Computer networks

Project documentation

Introduction

The project involves the creation of a simulation of a restaurant, consisting of a server and three clients: a customer, a table, and a kitchen appliance. Communication between these elements takes place using sockets in the TCP protocol. To handle multiple clients simultaneously, a forking mechanism is used that creates separate processes for each server-client connection. Each child process handles only one client connection, which means that the data and actions of one client do not interfere with other clients. This allows values unique only to that server-client connection to be stored.

General communication scheme

Client-server communication:

1. the client reads a command entered by the user in the console.
2. the client validates the command.
3. the command is encoded and sent to the server via sockets.
4. if parameters are needed to perform the operation, they are sent as for a command.

Server operation:

1. the server receives the command from the client via socket.
2. passes to the appropriate place in the code using if-else.
3. if parameters are needed to perform an operation, they are received in the same way as for a command and then written to the appropriate structures/variables.
4. based on the instructions received from the client, the server calls the appropriate methods.

Server communication with the client:

1. after the requested operation has been performed, the server generates responses to the client.
2. the responses may contain the results of the operation, errors, or specific values.
3. the server sends this information to the client via sockets.
4. for more complex operations, the server first sends the result of the operation and then informs the client of any errors or provides specific values. The client will expect to receive different messages depending on the result sent.

All transmitted information is encoded and decoded in appropriate buffers to ensure the correctness of the transmitted data. The size of the transmitted and received data is always declared.

Server

The server operates in a two-threaded mode. The first thread is responsible for waiting for the user to enter one of the commands in the console, such as:

1. *stop,*
2. *stat table {table_id},*
3. *stat status {status_name}.*

Upon receiving a command, the server executes the appropriate functions.

The second server thread handles socket communication. It waits for a command from the client, identifies the next action to perform and then accepts further information or starts performing the requested operation.
The server stores data about the restaurant in various structures, such as table layout, reservations, orders, and menus. This data can be stored in binary files, allowing all threads to share the same resources. This allows the server to manage and perform operations on the data efficiently.

Client:

The client is used exclusively for making restaurant reservations. It sends a request whether a table is available for the given parameters. The server checks the availability of a table, considering parameters such as name, number of people and date and time of the reservation. After the analysis, the server returns a list of available tables. Tables are selected according to the number of people. For example, two people will not be seated at a 4 or 6-person table. The customer receives a confirmation of the booking along with a unique booking code, which they will need to enter at the table.
Commands:

1. *find {surname} {nr_people} {DD-MM-YYYY} {HH:mm}*
2. *book {choice_nr}*

Table:

To unlock a table, the customer must enter their name and the booking code received when booking the table. The customer can then view the menu and place an order. The order consists of the menu items and their quantity:

1. *order {course_nr}: {dish_code1}-{quantity} {dish_code2}-{quantity} ... {dish_codeN}-{quantity}*
2. *order com1: A1-2 A2-3 F1-2*

The customer can place a maximum of 6 orders. For flexibility, orders of unknown length can be read using a function such as:

scanf("%[^:]: %[^]", course, order)

The customer also has the option to request a receipt for the order.

Kitchen device:

The kitchen appliance is responsible for preparing orders. By sending the `take' command to the server, it receives the order that has the earliest add date and a status of `waiting'. The server also changes the status of the order from 'waiting' to 'preparing'. The kitchen appliance stores the received order as an Order structure. Then, once the meal is prepared, the kitchen appliance sends a 'ready' command with the booking code and dish number to the server to change the order status from 'preparing' to 'served'.
It is also possible to display all orders that are currently in the 'preparation' state using the 'show' command. The server first sends an indication of whether any orders have been found in this state, then sends the number of orders found and, using a loop, sends the data of the individual orders.