Arcade Documentation

Roman Lopez, Stephane Fievez, Alexandre De Freitas Martins

Mars / Avril 2023

0.1 Introduction

Bienvenue dans la documentation du projet myTeams, ici vous trouverez toutes les informations nécessaires pour comprendre le fonctionnement de ce projet.

0.1.1 Le myTeams c'est quoi ?

Le myTeams est un projet de groupe qui consiste à créer un client et un serveur de messagerie instantanée en C. Ce projet est réalisé dans le cadre de la formation de l'Epitech.

0.2 Partie 1 : le Serveur

0.2.1 Le serveur c'est quoi ?

Le serveur est un programme qui permet de gérer les clients. Il est composé de plusieurs fonctions qui permettent de gérer les clients, les salons, les messages, les fichiers, etc.

0.2.2 Les différents protocoles utilisés

Le serveur utilise plusieurs protocoles pour communiquer avec les clients. Ces protocoles sont les suivants :

Protocal	Description	Code
HELP	Affiche la liste des commandes disponibles	214
LOGIN	set le user name utilisé par le client	220
LOGOUT	Permet de se déconnecter du serveur	221
USERS	Affiche la liste des utilisateurs qui existe sur le serveur	332
USER	Affiche les informations de l'utilisateur indiqué	331
SEND	Envoie un message a l'utilisateur indiqué	250
MESSAGES	Affiche la liste des messages de l'utilisateur indiqué	330
SUBSCRIBE	Permet de s'abonner événements d'une team et de ses sous-répertoires	230
SUBSCRIBED	Affiche la liste des teams dont l'utilisateur est abonné	231
UNSUBSCRIBE	Permet de se désabonner d'une team	232
USE	Permet de sélectionner une team pour executé une commande	225
CREATE	Permet de créer une team/chanel/thread/échanges	320
LIST	Affiche la liste des teams/channels/threads/échanges	150
INFO	Affiche les informations d'une team/channel/thread/échanges	151
ERROR	Affiche un message d'erreur	500

0.2.3 Les fonctions du serveur

Le Select

Le serveur utilise le "select" pour gérer les clients. Le "select" permet de gérer plusieurs clients en même temps. Il permet de gérer les entrées et sorties de chaque clients. Il permet aussi de gérer les déconnexions des clients.

```
void server_select(server_t *s)
{
    FD_ZERO(&s->read_fds);
    FD_SET(s->socket_fd, &s->read_fds);
    s->timeout.tv_sec = 0;
    s->timeout.tv_usec = 10000;
    s->r_select = select(s->socket_fd + 1, &s->read_fds, NULL, NULL,
    &s->timeout);
}

void client_select(client_t *c)
{
    FD_ZERO(&c->read_fds);
    FD_SET(c->cl_fd, &c->read_fds);
    c->timeout.tv_sec = 0;
    c->timeout.tv_usec = 10000;
    c->r_select = select(c->cl_fd + 1, &c->read_fds, NULL, NULL,
    &c->timeout);
}
```

Pour ajouter un client au select il faut utiliser la fonction suivante :

```
void handle_new_client(server_t *s)
{
    int client_socket = accept(s->socket_fd,
    (struct sockaddr *)&s->socket_address,
    (socklen_t *)&s->socket_address_len);
    if (client_socket < 0) {
        perror("accept"); return;
    }
    int actual_client = 0;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (s->clients[i].cl_fd == NO_SOCKET) {
            actual_client = i;
            s->clients[i] = create_client(s->port, client_socket);
            break;
        }
    }
    printf("Connection from %s:%d\n",
    inet_ntoa(s->clients[actual_client].socket_address.sin_addr),
    ntohs(s->clients[actual_client].socket_address.sin_port));
    write(s->clients[actual_client].cl_fd,
    "220 Welcome to my_teams\r\n", 23);
}
```

La structure du serveur

La structure contenant les informations du serveur est la suivante :

```
typedef struct server_s {
    int port;
    int r_select;
    int socket_fd;
    void *handler;
    fd_set read_fds;
    int socket_address_len;
    struct timeval timeout;
    struct sockaddr_in socket_address;
    struct commands_s *commands;

    struct client_s clients[MAX_CLIENTS];

    struct database_s *database;
} server_t;
```

La selection des commandes

La selection des commandes est gérée par la fonction suivante :

```
void fill_server(server_t *server, int server_port)
{
    server->port = server_port; server->r_select = 0;
    server->socket_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server->socket_fd < 0) {
        perror("socket_fd"); exit(84);
    }
    char *fct_names[] = {"/help", "/login", "/logout", "/users", "/user", "/send",
    "/messages", "/subscribe", "/subscribed", "/unsubscribe", "/use", "/create",
    "/list", "/info", NULL};
    void *fct_pointers[] = {&help, &login, &logout, &users, &user, &my_send,
    &messages, &subscribe, &subscribed, &unsubscribe, &use, &create,
    &list, &info, NULL};
    server->commands = fill_map(fct_names, fct_pointers);
    for (int i = 0; i < MAX_CLIENTS; i++) server->clients[i].cl_fd = NO_SOCKET;
    FD_ZERO(&server->read_fds);
    FD_SET(server->socket_fd, &server->read_fds);
    server->timeout.tv_sec = 0; server->timeout.tv_usec = 0;
    server->socket_address = create_socket_address(server_port);
    server->socket_address_len = sizeof(server->socket_address);
    server->database = create_database();
}
```

Toutes les commandes sont stockées dans un tableau de pointeurs de fonctions. Toutes les fonction de commande ont le même prototype et sont dans le dossier src/commands.

Exemple de fonction de commande

Voici un exemple, la fonction de la commande /login qui permet de se connecter au serveur :

```
void login(server_t *server, client_t *client)
{
    (void) client;
    (void) server;
    if (client->is_logged == true) {
        send(client->cl_fd, "You are already logged in.\r", 27, MSG_NOSIGNAL);
        return;
    }
    if (server->database->user->username != client->username) {
        send(client->cl_fd, "Username is incorrect.\r", 23, MSG_NOSIGNAL);
        return;
    }
    if (client->is_logged == false) {
        client->uuid = my_uuid();
        send(client->cl_fd, "You are now logged in.\r", 23, MSG_NOSIGNAL);
        client->is_logged = true;
    }
}
```

0.2.4 Les fonctions du client

La structure du client

La structure contenant les informations du client est la suivante :

```
typedef struct client_s {
    int srv_fd;
    int r_select;
    char *buffer;
    char *username;
    bool logged_in;
    fd_set read_fds;
    size_t len_buffer;
    int socket_address_len;
    struct timeval timeout;
    struct sockaddr_in socket_address;
    struct commands_s *commands;
    struct commands_lines_s *commands_lines;
} client_t;
```

La création du client

La création du client est gérée par la fonction suivante :

```
struct client_s create_client(char *ip, int port)
{
    client_t client;
    client.srv_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (client.srv_fd < 0) {
        printf("socket\n"); exit(84);
    }
    client.r_select = 0;
    client.len_buffer = 2048;
    client.buffer = malloc(sizeof(char) * client.len_buffer);
    FD_ZERO(&client.read_fds); FD_SET(client.srv_fd, &client.read_fds);
    client.timeout.tv_sec = 0; client.timeout.tv_usec = 0;
    client.socket_address = create_socket_address(port);
    client.socket_address_len = sizeof(client.socket_address);
    if (inet_pton(AF_INET, ip, &client.socket_address.sin_addr) <= 0) {
        printf("inet_pton"); exit(84);
    }
    if (connect(client.srv_fd, (struct sockaddr *)&client.socket_address,
    client.socket_address_len) < 0) {
        printf("connect\n"); exit(84);
    }
    return client;
}
```

La selection des commandes

Pour la selection des commandes, nous avons utilisé une map de fonctions. La map est gérée par la fonction suivante :