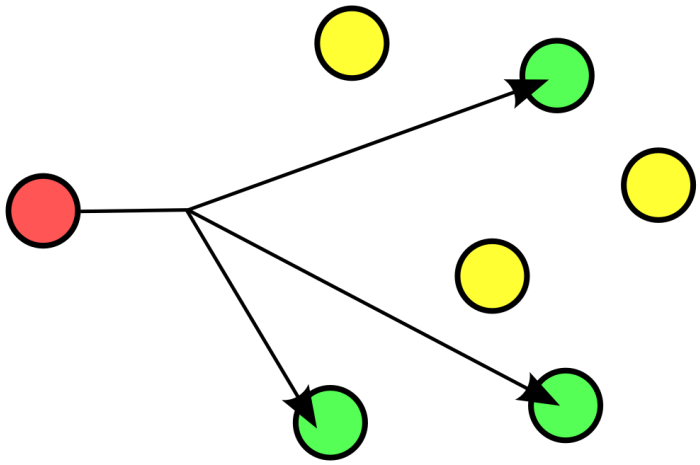
گزارش کار

Multicast Protocol



شبکه‌های کامپیوتری

غزل کلهر (۸۱۰۱۹۶۶۷۵)

محمدامین باقرشاهی (۸۱۰۱۹۷۴۶۴)

فهرست مطالب

مقدمه

در این پروژه به طراحی و پیاده‌سازی یک شبکه با پروتکل‌های multicast می‌پردازیم. به این منظور از پایپ‌های نام‌دار برای تعریف موجودیت‌های سیستم در قالب پراسس‌های جداگانه استفاده کردیم. در برنامه خود تعدادی گروه و کلاینت تعریف کردیم و با ایجاد یک توپولوژی جامع به رد و بدل پیام‌ها و فایل‌ها پرداختیم. به منظوری پیاده‌سازی آن از مفاهیمی که در درس فراگرفته بودیم استفاده کردیم و به پیاده‌سازی الگوریتم‌های مربوط به این شبکه مانند DVMRP پرداختیم.

ساختار پروژه

این پروژه از دو پوشه‌ی src و include تشکیل شده است. در پوشه‌ی include هدر فایل‌های مربوط به هر کلاس قرار گرفته است که در آن‌ها تعریف متدها و فیلدها و نیز کتابخانه‌ها قرار گرفته است. در پوشه‌ی src نیز فایل‌های cpp مربوط به کلاس‌ها آمده است که در آن‌ها بدنه‌ی تمامی متدها قرار گرفته است.

همچنین در پوشه‌ی اصلی پروژه یک MakeFile قرار دارد که از آن برای کامپایل و ساخت فایل‌های اجرایی پروژه بهره می‌گیریم.

در پوشه unicast-tables نیز جدول یونیکست متناظر با هر روتر آمده است.

نحوه اجرا

برای اجرای پروژه کافی است که با دستور cd multicast-protocol به پوشه اصلی پروژه وارد شویم. سپس با دستور make فایل‌ها را کامپایل می‌کنیم. در ادامه به بررسی نحوه اجرای هر یک از برنامه‌ها و دستورات نظیر آن‌ها می‌پردازیم.

دستورات سرور

ایجاد و اجرای سرور

برای ایجاد و سپس اجرای برنامه‌ی سرور دستور زیر را در ترمینال وارد می‌کنیم. تنها آرگومان آن بیانگر IP سرور است.
پس از اجرای این دستور سایر برنامه‌ها (کلاینت و ...) می‌توانند اجرا شوند و این آیپی داده شده را در آرگومان در دستور مربوط به اجرای خود وارد کنند.

ورودی

```
./Server.out <server_ip>
```

ورودی نمونه

```
./Server.out 1.1.2.2
```

توضیحات کد سرور

فیلد ها:

- string server_ip: این فیلد ip سرور را نگهداری می کند.
- int server_port: این فیلد پورت سرور را نگهداری می کند.
- pair<string, string> server_pipe: این فیلد پایپ های خواندن و نوشتنی که با استفاده از آن برنامه های دیگر می توانند به سرور متصل شوند را نگهداری می کند.
- map<string, pair<string, string>> clients_pipes: این فیلد یک map از ip کلاینت ها به پایپ های خواندن و نوشتن آن هاست. سرور با استفاده از این فیلد می تواند با کلاینت ها پیام رد و بدل کند.
- map<string, string> clients_ips: این فیلد یک map از نام کلاینت ها به ip آن ها است.
- map<string, vector<string>> clients_groups: این فیلد گروه هایی که کلاینت ها در آن ها عضو شده است را نگه داری می کند.
- map<string, pair<string, string>> groupservers_pipes: این فیلد یک map از نام گروه ها به پایپ های خواندن و نوشتن آن هاست. سرور با استفاده از این فیلد می تواند با سرور گروه ها پیام رد و بدل کند.
- map<string, pair<string, string>> router_pipes: این فیلد یک map از پورت روتر ها به پایپ های خواندن و نوشتن آن هاست. سرور با استفاده از این فیلد می تواند با روتر ها پیام رد و بدل کند.

- map<string, string> clients_ips: این فیلد یک map از نام گروه ها به ip آن ها است.

متد ها:

- void start()

این متد اصلی سرور است که در یک loop پایپ کلاینت ها، سرور گروه ها و پایپ خودش open می شوند و به لیست اضافه می‌شوند تا به آن ها گوش داده شود. در ادامه بر اساس اینکه دستوری از stdin دریافت شود، پیام connection از پایپ سرور، پیام از کلاینت ها و یا از سرور گروه ها دریافت شود به ترتیب توابع handle_command و handle_connection_message و handle_client_message و handle_groupserver_message فراخوانی می شود تا هندل شوند. در انتها نیز پایپ هایی که open شده اند close می شوند.

```cpp
void Server::start() {
    fd_set read_fds, copy_fds;
    FD_ZERO(&copy_fds);
    FD_SET(STDIN, &copy_fds);
    int max_fd = STDIN;
    int activity;
    char received_buffer[MAX_MESSAGE_SIZE] = {0};
    int server_pipe_fd = open(server_pipe.first.c_str(), O_RDWR);
    printf("Server is starting ...\n");
    while (true) {
        max_fd = server_pipe_fd;
        map<int, string> clients_fds = add_clients_to_set(copy_fds, max_fd);
        map<int, string> groupservers_fds = add_groupservers_to_set(copy_fds, max_fd);

        FD_SET(server_pipe_fd, &copy_fds);

        // Add fds to set
        memcpy(&read_fds, &copy_fds, sizeof(copy_fds));

        // Select
        cout << "> ";
        fflush(STDIN);
        activity = select(max_fd + 1, &read_fds, NULL, NULL, NULL);

        if (activity < 0)
            return;

        int ready_sockets = activity;
        for (int fd = 0; fd <= max_fd  &&  ready_sockets > 0; ++fd) {
            if (FD_ISSET(fd, &read_fds)) {
                memset(received_buffer, 0, sizeof received_buffer);

                // Receive command from command line.
                if (fd == 0) {
                    fgets(received_buffer, MAX_COMMAND_SIZE, stdin);
                    received_buffer[strlen(received_buffer) - 1] = '\0';
                    cout << "received command: " << received_buffer << endl;
                    handle_command(string(received_buffer));
                }

                // Receive connection message.
                else if (fd == server_pipe_fd) {
                    read(fd, received_buffer, MAX_MESSAGE_SIZE);
                    handle_connection_message(string(received_buffer));
                }

                // Receive message from clients.
                else if (clients_fds.find(fd) != clients_fds.end()) {
                    read(fd, received_buffer, MAX_MESSAGE_SIZE);
                    cout << "client message: " << received_buffer << endl;
                    handle_client_message(string(received_buffer), clients_fds[fd]);
                }

                // Receive message from group servers.
                else if (groupservers_fds.find(fd) != groupservers_fds.end()) {
                    read(fd, received_buffer, MAX_MESSAGE_SIZE);
                    cout << "group server message: " << received_buffer << endl;
                    handle_groupservers_message(string(received_buffer));
                }
            }
        }

        close_clients_fds(clients_fds);
        close_groupservers_fds(groupservers_fds);
        cout << "--------------- event ---------------" << endl;
    }

    close(server_pipe_fd);
}
```

- map<int, string> add_clients_to_set(fd_set& fds, int& max_fd)

در این متد پایپ های کلاینت هایی که به سرور متصل شده اند open می شوند و به لیست fd هایی که سرور به آنها گوش می دهد اضافه می شوند.

```
92   map<int, string> Server::add_clients_to_set(fd_set& fds, int& max_fd) {
93       map<int, string> clients_fds;
94       map<string, pair<string, string>>::iterator it;
95       for (it = clients_pipes.begin(); it != clients_pipes.end(); it++) {
96           cout << "listen to pipe: " << it->second.second << endl;
97           int client_fd = open(it->second.second.c_str(), O_RDWR);
98           clients_fds.insert({client_fd, it->first});
99           FD_SET(client_fd, &fds);
100          max_fd = (max_fd > client_fd) ? max_fd : client_fd;
101      }
102
103      return clients_fds;
104  }
```

- std::map<int, string> add_groupservers_to_set(fd_set& fds, int& max_fd)

در این متد پایپ های سرور گروه هایی که به سرور متصل شده اند open می شوند و به لیست fd هایی که سرور به آنها گوش می دهد اضافه می شوند.

```
106  map<int, string> Server::add_groupservers_to_set(fd_set& fds, int& max_fd) {
107      map<int, string> groupservers_fds;
108      map<string, pair<string, string>>::iterator it;
109      for (it = group_servers_pipes.begin(); it != group_servers_pipes.end(); it++) {
110          int groupserver_fd = open(it->second.second.c_str(), O_RDWR);
111          groupservers_fds.insert({groupserver_fd, it->first});
112          FD_SET(groupserver_fd, &fds);
113          max_fd = (max_fd > groupserver_fd) ? max_fd : groupserver_fd;
114      }
115
116      return groupservers_fds;
117  }
```

- void close_clients_fds(map<int, string> clients_fds)

این متد برای بسته شدن پایپ های کلاینت هاست.

```
120  void Server::close_clients_fds(map<int, string> clients_fds){
121      map<int, string>::iterator it;
122      for (it = clients_fds.begin(); it != clients_fds.end(); it++)
123          close(it->first);
124  }
```

- void close_groupservers_fds(map<int, string> groupservers_fds)

این متد برای بسته شدن پایپ های سرور گروه هاست.

```
126  void Server::close_groupservers_fds(map<int, string> groupservers_fds){
127      map<int, string>::iterator it;
128      for (it = groupservers_fds.begin(); it != groupservers_fds.end(); it++)
129          close(it->first);
130  }
```

- void handle_command(string command)

این متد بر اساس نوع دستور دریافتی متد مورد نظر برای هندل کردن آن را فراخوانی می کند.

```
132  void Server::handle_command(string command) {
133      vector<string> command_parts = split(command, COMMAND_DELIMITER);
134
135      if (command_parts.size() < 1)
136          return;
137
138      if (command_parts[ARG0] == SERVER_TO_ROUTER_CONNECT_CMD)
139          handle_connect_router(command_parts[ARG1]);
140
141      else printf("Unknown command.\n");
142  }
```

- void handle_connection_message(string message)

این متد زمانی فراخوانی می شود که یه پیام connection دریافت شود. در این متد براساس نوع فرستنده پیام متد مورد نظر برای هندل کردن آن فراخوانی می شود.

```
148  void Server::handle_connection_message(string message) {
149      vector<string> message_parts = split(message, MESSAGE_DELIMITER);
150      printf("Server::handle_connection_message\n");
151      if (message_parts[ARG0] == CLIENT_TO_SERVER_CONNECT_MSG)
152          handle_client_connect(message_parts[ARG1], message_parts[ARG2]);
153
154      if (message_parts[ARG0] == GROUPSERVER_TO_SERVER_CONNECT_MSG)
155          handle_group_server_connect(message_parts[ARG1], message_parts[ARG2]);
156
157      if (message_parts[ARG0] == ROUTER_TO_SERVER_CONNECT_MSG)
158          handle_router_connect(message_parts[ARG1]);
159  }
```

- void handle_client_connect(string name, string ip)

این متد زمانی صدا زده می شود پیام connection از یک کلاینت فرستاده شود. در این متد پایپ های خواندن و نوشتن مربوط به کلاینت ساخته می شوند و به لیست مربوطه اضافه می شوند.

```
161  void Server::handle_client_connect(string name, string ip) {
162      printf("Client with name %s and ip %s connects.\n", name.c_str(), ip.c_str());
163
164      pair<string, string> client_pipe = {(string(PIPE_ROOT_PATH) + SERVER_PIPE + CLIENT_PIPE + PIPE_NAME_DELIMITER +
165              (string(PIPE_ROOT_PATH) + SERVER_PIPE + CLIENT_PIPE + PIPE_NAME_DELIMITER + name + WRITE_PIPE)};
166
167      unlink(client_pipe.first.c_str());
168      mkfifo(client_pipe.first.c_str(), READ_WRITE);
169      unlink(client_pipe.second.c_str());
170      mkfifo(client_pipe.second.c_str(), READ_WRITE);
171
172      clients_ips.insert({name, ip});
173      clients_pipes.insert({name, client_pipe});
174  }
```

- void handle_group_server_connect(string name, string ip)

این متد زمانی صدا زده می شود پیام connection از یک سرور گروه فرستاده شود. در این متد پایپ های خواندن و نوشتن مربوط به سرور گروه ساخته می شوند و به لیست مربوطه اضافه می شوند.

```
176  void Server::handle_group_server_connect(string name, string ip) {
177      printf("Group server with name %s and ip %s connects.\n", name.c_str(), ip.c_str());
178
179      pair<string, string> group_server_pipe = {(string(PIPE_ROOT_PATH) + SERVER_PIPE + GROUPSERVER_PIPE + PIPE_NAME_DELIM
180              (string(PIPE_ROOT_PATH) + SERVER_PIPE + GROUPSERVER_PIPE + PIPE_NAME_DELIMITER + name + WRITE_PIPE)};
181
182      unlink(group_server_pipe.first.c_str());
183      mkfifo(group_server_pipe.first.c_str(), READ_WRITE);
184      unlink(group_server_pipe.second.c_str());
185      mkfifo(group_server_pipe.second.c_str(), READ_WRITE);
186
187      group_servers_pipes.insert({name, group_server_pipe});
188      groups_ip.insert({name, ip});
189  }
```

- void handle_router_connect(string port)

این متد زمانی صدا زده می شود پیام connection از یک روتر فرستاده شود. در این متد پایپ های خواندن و نوشتن مربوط به سرور گروه ساخته می شوند و به لیست مربوطه اضافه می شوند.

```
191  void Server::handle_router_connect(string port) {
192      printf("Router with port %s connects.\n", port.c_str());
193
194      pair<string, string> router_pipe = {(string(PIPE_ROOT_PATH) + SERVER_PIPE + ROUTER_PIPE + PIPE_NAME_DELIMITE
195              (string(PIPE_ROOT_PATH) + SERVER_PIPE + ROUTER_PIPE + PIPE_NAME_DELIMITER + port + WRITE_PIPE)};
196
197      unlink(router_pipe.first.c_str());
198      mkfifo(router_pipe.first.c_str(), READ_WRITE);
199      unlink(router_pipe.second.c_str());
200      mkfifo(router_pipe.second.c_str(), READ_WRITE);
201
202      routers_pipes.insert({port, router_pipe});
203  }
```

- void handle_client_message(string message, string client_name)

این متد زمانی فراخوانی می شود که پیامی از کلاینت به سرور برسد. براساس نوع این پیام متد مورد نظر برای هندل کردن آن فراخوانی می شود.

```
205  void Server::handle_client_message(string message, string client_name) {
206      vector<string> message_parts = split(message, MESSAGE_DELIMITER);
207
208      if (message_parts[ARG0] == GET_GROUP_LIST_MSG)
209          handle_get_group_list(client_name);
210
211      else if (message_parts[ARG0] == JOIN_GROUP_MSG)
212          handle_join_group(client_name, message_parts[ARG1]);
213
214      else if (message_parts[ARG0] == LEAVE_GROUP_MSG)
215          handle_leave_group(client_name, message_parts[ARG1]);
216
217      else printf("Unknown Message\n");
218  }
```

- void handle_get_group_list(string client_name)

این متد لیست گروه های موجود به همراه ip آن ها را آماده می کند و برای کلاینتی که آن را درخواست کرده است می فرستد.

```
220  void Server::handle_get_group_list(string client_name) {
221      string group_list = "Group List: ";
222      map<string, string>::iterator it;
223      for (it = groups_ip.begin(); it != groups_ip.end(); it++)
224          group_list += "(" + it->first + ", " + it->second + ") ";
225
226      int client_fd = open(clients_pipes[client_name].first.c_str(), O_RDWR);
227      write(client_fd, group_list.c_str(), group_list.size());
228      close(client_fd);
229  }
```

- void handle_join_group(string client_name, string group_name)

این تابع زمانی صدا می شود که کلاینت بخواهد به یک گروه اضافه شود. در ابتدا اطلاعات مربوط به عضویت کلاینت ها آپدیت می شود و در ادامه پیامی به روتر ها فرستاده می شود تا جدول های روتینگ خود را آپدیت کنند

```cpp
void Server::handle_join_group(std::string client_name, std::string group_name){
    if (group_servers_pipes.find(group_name) == group_servers_pipes.end()) {
        printf("Group does not exist.\n");
        return;
    }
    if (clients_groups.find(client_name) == clients_groups.end())
        clients_groups.insert({client_name, {group_name}});
    else
        clients_groups[client_name].push_back(group_name);

    printf("Client named %s joined group named %s.\n", client_name.c_str(), group_name.c_str());

    send_update_command(client_name, group_name, "join");
}
```

- void handle_leave_group(string client_name, string group_name)

این تابع زمانی صدا می شود که کلاینت بخواهد از یک گروه خارج شود. در ابتدا اطلاعات مربوط به عضویت کلاینت ها آپدیت می شود و در ادامه پیامی به روتر ها فرستاده می شود تا جدول های روتینگ خود را آپدیت کنند.

```cpp
void Server::handle_leave_group(std::string client_name, std::string group_name){
    if (clients_groups.find(client_name) != clients_groups.end() && clients_groups[client_name].size() > 0)
        clients_groups[client_name].erase(remove(clients_groups[client_name].begin(),
                clients_groups[client_name].end(), group_name), clients_groups[client_name].end());

    printf("Client named %s leaved group named %s.\n", client_name.c_str(), group_name.c_str());

    send_update_command(client_name, group_name, "leave");
}
```

- void send_update_command(string client_name, string group_name, string command)

در این متد پیامی به روتر ها فرستاده می شود تا جدول های روتینگ خود را آپدیت کنند.

```cpp
void Server::send_update_command(string client_name, string group_name, string command) {
    string client_ip = clients_ips[client_name];
    string message = command + MESSAGE_DELIMITER + client_ip + MESSAGE_DELIMITER + group_name;
    cout << "message: " << message << endl;
    map<string, pair<string, string>>::iterator it;
    for (it = routers_pipes.begin(); it != routers_pipes.end(); it++) {
        cout << "pipe: " << it->second.first << endl;
        int router_fd = open(it->second.first.c_str(), O_RDWR);
        write(router_fd, message.c_str(), message.size());
        close(router_fd);
    }
}
```

دستورات کلاینت

ایجاد و اجرای کلاینت

برای ایجاد و سپس اجرای برنامه‌ی یک کلاینت دستور زیر را در ترمینال وارد می‌کنیم. آرگومان اول آن نشان‌دهنده نام کلاینت است. آرگومان دوم آن بیانگر IP سرور است. آرگومان سوم و چهارم آن نیز به ترتیب بیانگر IP کلاینت و پورت روتر هستند.
پس از اجرای این دستور کاربر می‌تواند دستوراتی که در ادامه آمده است را در این ترمینال وارد کند.

ورودی

```
./Client.out <name> <server_ip> <client_ip> <router_port>
```

ورودی نمونه

```
SetIp Ghazal 1.1.2.2 1.1.1.1 8420
```

گرفتن لیست گروه‌ها

با استفاده از دستور زیر یک کلاینت می‌تواند لیست گروه‌ها را از سرور دریافت نماید.

ورودی

```
GetGroupList
```

ورودی نمونه

```
GetGroupList
```

عضو شدن در گروه

با استفاده از این دستور کلاینت می‌تواند در یک گروه عضو شد. تنها آرگومان این دستور نام گروهی است که کاربر قصد عضویت در آن را دارد.

ورودی

```
JoinGroup <group_name>
```

ورودی نمونه

```
JoinGroup DM
```

ترک کردن گروه

با استفاده از این دستور کلاینت می‌تواند یک گروه را ترک کند و دیگر عضو آن نباشد. همانند دستور قبلی تنها آرگومان این دستور نیز نام گروه موردنظر است.

ورودی

```
LeaveGroup <group_name>
```

ورودی نمونه

```
LeaveGroup DM
```

ارسال فایل

با استفاده از این دستور کلاینت می‌تواند یک فایل را در یک گروه که عضو آن است بفرستد. آرگومان اول این دستور نام فایل و آرگومان دوم آن نام گروه مقصد است.

ورودی

```
SendFile <file_name> <group_name>
```

ورودی نمونه

```
SendFile test-file DM
```

ارسال پیام

با استفاده از این دستور کلاینت می‌تواند یک پیام را در یک گروه که عضو آن است بفرستد. آرگومان اول این دستور محتوای پیام و آرگومان دوم آن نام گروه مقصد است. همچنین فرض شده است که در متن پیام هیچ فاصله‌ای بین کلمات وجود ندارد و اجزای پیام با ـ از یکدیگر جدا شده‌اند.

ورودی

```
SendMessage <message> <group_name>
```

ورودی نمونه

```
SendMessage Hi_everyone! DM
```

نمایش گروه‌ها

با استفاده از این دستور کلاینت لیست گروه‌هایی که در آن‌ها عضو است را نمایش می‌دهد.

ورودی

```
ShowGroups
```

ورودی نمونه

```
ShowGroups
```

توضیحات کد کلاینت

فیلد ها:

- string client_ip: این فیلد ip کلاینت را نگه داری می کند.
- string name: این فیلد نام کلاینت را نگه داری می کند.
- string server_ip: این فیلد ip سرور را نگه داری می کند.
- string router_port: این فیلد پورت روتری که به کلاینت متصل است را نگه داری می کند.

- string groups: این فیلد گروه هایی که کلاینت در آن ها عضو شده است را نگه داری می کند.
- client_to_server_pipe :pair<string, string>: این فیلد نگه دارنده دو پایپ خواندن و نوشتن بین کلاینت و سرور است.
- client_to_router_pipe :pair<string, string>: این فیلد نگه دارنده دو پایپ خواندن و نوشتن بین کلاینت و روتر است.

متد ها:

- void start();

این متد اصلی کلاینت است که در آن در یک loop منتظر دریافت دستور از stdin یا دریافت پیام از router می ماند. در صورتی که از stdin دستوری دریافت کند با فراخوانی تابع handle_command این دستور هندل می شود و در صورتی که پیامی از router دریافت کند، با استفاده از تابع handle_router_message آن را هندل می کند. در انتهای loop نیز پایپ هایی که open شده اند close می شوند.

```cpp
void Client::start() {
    fd_set read_fds, copy_fds;
    FD_ZERO(&copy_fds);
    FD_SET(STDIN, &copy_fds);
    int max_fd = STDIN;
    int activity;
    char received_buffer[MAX_MESSAGE_SIZE] = {0};
    while (true) {
        int router_pipe_fd = open(client_to_router_pipe.first.c_str(), O_RDWR);
        FD_SET(router_pipe_fd, &copy_fds);
        max_fd = router_pipe_fd;

        // Add fds to set
        memcpy(&read_fds, &copy_fds, sizeof(copy_fds));

        // Select
        cout << "> ";
        fflush(STDIN);
        activity = select(max_fd + 1, &read_fds, NULL, NULL, NULL);

        if (activity < 0)
            return;

        int ready_sockets = activity;
        for (int fd = 0; fd <= max_fd  &&  ready_sockets > 0; ++fd) {
            if (FD_ISSET(fd, &read_fds)) {
                memset(received_buffer, 0, sizeof received_buffer);

                // Command line input
                if (fd == 0) {
                    fgets(received_buffer, MAX_COMMAND_SIZE, stdin);
                    received_buffer[strlen(received_buffer) - 1] = '\0';
                    cout << "received command: " << received_buffer << endl;
                    handle_command(string(received_buffer));
                }

                // Router message
                if (fd == router_pipe_fd) {
                    read(fd, received_buffer, MAX_MESSAGE_SIZE);
                    handle_router_message(string(received_buffer));
                }
            }
        }

        close(router_pipe_fd);
        cout << "--------------- event ---------------" << endl;
    }
```

- void handle_command(std::string command);

این متد بر اساس نوع دستور دریافتی متد مورد نظر برای هندل کردن آن را فراخوانی می کند.

```
86  void Client::handle_command(string command) {
87      vector<string> command_parts = split(command, COMMAND_DELIMITER);
88
89      if (command_parts.size() < 1)
90          return;
91
92      else if (command_parts[ARG0] == SET_CLIENT_IP_CMD)
93          handle_set_ip(command_parts[ARG1]);
94
95      else if (command_parts[ARG0] == GET_GROUP_LIST_CMD)
96          handle_get_group_list();
97
98      else if (command_parts[ARG0] == JOIN_GROUP_CMD)
99          handle_join_group(command_parts[ARG1]);
100
101     else if (command_parts[ARG0] == LEAVE_GROUP_CMD)
102         handle_leave_group(command_parts[ARG1]);
103
104     else if (command_parts[ARG0] == SELECT_GROUP_CMD)
105         handle_select_group(command_parts[ARG1]);
106
107     else if (command_parts[ARG0] == SEND_FILE_CMD)
108         handle_send_file(command_parts[ARG1], command_parts[ARG2]);
109
110     else if (command_parts[ARG0] == SEND_MESSAGE_CMD)
111         handle_send_message(command_parts[ARG1], command_parts[ARG2]);
112
113     else if (command_parts[ARG0] == SHOW_GROUPS_CMD)
114         handle_show_groups();
115
116     else if (command_parts[ARG0] == SYNC_CMD)
117         handle_sync();
118
119     else if (command_parts[ARG0] == SIGN_OUT_CMD)
120         handle_sign_out();
121
122     else printf("Unknown command.\n");
123 }
```

- void handle_set_ip(std::string ip);

این متد برای هندل کردن دستور SetIp صدا زده می شود و در آن ip کلاینت مقداردهی می شود.

```
125 void Client::handle_set_ip(string ip) {
126     this->client_ip = ip;
127     printf("Client IP changed successfuly\n");
128 }
```

- void handle_get_group_list();

این متد برای هندل کردن دستور GetGroupList صدا زده می شود. در آن به سرور پیامی فرستاده می شود و لیست گروه ها به همراه ip آن ها دریافت می شود و نمایش داده می شود.

```
130  void Client::handle_get_group_list() {
131      int pipe_write_fd = open(client_to_server_pipe.second.c_str(), O_RDWR);
132      string message = string(GET_GROUP_LIST_MSG);
133      write(pipe_write_fd, message.c_str(), message.size());
134      close(pipe_write_fd);
135
136      char group_list[MAX_MESSAGE_SIZE] = {0};
137      memset(group_list, 0, sizeof group_list);
138      int pipe_read_fd = open(client_to_server_pipe.first.c_str(), O_RDWR);
139      read(pipe_read_fd, group_list, MAX_MESSAGE_SIZE);
140      close(pipe_read_fd);
141
142      cout << group_list << endl;
143  }
```

- void handle_join_group(std::string group_name);

این متد برای هندل کردن دستور JoinGroup صدا زده می شود. گروه مورد نظر به لیست گروه های کلاینت اضافه می شود. همچنین پیامی به سرور فرستاده می شود تا اطلاعات مربوط به عضویت کلاینت ها در گروه ها آپدیت شود.

```
145  void Client::handle_join_group(string group_name) {
146      cout << "send to pipe: " << client_to_server_pipe.second << endl;
147      int pipe_write_fd = open(client_to_server_pipe.second.c_str(), O_RDWR);
148      string message = string(JOIN_GROUP_MSG) + MESSAGE_DELIMITER + group_name;
149      write(pipe_write_fd, message.c_str(), message.size());
150      close(pipe_write_fd);
151
152      groups.push_back(group_name);
153  }
```

- void handle_leave_group(std::string group_name);

این متد برای هندل کردن دستور LeaveGroup صدا زده می شود. گروه مورد نظر hc لیست گروه های کلاینت حذف می شود. همچنین پیامی به سرور فرستاده می شود تا اطلاعات مربوط به عضویت کلاینت ها در گروه ها آپدیت شود.

```
155  void Client::handle_leave_group(string group_name) {
156      int pipe_write_fd = open(client_to_server_pipe.second.c_str(), O_RDWR);
157      string message = string(LEAVE_GROUP_MSG) + MESSAGE_DELIMITER + group_name;
158      write(pipe_write_fd, message.c_str(), message.size());
159      close(pipe_write_fd);
160
161      groups.erase(remove(groups.begin(), groups.end(), group_name), groups.end());
162  }
```

- void handle_send_file(std::string file_name, std::string group_name);

این متد برای هندل کردن دستور SendFile صدا زده می شود. در ابتدا محتوای فایل مورد نظر خوانده می شود و سپس پیامی حاوی محتوای فایل به روتر فرستاده می شود.

```
168  void Client::handle_send_file(string file_name, string group_name) {
169      string message = read_file_to_string(file_name);
170
171      int pipe_write_fd = open(client_to_router_pipe.second.c_str(), O_RDWR);
172      string sending_message = string(SEND_MSG) + MESSAGE_DELIMITER + group_name + MESSAGE_DELIMITER + message;
173      cout << "message: " << sending_message << endl;
174      write(pipe_write_fd, sending_message.c_str(), sending_message.size());
175      close(pipe_write_fd);
176  }
```

- void handle_send_message(std::string message, std::string group_name);

این متد برای هندل کردن دستور SendFile صدا زده می شود. در این متد پیامی حاوی پیام مورد نظر به روتر فرستاده می شود.

```
178  void Client::handle_send_message(string message, string group_name) {
179      int pipe_write_fd = open(client_to_router_pipe.second.c_str(), O_RDWR);
180      string sending_message = string(SEND_MSG) + MESSAGE_DELIMITER + group_name + MESSAGE_DELIMITER + message;
181      cout << "message: " << sending_message << endl;
182      write(pipe_write_fd, sending_message.c_str(), sending_message.size());
183      close(pipe_write_fd);
184  }
```

- void handle_show_groups()

این متد برای هندل کردن دستور ShowGroups صدا زده می شود. در آن گروه هایی که کلاینت عضو آن ها شده است نمایش داده می شود.

```
186  void Client::handle_show_groups() {
187      string result = "";
188      for(string group : groups)
189          result += group + " ";
190      printf("Joined groups: %s\n", result.c_str());
191  }
```

- void handle_router_message(std::string message)

این تابع زمانی صدا زده می شود که پیامی از روتر به کلاینت برسد. این متد پیام دریافتی را نمایش می دهد.

```
201  void Client::handle_router_message(std::string message) {
202      vector<string> message_parts = split(message, MESSAGE_DELIMITER);
203
204      cout << "incomming message: " << message_parts[ARG2] << endl;
205  }
```

دستورات سرور گروه

ایجاد و اجرای سرور گروه

برای ایجاد و سپس اجرای برنامه‌ی سرور گروه دستور زیر را در ترمینال وارد می‌کنیم. آرگومان اول آن نام گروه است و آرگومان دوم آن IP سرور است. به این ترتیب با وارد کردن این دستور گروه موردنظر ساخته می‌شود. در ادامه به بررسی دستوراتی که سرور گروه در ترمینال خود وارد می‌کند می‌پردازیم.

ورودی
`./GroupServer.out <group_name> <server_ip>`

ورودی نمونه
`./Server.out DM 1.1.2.2`

تنظیم کردن IP گروه

برای تنظیم کردن IP گروه از دستور زیر استفاده می‌کنیم. تنها آرگومان این دستور یک مقدار IP است که قرار است برای گروه تنظیم شود.

ورودی
`SetGroupIp <group_ip>`

ورودی نمونه
`SetGroupIp 1.1.3.4`

تنظیم کردن IP سرور گروه

به این منظور از دستور زیر استفاده می‌شود. تنها آرگومان این دستور IP ای است که می‌خواهد به عنوان IP سرور گروه تنظیم شود.

ورودی
`SetIp <groupserver_ip>`

ورودی نمونه
`SetIp 1.1.1.3`

اتصال به روتر

برای اینکه سرور گروه به یک روتر متصل شود از دستور زیر استفاده می‌کنیم. کافی است که در آرگومان اول این دستور پورت روتر موردنظر را وارد کنیم.

ورودی
`ConnectRouter <router_port> <server_port>`

ورودی نمونه
`ConnectRouter 8420 1.1.1.1`

اتصال به سرور

برای اینکه سرور گروه به یک سرور اصلی برنامه متصل شود از دستور زیر استفاده می‌کنیم. با از اجرای این دستور سرور گروه به سرور اصلی برنامه معرفی می‌شود.

ورودی

```
AddServer
```

ورودی نمونه

```
AddServer
```

توضیحات کد سرور گروه

فیلد ها:

- string group_name: این فیلد بیانگر نام گروه این سرور گروه است.
- string ip: این فیلد IP این سرور گروه است.
- string group_ip: این فیلد IP گروه متناظر با این سرور گروه است.
- string server_ip: این فیلد IP سرور است.
- pair<string, string> groupserver_to_server_pipe: این فیلد در اصل یک زوج مرتب است که عنصر اول و دوم آن به ترتیب پایپ‌های خواندن و نوشتن در سرور هستند.
- map<string,pair<string, string>> groupserver_to_routers_pipes: این فیلد بیانگر یک مپ است که کلید آن پورت روتر متصل و مقدار آن یک زوج مرتب از پایپ‌های خواندن و نوشتن مربوط به این روتر است.

متد ها:

- void start()

این متد اصلی سرور گروه است که در آن در یک loop منتظر دریافت دستور از stdin یا دریافت پیام از پایپ خود می‌ماند. در صورتی که از stdin دستوری دریافت کند با فراخوانی تابع handle_command این دستور هندل می‌شود و در صورتی که پیامی از پایپ خود دریافت کند، با استفاده از تابع handle_pip_message آن را هندل می‌کند. در انتهای loop نیز پایپ هایی که open شده‌اند close می‌شوند.

- std::map<int, string>add_routers_to_set(fd_set& fd,, int& max_fd)

در این متد پایپ های روترهایی که به این سرور گروه متصل شده‌اند open می‌شوند و به لیست fd هایی که روتر به آنها گوش می‌دهد اضافه می‌شوند.

```
map<int, string> GroupServer::add_routers_to_set(fd_set& fds, int& max_fd) {
    map<int, string> routers_fds;
    map<string, pair<string, string>>::iterator it;
    for (it = groupserver_to_routers_pipes.begin();
            it != groupserver_to_routers_pipes.end(); it++) {
        int router_fd = open(it->second.first.c_str(), O_RDWR);
        routers_fds.insert({router_fd, it->first});
        FD_SET(router_fd, &fds);
        max_fd = (max_fd > router_fd) ? max_fd : router_fd;
    }

    return routers_fds;
}
```

- void close_others_fds(map＜int, string＞ clients_fds)

این متد برای بسته شدن پایپ های اجزای سیستم که به این روتر متصل هستند نوشته شده است.

```
void Router::close_others_fds(map<int, string> others_fds){
    map<int, string>::iterator it;
    for (it = others_fds.begin(); it != others_fds.end(); it++)
        close(it->first);
}
```

- void handle_command(string command)

این متد بر اساس نوع دستور دریافتی متد مورد نظر برای هندل کردن آن را فراخوانی می کند.

```
void GroupServer::handle_command(string command) {
    vector<string> command_parts = split(command, COMMAND_DELIMITER);

    if (command_parts.size() < 1)
        return;

    if (command_parts[ARG0] == SET_GROUP_IP_CMD)
        handle_set_group_ip(command_parts[ARG1]);

    else if (command_parts[ARG0] == SET_GROUPSERVER_IP_CMD)
        handle_set_groupserver_ip(command_parts[ARG1]);

    else if (command_parts[ARG0] == GROUPSERVER_TO_ROUTER_CONNECT_CMD)
        handle_connect_router(command_parts[ARG1]);

    else if (command_parts[ARG0] == GROUPSERVER_TO_SERVER_CONNECT_CMD)
        handle_connect_server();

    else printf("Unknown command.\n");
}
```

- void handle_set_groupserver_ip(string groupserver_ip)

این متد IP سرور گروه را به مقدار داده شده در ورودی تنظیم می‌کند.

```
void GroupServer::handle_set_groupserver_ip(string groupserver_ip) {
    this->ip = groupserver_ip;
    printf("Group server IP changed successfuly\n");
}
```

- void handle_set_group_ip(string group_ip)

این متد IP گروه را به مقدار داده شده در ورودی تنظیم می‌کند.

```
void GroupServer::handle_set_group_ip(string group_ip) {
    this->group_ip = group_ip;
    printf("Group IP changed successfuly\n");
}
```

- void handle_connect_router(string router_port)

این متد اتصال یک روتر با پورت داده شه به سرور گروه را مدیریت می‌کند.

```
void GroupServer::handle_connect_router(string router_port) {
    // Send connection message to router.
    string router_pipe = PIPE_ROOT_PATH + string(router_port);
    int router_pipe_fd = open(router_pipe.c_str(), O_RDWR);
    string router_connect_message = string(GROUPSERVER_MESSAGE_PREFIX) + MESSAGE_DELIMITER
            + string("connect") + MESSAGE_DELIMITER + group_ip;
    write(router_pipe_fd, router_connect_message.c_str(), router_connect_message.size());
    close(router_pipe_fd);
    pair<string, string> groupserver_to_router_pipe = {(string(PIPE_ROOT_PATH) +
        ROUTER_PIPE + GROUPSERVER_PIPE + PIPE_NAME_DELIMITER + group_ip + READ_PIPE),
            (string(PIPE_ROOT_PATH) + ROUTER_PIPE + GROUPSERVER_PIPE +
                PIPE_NAME_DELIMITER + group_ip + WRITE_PIPE)};

    groupserver_to_routers_pipes.insert({router_port, groupserver_to_router_pipe});
}
```

- void handle_connect_server(string router_port)

این متد اتصال سرور به سرور گروه را مدیریت می‌کند.

```cpp
void GroupServer::handle_connect_server() {
    // Send connection message to server.
    string server_pipe = PIPE_ROOT_PATH + string(server_ip) + READ_PIPE;
    int server_pipe_fd = open(server_pipe.c_str(), O_RDWR);
    string connect_message = string(GROUPSERVER_TO_SERVER_CONNECT_MSG) +
        MESSAGE_DELIMITER + group_name + MESSAGE_DELIMITER + group_ip;
    write(server_pipe_fd, connect_message.c_str(), connect_message.size());
    close(server_pipe_fd);

    groupserver_to_server_pipe = {(string(PIPE_ROOT_PATH) + SERVER_PIPE +
        GROUPSERVER_PIPE + PIPE_NAME_DELIMITER + group_name + READ_PIPE),
        (string(PIPE_ROOT_PATH) + SERVER_PIPE + GROUPSERVER_PIPE +
        PIPE_NAME_DELIMITER + group_name + WRITE_PIPE)};
}
```

دستورات روتر

ایجاد و اجرای روتر

برای ایجاد و سپس اجرای برنامه‌ی روتر دستور زیر را در ترمینال وارد می‌کنیم. تنها آرگومان آن بیانگر IP سرور است. به این ترتیب با وارد کردن این دستور روتر موردنظر اجرا می‌شود. در ادامه به بررسی دستوراتی که می‌توان در ترمینال روتر وارد کرد می‌پردازیم.

ورودی

```
./Router.out <router_port> <server_ip>
```

ورودی نمونه

```
./Router.out 8420 1.1.2.2
```

اتصال روتر به روتر

برای اینکه روتر موردنظر به روتری دیگر متصل شود از دستور زیر استفاده می‌کنیم. توجه داریم که این اتصال از طریق یک لینک انجام می‌شود. آرگومان اول این دستور شماره پورت روتر هدف و آرگومان دوم آن نام لینکی است که می‌خواهیم بین دو روتر ایجاد شود.

ورودی

```
ConnectRouter <router_port> <link_name>
```

ورودی نمونه

```
ConnectRouter 8430 new_link
```

تغییر هزینه لینک

برای اینکه هزینه لینک بین این روتر و روتر دیگر را تغییر دهیم از دستور زیر استفاده می‌کنیم. در آرگومان اول این دستور شماره پورت روتر هدف و در آرگومان دوم هزینه موردنظر را وارد می‌کنیم. توجه داریم که در ابتدای ساخت یک لینک هزینه آن به صفر تنظیم می‌شود.

ورودی

```
ChangeCost <router_port> <cost>
```

ورودی نمونه

```
ChangeCost 8430 100
```

قطع اتصال یک روتر

در صورتی که بخواهیم اتصال این روتر را با یک روتر دیگر قطع کنیم از دستور زیر استفاده می‌کنیم. تنها آرگومان این دستور شماره پورت روتر هدف است.

ورودی

```
Disconnect <router_port>
```

ورودی نمونه

```
Disconnect 8430
```

نمایش جداول

برای نمایش جدول‌های یونیکست و مالتی‌کست یک روتر از دستور زیر استفاده می‌کنیم که در خروجی خود به ترتیب این دو جدول را نمایش می‌دهد.

ورودی
Show

ورودی نمونه
Show

توضیحات کد روتر

فیلد ها:

- string listen_port: این فیلد بیانگر شماره پورتی است که روتر به آن گوش می‌دهد. سایر اجزای سیستم نیز یک روتر را با پورت آن می‌شناسند.
- map<string, Link*> links: این فیلد در اصل یه مپ است که کلید آن نام لینک و مقدار آن اشاره‌گری به آبجکت لینک موردنظر است.
- pair<string, string> router_to_server_pipe: این فیلد در اصل یک زوج مرتب است که عنصر اول و دوم آن به ترتیب پایپ‌های خواندن و نوشتن در سرور هستند.
- map<string,pair<string, string>> clients_pipes: این فیلد بیانگر یک مپ است که کلید آن ip کلاینت متصل و مقدار آن یک زوج مرتب از پایپ‌های خواندن و نوشتن مربوط به این کلاینت است.
- map<string,pair<string, string>> group_servers_pipes: این فیلد بیانگر یک مپ است که کلید آن ip سرور گروه متصل و مقدار آن یک زوج مرتب از پایپ‌های خواندن و نوشتن مربوط به این سرور گروه است.
- map<string,vector<string>> multicast_table: این فیلد بیانگر جدول مالتی‌کست مربوط به این روتر است که به کمک یک مپ پیاده‌سازی شده است. کلید این مپ IP گروه و مقدار آن برداری است از عناصری هستند که مقصد پیام‌های ارسالی در این گروه هستند.

- unicast_table : این فیلد بیانگر جدول یونیکست مربوط به این روتر است که به کمک برداری از زوج مرتب‌ها پیاده‌سازی شده است. عنصر اول هر زوج در این بردار IP کلاینت و عنصر دوم آن شماره پورت روتری است که در کوتاه‌ترین مسیر کلاینت به این روتر قرار دارد.

متد ها:

- void start()

این متد اصلی روتر است که در آن در یک loop منتظر دریافت دستور از stdin یا دریافت پیام از سایر اجزای سیستم می‌ماند. در صورتی که از stdin دستوری دریافت کند با فراخوانی تابع handle_command این دستور هندل می‌شود و در صورتی که پیامی از سایر اجزای سیستم دریافت کند، با استفاده از تابع هندلر مربوط به آن المان آن را هندل می‌کند. در انتهای loop نیز پایپ هایی که open شده‌اند close می‌شوند.

- std::map<int, string>add_routers_to_set(fd_set& fd,, int& max_fd)

در این متد پایپ های روترهایی که به این روتر متصل شده‌اند open می‌شوند و به لیست fd هایی که روتر به آنها گوش می‌دهد اضافه می‌شوند.

```
map<int, string> Router::add_routers_to_set(fd_set& fds, int& max_fd) {
    map<int, string> routers_fds;
    map<string, Link*>::iterator it;
    for (it = links.begin(); it != links.end(); it++) {
        int router_fd = open(it->second->get_read_pipe().c_str(), O_RDWR);
        routers_fds.insert({router_fd, it->first});
        FD_SET(router_fd, &fds);
        max_fd = (max_fd > router_fd) ? max_fd : router_fd;
    }

    return routers_fds;
}
```

- std::map<int, string>add_clients_to_set(fd_set& fds, int& max_fd)

در این متد پایپ های کلاینت هایی که به این روتر متصل شده‌اند open می‌شوند و به لیست fd هایی که روتر به آنها گوش می‌دهد اضافه می‌شوند.

```
map<int, string> Router::add_clients_to_set(fd_set& fds, int& max_fd) {
    map<int, string> clients_fds;
    map<string, pair<string, string>>::iterator it;
    for (it = clients_pipes.begin(); it != clients_pipes.end(); it++) {
        int client_fd = open(it->second.second.c_str(), O_RDWR);
        clients_fds.insert({client_fd, it->first});
        FD_SET(client_fd, &fds);
        max_fd = (max_fd > client_fd) ? max_fd : client_fd;
    }

    return clients_fds;
}
```

- std::map<int, string>add_groupservers_to_set(fd_set& fds, int& max_fd)

در این متد پایپ های سرور گروه هایی که به این روتر متصل شده‌اند open می‌شوند و به لیست fd هایی که روتر به آنها گوش می‌دهد اضافه می‌شوند.

```
map<int, string> Router::add_groupservers_to_set(fd_set& fds, int& max_fd) {
    map<int, string> groupservers_fds;
    map<string, pair<string, string>>::iterator it;
    for (it = group_servers_pipes.begin(); it != group_servers_pipes.end(); it++) {
        int groupserver_fd = open(it->second.second.c_str(), O_RDWR);
        groupservers_fds.insert({groupserver_fd, it->first});
        FD_SET(groupserver_fd, &fds);
        max_fd = (max_fd > groupserver_fd) ? max_fd : groupserver_fd;
    }

    return groupservers_fds;
}
```

- void close_others_fds(map<int, string> clients_fds)

این متد برای بسته شدن پایپ های اجزای سیستم که به این روتر متصل هستند نوشته شده است.

```
void Router::close_others_fds(map<int, string> others_fds){
    map<int, string>::iterator it;
    for (it = others_fds.begin(); it != others_fds.end(); it++)
        close(it->first);
}
```

- void handle_command(string command)

این متد بر اساس نوع دستور دریافتی متد مورد نظر برای هندل کردن آن را فراخوانی می کند.

```
void Router::handle_command(string command) {
    vector<string> command_parts = split(command, COMMAND_DELIMITER);

    if (command_parts.size() < 1)
        return;

    else if (command_parts[ARG0] == ROUTER_TO_ROUTER_CONNECT_CMD)
        handle_connect_router(command_parts[ARG1], command_parts[ARG2]);

    else if (command_parts[ARG0] == CHANGE_COST_CMD)
        handle_change_cost(command_parts[ARG1], command_parts[ARG2]);

    else if (command_parts[ARG0] == DISCONNECT_LINK_CMD)
        handle_disconnect(command_parts[ARG1]);

    else if (command_parts[ARG0] == SHOW_CMD)
        handle_show();

    else printf("Unknown command.\n");
}
```

- void handle_connect_router(string router_port, string link_name)

این متد برای مدیریت اتصال دو روتر به یکدیگر نوشته شده است. در ورودی خود شماره پورت روتر مقصد و نام لینک را دریافت می‌کند. ابتدا روترهای خواندن و نوشتن مربوط به این اتصال را می‌سازد و سپس یک آبجکت لینک می‌سازد و اشاره‌گر به آن را همراه با نام آن در فیلد links درج می‌کند. در ادامه یک پیام مبتنی بر اتصال دو روتر به روتر هدف می‌فرسد که آن روتر نیز باخبر شده و از پایپ‌های مربوطه استفاده کند.

```
void Router::handle_connect_router(string router_port, string link_name) {
    make_router_router_pipes(router_port);

    string read_pipe = string(PIPE_ROOT_PATH) + router_port + '-' + listen_port;
    string write_pipe = string(PIPE_ROOT_PATH) + listen_port + '-' + router_port;

    printf("Router with port %s connects.\n", router_port.c_str());

    Link* link = new Link(link_name, read_pipe, write_pipe);

    links.insert({router_port, link});

    string message = string(ROUTER_MESSAGE_PREFIX) + MESSAGE_DELIMITER;
    message += string("connect") + MESSAGE_DELIMITER;
    message += listen_port + MESSAGE_DELIMITER;
    message += link_name;

    string pipe_path = string(PIPE_ROOT_PATH) + router_port;
    int connection_pipe_fd = open(pipe_path.c_str(), O_RDWR);
    write(connection_pipe_fd, message.c_str(), strlen(message.c_str()) + 1);
    close(connection_pipe_fd);
}
```

- void handle_connection_message(string pipe_message)

با استفاده از این متد در صورتی که روتر در پایپ خود پیامی که با connect شروع شود دریافت کند با بررسی سایر اجزای سیستم به المانی که می‌خواهد به آن وصل شود پی می‌برد و این اتصال را می‌پذیرد.

```
void Router::handle_connection_message(string pipe_message) {
    vector<string> message_parts = split(pipe_message, MESSAGE_DELIMITER);

    if (message_parts[ARG0] == ROUTER_MESSAGE_PREFIX && message_parts[ARG1] == "connect")
        accept_router_connect(message_parts[ARG2], message_parts[ARG3]);

    if (message_parts[ARG0] == CLIENT_MESSAGE_PREFIX && message_parts[ARG1] == "connect")
        accept_client_connect(message_parts[ARG2]);

    if (message_parts[ARG0] == GROUPSERVER_MESSAGE_PREFIX && message_parts[ARG1] == "connect")
        accept_groupserver_connect(message_parts[ARG2]);
}
```

- void accept_router_connect(string router_port, string link_name)

با استفاده از این متد روتر اتصال یک روتر را می‌پذیرد و لینک مربوط به آن اتصال را ساخته و در فیلد links درج می‌کند.

```
void Router::accept_router_connect(string router_port, string link_name) {
    string read_pipe = string(PIPE_ROOT_PATH) + router_port + '-' + listen_port;
    string write_pipe = string(PIPE_ROOT_PATH) + listen_port + '-' + router_port;

    printf("Router with port %s connects.\n", router_port.c_str());

    Link* link = new Link(link_name, read_pipe, write_pipe);

    links.insert({router_port, link});
}
```

- void make_router_router_pipes(string router_port)

با استفاده از این متد روتر پایپ‌های خواندن و نوشتن مربوط به اتصال خود با یک روتر را می‌سازد.

```
void Router::make_router_router_pipes(string router_port) {
    string read_pipe = string(PIPE_ROOT_PATH) + router_port + '-' + listen_port;
    string write_pipe = string(PIPE_ROOT_PATH) + listen_port + '-' + router_port;
    unlink(read_pipe.c_str());
    mkfifo(read_pipe.c_str(), READ_WRITE);
    unlink(write_pipe.c_str());
    mkfifo(write_pipe.c_str(), READ_WRITE);
}
```

- void accept_client_connect(string client_ip)

با استفاده از این متد روتر اتصال یک کلاینت را می‌پذیرد. سپس پایپ‌های مربوط به خواندن و نوشتن این اتصال را می‌سازد و در clients_pipes درج می‌کند.

```
void Router::accept_client_connect(string client_ip) {
    printf("client with ip %s connects.\n", client_ip.c_str());

    pair<string, string> client_pipe = {(string(PIPE_ROOT_PATH) +
        ROUTER_PIPE + CLIENT_PIPE + PIPE_NAME_DELIMITER + client_ip + READ_PIPE),
        (string(PIPE_ROOT_PATH) + ROUTER_PIPE + CLIENT_PIPE +
        PIPE_NAME_DELIMITER + client_ip + WRITE_PIPE)};

    unlink(client_pipe.first.c_str());
    mkfifo(client_pipe.first.c_str(), READ_WRITE);
    unlink(client_pipe.second.c_str());
    mkfifo(client_pipe.second.c_str(), READ_WRITE);

    clients_pipes.insert({client_ip, client_pipe});
}
```

- void accept_groupserver_connect(string groupserver_ip)

با استفاده از این متد روتر اتصال یک سرور گروه را می‌پذیرد. سپس پایپ‌های مربوط به خواندن و نوشتن این اتصال را می‌سازد و در group_servers_pipes درج می‌کند.

```
void Router::accept_groupserver_connect(string groupserver_ip) {
    printf("groupserver with ip %s connects.\n", groupserver_ip.c_str());

    pair<string, string> group_server_pipe = {(string(PIPE_ROOT_PATH) +
    ROUTER_PIPE + GROUPSERVER_PIPE + PIPE_NAME_DELIMITER + PIPE_NAME_DELIMITER +
        groupserver_ip + READ_PIPE),
        (string(PIPE_ROOT_PATH) + ROUTER_PIPE + GROUPSERVER_PIPE +
        PIPE_NAME_DELIMITER + groupserver_ip + WRITE_PIPE)};

    unlink(group_server_pipe.first.c_str());
    mkfifo(group_server_pipe.first.c_str(), READ_WRITE);
    unlink(group_server_pipe.second.c_str());
    mkfifo(group_server_pipe.second.c_str(), READ_WRITE);

    group_servers_pipes.insert({groupserver_ip, group_server_pipe});
}
```

- void handle_change_cost(string router_port, string cost)

با استفاده از این متد هزینه لینک اتصال بین این روتر و روتری که شماره پورت آن در آرگومان این تابع داده شده به مقدار موردنظر تنظیم می‌شود.

```
void Router::handle_change_cost(string router_port, string cost) {
    Link* link = links.find(router_port)->second;
    string write_pipe = link->get_write_pipe();

    string message = string(ROUTER_MESSAGE_PREFIX) + MESSAGE_DELIMITER;
    message += string("change_cost") + MESSAGE_DELIMITER;
    message += listen_port + MESSAGE_DELIMITER;
    message += cost;

    int write_pipe_fd = open(write_pipe.c_str(), O_RDWR);
    write(write_pipe_fd, message.c_str(), strlen(message.c_str()) + 1);
    close(write_pipe_fd);

    link->change_cost(stod(cost));
    printf("Cost of link with router %s changed to %s.\n", router_port.c_str(),
        cost.c_str());
}
```

- void handle_disconnect(string router_port)

با استفاده از این متد اتصال بین این روتر و روتر دیگر قطع می‌شود. لینک مربوط به این اتصال نیز از links حذف می‌شود.

```
void Router::handle_disconnect(string router_port) {
    Link* link = links.find(router_port)->second;
    string write_pipe = link->get_write_pipe();

    string message = string(ROUTER_MESSAGE_PREFIX) + MESSAGE_DELIMITER;
    message += string("disconnect") + MESSAGE_DELIMITER;
    message += listen_port;

    int write_pipe_fd = open(write_pipe.c_str(), O_RDWR);
    write(write_pipe_fd, message.c_str(), strlen(message.c_str()) + 1);
    close(write_pipe_fd);

    links.erase(router_port);
    printf("Link with router %s diconnected.\n", router_port.c_str());
}
```

- void show_unicast_table()

این متد برای چاپ جدول unicast مربوط به این روتر نوشته شده است.

```
void Router::show_unicast_table() {
    cout << "******   " << "Unicast Table" << "   ******" << endl;
    cout << "Client_IP" << " " << "Router_Port" << endl;
    for (auto & row : unicast_table) {
        cout << row.first << "    " << row.second << endl;
    }
    cout << endl;
}
```

- void show_multicast_table()

این متد برای چاپ جدول multicast مربوط به این روتر نوشته شده است.

```
void Router::show_multicast_table() {
    cout << "******   " << "Multicast Table" << "   ******" << endl;
    cout << "Group_IP" << " " << "Destinations" << endl;
    for(auto it = multicast_table.cbegin(); it != multicast_table.cend(); ++it) {
        std::cout << it->first << "  ";
        size_t length = it->second.size();
        for (size_t i = 0; i < length; i++) {
            cout << it->second[i];
            if (i == length - 1)
                cout << endl;
            else
                cout << ", ";
        }
    }
}
```

- void handle_show()

این متد برای فراخوانی دو متد قبلی برای مدیریت دستور نمایش نوشته شده است.

```
void Router::handle_show() {
    show_unicast_table();
    show_multicast_table();
}
```

- void handle_router_message(string router_message, string sender_router_port)

این متد برای مدیریت پیام‌هایی که از روتر دیگری رسیده است نوشته شده است.

```
void Router::handle_router_message(string router_message, string sender_router_port) {
    vector<string> message_parts = split(router_message, MESSAGE_DELIMITER);

    if (message_parts[ARG1] == "change_cost")
        accept_router_change_cost(message_parts[ARG2], message_parts[ARG3]);

    else if (message_parts[ARG1] == "disconnect")
        accept_router_disconnect(message_parts[ARG2]);

    else if (message_parts[ARG0] == SEND_MSG)
        handle_send(router_message, sender_router_port);

    else printf("Unknown message.\n");
}
```

- void handle_server_message(string server_message)

این متد برای مدیریت پیام‌هایی که از سرور رسیده است نوشته شده است.

```
void Router::handle_server_message(string server_message) {
    vector<string> message_parts = split(server_message, MESSAGE_DELIMITER);

    if (message_parts[ARG0] == "join")
        handle_join_update(message_parts[ARG1], message_parts[ARG2]);

    else if (message_parts[ARG0] == "leave")
        handle_leave_update(message_parts[ARG1], message_parts[ARG2]);

    else printf("Unknown message.\n");
}
```

- void accept_router_change_cost(string router_port, string cost)

این متد برای پذیرش پیام تغییر هزینه لینک که از سوی روتر هدف رسیده نوشته شده است.

```
void Router::accept_router_change_cost(string router_port, string cost) {
    Link* link = links.find(router_port)->second;
    link->change_cost(stod(cost));

    printf("Cost of link with router %s changed to %s.\n", router_port.c_str(), cost.c_str());
}
```

- void accept_router_change_cost(string router_port, string cost)

این متد برای پذیرش پیام قطع اتصال که از سوی روتر هدف رسیده نوشته شده است.

```
void Router::accept_router_disconnect(string router_port) {
    links.erase(router_port);
    printf("Link with router %s diconnected.\n", router_port.c_str());
}
```

- void handle_client_message(string client_message)

این متد برای مدیریت پیام‌هایی که از یک کلاینت رسیده است نوشته شده است.

```
void Router::handle_client_message(string client_message) {
    vector<string> message_parts = split(client_message, MESSAGE_DELIMITER);

    if (message_parts[ARG0] == SEND_MSG)
        handle_send(client_message, "");

    else printf("Unknown message.\n");
}
```

- void handle_send(string message, string sender_router_port)

این متد برای ارسال یک پیام نوشته شده است. ابتدا نام گروه از آرگومان اول دستور استخراج می‌شود. سپس سطر مربوط به آن را در جدول مالتی‌کست روتر پیدا می‌کنیم. حال به پایپ تمامی المان‌هایی که در ستون دوم این سطر قرار گرفته‌اند که می‌توانند روتر یا کلاینت باشند (در اتصال مستقیم) پیام مربوطه را می‌فرستیم.

```
void Router::handle_send(string message, string sender_router_port) {
    vector<string> message_parts = split(message, MESSAGE_DELIMITER);
    string group_name = message_parts[ARG1];

    if (multicast_table.find(group_name) == multicast_table.end())
        printf("group does not exists");
    else {
        for (string port : multicast_table[group_name]) {
            vector<string> port_parts = split(port, PORT_DELIMITER);
            if (port_parts[1] == "c") {
                string client_pipe = clients_pipes[port_parts[0]].first;
                int client_fd = open(client_pipe.c_str(), O_RDWR);
                write(client_fd, message.c_str(), message.size());
                close(client_fd);
            }
            else if (port_parts[1] == "r" && sender_router_port != port_parts[0]) {
                string router_pipe = links[port_parts[0]]->get_write_pipe();
                int router_fd = open(router_pipe.c_str(), O_RDWR);
                write(router_fd, message.c_str(), message.size());
                close(router_fd);
            }
        }
    }
}
```

- void handle_join_update(string client_ip, string group_ip)

این متد به این منظور نوشته شده است که هرگاه یک کلاینت عضو یک گروه شد سطر مربوط به آن گروه در جدول مالتی‌کست آپدیت شود.

```
void Router::handle_join_update(string client_ip, string group_ip) {
    map<string, vector<string>>::iterator it = multicast_table.find(group_ip);
    pair<string, string> dest = find_destination(client_ip);
    string value = dest.first + "-" + dest.second;
    if (it != multicast_table.end()) {
        it->second.push_back(value);
    }
    else {
        multicast_table.insert({group_ip, vector<string> {value}});
    }
    printf("join update\n");
}
```

- void handle_join_update(string client_ip, string group_ip)

این متد به این منظور نوشته شده است که هرگاه یک کلاینت عضو یک گروه شد سطر مربوط به آن گروه در جدول مالتی‌کست آپدیت شود.

```
void Router::handle_join_update(string client_ip, string group_ip) {
    map<string, vector<string>>::iterator it = multicast_table.find(group_ip);
    pair<string, string> dest = find_destination(client_ip);
    string value = dest.first + "-" + dest.second;
    if (it != multicast_table.end()) {
        it->second.push_back(value);
    }
    else {
        multicast_table.insert({group_ip, vector<string> {value}});
    }
    printf("join update\n");
}
```

- void handle_leave_update(string client_ip, string group_ip)

این متد به این منظور نوشته شده است که هرگاه یک کلاینت یک گروه را ترک کرد سطر مربوط به آن گروه در جدول مالتی‌کست آپدیت شود.

```
void Router::handle_leave_update(string client_ip, string group_ip) {
    map<string, vector<string>>::iterator it = multicast_table.find(group_ip);
    pair<string, string> dest = find_destination(client_ip);
    string value = dest.first + "-" + dest.second;
    it->second.erase(std::find(it->second.begin(), it->second.end(), value));
    printf("leave update\n");
}
```

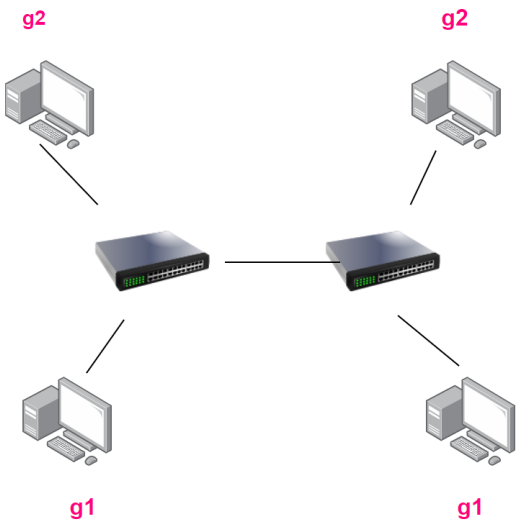
- pair<string, string> Router::find_destination(string client_ip)

این متد به این منظور نوشته شده است برای کلاینتی که IP آن داده شده است المانی که باید در جدول مالتی‌کست در سطر مربوط به یک گروه درج شود را پیدا کنیم. به این منظور از جدول یونیکست استفاده کردیم تا یا استفاده از آن جدول مالتی‌کست را پر کنیم. (الگوریتم IGMP)

```
pair<string, string> Router::find_destination(string client_ip) {
    pair<string, string> dest;
    for (size_t i = 0; i < unicast_table.size(); i++) {
        if (unicast_table[i].first == client_ip) {
            string target = unicast_table[i].second;
            if (target == listen_port)
                dest = make_pair(client_ip, "c");
            else
                dest = make_pair(target, "r");
            break;
        }
    }
    return dest;
}
```

تست

برای تست پروژه یک شبکه شامل ۲ روتر و ۴ کلاینت استفاده می کنیم (توپولوژی شبکه مربوطه در شکل زیر آمده است).



در ابتدا سرور را اجرا می کنیم و در ادامه دو روتر را اجرا می کنیم که دارای پورت های ۸۴۲۰ و ۸۴۳۰ هستند. سپس آن ها را به هم وصل می کنیم. در ادامه ۴ کلاینت با نام های c1 تا c4 اجرا می کنیم که به ترتیب دارای ip های ۱.۱.۱.۱، ۱.۱.۱.۲، ۱.۱.۱.۳ و ۱.۱.۱.۴ هستند. کلاینت اول و سوم به روتر اول متصل می شوند و کلاینت دوم و چهارم به روتر دوم. سپس دو کلاینت اول به گروه g1 و دو کلاینت دوم به گروه g2 درخواست join می دهند. حال کلاینت اول که عضو گروه g1 است یک پیام دارای محتوای "salam" به گروه g2 می فرستد و

کلاینت سوم و چهارم که عضو این گروه هستند آن را دریافت می کنند. نتایج این تست شامل موارد گفته شده و جدول های فروادینگ unicast و multicast در زیر آمده است.

```
amin@amin-PC:~/University/Semester6/CN/projects/CA3/multicast-protocol$ ./Client.out c1 10.10.10.10 1.1.1.1 8420
> JoinGroup g1
received command: JoinGroup g1
send to pipe: build/_s_c_c1_w
-------------- event --------------
> SendMessage salam g2
received command: SendMessage salam g2
message: Send%g2%salam
-------------- event --------------
>
```

کلاینت شماره ۱

```
amin@amin-PC:~/University/Semester6/CN/projects/CA3/multicast-protocol$ ./Client.out c2 10.10.10.10 1.1.1.2 8430
> JoinGroup g1
received command: JoinGroup g1
send to pipe: build/_s_c_c2_w
-------------- event --------------
>
```

کلاینت شماره ۲

```
amin@amin-PC:~/University/Semester6/CN/projects/CA3/multicast-protocol$ ./Client.out c3 10.10.10.10 1.1.1.3 8420
> JoinGroup g2
received command: JoinGroup g2
send to pipe: build/_s_c_c3_w
-------------- event --------------
> incomming message: salam
-------------- event --------------
>
```

کلاینت شماره ۳

```
amin@amin-PC:~/University/Semester6/CN/projects/CA3/multicast-protocol$ ./Client.out c4 10.10.10.10 1.1.1.4 8430
> JoinGroup g2
received command: JoinGroup g2
send to pipe: build/_s_c_c4_w
-------------- event --------------
> incomming message: salam
-------------- event --------------
>
```

کلاینت شماره ۴

```
> Show
received command: Show
******   Unicast Table   ******
Client_IP Router_Port
1.1.1.1   8420
1.1.1.2   8430
1.1.1.3   8420
1.1.1.4   8430

******   Multicast Table   ******
Group_IP Destinations
g1  1.1.1.1-c, 8430-r
g2  1.1.1.3-c, 8430-r
--------------- event ---------------
```

جدول فروارديڤنگ unicast و multicast برای روتر شماره ۱

```
> Show
received command: Show
******   Unicast Table   ******
Client_IP Router_Port
1.1.1.1   8420
1.1.1.2   8430
1.1.1.3   8420
1.1.1.4   8430

******   Multicast Table   ******
Group_IP Destinations
g1  8420-r, 1.1.1.2-c
g2  8420-r, 1.1.1.4-c
--------------- event ---------------
>
```

جدول فروارديڤنگ unicast و multicast برای روتر شماره ۲

نتیجه‌گیری

در این پروژه یک شبکه متشکل از کلاینت‌ها و روترها را شبیه‌سازی نمودیم و به پیاده‌سازی گروه و پروتکل مالتی‌کست پرداختیم. همچنین به طور گسترده با نحوه‌ی کار با پایپ‌های با نام در زبان C++ برای ایجاد یک برنامه multi-processing آشنا شدیم.