به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

شبکه‌های کامپیوتری

تمرین کامپیوتری 1

نام و نام خانوادگی

امیرحسین دبیری اقدم

شماره دانشجویی

810197502

خرداد ماه 1401

فهرست

چکیده

در این تمرین کامپیوتری ما به کمک روش socket programming و زبان برنامه نویسی C++ یک کلاینت برای سرور Chat roomای که در اختیار ما بود نوشتیم. به این ترتیب که هر کلاینت یک کانکشن TCP با سرور برقرار می‌کند و براساس پروتکل گفته شده در صورت پروژه به رد و بدل کردن درخواست‌ها و پیام‌ها با یکدیگر می‌پردازند.

بخش 1 - پیاده‌سازی

در این بخش جزئیات و نحوه پیاده‌سازی کلاینت به صورت مختصر بیان شده است:

برای سادگی در مدیریت درخواست‌ها، پیاده‌سازی به نحوی انجام شده است که پس از هر درخواست، کلاینت منتظر پاسخ سرور می‌ماند بنابراین مقدار Message ID در این پیاده‌سازی اهمیتی ندارد.

روند کلی برنامه را به تعدادی زیر تابع می‌شکنیم. در ابتدای تابع main براساس آرگومان‌های ورودی آدرس سرور ،پورت آن و نیز نام کاربر را مشخص می‌کنیم؛ اگر ورودی نامعتبر باشد (مثلاً پورت به درستی مشخص نشده باشد یا رنج عددی آن نادرست باشد یا ...) با پیغام *invalid argument* برنامه خاتمه می‌یابد. (اگر مشخصات سرور به عنوان آرگومان ورودی داده نشود مقدار پیش‌فرض 127.0.0.1:9000 در نظر گرفته می‌شود. همچنین اگر نام کاربری داده نشود به کمک تابع gen_random یک نام کاربری رندوم به صورت user_xyz در نظر گرفته می‌شود.)

در ادامه تابع کمکی connect_to_host براساس مقادیر بدست آمده برای آدرس سرور، با کمک توابع ارائه شده در کتابخانهٔ arpa/inet.h (مربوط به اینترفیس ارائه شده توسط سیستم عامل برای سوکت پروگرمینگ است) سعی می‌کند یک ارتباط TCP با سرور برقرار کند و اگر با خطا مواجه شود با پیغام متناسب (پیام‌هایی نظیر *failed to create socket*) موضوع به کاربر اطلاع داده می‌شود و برنامه پایان می‌یابد. در صورت موفقیت آمیز بودن ارتباط با سرور نام کاربری با پیام CONNECT برای سرور ارسال می‌شود و منتظر پیام CONNACK می‌مانیم زیرا در غیر اینصورت ارتباط توسط سرور قطع می‌شود.

حلقه اصلی برنامه بعد از اتصال به سرور اجرا می‌شود. در این حلقه به کمک تابع select (ارائه شده در کتابخانه arpa/inet.h) هر یک ثانیه یک بار ورودی stdin بررسی می‌شود. اگر دستوری توسط کاربر وارد نشده باشد با کمک تابع receive_message به سرور پیام RECEIVE ارسال می‌شود و منتظر می‌مانیم تا پیام RECEIVEREPLY توسط سرور (که شامل پیام‌هایی که برای کاربر ارسال شده است می‌باشد) برای ما ارسال شود. همچنین اگر کاربر دستوری وارد کرده باشد، با کمک تابع handle_commands دستورات ورودی کاربر بررسی و در صورت معتبر بودن دستور، اجرا می‌شود. به این ترتیب که اگر دستور list باشد تابع get_list اجرا شده که با ارسال پیام از نوع LIST به سرور شناسه کاربران دریافت می‌شود (از طریق پیام LISTREPLY) و بعد در یک حلقه for با استفاده از پیام INFO، شناسه کاربران به سرور ارسال شده و نام کاربری هر کدام دریافت شده (از طریق پیام INFOREPLY) و چاپ می‌شود. اگر دستور send باشد باید به فرمت send <User Name> <Message> باشد که اگر نباشد پیغام خطای *wrong input format* چاپ می‌شود؛ اگر فرمت صحیح باشد، لیست کاربران (دوباره) به کمک تابع get_list دریافت شده و بعد این لیست به همراه نام کاربری و متن پیام به تابع send_message داده می‌شود. در این تابع ابتدا در یک حلقه پیام‌هایی از نوع INFO ارسال می‌شود تا مشخص شود کدام یک از شناسه‌های بدست آمده از تابع get_list متناظر با نام کاربری وارد شده است. در این صورت اگر نام کاربری وارد شده در سرور موجود نباشد پیغام خطای *user not found* چاپ می‌شود. اگر نام کاربری در سرور موجود باشد، با استفاده از پیام از نوع SEND متن پیام کاربر را به سرور ارسال می‌کنیم و منتظر پیام سرور از نوع SENDREPLY می‌مانیم. در بخش payload پاسخ سرور یک مقدار status وجود دارد که اگر مقدار آن برابر با 0xff باشد یعنی پیام با موفقیت ارسال شده و اگر 0x00 باشد یعنی با خطا مواجه شده که با توجه به مقدار آن پیام *message sent* یا *sending failed* به کاربر نمایش داده می‌شود (البته در صورت پروژه ذکر

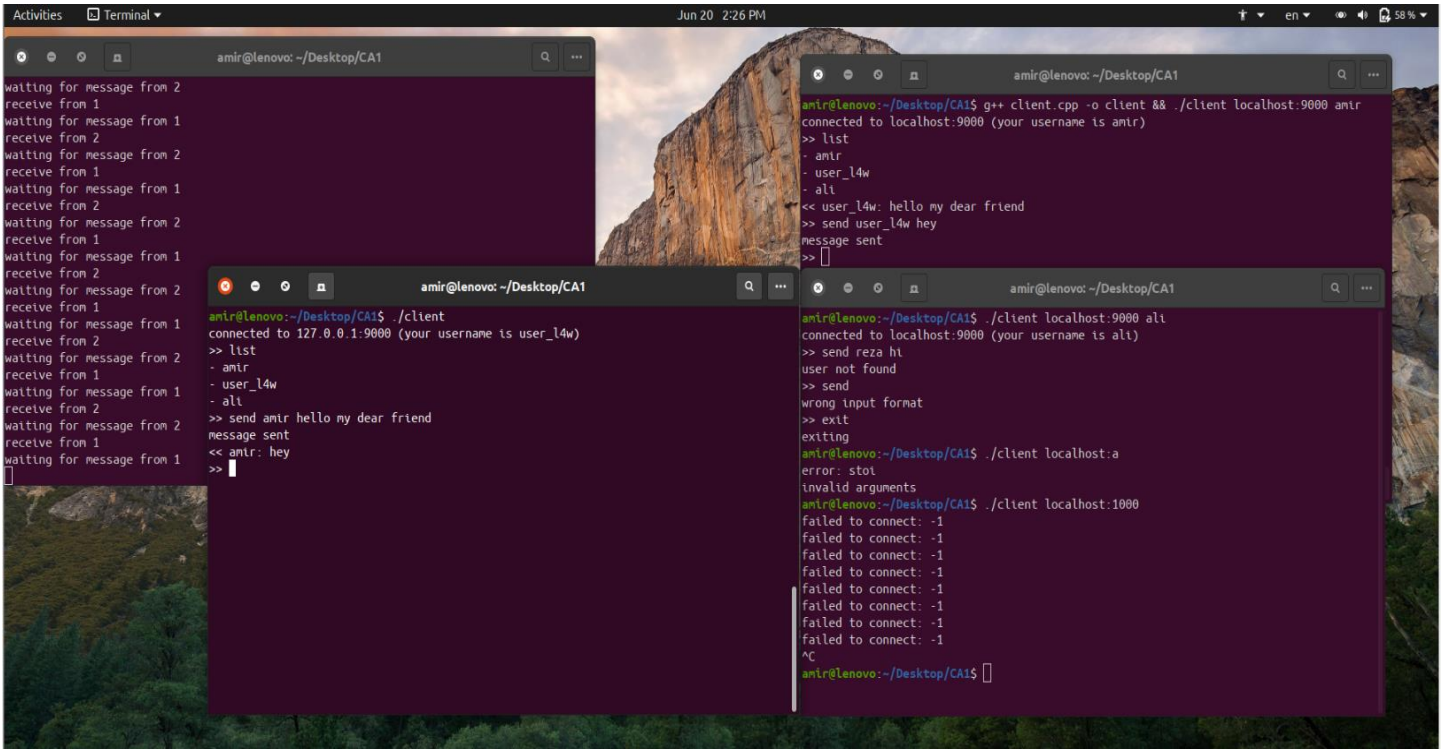
شده بود که اگر 0 باشد یعنی موفقیت در ارسال و اگر 1 باشد یعنی شکست در ارسال است اما با بررسی کد سرور متوجه شدم که صفر مربوط به خطا و 255 یا همان 0xff مربوط به موفقیت است). اگر دستور وارد شده معتبر نباشد خطای *wrong command* نمایش داده می‌شود و حلقه برای گرفتن دستور جدید از کاربر و... ادامه می‌یابد.

منطق کلی توابع کمکی فوق الذکر نظیرsend_username، get_list و ... مشابه یکدیگر است. در ابتدای هر کدام از این توابع یک آرایه از کاراکترها به عنوان بافر تعریف می‌شود و با یک پوینتر از نوع استراکچر Header که در فایل message.h تعریف شده است به آن اشاره می‌کنیم. و بعد مقادیر message_type و length آن هدر را متناسب با کاری که قرار است انجام شود تعیین می‌کنیم و با استفاده از تابع کمکی client_write آن را ارسال می‌کنیم. در ادامه نیز محتویات payload را در صورت لزوم متناسب با کاری که قرار است انجام شود را با client_write ارسال می‌کنیم (مثلاً ارسال نام کاربری به سرور در هنگام اتصال اولیه یا ...). همچنین برای دریافت پاسخ از سرور نیز با استفاده از تابع کمکی client_read ابتدا هدر پاسخ ارسال شده توسط سرور را می‌خوانیم و بعد یک متغیر به عنوان payload تعریف کرده و با همان تابع payload را هم دریافت کرده و در آن متغیر می‌ریزیم. توابع client_write و client_read بنیادی‌ترین توابع هستند که برای رد و بدل کردن پیام‌ها با سرور استفاده می‌شود (در کد سرور هم قطعه کد مشابهی وجود دارد برای ارتباط با کلاینت‌ها). این دو تابع از توابع write و read ارائه شده در کتابخانه arpa/inet.h در یک حلقه while استفاده می‌کنند و تا زمانی که به تعداد بایتی که در ورودی تابع گفته شده است write یا read انجام نشود این حلقه ادامه می‌یابد. همچنین در صورتی که در هر مرحلهٔ ارسال یا دریافت پیام‌ها خطایی رخ دهد پیغام مناسب (نظیر *wrong message length*، *failed to write*، *failed to read* یا ...) چاپ می‌شود و در صورتی که ادامه اجرای برنامه ممکن نباشد، برنامه پایان می‌یابد. همچنین برای هندل کردن exceptionها از کتابخانه مربوطه در C++ و بلوک‌های try و catch استفاده شده است.

کدهای تمام توابع مذکور در بخش ضمیمه‌ها در انتهای گزارش آمده است.

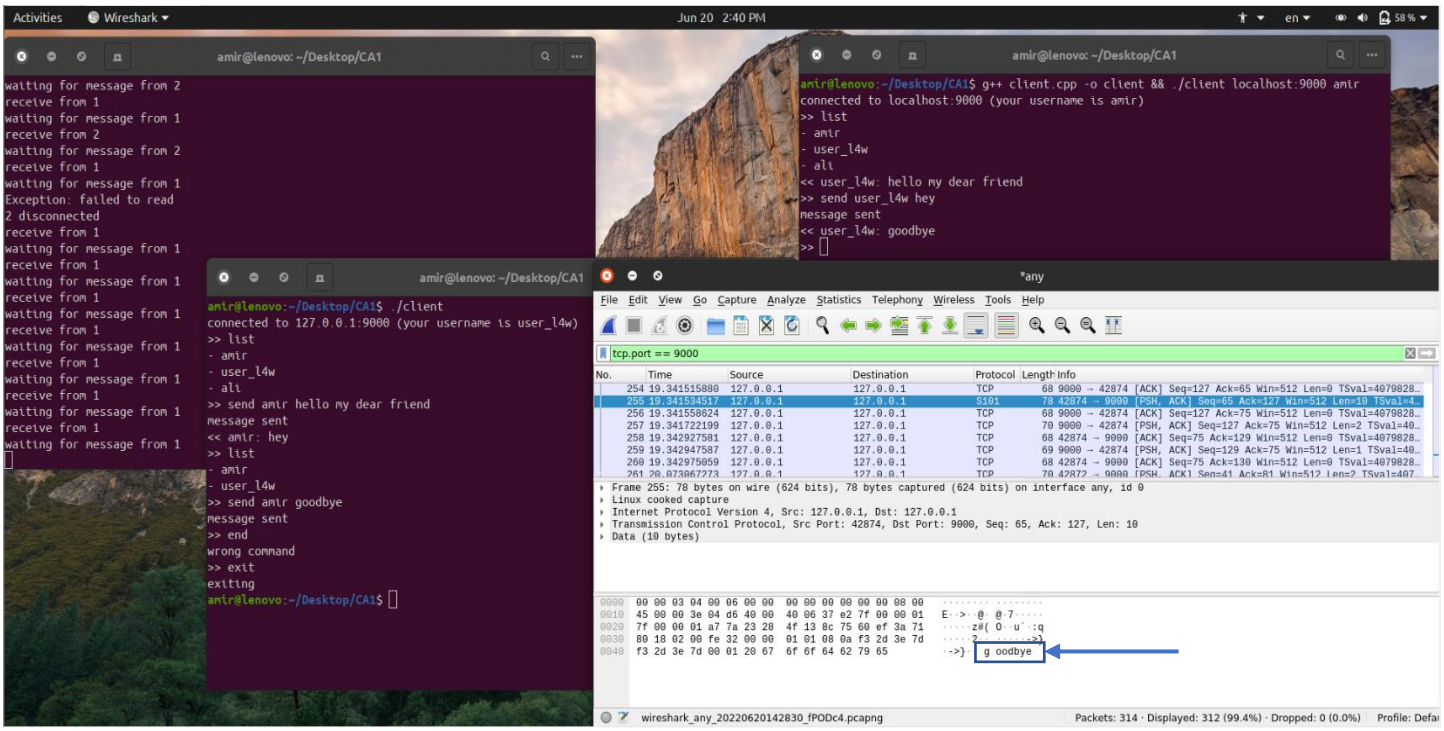
بخش 2- نمایی از اجرای چند کلاینت

در ادامه اجرای سرور و سه کلاینت در شکل 1 مشاهده می‌شود. همانطور که مشاهده می‌شود پیام‌ها به درستی رد و بدل شده‌اند و دستورات list، send و exit به درستی اجرا شده‌اند و به طور کلی کلاینت‌ها عملکرد مورد انتظار را از خود نشان داده‌اند.



شکل 1 – ارتباط سه کلاینت با یک سرور و رد و بدل پیام‌ها بین آن‌ها

همچنین در شکل 2 با استفاده از نرم‌افزار wireshark پکت‌های رد و بدل شده روی پورت 9000 با پروتکل TCP در هنگام اجرای سرور و کلاینت‌ها را مشاهده می‌کنیم و به طور خاص پکت مربوط به پیام *goodbye* ارسال شده توسط کاربر user_l4w به سرور برای کاربر amir و نیز پاسخ سرور به آن پیام و دیگر پیام‌های رد و بدل شده بین سرور و کلانیت‌ها مشاهده می‌شود...



شکل 2 - در هنگام اجرای سرور و کلاینت‌ها، پکت‌های رد و بدل شده روی پورت 9000 با پروتکل **TCP** توسط نرم‌افزار **wireshark** مشاهده می‌شود.

بخش 3 – ضمیمه‌ها

**برای پیاده‌سازی از کدهای ارائه شده برای سرور و نیز توضیحات جلسه توجیهی و فیلم آموزش برنامه نویسی سوکت کمک گرفته‌ام.*

کد مربوط به توابع کمکی نوشته شده، در ادامه آمده است:

```
void client_write(int fd, const uint8_t* buffer, size_t len)
{
    auto n = 0;
    while (n < len)
    {
        auto ret = write(fd, buffer + n, len - n);
        if (ret < 0)
            throw runtime_error("failed to write");
        if (ret == 0)
            throw runtime_error("socket closed");

        n += ret;
    }
}
```

```
void client_read(int fd, uint8_t* buffer, size_t len) {
    auto n = 0;
    while (n < len)
    {
        auto ret = read(fd, buffer + n, len - n);
        if (ret < 0)
            throw runtime_error("failed to read");
        if (ret == 0)
            throw runtime_error("socket closed");

        n += ret;
    }
}
```

```
int connect_to_host(string host, uint16_t port, string name)
{
    auto srv_addr = sockaddr_in{sin_family: AF_INET, sin_port: htons(port)};
    auto err = inet_pton(AF_INET, host.c_str(), &srv_addr.sin_addr);
    if (err < 0)
    {
        cerr << "failed to complete address " << err << endl;
        return EXIT_FAILURE;
    }

    auto fd = socket(AF_INET, SOCK_STREAM, 0);
```

```
    while(1)
    {
        if (fd < 0)
        {
            cerr << "failed to create socket " << fd << endl;
            goto wait;
        }
        err = connect(fd, (sockaddr*)&srv_addr, sizeof(srv_addr));
        if (err < 0)
        {
            cerr << "failed to connect: " << err << endl;
            goto exit;
        }

        send_username(fd, name);
        clog << "connected to " << host << ':' << port << " (your username is " << name << ")"
<< endl;
        cout << ">> ";
        cout.flush();
        break;

        exit:
            close(fd);

        wait:
            sleep(1);
    }
    return fd;
}
```

```
void send_username(int fd, string username)
{
    char buffer[MAX_PAYLOAD_LENGTH + sizeof(Header) + 1];
    Header* hdr_ptr;
    hdr_ptr = (Header*)buffer;

    hdr_ptr->message_type = CONNECT;
    hdr_ptr->length = username.length() + 2;
    client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
    client_write(fd, (uint8_t*)username.c_str(), username.length());

    while(hdr_ptr->message_type != CONNACK)
        client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
}
void receive_message(int fd)
{
    char buffer[MAX_PAYLOAD_LENGTH + sizeof(Header) + 1];
    Header* hdr_ptr;
    hdr_ptr = (Header*)buffer;
```

```
    hdr_ptr->message_type = RECEIVE;
    hdr_ptr->length = sizeof(Header);
    client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
    while(hdr_ptr->message_type != RECEIVEREPLY)
        client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));

    if (hdr_ptr->length > 4)
    {
        int payload_length = hdr_ptr->length - sizeof(Header) - 2;
        if (payload_length < 0)
            throw runtime_error("negative payload length");
        if (payload_length < 1)
            throw runtime_error("wrong message length");

        uint8_t payload[MAX_PAYLOAD_LENGTH + 1];
        client_read(fd, payload, 2);
        int uid = ntohs(*(uint16_t*)payload);

        client_read(fd, payload, payload_length);
        payload[payload_length] = 0;
        string msg = string((char*)payload);

        hdr_ptr->message_type = INFO;
        hdr_ptr->length = 2 + sizeof(Header);
        client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
        uint16_t buf = htons(uid);
        client_write(fd, (uint8_t*)&buf, sizeof(buf));
        while(hdr_ptr->message_type != INFOREPLY)
            client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));

        payload_length = hdr_ptr->length - sizeof(Header);
        if (payload_length < 0)
            throw runtime_error("negative payload length");
        if (payload_length < 1)
            throw runtime_error("wrong message length");

        client_read(fd, payload, payload_length);
        payload[payload_length] = 0;
        string user_name = string((char*)payload);

        cout << "\b\b\b<< " << user_name << ":" << msg << endl;
        cout << ">> ";
        cout.flush();
    }
}
```

```
int handle_commands(int fd)
{
    string line;
    if (!getline(cin, line) || line == "exit")
    {
        clog << "exiting" << endl;
        return 0;
    }

    else if (line == "list")
        get_list(fd, true);

    else if (line.find("send") != string::npos)
    {
        string buf;
        stringstream ss(line);
        vector<string> tokens;
        while (ss >> buf)
            tokens.push_back(buf);

        if (tokens[0] != "send" || tokens.size() < 3)
            cerr << "wrong input format" << endl;

        else
        {
            string username = tokens[1];
            string msg = "";
            for (int i = 2; i < tokens.size(); i++)
                msg += ' ' + tokens[i];
            auto ids = get_list(fd);
            send_message(fd, ids, username, msg);
        }
    }

    else
        cerr << "wrong command" << endl;

    cout << ">> ";
    cout.flush();

    return 1;
}
```

```
vector<int> get_list(int fd, bool verbose = false)
{
    char buffer[MAX_PAYLOAD_LENGTH + sizeof(Header) + 1];
    Header* hdr_ptr;
    hdr_ptr = (Header*)buffer;

    hdr_ptr->message_type = LIST;
    hdr_ptr->length = sizeof(Header);
    client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
    while(hdr_ptr->message_type != LISTREPLY)
        client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));

    int payload_length = hdr_ptr->length - sizeof(Header);
    if (payload_length < 0)
        throw runtime_error("negative payload length");
    if (payload_length < 1)
        throw runtime_error("wrong message length");
    uint8_t payload[MAX_PAYLOAD_LENGTH + 1];
    vector<int> ids;
    for (int i = 0; i < payload_length / 2; i++)
    {
        client_read(fd, payload, 2);
        ids.push_back(ntohs(*(uint16_t*)payload));
    }

    for (int i = 0; i < ids.size(); i++)
    {
        hdr_ptr->message_type = INFO;
        hdr_ptr->length = 2 + sizeof(Header);
        client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
        uint16_t buf = htons(ids[i]);
        client_write(fd, (uint8_t*)&buf, sizeof(buf));

        while(hdr_ptr->message_type != INFOREPLY)
            client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));

        payload_length = hdr_ptr->length - sizeof(Header);
        if (payload_length < 0)
            throw runtime_error("negative payload length");
        if (payload_length < 1)
            throw runtime_error("wrong message length");
        uint8_t payload[MAX_PAYLOAD_LENGTH + 1];
        client_read(fd, payload, payload_length);
        payload[payload_length] = 0;
        if (verbose)
        {
            string user_name = string((char*)payload);
            cout << "- " << user_name << endl;
        }
    }
```

```
    return ids;
}
```

```
void send_message(int fd, vector<int> ids, string username, string msg)
{
    char buffer[MAX_PAYLOAD_LENGTH + sizeof(Header) + 1];
    Header* hdr_ptr;
    hdr_ptr = (Header*)buffer;

    int uid = -1;
    for (int i = 0; i < ids.size(); i++)
    {
        hdr_ptr->message_type = INFO;
        hdr_ptr->length = 2 + sizeof(Header);
        client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
        uint16_t buf = htons(ids[i]);
        client_write(fd, (uint8_t*)&buf, sizeof(buf));

        while(hdr_ptr->message_type != INFOREPLY)
            client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));

        int payload_length = hdr_ptr->length - sizeof(Header);
        if (payload_length < 0)
            throw runtime_error("negative payload length");
        if (payload_length < 1)
            throw runtime_error("wrong message length");
        uint8_t payload[MAX_PAYLOAD_LENGTH + 1];
        client_read(fd, payload, payload_length);
        payload[payload_length] = 0;
        if (string((char*)payload) == username)
        {
            uid = ids[i];
            break;
        }
    }

    if (uid == -1)
        cerr << "user not found" << endl;

    else
    {
        hdr_ptr->message_type = SEND;
        hdr_ptr->length = 2 + sizeof(Header) + msg.length();
        client_write(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
        uint16_t buf = htons(uid);
        client_write(fd, (uint8_t*)&buf, sizeof(buf));
        client_write(fd, (uint8_t*)msg.c_str(), msg.length());
        while(hdr_ptr->message_type != SENDREPLY)
            client_read(fd, (uint8_t*)hdr_ptr, sizeof(*hdr_ptr));
```

```
        uint8_t payload;
        client_read(fd, &payload, 1);
        if (payload == 0xff)
            clog << "message sent" << endl;
        else
            cerr << "sending failed" << endl;
    }
}
```

```
string gen_random(const int len)
{
    static const char alphanum[] =
        "0123456789"
        "abcdefghijklmnopqrstuvwxyz";
    string tmp;
    tmp.reserve(len);
    for (int i = 0; i < len; ++i)
        tmp += alphanum[rand() % (sizeof(alphanum) - 1)];

    return tmp;
}
```

همچنین تابع main نیز به صورت زیر است:

```
int main(int argc, char** argv)
{
    srand((unsigned)time(NULL) * getpid());

    string name = "user_" + gen_random(3);
    string host = DEFAULT_SRV_HOST;
    int port = DEFAULT_SRV_PORT;
    /* handling input arguments*/
    try
    {
        if (argc > 1)
        {
            auto temp = string((char*)argv[1]);
            auto found = temp.find(":");
            if (found == string::npos)
            {
                cerr << "invalid arguments" << endl;
                return EXIT_FAILURE;
            }
            host = temp.substr(0, found);
            port = stoi(temp.substr(found + 1));
            if (port < 0 || port > 65536)
            {
                cerr << "invalid arguments" << endl;
                return EXIT_FAILURE;
```

```
            }
        }
        if (argc > 2)
            name = string((char*)argv[2]);
    }
    catch (exception& e)
    {
        cerr << "error: " << e.what() << endl;
        cerr << "invalid arguments" << endl;
        return EXIT_FAILURE;
    }

    auto fd = connect_to_host(host, port, name);
    if (fd == EXIT_FAILURE)
        return EXIT_FAILURE;

    /* main loop */
    while(1)
    {
        try
        {
            fd_set read_fds;
            FD_ZERO (&read_fds);
            FD_SET(STDIN_FILENO, &read_fds);
            auto tv = timeval{WAIT_TIME, 0};
            auto ret = select(2, &read_fds, NULL, NULL, &tv);

            if (ret < 0)
                throw runtime_error("select error");

            /* Asks server for new messages every 1 minute */
            else if (ret == 0)
                receive_message(fd);

            /* Handling user commands */
            if (FD_ISSET(STDIN_FILENO, &read_fds))
            {
                auto result = handle_commands(fd);
                if (!result)
                    break;
            }
        }
        catch (exception& e)
        {
            cerr << "error: " << e.what() << endl;
            cerr << "if the above error occurs repeatedly, consider restarting the app or your
system" << endl;
            cout << ">> ";
            cout.flush();
        }
    }
```

```
    return EXIT_SUCCESS;
}
```

❋نکته‌ای که لازم به ذکر است این است که کد سروری که ارائه شده بود نیازمند اندکی تغییر بود زیرا این کد روی سیستم من کامپایل نمی‌شد و لذا از راه حلی که در جلسه توجیهی هم گفته شد برای رفع مشکل استفاده کردم.