -1) {
3         m_info("socket_closed");
4         ::close(m_descriptor);
5         m_descriptor = -1;
6     }
7 }
8
9 // ...
10
11 ~socket() {
12     m_close_socket();
13 }
14
15 void close() {
16     m_close_socket();
17 }
```

## 6 软件测试

### 6.1 Server

#### 6.1.1 能在标准输出打印客户端发送的消息

在收到消息后打印即可。

```
1 void handle_connection(socket_server &socket, int client) {
2     while (true) {
3         auto msg = socket.recv(client);
4         if (msg.empty()) {
```

```
5         break;
6     }
7     std::cout << "server_received_from_" << client << " :_" << msg
8         << std::endl;
9     msg = msg.substr(0, msg.size() - 1);
10    socket.send_all(msg == "\n" ? "" : msg);
11    socket.send_all("");
12 }
13 }
```

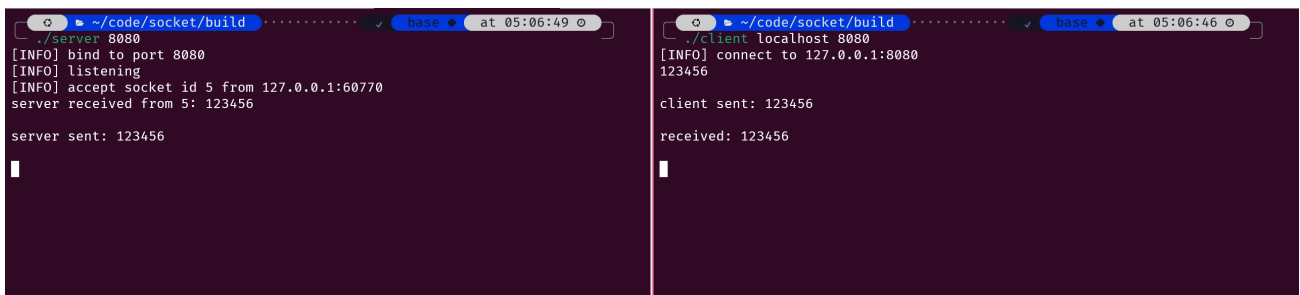


图 1: 能在标准输出打印客户端发送的消息

### 6.1.2 支持 5 个以上客户端同时发送消息并逐一打印

使用多线程，每个线程处理一个客户端的连接。

```
1  socket_server server{};
2
3  server.bind(port);
4  server.listen();
5
6  std::vector<std::thread> threads{};
7
8  while (true) {
9      int client = server.accept();
10     threads.emplace_back(handle_connection, std::ref(server), client);
11 }
12
13 for (auto &thread : threads) {
14     thread.join();
15 }
```

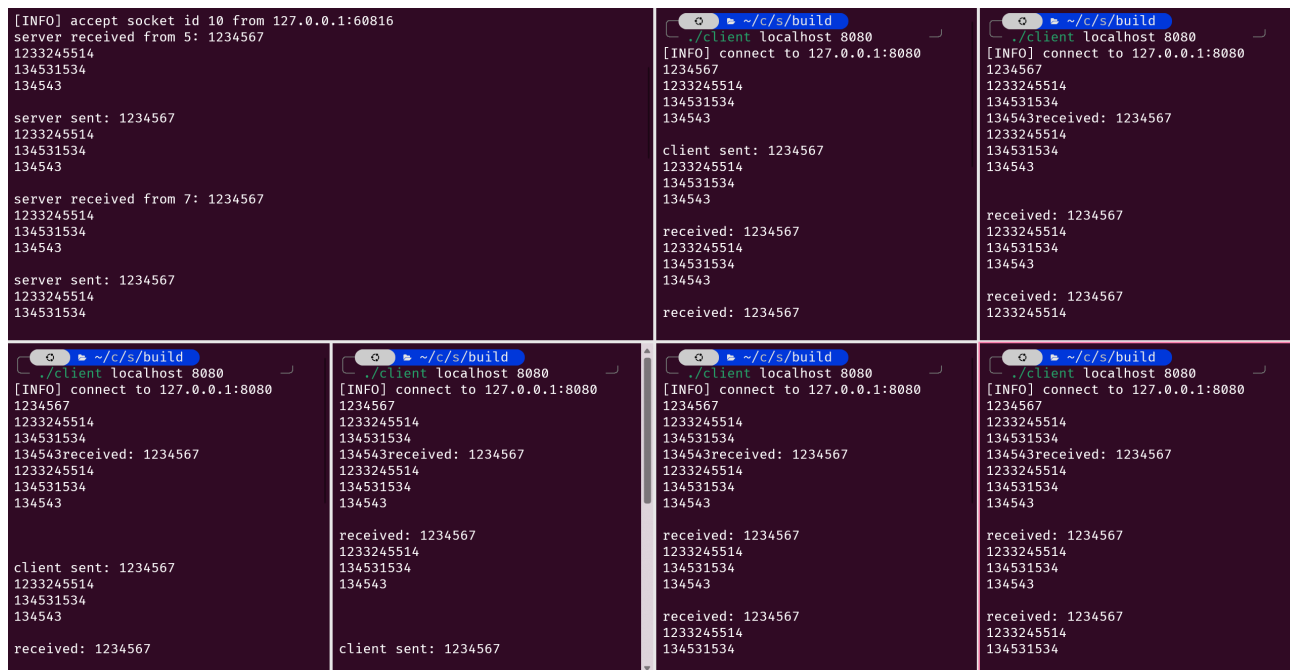


图 2: 支持 5 个以上客户端同时发送消息并逐一打印

### 6.1.3 绑定至错误的端口号时提示出错信息

对端口号进行检查，并在 `bind` 失败时抛出异常。

```

1 void bind(int port) {
2     if (port < 0 || port > 65535) {
3         m_close_socket();
4         throw std::runtime_error("invalid port");
5     }
6
7     sockaddr_in server_addr;
8     server_addr.sin_family = AF_INET;
9     server_addr.sin_addr.s_addr = INADDR_ANY;
10    server_addr.sin_port = htons(port);
11
12    if (::bind(m_descriptor, (sockaddr *)&server_addr,
13        sizeof(server_addr)) == -1) {
14        m_close_socket();
15        throw std::runtime_error(
16            "bind to port " + std::to_string(port) +
17            "\nfailed\nmaybe the port is already in use");
18    } else {
19        m_info("bind to port " + std::to_string(port));
20    }

```

```

21 }
22
23 // ...
24
25 int port{-1};
26 try {
27     port = std::stoi(argv[1]);
28 } catch (std::exception &e) {
29     std::cerr << "Invalid port" << std::endl;
30     show_usage();
31     return 1;
32 }
33
34 if (port < 0 || port > 65535) {
35     std::cerr << "Invalid port" << std::endl;
36     show_usage();
37     return 1;
38 }

```

The figure consists of three terminal screenshots showing the execution of a program named `server` in the directory `~/code/socket/build`.

- First screenshot (05:17:51):** The command `./server 22` is executed. The output shows "[INFO] socket closed" and a `terminate called after throwing an instance of 'std::runtime_error'` with the message `what(): bind to port 22 failed maybe the port is already in use`. The prompt then shows `[1] 17344 IOT instruction ./server 22`.
- Second screenshot (05:18:00):** The command `./server -10` is executed. The output shows `Invalid port` and `Usage: ./server <port>`. The status bar indicates an "IOT x" error.
- Third screenshot (05:18:39):** The command `./server 1` is executed. The output shows `Invalid port` and `Usage: ./server <port>`. The status bar indicates a "1 x" error.

图 3: 绑定至错误的端口号时提示出错信息

## 6.2 Client

### 6.2.1 能从标准输入或文件接收信息

使用 `getline` 函数从标准输入读取信息。

```

1 while (true) {
2     std::string msg;

```

```

3     std::getline(std::cin, msg);
4     client.send(msg);
5 }

```

使用 `std::fstream` 从文件流读取信息。

```

1  if (argc == 5) {
2     std::fstream in{argv[4]};
3     std::string str{};
4     std::string tmp{};
5     while (std::getline(in, tmp)) {
6         str.append(tmp).append("\n");
7     }
8     client.send(str);
9     client.send("");
10 }

```

```

[~/code/socket/build] ..... base at 05:27:44
./server 8080
[INFO] bind to port 8080
[INFO] listening
[INFO] accept socket id 5 from 127.0.0.1:60874
server received from 5: 1235131535
53443513531351
35443
355394

server sent: 1235131535
53443513531351
35443
355394

[INFO] accept socket id 6 from 127.0.0.1:60998
server received from 6: CXX = g++
ifeq ($(OS), Windows_NT)
    CXXFLAGS = -std=c++17 -lwsack32
else
    CXXFLAGS = -std=c++17
endif
BUILD_DIR = build

all: server client

server: src/server.cpp
$(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)

client: src/client.cpp
$(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)

clean:
ifeq ($(OS), Windows_NT)
    del /Q $(BUILD_DIR)\*
else
    rm -rf $(BUILD_DIR)/
endif

```

```

[~/code/socket/build] ..... base at 05:27:42
./client localhost 8080
[INFO] connect to 127.0.0.1:8080
1235131535
53443513531351
35443
355394

client sent: 1235131535
53443513531351
35443
355394

received: 1235131535
53443513531351
35443
355394

received: CXX = g++

```

```

[~/code/socket/build] ..... base at 05:28:08
./client localhost 8080 --file ../Makefile
[INFO] connect to 127.0.0.1:8080
client sent: CXX = g++
ifeq ($(OS), Windows_NT)
    CXXFLAGS = -std=c++17 -lwsack32
else
    CXXFLAGS = -std=c++17
endif
BUILD_DIR = build

all: server client

server: src/server.cpp
$(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)

client: src/client.cpp
$(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)

```

图 4: 能从标准输入或文件接收信息

### 6.2.2 标准输入信息以两次回车作为结束标志

在发送前进行判断即可。

```

1 void send(std::string_view msg) {
2     if (msg.empty()) {

```

```

3      if (m_last_send_enter) {
4          socket::send(m_server_descriptor,
5                      m_last_send + std::string{msg});
6          std::cout << "client sent: " << m_last_send + std::string{msg} <<
              std::endl;
7          if (msg == "exit") {
8              close();
9          }
10         m_last_send_enter = false;
11         m_last_send = "";
12         return;
13     }
14     m_last_send_enter = true;
15     m_last_send = "\n";
16     return;
17 }
18 m_last_send.append(msg).append("\n");
19 m_last_send_enter = true;
20 }

```

```

[~/code/socket/build] ..... base at 05:32:51
[server 8080]
[INFO] bind to port 8080
[INFO] listening
[INFO] accept socket id 5 from 127.0.0.1:60774
server received from 5: 12345

server sent: 12345

server received from 5:

server sent:

server received from 5: 123456
7891011

server sent: 123456
7891011

```

```

[~/code/socket/build] ..... base at 05:32:53
[client localhost 8080]
[INFO] connect to 127.0.0.1:8080
12345

client sent: 12345

received: 12345

client sent:

received:

123456
7891011

client sent: 123456
7891011

received: 123456
7891011

```

图 5: 标准输入信息以两次回车作为结束标志

### 6.2.3 连接至错误的 IP 地址/端口号时能提示错误信息

对 IP 地址和端口进行检查即可。

```

1 bool check_ipv4(std::string &addr) {
2     if (addr.empty() || addr.back() == '.' || addr.front() == '.') {
3         return false;
4     }

```

```
5     if (addr == "localhost") {
6         addr = "127.0.0.1";
7     }
8     int num{0};
9     int dot{0};
10    for (auto &c : addr) {
11        if (c == '.') {
12            if (num < 0 || num > 255) {
13                return false;
14            }
15            num = 0;
16            ++dot;
17            continue;
18        }
19        if (c < '0' || c > '9') {
20            return false;
21        }
22        num = num * 10 + (c - '0');
23    }
24    if (num < 0 || num > 255 || dot != 3) {
25        return false;
26    }
27    return true;
28 }
29
30 // ...
31
32 std::string ip{argv[1]};
33 if (!check_ipv4(ip)) {
34     std::cerr << "Invalid_ip" << std::endl;
35     show_usage();
36     return 1;
37 }
38
39 int port{-1};
40 try {
41     port = std::stoi(argv[2]);
42 } catch (std::exception &e) {
43     std::cerr << "Invalid_port" << std::endl;
44     show_usage();
45     return 1;
46 }
47
48 if (port < 0 || port > 65535) {
```

```
49     std::cerr << "Invalid_port" << std::endl;  
50     show_usage();  
51     return 1;  
52 }
```

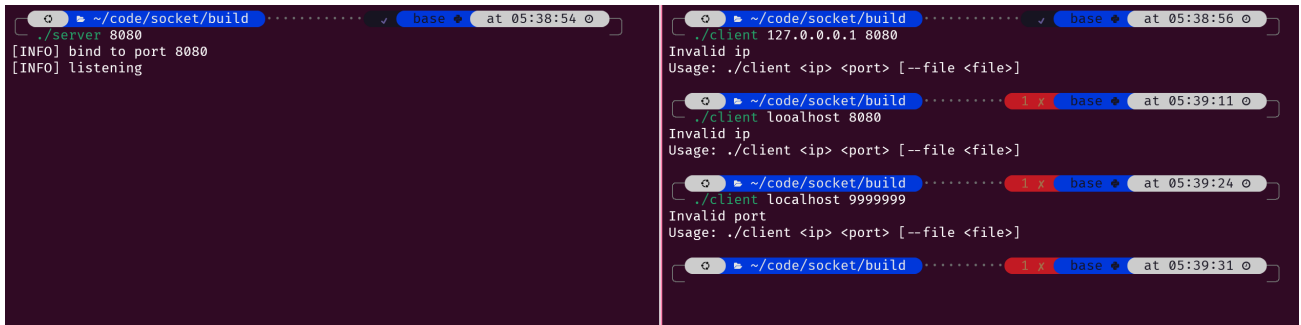


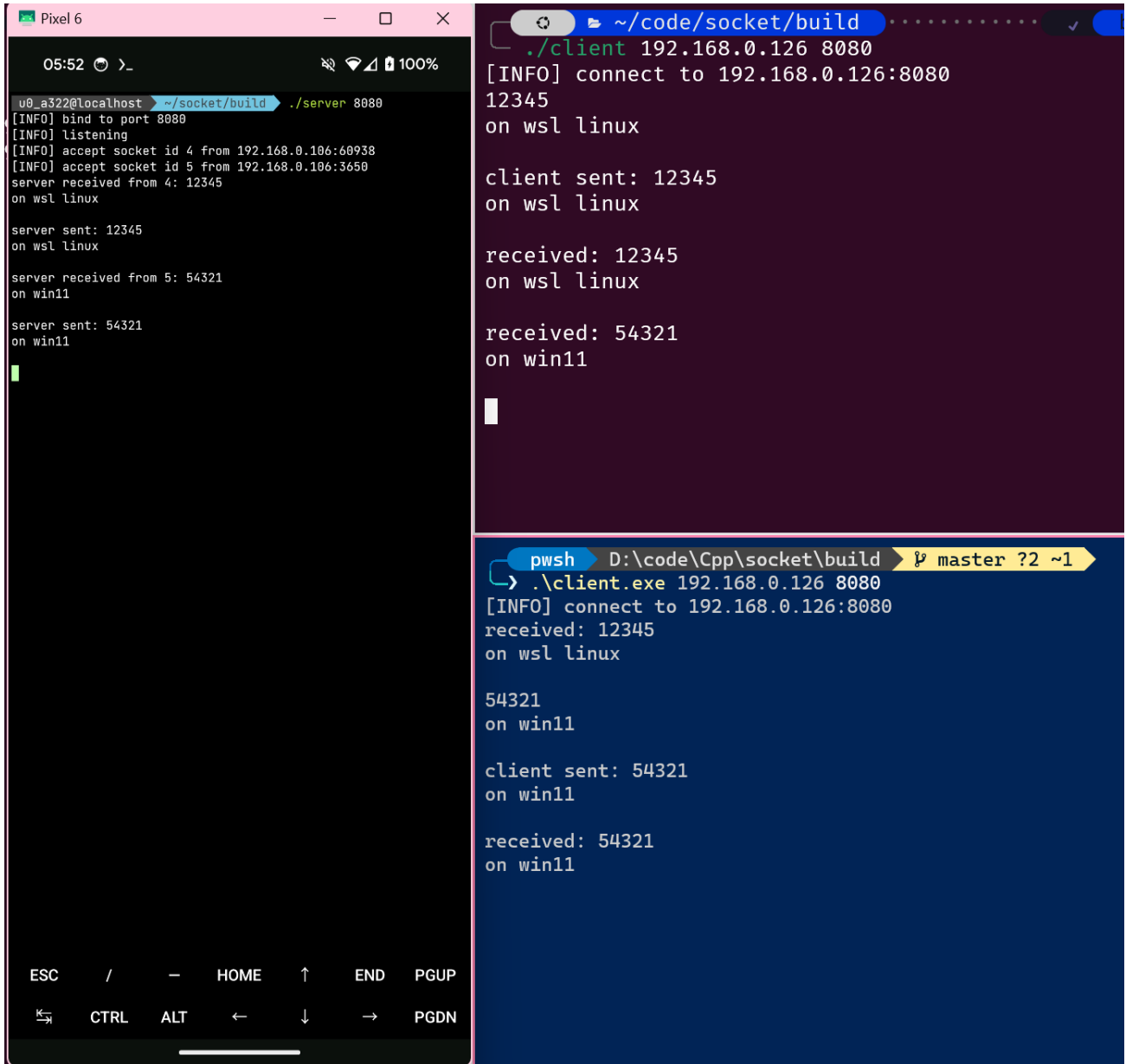
图 6: 连接至错误的 IP 地址/端口号时能提示错误信息

## 6.3 整体

### 6.3.1 支持在 **localhost** 及两台不同机器上运行

在 Windows 下, WSL Linux 虚拟机下和手机上运行:





```
Pixel 6
05:52
u0_a322@localhost ~/socket/build ./server 8080
[INFO] bind to port 8080
[INFO] listening
[INFO] accept socket id 4 from 192.168.0.106:60938
[INFO] accept socket id 5 from 192.168.0.106:3650
server received from 4: 12345
on wsl linux

server sent: 12345
on wsl linux

server received from 5: 54321
on win11

server sent: 54321
on win11

~/code/socket/build
./client 192.168.0.126 8080
[INFO] connect to 192.168.0.126:8080
12345
on wsl linux

client sent: 12345
on wsl linux

received: 12345
on wsl linux

received: 54321
on win11

D:\code\Cpp\socket\build
.\client.exe 192.168.0.126 8080
[INFO] connect to 192.168.0.126:8080
received: 12345
on wsl linux

54321
on win11

client sent: 54321
on win11

received: 54321
on win11
```

图 7: 支持在 localhost 及两台不同机器上运行

### 6.3.2 支持长文本消息（不少于 20KB）

如下，发送 579KB 的文件，可以正常工作。

```

A. caused a heated discussion B. spread widely
C. made a requirement D. proved the rule
28. Why did Tsinghua University once give up the rule?
  ① Because many students failed the swimming test.
  ② Because some people disagreed with the rule.
  ③ Because there were too many students.
  ④ Because there were not enough swimming pools.
A. ① B. ② C. ③ D. ④
29. What is the passage mainly about?
A. Different people hold different opinions about a new rule.
B. Tsinghua University has made a new rule of graduation.
C. Chinese students' health should be more and more valued.
D. There is a new chance to learn another life-saving ability.

server sent: g to the passage, what does it take for a student to graduate from Tsinghua university?
A. Some papers. B. High scores in exams.
C. Passing a swimming test. D. All of the above.
27. The underlined phrase "made waves" in Paragraph 3 most probably means _____.
A. caused a heated discussion B. spread widely
C. made a requirement D. proved the rule
28. Why did Tsinghua University once give up the rule?
  ① Because many students failed the swimming test.
  ② Because some people disagreed with the rule.
  ③ Because there were too many students.
  ④ Because there were not enough swimming pools.
A. ① B. ② C. ③ D. ④
29. What is the passage mainly about?
A. Different people hold different opinions about a new rule.
B. Tsinghua University has made a new rule of graduation.
C. Chinese students' health should be more and more valued.
D. There is a new chance to learn another life-saving ability.

server sent: exit
[INFO] client 5 disconnected

```

```

26. Accordi
received: g to the passage, what does it take for a student to graduate from Tsinghua university?
A. Some papers. B. High scores in exams.
C. Passing a swimming test. D. All of the above.
27. The underlined phrase "made waves" in Paragraph 3 most probably means _____.
A. caused a heated discussion B. spread widely
C. made a requirement D. proved the rule
28. Why did Tsinghua University once give up the rule?
  ① Because many students failed the swimming test.
  ② Because some people disagreed with the rule.
  ③ Because there were too many students.
  ④ Because there were not enough swimming pools.
A. ① B. ② C. ③ D. ④
29. What is the passage mainly about?
A. Different people hold different opinions about a new rule.
B. Tsinghua University has made a new rule of graduation.
C. Chinese students' health should be more and more valued.
D. There is a new chance to learn another life-saving ability.

exit

client sent: exit

[INFO] server disconnected
[INFO] disconnected
[INFO] socket closed

```

```

total 752K
-rwxr-xr-x 1 pdli pdli 66K Jan 12 03:00 client
-rw-r--r-- 1 pdli pdli 579K Jan 12 06:01 longtext.txt
-rwxr-xr-x 1 pdli pdli 99K Jan 12 03:00 server

```

图 8: 支持长文本消息（不少于 20KB）

## 6.4 容错性好，无闪退

可以正常工作，无闪退，容错性好。

## 6.5 加分项

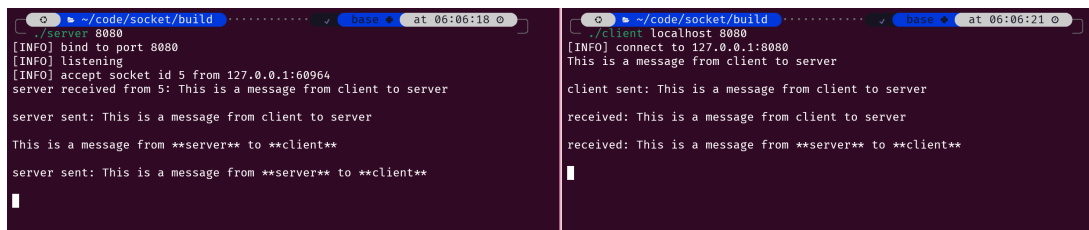
### 6.5.1 支持双工通信

在 server 段开启一个线程发送消息即可。

```

1 void input(socket_server &socket) {
2     while (true) {
3         std::string msg{};
4         std::getline(std::cin, msg);
5         socket.send_all(msg);
6         if (msg == "exit") {
7             socket.send_all("");
8             socket.close();
9             exit(0);
10        }
11    }
12 }

```



```
[~/code/socket/build] [base] at 06:06:18
[INFO] bind to port 8080
[INFO] listening
[INFO] accept socket id 5 from 127.0.0.1:60964
server received from 5: This is a message from client to server
server sent: This is a message from client to server
This is a message from **server** to **client**
server sent: This is a message from **server** to **client**

[~/code/socket/build] [base] at 06:06:21
[client] connect to 127.0.0.1:8080
This is a message from client to server
client sent: This is a message from client to server
received: This is a message from client to server
received: This is a message from **server** to **client**
```

图 9: 支持双工通信

## 7 实验结果总结

在本次基于 **Socket** 编程的实验中,我深入探索了网络通信编程的世界,成功实现了 Client 和 Server 之间的通信,并达成了多项实验要求,收获颇丰。

在实验过程中,我首先熟练掌握了 **Socket** 编程的核心接口。通过 `socket()` 函数创建套接字,清晰地理解了 `AF_INET` 与 `SOCK_STREAM` 参数的含义及其在 IPv4 和 TCP 协议下的作用。在绑定端口号的 `bind()` 函数实现中,学会了对端口号的有效性进行严谨检查,以及如何处理绑定失败的情况,这对于确保服务器正确运行在指定端口至关重要。

`listen()` 函数让我明白了如何设置服务器的监听状态以及最大连接数,而 `accept()` 函数则实现了服务器对客户端连接请求的接收与处理。在客户端的 `connect()` 函数中,掌握了如何根据指定的 IP 地址和端口号与服务器建立连接,并能准确地对错误的 IP 地址和端口号进行检测与提示,增强了程序的健壮性。

数据的发送与接收通过 `send()` 和 `recv()` 函数得以实现,我不仅学会了基本的数据传输操作,还考虑到了长文本消息的传输需求,通过合理的缓冲区管理,成功实现了对不少于 20KB 长文本消息的支持,确保了大数据量传输的稳定性和可靠性。

为了满足多个客户端同时与服务器通信的要求,我运用多线程技术,为每个客户端连接创建独立的线程来处理,实现了服务器对多个客户端消息的并发处理与逐一打印。这不仅提升了程序的并发性能,也让我对多线程编程在网络通信中的应用有了更深入的理解。

在实验的整体测试中,程序表现出了良好的兼容性,能够在 `localhost` 以及不同机器上稳定运行,无论是在 Windows 系统、Linux 系统还是通过手机进行测试,都能正确地建立连接并进行通信。同时,程序的容错性也得到了有效验证,在面对各种异常情况,如错误的端口绑定、错误的连接信息以及客户端与服务器之间的异常断开等,都能妥善处理,不会出现闪退等不稳定现象。

此外,我还成功实现了加分项中的双工通信功能。通过在服务器端开启额外的线程用于接收用户输入并向客户端发送消息,使得客户端与服务器之间能够实现双向的信息交互,进一步丰富了网络通信程序的功能。

通过本次实验,我不仅掌握了 **Socket** 编程的技术要点,还在实践中提升了自己解决问题的能力、对网络通信原理的理解以及多线程编程的应用水平。这些收获将为我今后从事网络相关的编程开发工作奠定坚实的基础,也让我对网络编程领域充满了更浓厚的兴趣和探索欲望。

## 8 附录

### 8.1 代码结构

```
socket
|-- Makefile
|-- build
|   |-- client
|   |-- longtext.txt
|   `-- server
`-- src
    |-- client.cpp
    |-- include
    |   |-- socket.hpp
    |   |-- socket_client.hpp
    |   `-- socket_server.hpp
    `-- server.cpp

3 directories, 9 files
```

### 8.2 源代码

#### 8.2.1 Makefile

```
1 CXX = g++
2 ifeq ($(OS), Windows_NT)
3     CXXFLAGS = -std=c++17 -lwsck32
4 else
5     CXXFLAGS = -std=c++17
6 endif
7 BUILD_DIR = build
8
9 all: server client
10
11 server: src/server.cpp
12     $(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)
13
14 client: src/client.cpp
15     $(CXX) $^ -o $(BUILD_DIR)/$@ $(CXXFLAGS)
16
17 clean:
18 ifeq ($(OS), Windows_NT)
19     del /Q $(BUILD_DIR)\*
```

```
20 else
21     rm -rf $(BUILD_DIR)/*
22 endif
```

### 8.2.2 socket.hpp

```
1  #pragma once
2
3  #ifndef _SIMPLE_SOCKET_H_
4  #define _SIMPLE_SOCKET_H_
5
6  #include <iostream>
7  #include <stdexcept>
8  #include <string>
9  #include <string_view>
10 #include <unistd.h>
11
12 #ifdef _WIN32
13 #include <winsock2.h>
14 #include <ws2tcpip.h>
15 #else
16 #include <arpa/inet.h>
17 #include <sys/socket.h>
18 #endif
19
20 class socket {
21 public:
22     socket() : m_descriptor(-1) {
23 #ifdef _WIN32
24         WSADATA wsa_data;
25         if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0) {
26             throw std::runtime_error("WSAStartup_failed");
27         }
28 #endif
29         m_descriptor = ::socket(AF_INET, SOCK_STREAM, 0);
30         if (m_descriptor < 0) {
31             throw std::runtime_error("create_socket_failed");
32         }
33     }
34
35     virtual ~socket() noexcept { m_close_socket(); }
36
37     void bind(int port) {
```

```
38     if (port < 0 || port > 65535) {
39         m_close_socket();
40         throw std::runtime_error("invalid_port");
41     }
42
43     sockaddr_in server_addr;
44     server_addr.sin_family = AF_INET;
45     server_addr.sin_addr.s_addr = INADDR_ANY;
46     server_addr.sin_port = htons(port);
47
48     if (::bind(m_descriptor, (sockaddr *)&server_addr,
49             sizeof(server_addr)) == -1) {
50         m_close_socket();
51         throw std::runtime_error(
52             "bind_to_port_" + std::to_string(port) +
53             "_failed\nmaybe_the_port_is_already_in_use");
54     } else {
55         m_info("bind_to_port_" + std::to_string(port));
56     }
57 }
58
59 void listen(int n = 10) {
60     if (::listen(m_descriptor, n) == -1) {
61         m_close_socket();
62         throw std::runtime_error("listen_socket_failed");
63     } else {
64         m_info("listening");
65     }
66 }
67
68 int accept() {
69     sockaddr_in client_addr;
70     socklen_t client_addr_len = sizeof(client_addr);
71     int connected =
72         ::accept(m_descriptor, (sockaddr *)&client_addr, &client_addr_len);
73     if (connected == -1) {
74         m_close_socket();
75         throw std::runtime_error("accept_socket_failed");
76     } else {
77         m_info("accept_socket_id_" + std::to_string(connected) + "_from_" +
78             std::string(inet_ntoa(client_addr.sin_addr)) + ":" +
79             std::to_string(ntohs(client_addr.sin_port)));
80     }
81     return connected;
```

```
82     }
83
84     int connect(const char *ip, int port) {
85         sockaddr_in server_addr;
86         server_addr.sin_family = AF_INET;
87         server_addr.sin_addr.s_addr = inet_addr(ip);
88         server_addr.sin_port = htons(port);
89
90         int connected = ::socket(AF_INET, SOCK_STREAM, 0);
91
92         if (::connect(connected, (sockaddr *)&server_addr,
93                     sizeof(server_addr)) == -1) {
94             m_close_socket();
95             throw std::runtime_error("connect_to_ " + std::string(ip) + ":" +
96                                     std::to_string(port) + "_failed");
97         } else {
98             m_info("connect_to_ " + std::string(ip) + ":" +
99                  std::to_string(port));
100        }
101        return connected;
102    }
103
104    int connect(std::string ip, int port) { return connect(ip.c_str(), port); }
105
106    int connect(std::string_view ip, int port) {
107        return connect(ip.data(), port);
108    }
109
110    void send(int to, const char *data, int len) {
111        if (::send(to, data, len, 0) == -1) {
112            m_error("send_data_to_ " + std::to_string(to) + "_failed");
113        }
114    }
115
116    void send(int to, std::string_view data) {
117        send(to, data.data(), data.size());
118    }
119
120    bool recv(int from, char *data, int len) {
121        int read_num = ::recv(from, data, len, 0);
122        if (read_num == -1) {
123            m_error("recv_data_from_ " + std::to_string(from) + "_failed");
124            return false;
125        }
126    }
```

```
126     if (read_num == 0) {
127         return false;
128     }
129     return true;
130 }
131
132 void close() { m_close_socket(); }
133
134 private:
135     int m_descriptor;
136
137     void m_close_socket() {
138         if (m_descriptor != -1) {
139             m_info("socket_closed");
140             ::close(m_descriptor);
141             m_descriptor = -1;
142         }
143     }
144
145     void m_info(const std::string &msg) {
146         std::cout << "[INFO]" << msg << std::endl;
147     }
148
149     void m_error(const std::string &msg) {
150         std::cerr << "[ERROR]" << msg << std::endl;
151     }
152 };
153
154 #endif // _SIMPLE_SOCKET_H_
```

### 8.2.3 socket\_client.hpp

```
1  #pragma once
2
3  #ifndef _CLIENT_H_
4  #define _CLIENT_H_
5
6  #include "socket.hpp"
7  #include <cstring>
8  #include <iostream>
9  #include <unistd.h>
10
11  class socket_client : socket {
```



```
12 public:
13     socket_client()
14         : socket(), m_server_descriptor(-1), m_last_send_enter(false) {}
15
16     void connect(std::string_view ip, int port) {
17         m_server_descriptor = socket::connect(ip, port);
18     }
19
20     void send(std::string_view msg) {
21         if (msg.empty()) {
22             if (m_last_send_enter) {
23                 socket::send(m_server_descriptor,
24                             m_last_send + std::string{msg});
25                 std::cout << "client_send:_" << m_last_send + std::string{msg}
26                             << std::endl;
27                 if (msg == "exit") {
28                     close();
29                 }
30                 m_last_send_enter = false;
31                 m_last_send = "";
32                 return;
33             }
34             m_last_send_enter = true;
35             m_last_send = "\n";
36             return;
37         }
38         m_last_send.append(msg).append("\n");
39         m_last_send_enter = true;
40     }
41
42     std::string recv(int size = 1024) {
43         char* buffer = new char[size];
44         std::memset(buffer, 0, size);
45         bool ok = socket::recv(m_server_descriptor, buffer, size);
46         auto res = std::string(buffer);
47         delete[] buffer;
48         if (!ok || res == "exit\n") {
49             res = "";
50             std::cout << "[INFO]_server_disconnected" << std::endl;
51             close();
52         }
53         return res;
54     }
```

```
55     void close() {
56         if (m_server_descriptor != -1) {
57             std::cout << "[INFO]_disconnected" << std::endl;
58             ::close(m_server_descriptor);
59             m_server_descriptor = -1;
60         }
61         socket::close();
62     }
63
64     private:
65         int m_server_descriptor;
66         bool m_last_send_enter;
67         std::string m_last_send{};
68 };
69
70 #endif // _CLIENT_H_
```

#### 8.2.4 socket\_server.hpp

```
1  #pragma once
2
3  #ifndef _SERVER_H_
4  #define _SERVER_H_
5
6  #include "socket.hpp"
7  #include <cstring>
8  #include <iostream>
9  #include <vector>
10
11 class socket_server : socket {
12     public:
13         socket_server() : socket() {}
14
15         void bind(int port) { socket::bind(port); }
16
17         void listen(int n = 10) { socket::listen(n); }
18
19         int accept() {
20             int connected = socket::accept();
21             m_connections.push_back(connected);
22             return connected;
23         }
24 }
```

```
25 void send_all(std::string_view msg) {
26     if (msg.empty()) {
27         if (m_last_send_enter) {
28             m_send_all(m_last_send + std::string{msg});
29             std::cout << "server_send:_" << m_last_send + std::string{msg}
30                 << std::endl;
31             if (msg == "exit") {
32                 close();
33             }
34             m_last_send_enter = false;
35             m_last_send = "";
36             return;
37         }
38         m_last_send_enter = true;
39         m_last_send = "\n";
40         return;
41     }
42     m_last_send.append(msg).append("\n");
43     m_last_send_enter = true;
44 }
45
46 void send(int to, std::string_view msg) {
47     if (msg.empty()) {
48         if (m_last_send_enter) {
49             socket::send(to, m_last_send + std::string{msg});
50             if (msg == "exit") {
51                 m_close(to);
52                 return;
53             }
54             std::cout << "server_send:_" << m_last_send + std::string{msg}
55                 << std::endl;
56             m_last_send_enter = false;
57             m_last_send = "";
58             return;
59         }
60         m_last_send_enter = true;
61         m_last_send = "\n";
62         return;
63     }
64     m_last_send.append(msg).append("\n");
65     m_last_send_enter = true;
66 }
67
68 const std::string recv(int from, int size = 1024) {
```

```
69     char *buffer = new char[size];
70     std::memset(buffer, 0, size);
71     bool ok = socket::recv(from, buffer, size);
72     auto res = std::string(buffer);
73     delete[] buffer;
74     if (!ok || res == "exit\n") {
75         send(from, "exit");
76         send(from, "");
77         std::cout << "[INFO]_client_" << from << "_disconnected"
78             << std::endl;
79         m_close(from);
80         res = "";
81     }
82     return res;
83 }
84
85 void close() { socket::close(); }
86
87 private:
88     std::vector<int> m_connections{};
89
90     bool m_last_send_enter{false};
91
92     std::string m_last_send{};
93
94     void m_close(int client) {
95         for (auto it = m_connections.begin(); it != m_connections.end(); ++it) {
96             if (*it == client) {
97                 m_connections.erase(it);
98                 break;
99             }
100         }
101     }
102
103     void m_send_all(std::string_view msg) {
104         for (auto &client : m_connections) {
105             socket::send(client, msg);
106         }
107     }
108 };
109
110 #endif // _SERVER_H_
```

## 8.2.5 client.cpp

```
1  #include "include/socket_client.hpp"
2  #include <exception>
3  #include <fstream>
4  #include <iostream>
5  #include <string>
6  #include <thread>
7
8  bool check_ipv4(std::string &addr) {
9      if (addr.empty() || addr.back() == '.' || addr.front() == '.') {
10         return false;
11     }
12     if (addr == "localhost") {
13         addr = "127.0.0.1";
14     }
15     int num{0};
16     int dot{0};
17     for (auto &c : addr) {
18         if (c == '.') {
19             if (num < 0 || num > 255) {
20                 return false;
21             }
22             num = 0;
23             ++dot;
24             continue;
25         }
26         if (c < '0' || c > '9') {
27             return false;
28         }
29         num = num * 10 + (c - '0');
30     }
31     if (num < 0 || num > 255 || dot != 3) {
32         return false;
33     }
34     return true;
35 }
36
37 void receive(socket_client &socket) {
38     while (true) {
39         auto msg = socket.recv();
40         if (msg.empty()) {
41             socket.close();
42             exit(0);
```

```
43     }
44     std::cout << "received:_" << msg << std::endl;
45 }
46 }
47
48 int main(const int argc, const char **argv) {
49     const auto show_usage = [argv]() {
50         std::cerr << "Usage:_" << argv[0] << "_<ip>_<port>_ [--file_<file>]"
51             << std::endl;
52     };
53
54     if ((argc != 3 && argc != 5) ||
55         (argc == 5 && std::string{argv[3]} != "--file")) {
56         std::cerr << "Invalid_arguments" << std::endl;
57         show_usage();
58         return 1;
59     }
60
61     std::string ip{argv[1]};
62     if (!check_ipv4(ip)) {
63         std::cerr << "Invalid_ip" << std::endl;
64         show_usage();
65         return 1;
66     }
67
68     int port{-1};
69     try {
70         port = std::stoi(argv[2]);
71     } catch (std::exception &e) {
72         std::cerr << "Invalid_port" << std::endl;
73         show_usage();
74         return 1;
75     }
76
77     if (port < 0 || port > 65535) {
78         std::cerr << "Invalid_port" << std::endl;
79         show_usage();
80         return 1;
81     }
82
83     socket_client client{};
84     client.connect(ip, port);
85
86     std::thread recv_t(receive, std::ref(client));
```

```
87
88     if (argc == 5) {
89         std::fstream in{argv[4]};
90         std::string str{};
91         std::string tmp{};
92         while (std::getline(in, tmp)) {
93             str.append(tmp).append("\n");
94         }
95         client.send(str);
96         client.send("");
97     }
98
99     while (true) {
100         std::string msg;
101         std::getline(std::cin, msg);
102         client.send(msg);
103     }
104
105     recv_t.join();
106 }
```

#### 8.2.6 server.cpp

```
1  #include "include/socket_server.hpp"
2  #include <functional>
3  #include <iostream>
4  #include <thread>
5
6  void handle_connection(socket_server &socket, int client) {
7      while (true) {
8          auto msg = socket.recv(client);
9          if (msg.empty()) {
10             break;
11         }
12         std::cout << "server_ received_from_" << client << " :_" << msg
13             << std::endl;
14         msg = msg.substr(0, msg.size() - 1);
15         socket.send_all(msg == "\n" ? "" : msg);
16         socket.send_all("");
17     }
18 }
19
20 void input(socket_server &socket) {
```

```
21     while (true) {
22         std::string msg{};
23         std::getline(std::cin, msg);
24         socket.send_all(msg);
25         if (msg == "exit") {
26             socket.send_all("");
27             socket.close();
28             exit(0);
29         }
30     }
31 }
32
33 int main(const int argc, const char **argv) {
34     const auto show_usage = [argv]() {
35         std::cerr << "Usage: _" << argv[0] << "_<port>" << std::endl;
36     };
37
38     if (argc != 2) {
39         show_usage();
40         return 1;
41     }
42
43     int port{-1};
44     try {
45         port = std::stoi(argv[1]);
46     } catch (std::exception &e) {
47         std::cerr << "Invalid _port" << std::endl;
48         show_usage();
49         return 1;
50     }
51
52     if (port < 0 || port > 65535) {
53         std::cerr << "Invalid _port" << std::endl;
54         show_usage();
55         return 1;
56     }
57
58     socket_server server{};
59
60     server.bind(port);
61     server.listen();
62
63     std::vector<std::thread> threads{};
64
```



```
65     std::thread input_t{input, std::ref(server)};
66
67     while (true) {
68         int client = server.accept();
69         threads.emplace_back(handle_connection, std::ref(server), client);
70     }
71
72     for (auto &thread : threads) {
73         thread.join();
74     }
75 }
```