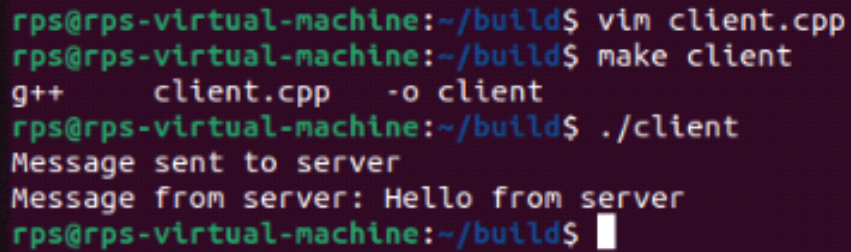


DATE :- 26-07-2024

Design and implement a network service that reliably handles concurrent client connections while ensuring graceful termination in response to external signals (e.g., SIGTERM, SIGINT). The service must maintain data consistency and avoid resource leaks throughout its lifecycle.

Client :-



```
rps@rps-virtual-machine:~/build$ vim client.cpp
rps@rps-virtual-machine:~/build$ make client
g++    client.cpp    -o client
rps@rps-virtual-machine:~/build$ ./client
Message sent to server
Message from server: Hello from server
rps@rps-virtual-machine:~/build$
```

```

#include <iostream>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>

void start_client(const std::string& server_ip, int port) {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    std::string message = "Hello from client";
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);
    if (inet_pton(AF_INET, server_ip.c_str(), &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return;
    }
    if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection failed" << std::endl;
        return;
    }
    send(sock, message.c_str(), message.size(), 0);
    std::cout << "Message sent to server" << std::endl;
    int valread = read(sock, buffer, 1024);
    std::cout << "Message from server: " << buffer << std::endl;
    close(sock);
}

int main() {
    std::string server_ip = "127.0.0.1";
    int port = 8080;
    start_client(server_ip, port);
    return 0;
}

```

Server :-

```
rps@rps-virtual-machine:~/sockets/today$ cd
rps@rps-virtual-machine:~$ mkdir build
rps@rps-virtual-machine:~$ cd build
rps@rps-virtual-machine:~/build$ vim server.cpp
rps@rps-virtual-machine:~/build$ make server
g++    server.cpp    -o server
rps@rps-virtual-machine:~/build$ ./server
Starting server on port 8080
Server is listening on port 8080
Message from client: Hello from client
^CServer shutting down gracefully
```

```
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>
#include <atomic>
#include <signal.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>

// Signal handling
std::atomic<bool> running(true);

void signal_handler(int signum) {
    if (signum == SIGINT || signum == SIGTERM) {
        running = false;
    }
}

void setup_signal_handling() {
    struct sigaction sa;
    sa.sa_handler = signal_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGINT, &sa, nullptr);
    sigaction(SIGTERM, &sa, nullptr);
}

// Client handler
void handle_client(int client_socket) {
    char buffer[1024] = {0};
    std::string hello = "Hello from server";

    // Read client message
```

```

    // Read client message
    read(client_socket, buffer, 1024);
    std::cout << "Message from client: " << buffer << std::endl;

    // Send response to client
    send(client_socket, hello.c_str(), hello.size(), 0);

    // Close client socket
    close(client_socket);
}

// Server implementation
std::vector<std::thread> client_threads;
std::mutex threads_mutex;

void start_server(int port, std::atomic<bool>& running) {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
        perror("setsockopt");
    close(server_fd);
}

```

```

// Binding the socket to the network address and port
if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
    perror("bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}
if (listen(server_fd, 3) < 0) {
    perror("listen");
    close(server_fd);
    exit(EXIT_FAILURE);
}
std::cout << "Server is listening on port " << port << std::endl;
while (running) {
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        if (running) {
            perror("accept");
        }
        break;
    }
    std::lock_guard<std::mutex> lock(threads_mutex);
    client_threads.emplace_back(std::thread(handle_client, new_socket));
}
close(server_fd);
std::lock_guard<std::mutex> lock(threads_mutex);
for (auto& thread : client_threads) {
    if (thread.joinable()) {
        thread.join();
    }
}
}
int main() {
    setup_signal_handling();
    int port = 8080;
    std::cout << "Starting server on port " << port << std::endl;
    start_server(port, running);
}

```

```

int main() {
    setup_signal_handling();
    int port = 8080;
    std::cout << "Starting server on port " << port << std::endl;
    start_server(port, running);
    std::cout << "Server shutting down gracefully" << std::endl;
    return 0;
}

```

MESSAGE FROM CLIENT TO SERVER :-

SERVER CODE :-



```

#include <iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("172.20.0.29");
    address.sin_port = htons(PORT);

    // Bind the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
}

```

```

// Bind the socket to the network address and port
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("listen");
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::cout << "Server listening on 172.20.0.29:" << PORT << std::endl;

// Accept a connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
    perror("accept");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Communicate with the client
while (true) {
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;
    send(new_socket, buffer, strlen(buffer), 0);
}

```

```

// Communicate with the client
while (true) {
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;
    send(new_socket, buffer, strlen(buffer), 0);
}

close(new_socket);
close(server_fd);
return 0;
}

```

CLIENT CODE :-

```

rps@rps-virtual-machine:~/sockets$ cat client1.cpp
#include <iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return -1;
    }

    // Define the server address
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "172.20.0.46", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed" << std::endl;
    }
}

```

```

// Connect to the server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    std::cerr << "Connection Failed" << std::endl;
    return -1;
}

while (true) {
    std::cout << "Enter message to send (type 'exit' to close): ";
    std::string message;
    std::getline(std::cin, message);

    if (message == "exit") {
        break;
    }

    send(sock, message.c_str(), message.length(), 0);
    std::cout << "Message sent" << std::endl;

    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(sock, buffer, BUFFER_SIZE);
    std::cout << "Received from server: " << buffer << std::endl;
}

close(sock);
return 0;
}
rps@rps-virtual-machine:~/sockets$

```

## CONVERSATION BETWEEN SERVER AND CLIENT :-

### SERVER CODE :-

```

rps@rps-virtual-machine:~$ vim server3.cpp
rps@rps-virtual-machine:~$ make server3
g++ server3.cpp -o server3
rps@rps-virtual-machine:~$ ./server3
Server listening on port 65432
Received from client: hii ameesha
Enter message to client: hii harika
Received from client: how are you?
Enter message to client: good

```



```

#include<iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("172.20.0.29"); // Bind to any available address
    address.sin_port = htons(PORT);

    // Bind the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    std::cout << "Server listening on port " << PORT << std::endl;

    // Accept a connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Communication loop
    while (true) {
        // Read from client
        memset(buffer, 0, BUFFER_SIZE);
        int valread = read(new_socket, buffer, BUFFER_SIZE);
        if (valread <= 0) {
            break;
        }
        std::cout << "Received from client: " << buffer << std::endl;

        // Process the received message (if needed)

        // Example: Check if client wants to end communication
        if (strcmp(buffer, "quit") == 0) {
            std::cout << "Client has requested to quit. Closing connection." << std::endl;
            break;
        }
    }
}

```

```

while (true) {
    // Read from client
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;

    // Process the received message (if needed)

    // Example: Check if client wants to end communication
    if (strcmp(buffer, "quit") == 0) {
        std::cout << "Client has requested to quit. Closing connection." << std::endl;
        break;
    }

    // Server's response
    std::string server_response;
    std::cout << "Enter message to client: ";
    std::getline(std::cin, server_response);

    // Send response back to client
    send(new_socket, server_response.c_str(), server_response.size(), 0);
}

// Close sockets
close(new_socket);
close(server_fd);
return 0;
}

```

## CLIENT CODE :-

```

rps@rps-virtual-machine:~/sockets$ ./client1
Enter message to send (type 'exit' to close): hii ameesha
Message sent
Received from server: hii ameesha
Enter message to send (type 'exit' to close): exit
rps@rps-virtual-machine:~/sockets$ ./client1
Enter message to send (type 'exit' to close): hii ameesha
Message sent
Received from server: hii harika
Enter message to send (type 'exit' to close): how are you?
Message sent
Received from server: good
Enter message to send (type 'exit' to close): exit
rps@rps-virtual-machine:~/sockets$

```

```
rps@rps-virtual-machine:~/sockets$ cat client1.cpp
#include <iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return -1;
    }

    // Define the server address
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "172.20.0.46", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed" << std::endl;
    }
}
```

```

// Connect to the server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    std::cerr << "Connection Failed" << std::endl;
    return -1;
}

while (true) {
    std::cout << "Enter message to send (type 'exit' to close): ";
    std::string message;
    std::getline(std::cin, message);

    if (message == "exit") {
        break;
    }

    send(sock, message.c_str(), message.length(), 0);
    std::cout << "Message sent" << std::endl;

    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(sock, buffer, BUFFER_SIZE);
    std::cout << "Received from server: " << buffer << std::endl;
}

close(sock);
return 0;
}
rps@rps-virtual-machine:~/sockets$

```

CREATE A TEXT FILE AND THEN SEND FROM SERVER TO CLIENT AND VICE VERSA?

RECEIVED FROM CLIENT :-

SERVER CODE :-

```

rps@rps-virtual-machine:~/files$ vim server.cpp
rps@rps-virtual-machine:~/files$ make server
g++    server.cpp    -o server
rps@rps-virtual-machine:~/files$ ./server
Server listening on port 65432
Received from client: recieved
Enter message to client: hii
rps@rps-virtual-machine:~/files$

```



```

#include<iostream>
#include <fstream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("172.20.0.29"); // Bind to any available address
    address.sin_port = htons(PORT);

    // Bind the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
}

```

```

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("listen");
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::cout << "Server listening on port " << PORT << std::endl;

// Accept a connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
    perror("accept");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Read the contents of the text file
std::ifstream file("example.txt");
if (!file) {
    perror("File open failed");
    close(new_socket);
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::string file_contents((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
file.close();

```

```

send(new_socket, file_contents.c_str(), file_contents.size(), 0);

// Communication loop
while (true) {
    // Read from client
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;

    // Process the received message (if needed)

    // Example: Check if client wants to end communication
    if (strcmp(buffer, "quit") == 0) {
        std::cout << "Client has requested to quit. Closing connection." << std::endl;
        break;
    }

    // Server's response
    std::string server_response;
    std::cout << "Enter message to client: ";
    std::getline(std::cin, server_response);

    // Send response back to client
    send(new_socket, server_response.c_str(), server_response.size(), 0);
}

// Close sockets
close(new_socket);
close(server_fd);
return 0;
}

```

#### CLIENT CODE :-

```

rps@rps-virtual-machine:~/files$ ./client
Received from server:
hii
hlo

Enter message to send (type 'quit' to close): recieved
Message sent
Received from server: hii
Enter message to send (type 'quit' to close): quit
rps@rps-virtual-machine:~/files$

```

```
rps@rps-virtual-machine:~/files$ cat client.cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return -1;
    }

    // Define the server address
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "172.20.0.46", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
```

```

// Connect to the server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    std::cerr << "Connection Failed" << std::endl;
    return -1;
}

// Receive the contents of the text file from the server
memset(buffer, 0, BUFFER_SIZE);
int valread = read(sock, buffer, BUFFER_SIZE);
if (valread > 0) {
    std::cout << "Received from server:\n" << buffer << std::endl;
}

while (true) {
    std::cout << "Enter message to send (type 'quit' to close): ";
    std::string message;
    std::getline(std::cin, message);

    if (message == "quit") {
        break;
    }

    send(sock, message.c_str(), message.length(), 0);
    std::cout << "Message sent" << std::endl;

    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(sock, buffer, BUFFER_SIZE);
    std::cout << "Received from server: " << buffer << std::endl;
}

close(sock);
return 0;
}
rps@rps-virtual-machine:~/files$

```

RECEIVED FILES FROM CLIENT TO SERVER :-

SERVER CODE :-



```

rps@rps-virtual-machine:~/files$ vim server.cpp
rps@rps-virtual-machine:~/files$ make server
g++      server.cpp      -o server
.rps@rps-virtual-machine:~/files$ ./server
Server listening on port 65432
Received from client: recieved
Enter message to client: hii
rps@rps-virtual-machine:~/files$ vim server.cpp
rps@rps-virtual-machine:~/files$ make server
g++      server.cpp      -o server
.rps@rps-virtual-machine:~/files$ ./server
Server listening on port 65432
Received from client: hii
welcome unix
training

Enter message to client: received
Received from client: recieved
Enter message to client: exit
Received from client: ok
Enter message to client: ^C

```

```

#include<iostream>
#include <fstream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 65432
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("172.20.0.29"); // Bind to any available address
    address.sin_port = htons(PORT);

    // Bind the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
}

```

```

if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("listen");
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::cout << "Server listening on port " << PORT << std::endl;

// Accept a connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
    perror("accept");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Read the contents of the text file
std::ifstream file("example.txt");
if (!file) {
    perror("File open failed");
    close(new_socket);
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::string file_contents((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
file.close();

```

```

// Read the contents of the text file
std::ifstream file("example.txt");
if (!file) {
    perror("File open failed");
    close(new_socket);
    close(server_fd);
    exit(EXIT_FAILURE);
}

std::string file_contents((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
file.close();

// Send the contents of the text file to the client
send(new_socket, file_contents.c_str(), file_contents.size(), 0);

// Communication loop
while (true) {
    // Read from client
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;

    // Process the received message (if needed)

    // Example: Check if client wants to end communication
    if (strcmp(buffer, "quit") == 0) {
        std::cout << "Client has requested to quit. Closing connection." << std::endl;
        break;
    }
}

```

```

// Send the contents of the text file to the client
send(new_socket, file_contents.c_str(), file_contents.size(), 0);

// Communication loop
while (true) {
    // Read from client
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread <= 0) {
        break;
    }
    std::cout << "Received from client: " << buffer << std::endl;

    // Process the received message (if needed)

    // Example: Check if client wants to end communication
    if (strcmp(buffer, "quit") == 0) {
        std::cout << "Client has requested to quit. Closing connection." << std::endl;
        break;
    }

    // Server's response
    std::string server_response;
    std::cout << "Enter message to client: ";
    std::getline(std::cin, server_response);

    // Send response back to client
    send(new_socket, server_response.c_str(), server_response.size(), 0);
}

// Close sockets
close(new_socket);
close(server_fd);
return 0;
}

-- INSERT --

```

## CLIENT CODE :-

```

rps@rps-virtual-machine:~/files$ ./client2
File contents received from server:
hii
hlo

Enter message to server: recieved
Received from server: received
Enter message to server: ok
^C
rps@rps-virtual-machine:~/files$

```

```
rps@rps-virtual-machine:~/files$ cat client2.cpp
#include <iostream>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fstream>
#include <vector>

#define PORT 65432
#define BUFFER_SIZE 1024

void sendFile(int sock, const std::string& filename) {
    std::ifstream file(filename, std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        std::cerr << "Error opening file: " << filename << std::endl;
        return;
    }

    std::streamsize file_size = file.tellg();
    file.seekg(0, std::ios::beg);

    std::vector<char> buffer(file_size);
    if (file.read(buffer.data(), file_size)) {
        send(sock, buffer.data(), file_size, 0);
    } else {
        std::cerr << "Error reading file: " << filename << std::endl;
    }
}
```



```

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "172.20.0.46", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address or Address not supported" << std::endl;
        close(sock);
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection failed" << std::endl;
        close(sock);
        return -1;
    }

    std::string file_to_send = "example.txt";
    sendFile(sock, file_to_send);

    std::string file_contents;
    ssize_t bytes_received;
    while ((bytes_received = recv(sock, buffer, BUFFER_SIZE, 0)) > 0) {
        file_contents.append(buffer, bytes_received);
        if (bytes_received < BUFFER_SIZE) break;
    }
}

```

```

std::string file_contents;
ssize_t bytes_received;
while ((bytes_received = recv(sock, buffer, BUFFER_SIZE, 0)) > 0) {
    file_contents.append(buffer, bytes_received);
    if (bytes_received < BUFFER_SIZE) break;
}

if (bytes_received < 0) {
    std::cerr << "Error receiving file data" << std::endl;
    close(sock);
    return -1;
}

std::cout << "File contents received from server:" << std::endl;
std::cout << file_contents << std::endl;

while (true) {
    std::string message;
    std::cout << "Enter message to server: ";
    std::getline(std::cin, message);

    send(sock, message.c_str(), message.size(), 0);

    if (message == "quit") {
        std::cout << "Ending communication with server." << std::endl;
        break;
    }

    memset(buffer, 0, BUFFER_SIZE);

```

```

memset(buffer, 0, BUFFER_SIZE);
ssize_t valread = recv(sock, buffer, BUFFER_SIZE, 0);
if (valread > 0) {
    std::cout << "Received from server: " << buffer << std::endl;
} else if (valread == 0) {
    std::cout << "Server closed the connection." << std::endl;
    break;
} else {
    std::cerr << "Error receiving data from server" << std::endl;
    break;
}

close(sock);
return 0;
}

```

