Отчет по реализации чата на сокетах на языке C

Серверная часть

Подключаемые библиотеки

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
```

Эти библиотеки используются для стандартного ввода-вывода, работы с памятью, строками, системными вызовами, сетевыми операциями и потоками.

Определение констант

```c
#define PORT 8080
#define BUFFER_SIZE 1024
#define MAX_CLIENTS 10
```

- `PORT`: Порт, на котором будет работать сервер.
- `BUFFER_SIZE`: Размер буфера для сообщений.
- `MAX_CLIENTS`: Максимальное количество клиентов, которые могут подключиться к серверу одновременно.

Глобальные переменные

```c
int client_sockets[MAX_CLIENTS];
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;
```

- `client_sockets`: Массив для хранения сокетов клиентов.
- `clients_mutex`: Мьютекс для защиты доступа к массиву клиентов.

Функция для отправки сообщения всем клиентам, кроме отправителя

```c
void send_message_to_all(char *message, int exclude_sock) {
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] != 0 && client_sockets[i] != exclude_sock) {
            send(client_sockets[i], message, strlen(message), 0);
```

```
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
```

Эта функция отправляет сообщение всем подключенным клиентам, кроме клиента-отправителя.

Функция для обработки каждого клиента

```
void *handle_client(void *socket_desc) {
    int sock = *(int *)socket_desc;
    char buffer[BUFFER_SIZE];
    int read_size;
    struct sockaddr_in client_addr;
    socklen_t addr_size = sizeof(struct sockaddr_in);
    getpeername(sock, (struct sockaddr*)&client_addr, &addr_size);
    char client_ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &client_addr.sin_addr, client_ip, INET_ADDRSTRLEN);

    char join_message[BUFFER_SIZE];
    snprintf(join_message, sizeof(join_message), "Client %s joined the c-
chat\n", client_ip);
    send_message_to_all(join_message, sock);

    while ((read_size = recv(sock, buffer, BUFFER_SIZE, 0)) > 0) {
        buffer[read_size] = '\0';
        char message[BUFFER_SIZE + INET_ADDRSTRLEN + 3];
        snprintf(message, sizeof(message), "%s: %s", client_ip, buffer);
        send_message_to_all(message, sock);
    }

    close(sock);
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] == sock) {
            client_sockets[i] = 0;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);

    char leave_message[BUFFER_SIZE];
    snprintf(leave_message, sizeof(leave_message), "Client %s left the c-
chat\n", client_ip);
    send_message_to_all(leave_message, sock);

    free(socket_desc);
    pthread_exit(NULL);
}
```

- `handle_client`: Функция для обработки подключения клиента. Она получает сообщение от клиента и отправляет его всем другим клиентам. Также отправляет уведомления о подключении и отключении клиентов.

Главная функция сервера

```
int main() {
    int server_socket, client_socket, *new_sock;
    struct sockaddr_in server, client;
    socklen_t client_len = sizeof(client);

    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Could not create socket");
        return 1;
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    if (bind(server_socket, (struct sockaddr *)&server, sizeof(server)) <
0) {
        perror("Bind failed");
        close(server_socket);
        return 1;
    }

    listen(server_socket, 3);
    printf("Waiting for incoming connections...\n");

    while ((client_socket = accept(server_socket, (struct sockaddr
*)&client, &client_len))) {
        printf("Connection accepted from %s:%d\n",
inet_ntoa(client.sin_addr), ntohs(client.sin_port));

        pthread_mutex_lock(&clients_mutex);
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (client_sockets[i] == 0) {
                client_sockets[i] = client_socket;
                break;
            }
        }
        pthread_mutex_unlock(&clients_mutex);

        pthread_t client_thread;
        new_sock = malloc(1);
        *new_sock = client_socket;

        if (pthread_create(&client_thread, NULL, handle_client, (void
*)new_sock) < 0) {
            perror("Could not create thread");
```

```
            return 1;
        }

        printf("Handler assigned\n");
    }

    if (client_socket < 0) {
        perror("Accept failed");
        close(server_socket);
        return 1;
    }

    close(server_socket);
    return 0;
}
```

- В `main` функция создает сокет сервера и привязывает его к указанному порту и адресу.
- Затем она начинает слушать входящие подключения с помощью `listen`.
- Когда новое подключение принимается с помощью `accept`, создается новый поток для обработки клиента с помощью `pthread_create`.
- Массив `client_sockets` используется для отслеживания всех подключенных клиентов.

Клиентская часть

Подключаемые библиотеки

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
```

Эти библиотеки используются для стандартного ввода-вывода, работы с памятью, строками, системными вызовами, сетевыми операциями и потоками.

Определение констант

```
#define BUFFER_SIZE 1024
#define ENV_FILE ".env"
```

- `BUFFER_SIZE`: Размер буфера для сообщений.
- `ENV_FILE`: Имя файла окружения, из которого будут считываться IP-адрес и порт сервера.

Функция для получения сообщений от сервера

```c
void *receive_messages(void *socket_desc) {
    int sock = *(int *)socket_desc;
    char buffer[BUFFER_SIZE];
    int read_size;

    while ((read_size = recv(sock, buffer, BUFFER_SIZE, 0)) > 0) {
        buffer[read_size] = '\0';
        printf("%s", buffer);
    }

    if (read_size == 0) {
        printf("Server disconnected\n");
    } else if (read_size == -1) {
        perror("recv failed");
    }

    pthread_exit(NULL);
}
```

- `receive_messages`: Функция, выполняемая в отдельном потоке, которая непрерывно получает сообщения от сервера и выводит их на экран.

Главная функция клиента

```c
int main() {
    int sock;
    struct sockaddr_in server;
    char message[BUFFER_SIZE];
    pthread_t receive_thread;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
        perror("Could not create socket");
        return 1;
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("158.160.137.120");
    server.sin_port = htons(PORT);

    if (connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
        perror("Connect failed");
        close(sock);
        return 1;
    }

    printf("Connected to server\n");

    if (pthread_create(&receive_thread, NULL, receive_messages, (void
*)&sock) < 0) {
```

```
        perror("Could not create thread");
        close(sock);
        return 1;
    }

    while (1) {
        fgets(message, BUFFER_SIZE, stdin);
        if (send(sock, message, strlen(message), 0) < 0) {
            perror("Send failed");
            close(sock);
            return 1;
        }
    }

    close(sock);
    return 0;
}
```

- В `main` функция читает параметры подключения (IP-адрес и порт) из файла `.env` с помощью `read_env`.
- Затем создается сокет и устанавливается соединение с сервером с помощью `connect`.
- После успешного подключения создается поток для получения сообщений от сервера с помощью `pthread_create`.
- Главный поток программы читает сообщения с клавиатуры и отправляет их на сервер.