Operating Sistem Project

VideoGame (3 pepole)
Implement a distributed videogame

Cariggi Gianmarco Ceccarelli Riccardo Francesco Fabbi

June 29, 2018

Contents

Chapter 1

Introduction

1.1 Server Side

1.1.1 TCP part

1. registering a new client when it comes online, by registering his username and password;
2. deregistering a client when it goes offline, but saving his status, in order to restore whenever it goes online again;
3. sending the map, when the client requests it;
4. send periodically a test packet to check connection with clients;

1.1.2 UDP Part

1. the server receives periodic upates from the client in the form of <timestamp, translational acceleration, rotational acceleration>;
2. Each "epoch" it integrates the messages from the clients, and sends back a state update;
3. the server sends to each connected client the position of the agents around him;

1.2 Client Side

1.2.1 Starting

1. connects to the server (TCP), by registering with a login protocol;
2. requests the map, and gets an ID from the server (TCP);

1.2.2 Periodically

1. receives udates on the state from the server(TCP-UDP);
2. updates the viewer (provided)
3. reads either keyboard or joystick
4. sends the <UDP> packet of the control to the server

Chapter 2

Work

2.1 Server Side

When we launch the server from terminal, we need to specify elevation map and surface texture. Then it starts initializing all it needs to work: data structures for clients, UDP socket, world and semaphores. Also, it defines a signal handler for many signals, in order to shut down properly whenever we close the server or a problem occurs. Next, the server defines two UDP threads: first one for receiving updates from all the connected clients, the second one for updating the world and sending them the world updates. We'll specify these threads' behavior next. UDP socket listen and communicate by using a port defined in common.h (SERVER_PORT_UDP). Server defines a TCP socket similarly. Like UDP socket, TCP port is defined in common.h (SERVER_PORT_TCP). Whenever a client connects, the server creates and launches a thread for communicate with it via TCP. This thread handles the login, checking if it's a new user, or if it's reconnecting to the game after some time. We use the data structures defined before for this purpose. We have two of them: the first one, saves client's username and password and it's the one we use for checking if it's reconnecting, while in the second one there are all the informations we need to know about the client, like id, texture, address for UDP communication, TCP socket, position and, most of all, it's status, in order to realize if it's connected, disconnected, or it never was connected at all. When login is finished, TCP part protocol starts: in order, server waits for an Id request, then calculates an Id and sends it to the client, waits for client's texture (so can save it) and sends it back, then waits for an elevation map request and sends it, same for the map. Later, if other clients are connected, server send them the texture of the new client, and to the latter the textures of the others, in order to let him initialize his world properly. This second passage is also useful for determining if someone disconnected, because if server tries to send something to a client using a TCP socket, if this process returns an error, then it realizes that that particular client disconnected, and can update its status. Like we said before, there are two UDP threads, one for

receive updates from clients, and one for send them the world update. First one receives the force values of the client and updates its vehicle's force (translational and rotational forces), second one updates the world and send it to all of the connected clients. If a client disconnects, server saves its last known position and if that particular client reconnects, sends him that position. When server shuts down, it closes sockets, terminates threads and removes all those data structures no more needed.

2.2 Client Side

When launching a new client, we need to specify the IP address of the server, and the texture of the vehicle it will use. Client can work only if server is on, otherwise will receive an error message and shut down. Like the server, client creates UDP and TCP sockets, two UDP threads, one TCP thread and a handler for many signals, in order to shut down properly. Ports for TCP and UDP sockets are defined in common.h. When it connects, in order to join the game, it must specify an username and a password. These informations are necessary, and helps server to understand if the client is a new one, or and old one which disconnected before and it's reconnecting now. After login protocol, it starts asking for all the info it needs: id, elevation map and map. Also it sends its texture to server, and receives it back. Next, he will receive the textures of all those clients already connected.Periodically, client sends its forces values to server (translational and rotational forces) using one thread UDP, the other one will be used for receive the world from server and update all the clients' positions. When client shut down, it closes sockets, terminates threads and removes all those data structures no more needed.

2.3 General

In this project, we defined some new version for recv and send functions. When both the server or the client need to receive some packets via TCP, they use a function called recv_TCP_packet, which results useful for receiving the exact amount of byte we need to receive. Otherwise, when we need to receive normal strings, or a UDP packet, we use a standard recv/recvfrom, but still we defined a function called recv_TCP/recv_UDP. Same for send functions (both TCP and UDP). Thus, in order to semplify error handling and avoid code excess.

Chapter 3

Usage

First of all, compile the videogame. In the directory where the folder is located, open a terminal and digit:

```
make
```

This will create two executable files: so_game_server and so_game_client.

3.1 Server

```
./so_game_server <elevation_map_directory>/<elevation\_map>.pgm
 <map_texture_directory>/<map_texture>.ppm (without ‘‘<>’’)
```

3.2 Client

```
./so_game_client <Server IP address> (without ‘‘<>’’)
 <vehicle_texture_directory>/<client_texture>.ppm
```

3.3 Example

Execute client and server on different terminals

3.3.1 For Server

```
./so_game_server ./images/maze.pgm ./images/maze.ppm
```

3.3.2 For Client

```
./so_game_client 127.0.0.1 ./images/arrow-right.ppm
```