Client

File: /home/varsha/Dropbox/work/tea...23-master/v1/client/submit.cppPage 1 of 2

```cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

using namespace std;

const int BUFFER_SIZE = 1024;
const int MAX_FILE_SIZE_BYTES = 4;
const int MAX_TRIES = 5;

int send_file(int sockfd, string file_path)
{
    char buffer[BUFFER_SIZE];
    bzero(buffer, BUFFER_SIZE);
    FILE *file = fopen(file_path.c_str(), "rb");
    if (!file)
    {
        perror("Error opening file");
        return -1;
    }

    fseek(file, 0L, SEEK_END);
    int file_size = ftell(file);
    fseek(file, 0L, SEEK_SET);
    char file_size_bytes[MAX_FILE_SIZE_BYTES];
    memcpy(file_size_bytes, &file_size, sizeof(file_size));
    if (send(sockfd, &file_size_bytes, sizeof(file_size_bytes), 0) == -1)
    {
        perror("Error sending file size");
        fclose(file);
        return -1;
    }

    while (!feof(file))
    {
        size_t bytes_read = fread(buffer, 1, sizeof(buffer), file);
        if (send(sockfd, buffer, bytes_read, 0) == -1)
        {
            perror("Error sending file data");
            fclose(file);
            return -1;
        }
        bzero(buffer, BUFFER_SIZE);
    }
    fclose(file);
    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        cerr << "Usage: ./submit  <serverIP:port>  <sourceCodeFileTobeGraded>\n";
        return -1;
    }

    string server_ip;
    int server_port;
    string ip_port = string(argv[1]);
```

```
        server_ip = ip_port.substr(0, colon_pos);
        server_port = stoi(ip_port.substr(colon_pos + 1));
    }
    else
    {
        cerr << "Invalid input format\n";
        return -1;
    }

    string file_path = argv[2];

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        perror("Socket creation failed");
        return -1;
    }

    struct sockaddr_in serv_addr;
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(server_port);
    inet_pton(AF_INET, server_ip.c_str(), &serv_addr.sin_addr.s_addr);

    int tries = 0;
    while (true)
    {
        if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == 0)
            break;
        sleep(1);
        tries += 1;
        if (tries == MAX_TRIES)
        {
            cout << "Server not responding\n";
            return -1;
        }
    }

    if (send_file(sockfd, file_path) != 0)
    {
        cout << "Error sending source file\n";
        close(sockfd);
        return -1;
    };

    size_t bytes_read;
    char buffer[BUFFER_SIZE];
    while (true)
    {
        bytes_read = recv(sockfd, buffer, BUFFER_SIZE, 0);
        if (bytes_read <= 0)
            break;
        write(STDOUT_FILENO, buffer, bytes_read);
    }

    close(sockfd);

    return 0;
}
```

Server


```cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include <thread>
#include <filesystem>
#include <vector>
#include <queue>
#include <mutex>
#include <condition_variable>
#include <unistd.h>
#include <netinet/in.h>

using namespace std;
namespace fs = std::filesystem;

const int BUFFER_SIZE = 1024;
const int MAX_FILE_SIZE_BYTES = 4;
const int MAX_QUEUE = 50;

const char SUBMISSIONS_DIR[] = "./submissions/";
const char EXECUTABLES_DIR[] = "./executables/";
const char OUTPUTS_DIR[] = "./outputs/";
const char COMPILER_ERROR_DIR[] = "./compiler_error/";
const char RUNTIME_ERROR_DIR[] = "./runtime_error/";
const char EXPECTED_OUTPUT[] = "./expected/output.txt";

const char PASS_MSG[] = "PASS\n";
const char COMPILER_ERROR_MSG[] = "COMPILER ERROR\n";
const char RUNTIME_ERROR_MSG[] = "RUNTIME ERROR\n";
const char OUTPUT_ERROR_MSG[] = "OUTPUT ERROR\n";

int recv_file(int sockfd, string file_path)
{
    char buffer[BUFFER_SIZE];
    bzero(buffer, BUFFER_SIZE);
    FILE *file = fopen(file_path.c_str(), "wb");
    if (!file)
    {
        perror("Error opening file");
        return -1;
    }

    char file_size_bytes[MAX_FILE_SIZE_BYTES];
    if (recv(sockfd, file_size_bytes, sizeof(file_size_bytes), 0) == -1)
    {
        perror("Error receiving file size");
        fclose(file);
        return -1;
    }
    int file_size;
    memcpy(&file_size, file_size_bytes, sizeof(file_size_bytes));

    size_t bytes_read = 0;
    while (true)
    {
        bytes_read += recv(sockfd, buffer, BUFFER_SIZE, 0);
        if (bytes_read <= 0)
        {
            perror("Error receiving file data");
            fclose(file);
            return -1;
        }
        fwrite(buffer, 1, bytes_read, file);
        bzero(buffer, BUFFER_SIZE);
        if (bytes_read >= file_size)
            break;
```

```
    }
    fclose(file);
    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        cerr << "Usage: ./server  <port>\n";
        return -1;
    }

    int port = stoi(argv[1]);

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        perror("Socket creation failed");
        return 1;
    }
    struct sockaddr_in serv_addr;
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);
    int iSetOption = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char *)&iSetOption, sizeof(iSetOption));
    if (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) != 0)
    {
        perror("Bind failed");
        close(sockfd);
        return -1;
    }
    if (listen(sockfd, MAX_QUEUE) != 0)
    {
        perror("Listen failed");
        close(sockfd);
        return -1;
    }

    cout << "Server listening on port: " << port << "\n";

    try
    {
        if (!fs::exists(SUBMISSIONS_DIR))
            fs::create_directory(SUBMISSIONS_DIR);
        if (!fs::exists(EXECUTABLES_DIR))
            fs::create_directory(EXECUTABLES_DIR);
        if (!fs::exists(OUTPUTS_DIR))
            fs::create_directory(OUTPUTS_DIR);
        if (!fs::exists(COMPILER_ERROR_DIR))
            fs::create_directory(COMPILER_ERROR_DIR);
        if (!fs::exists(RUNTIME_ERROR_DIR))
            fs::create_directories(RUNTIME_ERROR_DIR);
    }
    catch (fs::filesystem_error &e)
    {
        cerr << "Error creating directories: " << e.what() << "\n";
        close(sockfd);
        return -1;
    }

    struct sockaddr_in client_addr;
```

```
    {
        int client_sockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

        string source_file = SUBMISSIONS_DIR + string("file.cpp");
        string executable = EXECUTABLES_DIR + string("file.o");
        string output_file = OUTPUTS_DIR + string("file.txt");
        string compiler_error_file = COMPILER_ERROR_DIR + string("file.err");
        string runtime_error_file = RUNTIME_ERROR_DIR + string("file.err");

        string compile_command = "g++ " + source_file + " -o " + executable + " > /dev/
null 2> " + compiler_error_file;
        string run_command = executable + " > " + output_file + " 2> " +
runtime_error_file;
        string compare_command = "diff " + output_file + " " + EXPECTED_OUTPUT + " > /dev/
null 2> /dev/null";

        string result_msg = "";
        string result_details = "";

        if (recv_file(client_sockfd, source_file) != 0)
        {
            close(client_sockfd);
            continue;
        }

        if (system(compile_command.c_str()) != 0)
        {
            result_msg = COMPILER_ERROR_MSG;
            result_details = compiler_error_file;
        }
        else if (system(run_command.c_str()) != 0)
        {
            result_msg = RUNTIME_ERROR_MSG;
            result_details = runtime_error_file;
        }
        else if (system(compare_command.c_str()) != 0)
        {
            result_msg = OUTPUT_ERROR_MSG;
            result_details = output_file;
        }
        else
        {
            result_msg = PASS_MSG;
        }

        if (send(client_sockfd, result_msg.c_str(), strlen(result_msg.c_str()), 0) == -1)
        {
            perror("Error sending result message");
            close(client_sockfd);
            continue;
        }

        if (!result_details.empty())
        {
            char buffer[BUFFER_SIZE];
            bzero(buffer, BUFFER_SIZE);

            FILE *file = fopen(result_details.c_str(), "rb");
            while (!feof(file))
            {
                size_t bytes_read = fread(buffer, 1, sizeof(buffer), file);
                if (send(client_sockfd, buffer, bytes_read, 0) == -1)
```

```
                bzero(buffer, BUFFER_SIZE);
            }
            fclose(file);
        }
        close(client_sockfd);
    }

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