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:	2022 级
实验编号:	Lab 07	实验名称:	socket 编程
姓　　名:	李鹏达	学　　号:	10225101460

1 实验目的

1) 熟悉 socket 编程的基本原理
2) 掌握简单的套接字编程
3) 掌握通过 socket 编程实现 C/S 程序的基本方法
4) 了解应用层和运输层的作用及相关协议的工作原理和机制

2 实验内容与实验步骤

2.1 实验内容

实现 Client 和 Server 的通信，并满足以下要求：

Server			Client			整个系统			Bonus
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 localhost 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

2.2 实验步骤

2.2.1 创建 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

```
    socket() : m_descriptor(-1) {
#ifdef _WIN32
        WSADATA wsa_data;
        if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0) {
            throw std::runtime_error("WSAStartup failed");
        }
#endif
        m_descriptor = ::socket(AF_INET, SOCK_STREAM, 0);
        if (m_descriptor < 0) {
            throw std::runtime_error("create socket failed");
        }
    }
```

其中，AF_INET 表示使用 IPv4 协议，SOCK_STREAM 表示使用 TCP 协议。

2.2.2 bind

使用 bind 函数将套接字与端口号绑定。

bind

```
void bind(int port) {
    if (port < 0 || port > 65535) {
        m_close_socket();
        throw std::runtime_error("invalid port");
    }

    sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);

    if (::bind(m_descriptor, (sockaddr *)&server_addr,
               sizeof(server_addr)) == -1) {
        m_close_socket();
        throw std::runtime_error(
            "bind to port " + std::to_string(port) +
            " failed\n maybe the port is already in use");
    } else {
        m_info("bind to port " + std::to_string(port));
    }
}
```

2.2.3 listen

使用 listen 函数监听端口。

listen

```
void listen(int n = 10) {
    if (::listen(m_descriptor, n) == -1) {
        m_close_socket();
        throw std::runtime_error("listen socket failed");
    } else {
        m_info("listening");
    }
}
```

其中，n 表示最大连接数。

2.2.4 accept

使用 accept 函数接受客户端的连接。

accept

```
int accept() {
    sockaddr_in client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    int connected =
        ::accept(m_descriptor, (sockaddr *)&client_addr, &client_addr_len);
    if (connected == -1) {
        m_close_socket();
        throw std::runtime_error("accept socket failed");
    } else {
        m_info("accept socket id " + std::to_string(connected) + " from " +
              std::string(inet_ntoa(client_addr.sin_addr)) + ":" +
              std::to_string(ntohs(client_addr.sin_port)));
    }
    return connected;
}
```

2.2.5 connect

使用 connect 函数主动与服务器建立连接。

connect

```
int connect(const char *ip, int port) {
```

```
    sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    server_addr.sin_port = htons(port);

    int connected = ::socket(AF_INET, SOCK_STREAM, 0);

    if (::connect(connected, (sockaddr *)&server_addr,
                  sizeof(server_addr)) == -1) {
        m_close_socket();
        throw std::runtime_error("connect to " + std::string(ip) + ":" +
                                 std::to_string(port) + " failed");
    } else {
        m_info("connect to " + std::string(ip) + ":" +
               std::to_string(port));
    }
    return connected;
}

int connect(std::string_view ip, int port) {
    return connect(ip.data(), port);
}
```

2.2.6 send

使用 send 函数发送数据。

send

```
void send(int to, const char *data, int len) {
    if (::send(to, data, len, 0) == -1) {
        m_error("send data to " + std::to_string(to) + " failed");
    }
}

void send(int to, std::string_view data) {
    send(to, data.data(), data.size());
}
```

2.2.7 recv

使用 recv 函数接收数据。

recv

```
bool recv(int from, char *data, int len) {
    int read_num = ::recv(from, data, len, 0);
    if (read_num == -1) {
        m_error("recv data from " + std::to_string(from) + " failed");
        return false;
    }
    if (read_num == 0) {
        return false;
    }
    return true;
}
```

2.2.8 close

使用 close 函数关闭套接字。

close

```
void m_close_socket() {
    if (m_descriptor != -1) {
        m_info("socket closed");
        ::close(m_descriptor);
        m_descriptor = -1;
    }
}

// ...

~socket() {
    m_close_socket();
}

void close() {
    m_close_socket();
}
```

3 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

4 实验结果与分析

4.1 Server

4.1.1 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。

```
void handle_connection(socket_server &socket, int client) {
    while (true) {
        auto msg = socket.recv(client);
        if (msg.empty()) {
            break;
        }
        std::cout << "server received from " << client << ": " << msg
                  << std::endl;
        msg = msg.substr(0, msg.size() - 1);
        socket.send_all(msg == "\n" ? "" : msg);
        socket.send_all("");
    }
}
```

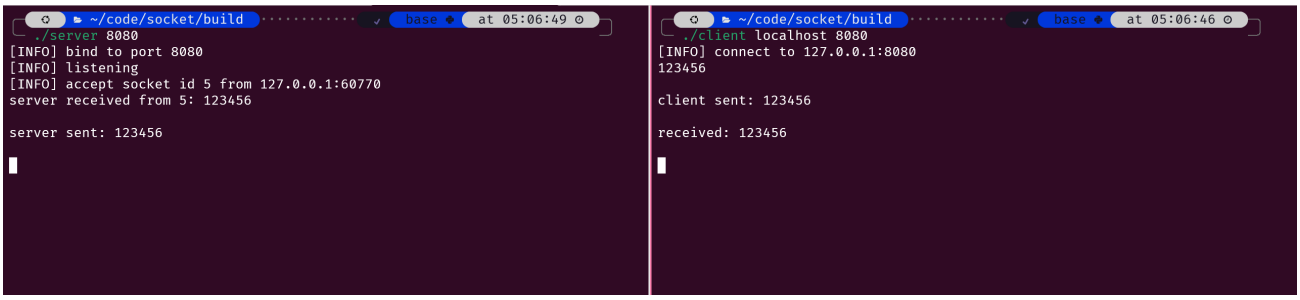

图 1: 能在标准输出打印客户端发送的消息

4.1.2 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。

```
socket_server server{};
```

```
server.bind(port);
server.listen();

std::vector<std::thread> threads{};

while (true) {
    int client = server.accept();
    threads.emplace_back(handle_connection, std::ref(server), client);
}

for (auto &thread : threads) {
    thread.join();
}
```

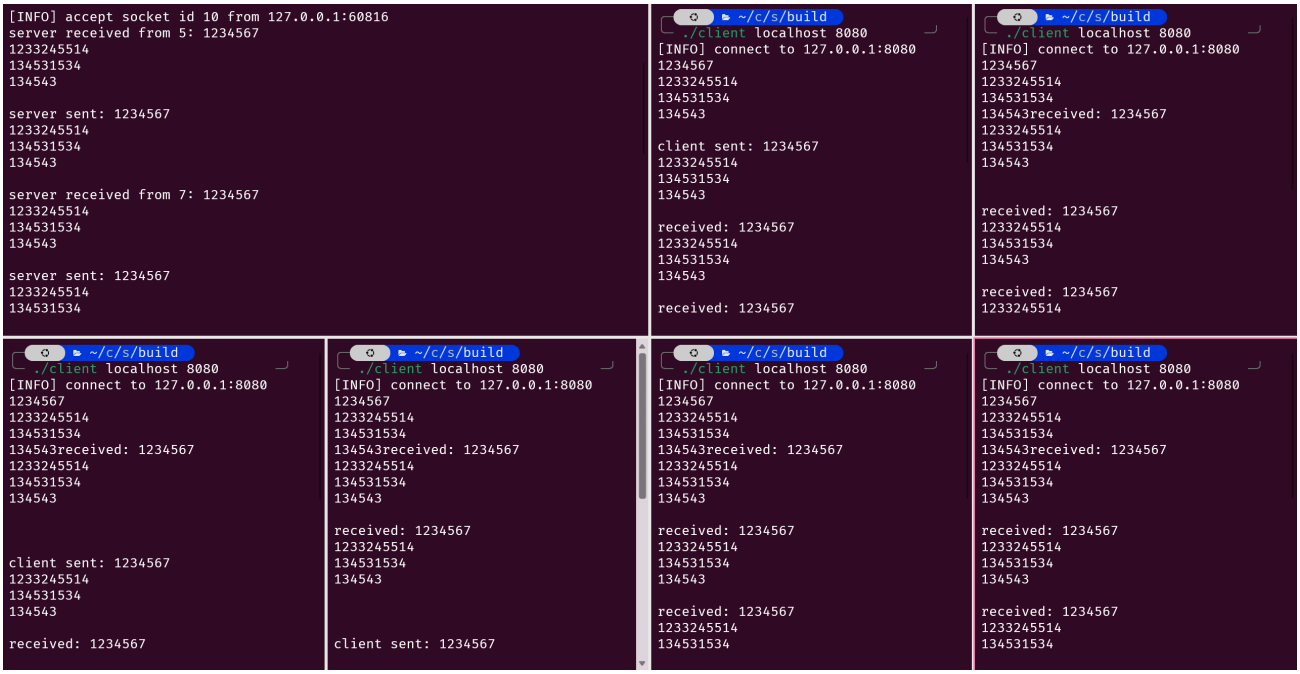

图 2: 支持 5 个以上客户端同时发送消息并逐一打印

4.1.3 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。

```
void bind(int port) {
    if (port < 0 || port > 65535) {
        m_close_socket();
        throw std::runtime_error("invalid port");
```

```
    }

    sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);

    if (::bind(m_descriptor, (sockaddr *)&server_addr,
               sizeof(server_addr)) == -1) {
        m_close_socket();
        throw std::runtime_error(
            "bind to port " + std::to_string(port) +
            " failed\n maybe the port is already in use");
    } else {
        m_info("bind to port " + std::to_string(port));
    }
}

// ...

int port{-1};
try {
    port = std::stoi(argv[1]);
} catch (std::exception &e) {
    std::cerr << "Invalid port" << std::endl;
    show_usage();
    return 1;
}

if (port < 0 || port > 65535) {
    std::cerr << "Invalid port" << std::endl;
    show_usage();
    return 1;
}
```

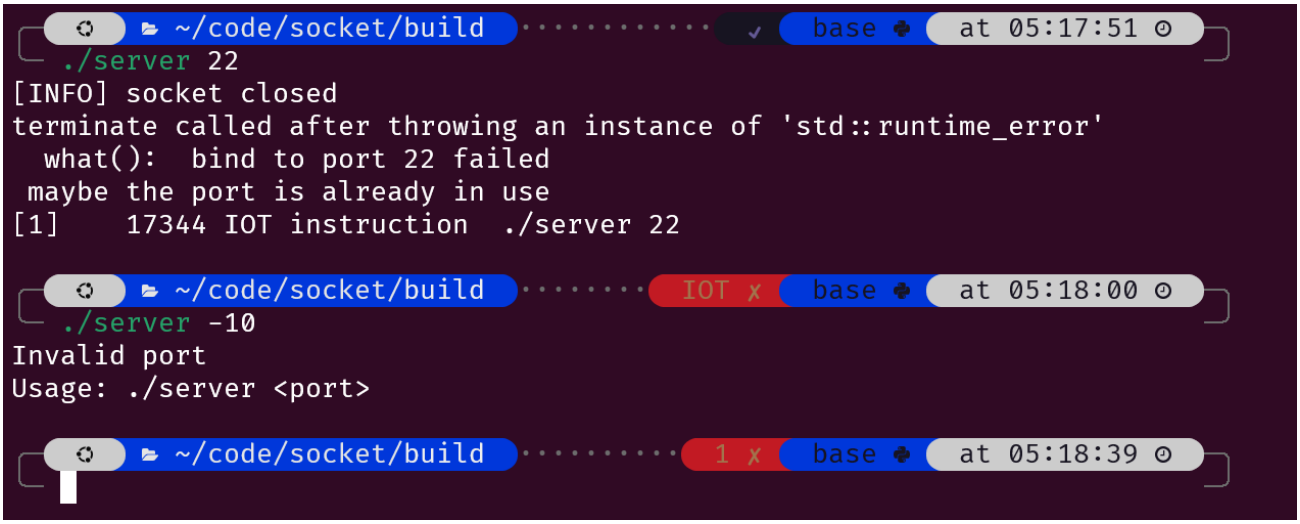

图 3: 绑定至错误的端口号时提示出错信息

4.2 Client

4.2.1 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。

```
while (true) {
    std::string msg;
    std::getline(std::cin, msg);
    client.send(msg);
}
```

使用 `std::fstream` 从文件流读取信息。

```
if (argc == 5) {
    std::fstream in{argv[4]};
    std::string str{};
    std::string tmp{};
    while (std::getline(in, tmp)) {
        str.append(tmp).append("\n");
    }
    client.send(str);
    client.send("");
}
```

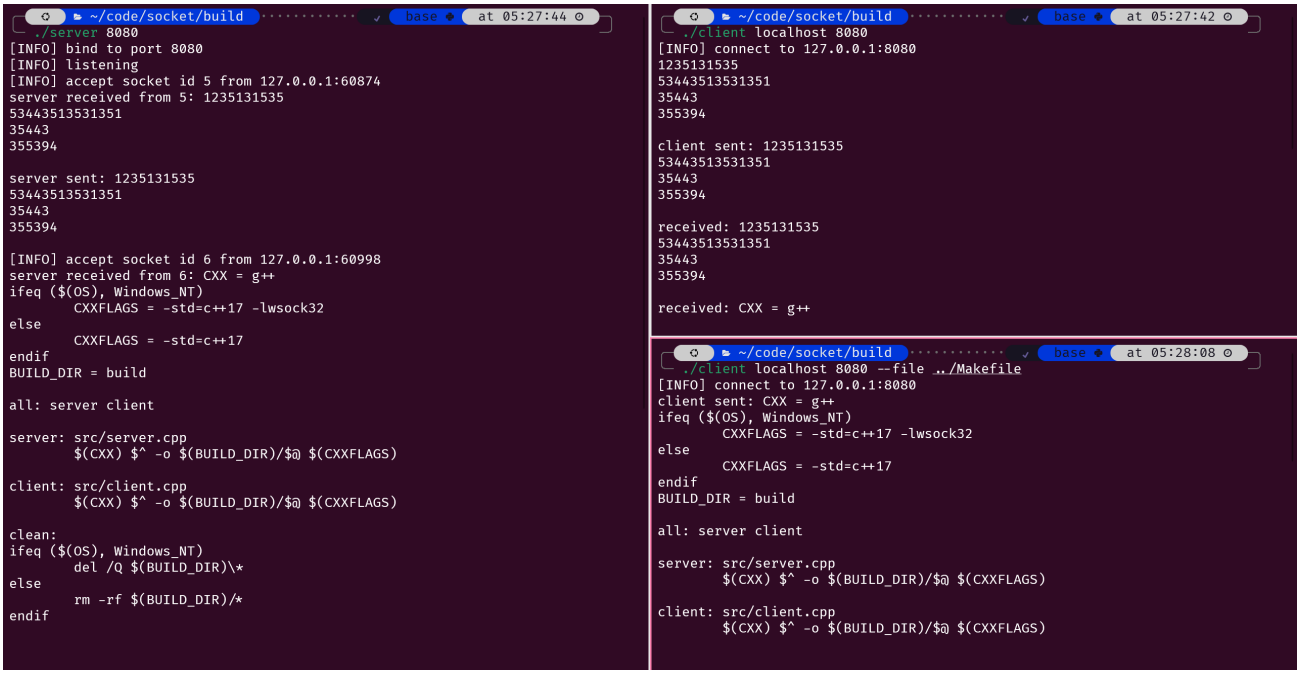

图 4: 能从标准输入或文件接收信息

4.2.2 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。

```
void send(std::string_view msg) {
    if (msg.empty()) {
        if (m_last_send_enter) {
            socket::send(m_server_descriptor,
                         m_last_send + std::string{msg});
            std::cout << "client sent: " << m_last_send + std::string{msg} <<
                std::endl;
            if (msg == "exit") {
                close();
            }
            m_last_send_enter = false;
            m_last_send = "";
            return;
        }
        m_last_send_enter = true;
        m_last_send = "\n";
        return;
    }
    m_last_send.append(msg).append("\n");
    m_last_send_enter = true;
```

```
}
```

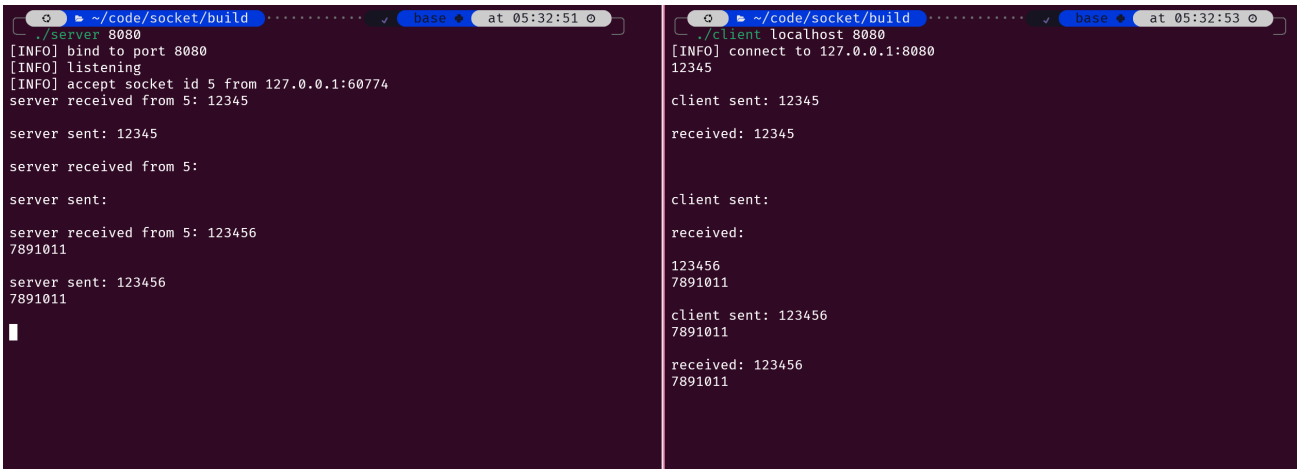

图 5: 标准输入信息以两次回车作为结束标志

4.2.3 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。

```cpp
bool check_ipv4(std::string &addr) {
    if (addr.empty() || addr.back() == '.' || addr.front() == '.') {
        return false;
    }
    if (addr == "localhost") {
        addr = "127.0.0.1";
    }
    int num{0};
    int dot{0};
    for (auto &c : addr) {
        if (c == '.') {
            if (num < 0 || num > 255) {
                return false;
            }
            num = 0;
            ++dot;
            continue;
        }
        if (c < '0' || c > '9') {
            return false;
        }
        num = num * 10 + (c - '0');
```

```
    }
    if (num < 0 || num > 255 || dot != 3) {
        return false;
    }
    return true;
}

// ...

std::string ip{argv[1]};
if (!check_ipv4(ip)) {
    std::cerr << "Invalid ip" << std::endl;
    show_usage();
    return 1;
}

int port{-1};
try {
    port = std::stoi(argv[2]);
} catch (std::exception &e) {
    std::cerr << "Invalid port" << std::endl;
    show_usage();
    return 1;
}

if (port < 0 || port > 65535) {
    std::cerr << "Invalid port" << std::endl;
    show_usage();
    return 1;
}
```

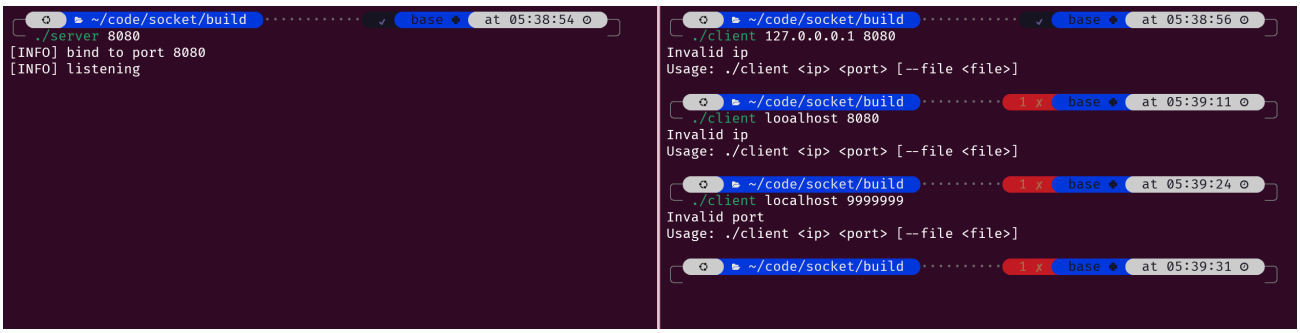

图 6: 连接至错误的 IP 地址/端口号时能提示错误信息

4.3 整个系统

4.3.1 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：

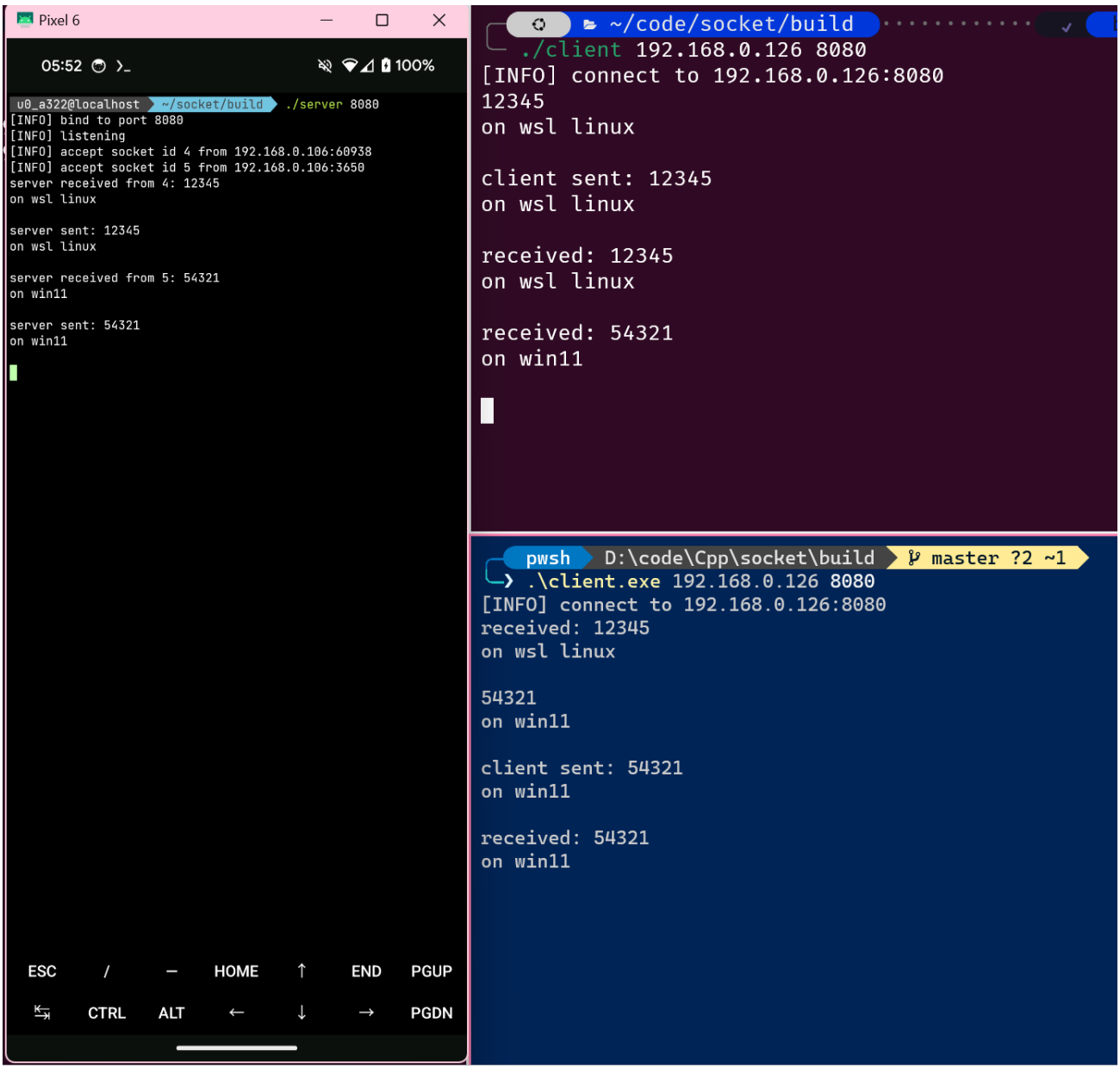

图 7: 支持在 localhost 及两台不同机器上运行

4.3.2 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。

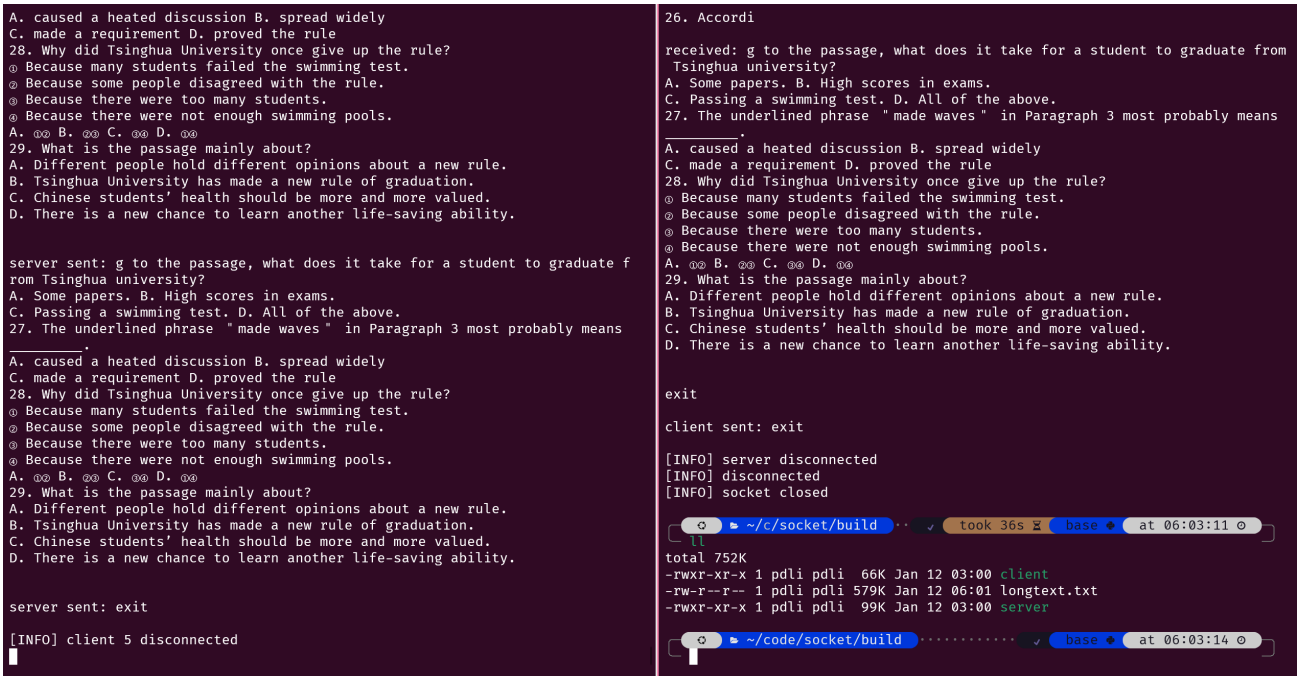

图 8: 支持长文本消息（不少于 20KB）

4.4 容错性好，无闪退

可以正常工作，无闪退，容错性好。

4.5 Bonus

4.5.1 支持双工通信

在 server 段开启一个线程发送消息即可。

```
void input(socket_server &socket) {
    while (true) {
        std::string msg{};
        std::getline(std::cin, msg);
        socket.send_all(msg);
        if (msg == "exit") {
            socket.send_all("");
            socket.close();
            exit(0);
        }
    }
}
```

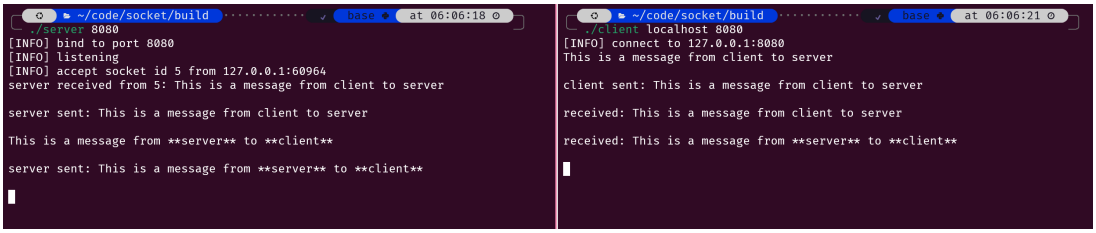

图 9: 支持双工通信

5 实验结果总结

在本次实验中，我掌握了 socket 编程的基本要领。

6 附录

6.1 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
```

6.2 源代码

6.2.1 Makefile

```
CXX = g++
ifeq ($(OS), Windows_NT)
    CXXFLAGS = -std=c++17 -lwsock32
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
```

6.2.2 socket.hpp

```
#pragma once

#ifndef _SIMPLE_SOCKET_H_
#define _SIMPLE_SOCKET_H_

#include <iostream>
#include <stdexcept>
#include <string>
#include <string_view>
#include <unistd.h>

#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <arpa/inet.h>
#include <sys/socket.h>
#endif

class socket {
  public:
    socket() : m_descriptor(-1) {
```

```
#ifdef _WIN32
        WSADATA wsa_data;
        if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0) {
            throw std::runtime_error("WSAStartup failed");
        }
#endif
        m_descriptor = ::socket(AF_INET, SOCK_STREAM, 0);
        if (m_descriptor < 0) {
            throw std::runtime_error("create socket failed");
        }
    }

    virtual ~socket() noexcept { m_close_socket(); }

    void bind(int port) {
        if (port < 0 || port > 65535) {
            m_close_socket();
            throw std::runtime_error("invalid port");
        }

        sockaddr_in server_addr;
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = INADDR_ANY;
        server_addr.sin_port = htons(port);

        if (::bind(m_descriptor, (sockaddr *)&server_addr,
                   sizeof(server_addr)) == -1) {
            m_close_socket();
            throw std::runtime_error(
                "bind to port " + std::to_string(port) +
                " failed\n maybe the port is already in use");
        } else {
            m_info("bind to port " + std::to_string(port));
        }
    }

    void listen(int n = 10) {
        if (::listen(m_descriptor, n) == -1) {
            m_close_socket();
            throw std::runtime_error("listen socket failed");
        } else {
            m_info("listening");
        }
    }
```

```

    int accept() {
        sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        int connected =
            ::accept(m_descriptor, (sockaddr *)&client_addr, &client_addr_len);
        if (connected == -1) {
            m_close_socket();
            throw std::runtime_error("accept socket failed");
        } else {
            m_info("accept socket id " + std::to_string(connected) + " from " +
                   std::string(inet_ntoa(client_addr.sin_addr)) + ":" +
                   std::to_string(ntohs(client_addr.sin_port)));
        }
        return connected;
    }

    int connect(const char *ip, int port) {
        sockaddr_in server_addr;
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr(ip);
        server_addr.sin_port = htons(port);

        int connected = ::socket(AF_INET, SOCK_STREAM, 0);

        if (::connect(connected, (sockaddr *)&server_addr,
                      sizeof(server_addr)) == -1) {
            m_close_socket();
            throw std::runtime_error("connect to " + std::string(ip) + ":" +
                                     std::to_string(port) + " failed");
        } else {
            m_info("connect to " + std::string(ip) + ":" +
                   std::to_string(port));
        }
        return connected;
    }

    int connect(std::string ip, int port) { return connect(ip.c_str(), port); }

    int connect(std::string_view ip, int port) {
        return connect(ip.data(), port);
    }

    void send(int to, const char *data, int len) {
```

```
        if (::send(to, data, len, 0) == -1) {
            m_error("send data to " + std::to_string(to) + " failed");
        }
    }

    void send(int to, std::string_view data) {
        send(to, data.data(), data.size());
    }

    bool recv(int from, char *data, int len) {
        int read_num = ::recv(from, data, len, 0);
        if (read_num == -1) {
            m_error("recv data from " + std::to_string(from) + " failed");
            return false;
        }
        if (read_num == 0) {
            return false;
        }
        return true;
    }

    void close() { m_close_socket(); }

  private:
    int m_descriptor;

    void m_close_socket() {
        if (m_descriptor != -1) {
            m_info("socket closed");
            ::close(m_descriptor);
            m_descriptor = -1;
        }
    }

    void m_info(const std::string &msg) {
        std::cout << "[INFO] " << msg << std::endl;
    }

    void m_error(const std::string &msg) {
        std::cerr << "[ERROR] " << msg << std::endl;
    }
};

#endif // _SIMPLE_SOCKET_H_
```

6.2.3 socket_client.hpp

```
#pragma once

#ifndef _CLIENT_H_
#define _CLIENT_H_

#include "socket.hpp"
#include <cstring>
#include <iostream>
#include <unistd.h>

class socket_client : socket {
public:
    socket_client()
        : socket(), m_server_descriptor(-1), m_last_send_enter(false) {}

    void connect(std::string_view ip, int port) {
        m_server_descriptor = socket::connect(ip, port);
    }

    void send(std::string_view msg) {
        if (msg.empty()) {
            if (m_last_send_enter) {
                socket::send(m_server_descriptor,
                             m_last_send + std::string{msg});
                std::cout << "client sent: " << m_last_send + std::string{msg}
                    << std::endl;
                if (msg == "exit") {
                    close();
                }
                m_last_send_enter = false;
                m_last_send = "";
                return;
            }
            m_last_send_enter = true;
            m_last_send = "\n";
            return;
        }
        m_last_send.append(msg).append("\n");
        m_last_send_enter = true;
    }

    std::string recv(int size = 1024) {
```

```
        char* buffer = new char[size];
        std::memset(buffer, 0, size);
        bool ok = socket::recv(m_server_descriptor, buffer, size);
        auto res = std::string(buffer);
        delete[] buffer;
        if (!ok || res == "exit\n") {
            res = "";
            std::cout << "[INFO] server disconnected" << std::endl;
            close();
        }
        return res;
    }

    void close() {
        if (m_server_descriptor != -1) {
            std::cout << "[INFO] disconnected" << std::endl;
            ::close(m_server_descriptor);
            m_server_descriptor = -1;
        }
        socket::close();
    }

private:
    int m_server_descriptor;
    bool m_last_send_enter;
    std::string m_last_send{};
};

#endif // _CLIENT_H_
```

6.2.4 socket_server.hpp

```
#pragma once

#ifndef _SERVER_H_
#define _SERVER_H_

#include "socket.hpp"
#include <cstring>
#include <iostream>
#include <vector>

class socket_server : socket {
```

```
public:
  socket_server() : socket() {}

  void bind(int port) { socket::bind(port); }

  void listen(int n = 10) { socket::listen(n); }

  int accept() {
      int connected = socket::accept();
      m_connections.push_back(connected);
      return connected;
  }

  void send_all(std::string_view msg) {
      if (msg.empty()) {
          if (m_last_send_enter) {
              m_send_all(m_last_send + std::string{msg});
              std::cout << "server sent: " << m_last_send + std::string{msg}
                        << std::endl;
              if (msg == "exit") {
                  close();
              }
              m_last_send_enter = false;
              m_last_send = "";
              return;
          }
          m_last_send_enter = true;
          m_last_send = "\n";
          return;
      }
      m_last_send.append(msg).append("\n");
      m_last_send_enter = true;
  }

  void send(int to, std::string_view msg) {
      if (msg.empty()) {
          if (m_last_send_enter) {
              socket::send(to, m_last_send + std::string{msg});
              if (msg == "exit") {
                  m_close(to);
                  return;
              }
              std::cout << "server sent: " << m_last_send + std::string{msg}
                        << std::endl;
```

```
                m_last_send_enter = false;
                m_last_send = "";
                return;
            }
            m_last_send_enter = true;
            m_last_send = "\n";
            return;
        }
        m_last_send.append(msg).append("\n");
        m_last_send_enter = true;
    }

    const std::string recv(int from, int size = 1024) {
        char *buffer = new char[size];
        std::memset(buffer, 0, size);
        bool ok = socket::recv(from, buffer, size);
        auto res = std::string(buffer);
        delete[] buffer;
        if (!ok || res == "exit\n") {
            send(from, "exit");
            send(from, "");
            std::cout << "[INFO] client " << from << " disconnected"
                      << std::endl;
            m_close(from);
            res = "";
        }
        return res;
    }

    void close() { socket::close(); }

  private:
    std::vector<int> m_connections{};

    bool m_last_send_enter{false};

    std::string m_last_send{};

    void m_close(int client) {
        for (auto it = m_connections.begin(); it != m_connections.end(); ++it) {
            if (*it == client) {
                m_connections.erase(it);
                break;
            }
```

```
        }
    }

    void m_send_all(std::string_view msg) {
        for (auto &client : m_connections) {
            socket::send(client, msg);
        }
    }
};

#endif // _SERVER_H_
```

6.3 client.cpp

```
#include "include/socket_client.hpp"
#include <exception>
#include <fstream>
#include <iostream>
#include <string>
#include <thread>

bool check_ipv4(std::string &addr) {
    if (addr.empty() || addr.back() == '.' || addr.front() == '.') {
        return false;
    }
    if (addr == "localhost") {
        addr = "127.0.0.1";
    }
    int num{0};
    int dot{0};
    for (auto &c : addr) {
        if (c == '.') {
            if (num < 0 || num > 255) {
                return false;
            }
            num = 0;
            ++dot;
            continue;
        }
        if (c < '0' || c > '9') {
            return false;
        }
        num = num * 10 + (c - '0');
```

```
    }
    if (num < 0 || num > 255 || dot != 3) {
        return false;
    }
    return true;
}

void receive(socket_client &socket) {
    while (true) {
        auto msg = socket.recv();
        if (msg.empty()) {
            socket.close();
            exit(0);
        }
        std::cout << "received: " << msg << std::endl;
    }
}

int main(const int argc, const char **argv) {
    const auto show_usage = [argv]() {
        std::cerr << "Usage: " << argv[0] << " <ip> <port> [--file <file>]"
                  << std::endl;
    };

    if ((argc != 3 && argc != 5) ||
        (argc == 5 && std::string{argv[3]} != "--file")) {
        std::cerr << "Invalid arguments" << std::endl;
        show_usage();
        return 1;
    }

    std::string ip{argv[1]};
    if (!check_ipv4(ip)) {
        std::cerr << "Invalid ip" << std::endl;
        show_usage();
        return 1;
    }

    int port{-1};
    try {
        port = std::stoi(argv[2]);
    } catch (std::exception &e) {
        std::cerr << "Invalid port" << std::endl;
        show_usage();
```

```
        return 1;
    }

    if (port < 0 || port > 65535) {
        std::cerr << "Invalid port" << std::endl;
        show_usage();
        return 1;
    }

    socket_client client{};
    client.connect(ip, port);

    std::thread recv_t(receive, std::ref(client));

    if (argc == 5) {
        std::fstream in{argv[4]};
        std::string str{};
        std::string tmp{};
        while (std::getline(in, tmp)) {
            str.append(tmp).append("\n");
        }
        client.send(str);
        client.send("");
    }

    while (true) {
        std::string msg;
        std::getline(std::cin, msg);
        client.send(msg);
    }

    recv_t.join();
}
```

6.3.1 server.cpp

```
#include "include/socket_server.hpp"
#include <functional>
#include <iostream>
#include <thread>

void handle_connection(socket_server &socket, int client) {
    while (true) {
```

```
        auto msg = socket.recv(client);
        if (msg.empty()) {
            break;
        }
        std::cout << "server received from " << client << ": " << msg
                  << std::endl;
        msg = msg.substr(0, msg.size() - 1);
        socket.send_all(msg == "\n" ? "" : msg);
        socket.send_all("");
    }
}

void input(socket_server &socket) {
    while (true) {
        std::string msg{};
        std::getline(std::cin, msg);
        socket.send_all(msg);
        if (msg == "exit") {
            socket.send_all("");
            socket.close();
            exit(0);
        }
    }
}

int main(const int argc, const char **argv) {
    const auto show_usage = [argv]() {
        std::cerr << "Usage: " << argv[0] << " <port>" << std::endl;
    };

    if (argc != 2) {
        show_usage();
        return 1;
    }

    int port{-1};
    try {
        port = std::stoi(argv[1]);
    } catch (std::exception &e) {
        std::cerr << "Invalid port" << std::endl;
        show_usage();
        return 1;
    }

```

```
    if (port < 0 || port > 65535) {
        std::cerr << "Invalid port" << std::endl;
        show_usage();
        return 1;
    }

    socket_server server{};

    server.bind(port);
    server.listen();

    std::vector<std::thread> threads{};

    std::thread input_t{input, std::ref(server)};

    while (true) {
        int client = server.accept();
        threads.emplace_back(handle_connection, std::ref(server), client);
    }

    for (auto &thread : threads) {
        thread.join();
    }
}
```