Trabalho 1 - Redes de computadores

Fabrício Samapio - nUSP: 12547423

October 14, 2023

1 Servidor

1.1 Inicio e thread para input

```cpp
int main() {
    // Create a socket
    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        std::cerr << "Error creating socket" << std::endl;
        return -1;
    }

    // Bind the socket to an address and port
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(PORT);
    serverAddress.sin_addr.s_addr = INADDR_ANY;

    if (bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == -1) {
        std::cerr << "Error binding socket" << std::endl;
        return -1;
    }

    // Listen for incoming connections
    if (listen(serverSocket, MAX_CONNECTIONS) == -1) {
        std::cerr << "Error listening on socket" << std::endl;
        return -1;
    }
    std::cout << "Server listening on port " << PORT << " ..." << std::endl;

    std::thread inputThread(handle_input, serverSocket);
```

```cpp
void handle_input(int serverSocket) {
    char buffer[1024];
    std::cout << "Digite 'EXIT' para fechar o server" << std::endl;
    std::cin.getline(buffer, 1024);

        if (strcmp("EXIT", buffer) == 0){
        for (int client : client_sockets) close(client);
            close(serverSocket);
            exit(0);
        }
}
```

Esse é o começo do código do sevidor, ela serve para criar e bindar o socket do servidor; feito isso, começa a escutar para novas conexões. Também é inicializado a thread que cuidará dos inputs do servidor.

1.2 Aceitando conexões dos clientes e threads para cliente

```cpp
std::vector<std::thread> threads;
while(true) {
    // Accept incoming connections
    sockaddr_in clientAddress;
    socklen_t clientAddressSize = sizeof(clientAddress);

    int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddressSize);
    if (clientSocket == -1) {
        std::cerr << "Error accepting client connection" << std::endl;
        return -1;
    }

    // Save the client socket and create a thread for this client
    client_sockets.push_back(clientSocket);
    threads.emplace_back(handle_client, clientSocket);
}
```

```cpp
// Receive the name of the new client
char buffer[1024];
char client_name[16];
int bytesRead = recv(clientSocket, client_name, sizeof(client_name), 0);
if (bytesRead <= 0) {
    std::cerr << "Connection closed by client" << std::endl;
    close(clientSocket);
    return;
}
client_name[bytesRead] = '\0';
std::cout << "New client [ " << client_name << " ] connected!" << std::endl;

// Echo the received data back to the client
send(clientSocket, client_name, strlen(client_name), 0);

// Send and receive data
while (true) {
    bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);
    if (bytesRead <= 0 || !strcmp(buffer, "exit")) {
        std::cerr << "Connection closed by the client [ " << client_name << " ]" << std::endl;
        break;
    }
    buffer[bytesRead] = '\0';
    char data[1024];
    sprintf(data, "[ %s ]: %s\n", client_name, buffer);

    std::cout << data;

    // Send the data to all users
    for (int client : client_sockets) {
        if (client != clientSocket) send(client, data, strlen(data), 0);
        else {
            if (!strcmp(buffer, "exit")) send(client, data, strlen(data), 0);
        }
    }
}
close(clientSocket);
```

Nessa parte, após cada conexão aceita, um socket é criado e uma thread para cuidar do client é feita. Na thread, é recebido o nome que o client enviou e após isso fica recebendo as mensagens do client e repassando a mesma para todos os outros clientes.

2 Cliente

2.1 Conexão do cliente com o sevidor

```cpp
int main() {
    // Create a socket
    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1) {
        std::cerr << "Error creating socket" << std::endl;
        return -1;
    }

    // Connect to the server
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddress.sin_addr);

    if (connect(clientSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == -1) {
        std::cerr << "Error connecting to server" << std::endl;
        return -1;
    }
    std::cout << "Connected to server" << std::endl;

    // Send the name
    char buffer[1024];
    std::cout << "Enter your name: ";
    std::cin.getline(buffer, sizeof(buffer));

    send(clientSocket, buffer, strlen(buffer), 0);

    int bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);
    if (bytesRead <= 0) {
        std::cerr << "Connection closed by server" << std::endl;
        close(clientSocket);
        return -1;
    }

    // Send and receive data
    std::thread sendThread(sendData, clientSocket);
    std::thread receiveThread(recieveData, clientSocket);
```

Nessa parte do código, é criado o socket do cliente e é efetuada a conexão com o servidor. Após isso, é enviado o seu nick que será utilizado o chat e então é inicializado as threads de envio e recebimento de dados.

2.2 Threads de recebimento e envio de dados

```cpp
// Function for the thread that send data to the server
void sendData(int clientSocket) {
    char buffer[1024];
    std::cout << "Enter a message ('EXIT' to close the connection): " << std::endl;
    while (true) {
        std::cin.getline(buffer, sizeof(buffer));
        send(clientSocket, buffer, strlen(buffer), 0);

        if (strcmp(buffer, "EXIT") == 0) {
            break;
        }
    }
    stopThread.store(true, std::memory_order_release);
}

// Function for the thread that receive data from the server
void recieveData(int clientSocket) {
    char buffer[1024];
    while (true) {
        int bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesRead <= 0) {
            std::cerr << "Connection closed" << std::endl;
            break;
        }
        if (stopThread) break;
        buffer[bytesRead] = '\0';
        std::cout << buffer;
    }
    stopThread.store(true, std::memory_order_release);
}
```

A função de cima é a responsável com enviar para o servidor os inputs que o client digitar. Já a segunda é responsável por receber todas as mensagens enviada pelo servidor, advindas de inputs de todos os outros clientes.