B4 - Network Programming

B-NWP-400

MyTeams Documentation

Styled RFC Documentation

MyTeams

How to run the project

After cloning the project, you need run the following commands to compile the project:

```
Terminal
~/B-NWP-400> make
```

Add the following environment variables to your shell:

```
Terminal
~/B-NWP-400> export LD_LIBRARY_PATH=$(pwd)/libs/myteams/
```

Then, you can run the server with the following command:

```
Terminal
~/B-NWP-400> ./myteams_server [port]
```

And connect to the server using the client:

```
Terminal
~/B-NWP-400> ./myteams_cli [ip] [port]
```

How to debug

You can use the following command to compile the project with the debug flags:

```
Terminal
~/B-NWP-400> make debug
```

Then you can run the same binaries as before with the `-debug` suffix, here's an example using `gdb`:

```
Terminal
~/B-NWP-400> gdb ./myteams_server-debug
~/B-NWP-400> gdb ./myteams_cli-debug
```

You can also use `valgrind` to check for memory leaks:

```
Terminal
~/B-NWP-400> valgrind ./myteams_server-debug [port]
~/B-NWP-400> valgrind ./myteams_cli-debug [ip] [port]
```

You can also use the `--leak-check=full` flag to get more information about the memory leaks.

Or use the vs-code launch configurations to debug the project with the graphical interface.

RFC Documentation

This is a styled RFC documentation, you can find the raw RFC file in the project repository, under the `doc/RFC` directory.

Independent Submission

Category: Informational

April 2024

A1ex, (h)Adam, Zowks

Status of this Memo

This memo is the specification of the my_teams project. Distribution of this memo is unlimited.
This Informational memo is published for the general information of the Internet community, and does not represent an Internet community consensus or recommendation.
Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://github.com/EpitechPromo2027/B-NWP-400-LIL-4-1-myteams-alexandre.barberis.

Copyright Notice

This document is not subject to copyright. It can be modified, and derivative works of it can be created and published.

Table of Contents

- MyTeams
  - How to run the project
  - How to debug
  - RFC Documentation
    - Status of this Memo
    - Copyright Notice
    - Table of Contents
    - 2. Introduction
    - 3. Client
      - 3.1. Overview
      - 3.2. Commands
      - 3.3. Sending Packets
      - 3.4. Client Packet Types
      - 3.5. Receiving Packets

- 4. Server
  - 4.1. Overview
  - 4.2. Receiving Packets
  - 4.3. Sending Packets
  - 4.4. Server Packet Types
- 5. Handbook
  - 5.1. Create a packet
  - 5.2. Handle a command
  - 5.3. Handle a client packet
  - 5.4. Handle a server packet
- 6. Security Considerations
- 7. Author's Address

2. Introduction

The my_teams project is a simple communication application that allows users to create teams, send messages, and interact with other users. The application is composed of a server and a CLI client.
The server is responsible for managing the teams, users, and messages.
It listens for incoming connections from clients and handles the communication between them.
The CLI client is a simple command-line interface that allows users to interact with the server.
It provides commands to create teams, send messages, and join teams, among others.

3. Client

3.1. Overview The CLI client must be able to connect to the server.
Therefore, it need the server's IP address and port number as arguments when started.
The client is composed of multiple components:
The CLI parser is responsible for parsing the user's input and converting it into commands that can be used by the command handler (see 3.2.).
The command handler is responsible for executing the commands received from the CLI parser. It creates the appropriate packet and sends it to the server.
The packet handler is responsible for receiving packets from the server and process them into a format that can be displayed to the user.

3.2. Commands The client supports the following commands:

- `/help`: Show help message.
- `/login "username"`: Log in with the given username.
- `/logout`: Disconnect the client from the server.
- `/users`: Get the list of all users.
- `/user "user_uuid"`: Get the informations of the requested user.
- `/send "user_uuid" "message"`: Send a message to specified user.
- `/messages "user_uuid"`: Get all messages exchanged with the specified user.
- `/subscribe "team_uuid"`: Subscribe to the specified team events.
- `/unsubscribe "team_uuid"`: Unsubscribe from the specified team.
- `/subscribed ?"team_uuid"`: Get the list of all subscribed teams, or list all users subscribed to a team.
- `/use ?"team_uuid|channel_uuid|thread_uuid"`: Reset the command context, or set the command context to `.._uuid`.

- `/create`: Based on the context, create the sub resource.
- `/list`: Based on the context, list all the sub resources.
- `/info`: Based on the context, list the current resource informations.

Commands are parsed by the CLI parser and sent to the command handler in the form of a structure containing the command and its arguments:

```
typedef struct {
    string_t *name;
    vec_t *arguments;
} command_t;
```

The command handler then processes the command and creates the appropriate packet to send to the server.

3.3. Sending Packets The client sends packets to the server using the following format:

```
typedef struct {
    packet_type_t type;
    ... // packet specific fields
} ..._packet_t;
```

Structures are named as follow: `<cl|sv>_(packet_type)_packet_t`
Where packet_type is the type of packet being sent (see 3.4.).
And `<cl|sv>` is the origin of the packet *(client or server)*.
This structures are shared between the client and the server along side with encoding and decoding functions to convert them to and from a binary format.

3.4. Client Packet Types The client uses the following packet types:

```
- PACKET_TYPE_CL_USER_LOGGED_IN
- PACKET_TYPE_CL_USER_LOGGED_OUT
- PACKET_TYPE_CL_GET_ALL_USERS
- PACKET_TYPE_CL_GET_USER
- PACKET_TYPE_CL_SEND_PRIVATE_MESSAGE
- PACKET_TYPE_CL_GET_ALL_MESSAGES
- PACKET_TYPE_CL_SUBSCRIBE_TO_TEAM
- PACKET_TYPE_CL_UNSUBSCRIBE_FROM_TEAM
- PACKET_TYPE_CL_LIST_SUBSCRIBED_TEAMS
- PACKET_TYPE_CL_LIST_USERS_SUBSCRIBED_TO_TEAM
- PACKET_TYPE_CL_CREATE_TEAM
- PACKET_TYPE_CL_CREATE_CHANNEL
- PACKET_TYPE_CL_CREATE_THREAD
- PACKET_TYPE_CL_CREATE_REPLY
- PACKET_TYPE_CL_LIST_TEAMS
- PACKET_TYPE_CL_LIST_CHANNELS
- PACKET_TYPE_CL_LIST_THREADS
- PACKET_TYPE_CL_LIST_REPLIES
- PACKET_TYPE_CL_GET_CURRENT_USER_INFO
- PACKET_TYPE_CL_GET_TEAM_INFO
- PACKET_TYPE_CL_GET_CHANNEL_INFO
- PACKET_TYPE_CL_GET_THREAD_INFO
```

3.5. Receiving Packets The client receives packets from the server using the same format as the packets sent to the server (see 3.3.).
Packets are decoded and processed by the packet handler, which then displays the information to the user.

4. Server

4.1. Overview The server is responsible for managing the teams, users, messages, etc.
It listens for incoming connections from clients and handles the communication between them.
The server receives packets from the clients, decode them and do the appropriate action(s) based on the packet type.
Send packets to the clients in response to their requests.
Or broadcast packets to all clients connected to the server.
Data is persistent and stored in custom databases.

4.2. Receiving Packets The server receives packets from the clients using the same format as the packets sent to the server (see 3.3.).
Packets are decoded and processed by the server, which then performs the appropriate action(s) based on the packet type.

4.3. Sending Packets The server sends packets to the clients using the same format as the packets sent to the server (see 3.3.).
Packets are encoded and sent to the clients in response to their requests or broadcasted to all clients connected to the server.

```
- PACKET_TYPE_SV_USER_LOGGED_IN
- PACKET_TYPE_SV_USER_LOGGED_OUT
- PACKET_TYPE_SV_DISCONNECTION_SUCCESS
- PACKET_TYPE_SV_GET_ALL_USERS
- PACKET_TYPE_SV_GET_USER
- PACKET_TYPE_SV_SEND_PRIVATE_MESSAGE
- PACKET_TYPE_SV_GET_ALL_PRIVATE_MESSAGE
- PACKET_TYPE_SV_SUBSCRIBE_TO_TEAM
- PACKET_TYPE_SV_UNSUBSCRIBE_FROM_TEAM
- PACKET_TYPE_SV_LIST_SUBSCRIBED_TEAMS
- PACKET_TYPE_SV_LIST_USERS_SUBSCRIBED_TO_TEAM
- PACKET_TYPE_SV_TEAM_CREATED
- PACKET_TYPE_SV_CREATED_TEAM_SUCCESSFULLY
- PACKET_TYPE_SV_CHANNEL_CREATED
- PACKET_TYPE_SV_CREATED_CHANNEL_SUCCESSFULLY
- PACKET_TYPE_SV_THREAD_CREATED
- PACKET_TYPE_SV_CREATED_THREAD_SUCCESSFULLY
- PACKET_TYPE_SV_REPLY_CREATED
- PACKET_TYPE_SV_CREATED_REPLY_SUCCESSFULLY
- PACKET_TYPE_SV_LIST_TEAMS
- PACKET_TYPE_SV_LIST_CHANNELS
- PACKET_TYPE_SV_LIST_THREADS
- PACKET_TYPE_SV_LIST_REPLIES
- PACKET_TYPE_SV_GET_CURRENT_USER_INFO
- PACKET_TYPE_SV_GET_TEAM_INFO
- PACKET_TYPE_SV_GET_CHANNEL_INFO
- PACKET_TYPE_SV_GET_THREAD_INFO
- PACKET_TYPE_SV_ERROR_UNKNOWN_TEAM
- PACKET_TYPE_SV_ERROR_UNKNOWN_CHANNEL
- PACKET_TYPE_SV_ERROR_UNKNOWN_THREAD
- PACKET_TYPE_SV_ERROR_UNKNOWN_USER
- PACKET_TYPE_SV_ERROR_UNAUTHORIZED
- PACKET_TYPE_SV_ERROR_ALREADY_EXIST
```

4.4. Server Packet Types

5.1. Create a packet Append to `protocol/include/protocol/codec.h` union the new packet type:

```
typedef union {
    ...
    cl_<packet_type>_packet_t cl_<packet_type>;
    sv_<packet_type>_packet_t sv_<packet_type>;
} packet_t;
```

Go to `protocol/include/protocol/packet` and create a new file named `<packet_type>.h` and add two packet structures (see 3.3.) for the client and server.
And add the definition of the encoding and decoding functions for both client and server that will be implemented in the next step.
Then go to `protocol/src/protocol/packet` and create two new files named `cl_<packet_type>_packet.c` and `sv_<packet_type>_packet.c` and implement the encoding and decoding functions like so:

```
void <cl|sv>_<packet_type>_packet_encode(
    <cl|sv>_<packet_type>_packet_t *self,
    packet_t *packet
) { ... }

bool <cl|sv>_<packet_type>_packet_decode(
    <cl|sv>_<packet_type>_packet_t *self,
    packet_t *packet
) { return ... }

void <cl|sv>_<packet_type>_packet_destroy(
    <cl|sv>_<packet_type>_packet_t *self
)
{
    ...
    deallocate(self);
}
```

You can now append to :

- `protocol/src/protocol/codec/packet_push_cl_sv_packet.c`
- `protocol/src/protocol/codec/packet_pop_cl_sv_packet.c`
- `protocol/src/protocol/codec/cl_sv_packet_destroy.c`

the functions like so :

```
... handlers[PACKET_TYPE_COUNT] = {
    ...
    [PACKET_TYPE_<cl|sv>_<packet_type>] =
    (cl_sv_packet_<encode|decode|destroy>_t)
    <cl|sv>_<packet_type>_packet_<encode|decode|destroy>,
};
```

5.2. Handle a command Go to `cli/includes/cli.h` and add to the *Command Handler* section a function declaration like so:

```
void cli_handle_<packet_type>_command(cli_t *self, command_t *cmd);
```

Then go to `cli/src/cli/commands/` and create a new file named `cli_handle_command_<packet_type>.c` and implement the function like so:

```
void cli_handle_<packet_type>_command(cli_t *self, command_t *cmd)
{
    cl_<packet_type>_packet_t packet;

     ...

    stream_send_packet(self->stream, (cl_sv_packet_t *)&packet);
}
```

Append the function to the `cli/src/cli/cli_handle_input.c` file like so:

```
static const command_handler_t HANDLERS[PACKET_TYPE_COUNT] = {
    ...
    { "<command_name>", cli_handle_<packet_type>_command },
};
```

5.3. Handle a client packet Go to `server/includes/client.h` and add to the *Handlers* section a function declaration like so:

```
void client_handle_<packet_type>(
    client_t *self,
    packet_t *packet
);
```

Then append to `server/src/client/client_handle_packet.c` your function:

```
static const packet_handler_t PACKET_HANDLERS[PACKET_TYPE_COUNT] = {
    ...
    [PACKET_TYPE_<cl|sv>_<packet_type>] =
    {true, (void *)client_handle_<packet_type>},
};
```

Lastly, create a new file `server/src/client/handlers/client_handle_<packet_type>.c`:

```
void client_handle_<packet_type>(
    client_t *self,
    packet_t *packet
) { ... }
```

5.4. Handle a server packet Go to `cli/include/cli.h` and add to the *Packet Handler* section a function declaration like so:

```
void cli_handle_packet_sv_<packet_type>(
    cli_t *self,
    sv_<packe_type>_packet_t *packet
);
```

Then, go to `cli/src/cli/cli_handle_packets.c` and append your function:

```
static const packet_handler_t HANDLERS[PACKET_TYPE_COUNT] = {
    ...
    [PACKET_TYPE_SV_<packet_type>] =
    (void *)cli_handle_packet_sv_<packet_type>,
};
```

Lastly, implement your function into a new file `cli/src/cli/packets/cli_handle_packet_sv_<packet_type>.c`:

```
void cli_handle_packet_sv_<packet_type>(
    cli_t *self,
    sv_<packet_type>_packet_t *packet
) { ... }
```

6. Security Considerations

This document has no security considerations.

7. Author's Address

A1ex, Epitech, a1ex@a1ex.fr

(h)Adam, Epitech, the.hadam@example.com

Zowks, Epitech, zowks@example.com