ГУО “БГУИР”

Инженерно-экономический факультет

Кафедра экономической информатики

Отчёт по

Лабораторной работе № 1

**Создание последовательного сервера с установлением логического соединения TCP**

Подготовил:

Студент гр.110101

Пантус Р. В.

Проверил:

Бутов А. А.

Минск 2023

**Цель работы:** изучить методы создания серверов с установлением логического соединения *TCP*, используя алгоритм последовательной обработки запросов.

**Вариант 3:** Клиент посылает два числа серверу m и n, сервер возвращает m!+n! этих чисел назад клиенту.

Код клиентской части:

#include <iostream>

#include <cstring>

#include <sstream>

#include <utility>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <arpa/inet.h>

class Client {

public:

Client(const int& kPort\_ = 9984,

std::string kIpAddress\_ = "127.0.0.1") : kPort(kPort\_), kIpAddress(std::move(kIpAddress\_))

{

client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_fd < 0) {

throw std::runtime\_error("Cant create socket descriptor.");

}

server\_address.sin\_family = AF\_INET; // IPv4

server\_address.sin\_port = htons(kPort);

if (inet\_pton(AF\_INET, kIpAddress.c\_str(), &server\_address.sin\_addr) <= 0) {

throw std::runtime\_error("Cant convert IP address from text to binary form");

}

if (connect(client\_fd, (sockaddr\*)&server\_address, sizeof(server\_address)) < 0) {

throw std::runtime\_error("Cant connect to the server with ip: " + kIpAddress);

}

}

void sendMessage(const std::string& message) const {

send(client\_fd, message.c\_str(), message.size(), 0);

}

void sendMessage(int m, int n) {

// create message

m = htonl(m), n = htonl(n);

send(client\_fd, &m, sizeof(m), 0);

send(client\_fd, &n, sizeof(n), 0);

read(client\_fd, &received\_answer, sizeof(received\_answer));

received\_answer = ntohl(received\_answer);

std::cout << received\_answer << std::endl;

}

private:

const int kPort = 0;

int32\_t received\_answer = 0; // variable that stores sum of factorials

const std::string kIpAddress;

sockaddr\_in server\_address{}; // server information

int client\_fd = 0; // socket information

};

int main() {

const std::string& kMessage = "Input two non-negative integers -> get sum of factorials\nelse -> quit";

const std::string& kInputMsg = "Input: ";

Client client;

std::cout << kMessage << std::endl;

while (true) {

std::cout << std::endl << kInputMsg;

std::string request;

std::getline(std::cin, request);

std::stringstream input(request);

int m = 0, n = 0;

input >> m >> n;

char c = 0;

if (m < 0 || n < 0 || input.fail() || input.get(c)) {

std::cout << "Bye!" << std::endl;

break;

} else {

try {

client.sendMessage(m, n);

} catch (std::runtime\_error& e) {

std::cout << e.what() << std::endl;

}

}

}

return 0;

}

Код серверной части:

#include <iostream>

#include <stdexcept>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <arpa/inet.h>

#include "combinatorics.h"

class Server {

public:

Server() {

socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_fd < 0) {

throw std::runtime\_error("Cant create socket descriptor.");

}

// setup server information

server\_address.sin\_family = AF\_INET;

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

server\_address.sin\_port = htons(PORT);

// connect socket with port

if (bind(socket\_fd, reinterpret\_cast<sockaddr\*>(&server\_address), sizeof(server\_address)) < 0) {

throw std::runtime\_error("Cant connect socket with port.");

}

// put server in a passive mode

if (listen(socket\_fd, MAX\_QUERIES) < 0) {

throw std::runtime\_error("Cant listen to socket");

}

std::cout << "Listening! Port: " << PORT << " IpAddress: localhost" << std::endl;

}

~Server() {

close(socket\_fd);

}

int32\_t GetNumber(const int& new\_socket, const std::string& client\_readable\_ip) const {

int32\_t getted\_number = 0;

int retval = read(new\_socket, &getted\_number, sizeof(getted\_number));

if (!retval) {

throw std::runtime\_error("connection lost with: " + client\_readable\_ip);

}

return (int32\_t)ntohl(getted\_number); // convert from internet type order to host type

}

bool IsSocketConnected() const {

int error = 0; socklen\_t error\_len = sizeof(int);

int retval = getsockopt(socket\_fd, SOL\_SOCKET, SO\_ERROR, &error, &error\_len);

return !retval && !error;

}

[[noreturn]] void run() {

Combinatorics comb;

while (true) {

int new\_socket = accept(socket\_fd, (sockaddr\*)(&server\_address), (socklen\_t\*)&address\_length);

if (new\_socket < 0) {

close(new\_socket);

throw std::runtime\_error("Error while Accepting on socket");

}

std::string client\_readable\_ip(INET\_ADDRSTRLEN, '\0');

inet\_ntop(server\_address.sin\_family,

(sockaddr\*)&server\_address.sin\_addr,

client\_readable\_ip.data(),

client\_readable\_ip.size());

std::cout << "Connection established with: " + client\_readable\_ip << std::endl;

while (true) {

try {

auto received\_m = GetNumber(new\_socket, client\_readable\_ip);

auto received\_n = GetNumber(new\_socket, client\_readable\_ip);

int32\_t response = 0;

try {

response = htonl(comb.getSumOfFactorials(received\_n, received\_m));

} catch (std::out\_of\_range &e) {

std::cerr << '[' << e.what() << "] ";

}

ShowMessage(kRequestMsg, received\_m, received\_n, kResponseMsg, ntohl(response));

send(new\_socket, &response, sizeof(response), 0);

} catch (std::runtime\_error& e) {

// connection lost

std::cout << e.what() << std::endl;

break;

}

}

}

}

private: // functions

template <class T>

void ShowMessage(const T& message) {

std::cout << message << std::endl;

}

template <class T, class... Args>

void ShowMessage(const T& message, Args... args) {

std::cout << message << ' ';

ShowMessage(args...);

}

private: // variables

const int PORT = 9984; // TODO: возможность задать автоматически

const int MAX\_QUERIES = 3;

sockaddr\_in server\_address{}; // server information

const int address\_length = sizeof(server\_address);

int socket\_fd = 0; // file descriptor

const std::string kRequestMsg = "Request:";

const std::string kResponseMsg = "Response:";

};

int main() {

try {

Server server;

server.run();

} catch (std::runtime\_error& e) {

std::cerr << e.what() << std::endl;

return EXIT\_FAILURE;

}

}

-----combinatorics.h------

#pragma once

#include <iostream>

#include <vector>

#include <iostream>

#include <vector>

#include <array>

class Combinatorics {

public:

Combinatorics() {

factorial.resize(kMaxN, 1);

invfact.resize(kMaxN, 1);

for (int i = 1; i < kMaxN; ++i) {

factorial[i] = (factorial[i - 1] \* i) % kMod;

}

int n = kMaxN - 1;

invfact[n] = BinPow(factorial[n], kMod - 2);

for (int i = n; i > 0; --i) {

invfact[i - 1] = invfact[i] \* i % kMod;

}

}

int32\_t BinPow(int32\_t a, int32\_t x) {

if (x == 0) {

return 1;

} else if (x & 1) {

return (BinPow(a, x - 1) \* a) % kMod ;

} else {

int32\_t tmp = BinPow(a, x / 2);

return (tmp \* tmp) % kMod;

}

}

int32\_t C(int n, int r) {

auto invdenominator = (invfact[r] \* invfact[n - r]) % kMod;

return (factorial[n] \* invdenominator) % kMod;

}

int32\_t getFactorial(int32\_t n) {

if (n < 0 || n >= kMaxN) {

throw std::out\_of\_range("Invalid range");

}

return factorial[n];

}

int32\_t getSumOfFactorials(int32\_t n, int32\_t m) {

return (getFactorial(n) + getFactorial(m)) % kMod;

}

private:

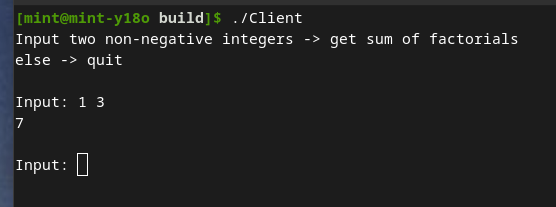
static const int kMaxN = 1e6, kMod = 1e9 + 7;

std::vector<int32\_t> factorial;

std::vector<int32\_t> invfact;

};

Скриншот работы клиентской части:



Скриншот работы серверной части:

