1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «**Сетевые взаимодействия**
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 5131001/20001 Маронова К.Д.

<*подпись*>

1. Преподаватель
2. Огнёв Р.А.

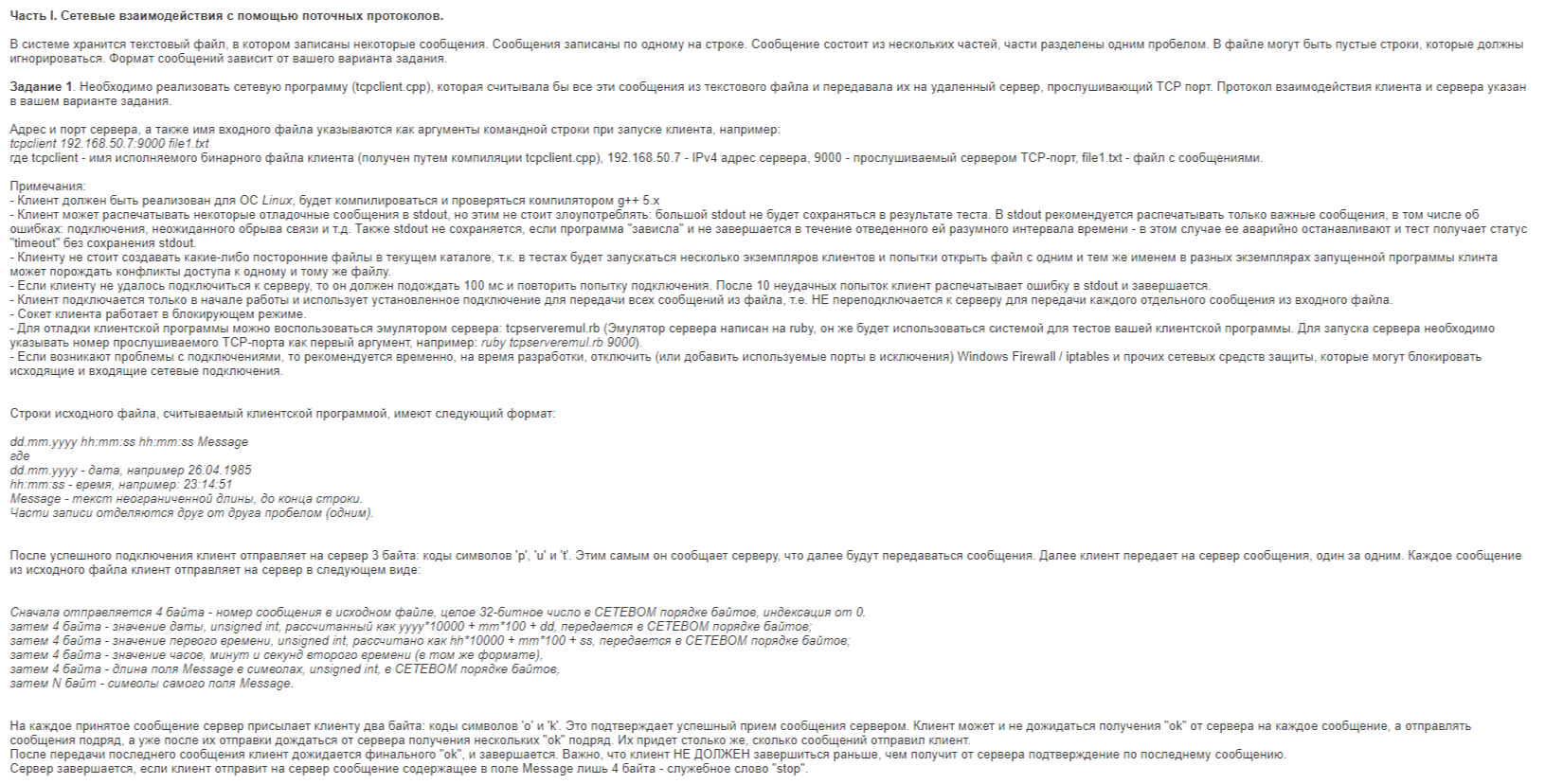
<*подпись*>

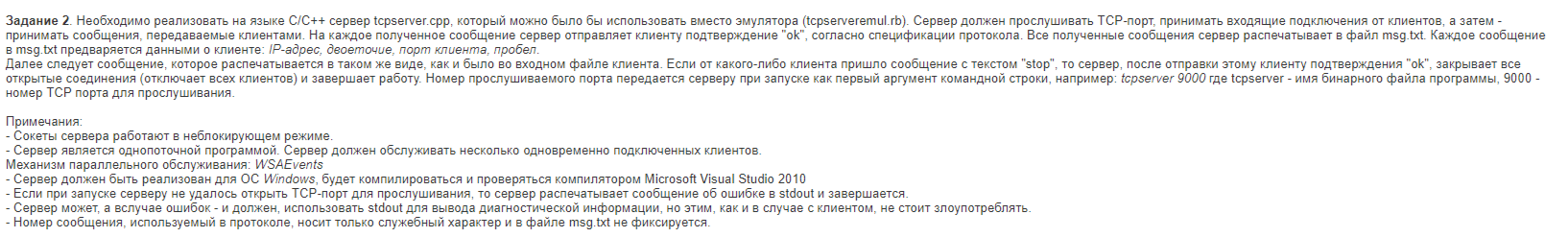
1. Санкт-Петербург
2. 2024

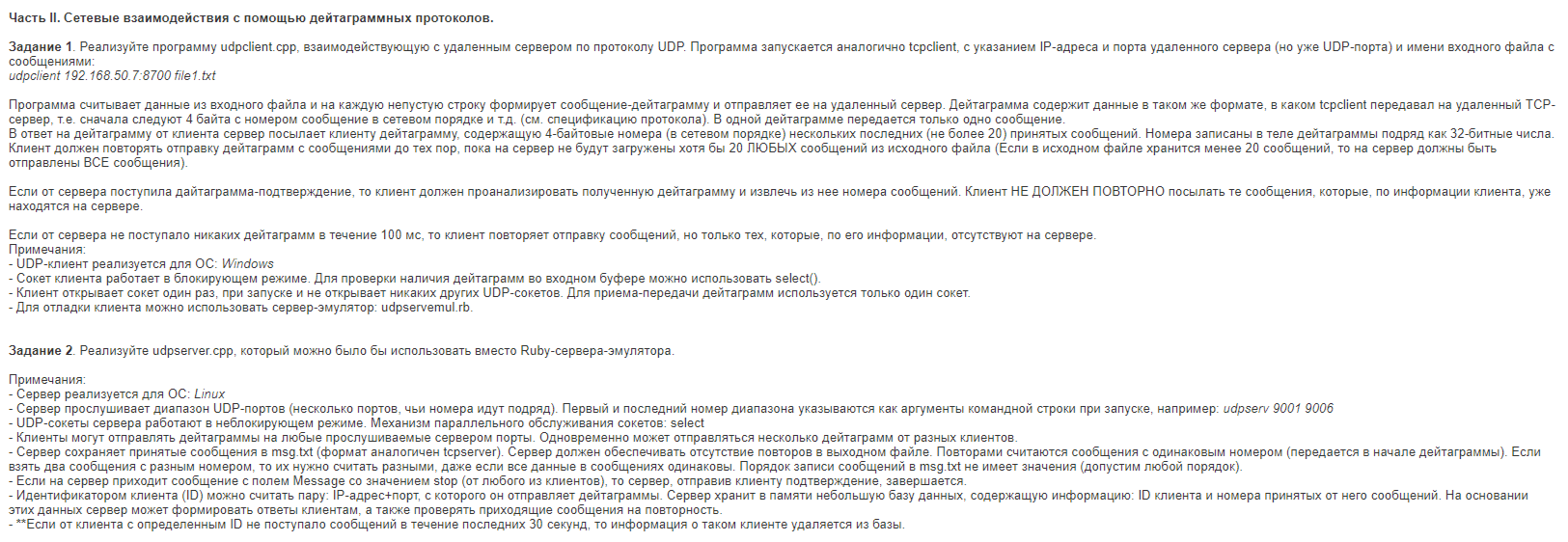
**1. Цель работы**

Изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

**2. Постановка задачи**







*Рисунок 1. «Вариант задания».*

**3. Описание решения**

**Описание структур данных и функций для tcpclient.cpp**

*Структуры данных:*

* struct sockaddr\_in: используется для хранения адреса и порта удаленного узла.
* char consol\_str[MAX\_MASSAGE\_LEN]: массив символов для хранения вводимой пользователем строки.
* char ip\_adress[15]: массив символов для хранения IP-адреса сервера.
* char file\_name[50]: массив символов для хранения имени файла.
* char port\_str[5]: массив символов для хранения номера порта сервера.

*Функции*:

|  |  |
| --- | --- |
| Имя функции | Назначение |
| int sock\_err(const char function, int s)\* | Обрабатывает ошибки сокета и выводит сообщение об ошибке. |
| void s\_close(int s) | Закрывает сокет. |
| unsigned int get\_host\_ipn(const char name)\* | Определяет IP-адрес узла по его имени. |
| unsigned int time\_convert(char time)\* | Преобразует время из формата HH:MM:SS в число. |
| unsigned int date\_convert(char date)\* | Преобразует дату из формата DD.MM.YYYY в число. |
| int send\_request(int s, char file\_name)\* | Отправляет http-запрос на удаленный сервер. |

**Описание структур данных и функций для tcpserver.cpp**

*Структуры данных:*

* struct sockaddr\_in: используется для хранения адреса и порта клиента.

*Функции***:**

|  |  |
| --- | --- |
| Имя функции | Назначение |
| int init() | Инициализирует сетевую библиотеку Winsock. |
| void deinit() | Очищает сетевую библиотеку Winsock. |
| int sock\_err(const char function, int s)\* | Обрабатывает ошибки сокета и выводит сообщение об ошибке. |
| void s\_close(int s) | Закрывает сокет. |
| int set\_non\_block\_mode(int s) | Устанавливает сокет в неблокирующий режим. |
| int get\_client\_data(int socket, int port, int ip) | Получает данные от клиента, парсит их и записывает в файл. |
| void delete\_and\_shift\_array(int array, int index)\* | Удаляет элемент из массива со сдвигом элементов. |

**Описание структур данных и функций для udpclient.cpp**

*Структуры данных:*

* **struct Message:**
  + msg\_id: uint32\_t - идентификатор сообщения
  + date: uint32\_t - дата
  + time1: uint32\_t - время 1
  + time2: uint32\_t - время 2
  + text\_len: uint32\_t - длина текста
  + text: char[35000 - 20] - текст сообщения
* **struct sockaddr\_in:**
  + sin\_family: short - тип адреса (AF\_INET)
  + sin\_port: short - порт
  + sin\_addr: in\_addr - IP-адрес

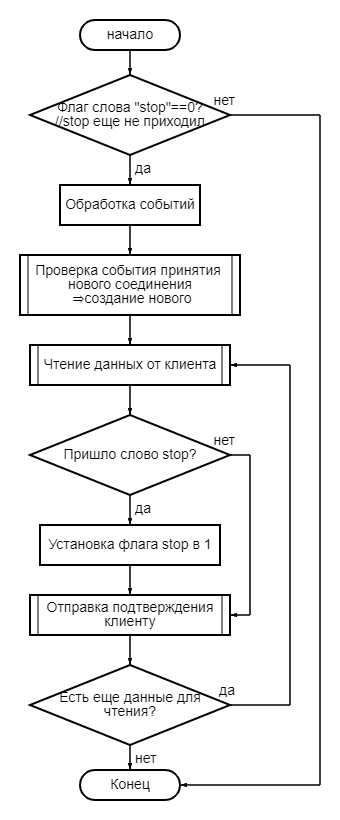
*Функции***:**

|  |  |
| --- | --- |
| Имя функции | Назначение |
| int init() | Инициализирует Winsock. |
| void deinit() | Очищает Winsock. |
| int sock\_err(const char function, int s)\* | Обрабатывает ошибки сокета. |
| void s\_close(int s) | Закрывает сокет. |
| unsigned int get\_host\_ipn(const char name)\* | Получает IP-адрес хоста. |
| unsigned int time\_convert(char time)\* | Преобразует время в формат uint32\_t. |
| unsigned int date\_convert(char date)\* | Преобразует дату в формат uint32\_t. |
| void send\_message(int s, struct Message msg, struct sockaddr\_in addr)\*\* | Отправляет сообщение на сервер. |
| void get\_lines(FILE file)\* | Считывает строки из файла и заполняет массив file\_lines. |
| unsigned int recv\_response(int s) | Принимает ответ от сервера. |

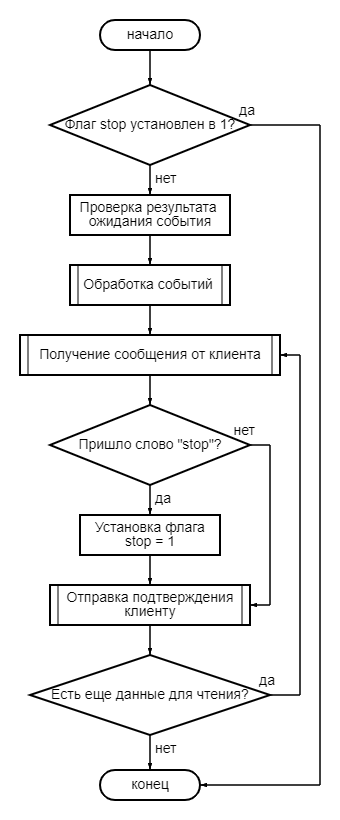
**Описание структур данных и функций для udpserver.cpp**

*Функции:*

|  |  |
| --- | --- |
| Имя функции | Назначение |
| int set\_non\_block\_mode(int s) | Устанавливает сокет в неблокирующий режим. |
| int sock\_err(const char function, int s)\* | Обрабатывает ошибки сокета. |
| void s\_close(int s) | Закрывает сокет. |
| void send\_to(int client\_ind, long acceptable\_msg\_table[500][500], sockaddr\_in addr)\* | Отправляет дейтаграмму клиенту. |
| void recv\_msg(struct sockaddr\_in addr, char buf, int ind, long acceptable\_msg\_table[500][500])\*\* | Обрабатывает сообщение от клиента. |



*Рисунок 2. «Блок-схема tcp server».*



*Рисунок 3. «Блок-схема udp server».*

**4. Выводы**

В ходе выполнения данной лабораторной работы были разработаны программы клиентов и серверов для разных операционных систем, взаимодействующие по разным протоколам. Были изучены блокирующий и неблокирующий режимы сокетов.

**Листинг программы:**

**tcpclient:**

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_MASSAGE\_LEN 341000

int sock\_err(const char\* function, int s)

{

int err;

err = errno;

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

close(s);

}

// Функция определяет IP-адрес узла по его имени.

// Адрес возвращается в сетевом порядке байтов.

unsigned int get\_host\_ipn(const char\* name)

{

struct addrinfo\* addr = 0;

unsigned int ip4addr = 0;

// Функция возвращает все адреса указанного хоста

// в виде динамического однонаправленного списка

if (0 == getaddrinfo(name, 0, 0, &addr))

{

struct addrinfo\* cur = addr;

while (cur)

{

// Интересует только IPv4 адрес, если их несколько - то первый

if (cur->ai\_family == AF\_INET)

{

ip4addr = ((struct sockaddr\_in\*) cur->ai\_addr)->sin\_addr.s\_addr;

break;

}

cur = cur->ai\_next;

}

freeaddrinfo(addr);

}

return ip4addr;

}

unsigned int time\_convert(char\* time){

char \*token\_time = strtok(time, ":");

int hour = atoi(token\_time)\*10000;

token\_time = strtok(NULL, ":");

int minute = atoi(token\_time)\*100;

token\_time = strtok(NULL, ":");

int seconds = atoi(token\_time);

return hour+minute+seconds;

}

unsigned int date\_convert(char\* date){

char \*token\_date = strtok(date, ".");

int day = atoi(token\_date);

token\_date = strtok(NULL, ".");

int month = atoi(token\_date)\*100;

token\_date = strtok(NULL, ".");

int year = atoi(token\_date)\*10000;

return day+month+year;

}

// Отправляет http-запрос на удаленный сервер

int send\_request(int s, char\* file\_name)

{

}

int main(int argc, char\* argv[])

{

int s;

struct sockaddr\_in addr;

//char consol\_str[MAX\_MASSAGE\_LEN];

char ip\_adress[15];

char file\_name[50];

char port\_str[5];

if(argc<2){

printf("Less arguments!\n");

return 0;

}

char\* token = strtok(argv[1], ":");

strncpy(ip\_adress, token, 15);

token = strtok(NULL, ":");

strncpy(port\_str, token, 5);

int port=atoi(port\_str);

//printf("%d\n",port);

//printf("пока все ок \n");

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

if (s < 0) return sock\_err("socket", s);

// Заполнение структуры с адресом удаленного узла

//printf("пока все ок \n");

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

addr.sin\_addr.s\_addr = get\_host\_ipn(argv[1]);

int try\_cnt=10;

int success\_flag=1;

//printf("заход в цикл вайл\n");

while(try\_cnt){

//printf("сейчас будет коннект\n");

if (connect(s, (struct sockaddr\*) &addr, sizeof(addr)) == 0) break;

printf("we keep trying to connecting\n");

try\_cnt--;

usleep(100000);

}

//printf("выход из цикла\n");

//////////////////////////////////////////////////

if(try\_cnt<=0){

success\_flag=0;

s\_close(s);

return sock\_err("connect", s);

}

if(success\_flag){

printf("success connection\n");

try\_cnt = send(s, "put", 3 \* sizeof(char), 0);

printf("put was sended\n");

}

// ВМЕСТО СЕНД РЕКВЕСТА //

/\* ТЕСТИРОВАНИЕ РАБОТЫ ПАРСИНГА СТРОК \*/

FILE \*file = fopen(argv[2], "r");

char line[MAX\_MASSAGE\_LEN]={0};

unsigned int msg\_id,msg\_cnt=0;

if (file==NULL){

printf("Error with file oppening\n");

return 0;

}

int reply=0;

int result;

char ok[3]={0};

while (fgets(line, MAX\_MASSAGE\_LEN, file)) {

// Парсинг строки

if(line[0]=='\n') continue;

msg\_id++;

msg\_cnt++;

char \*token = strtok(line, " "); // разделение строки по пробелам

char date\_str[11], time1\_str[9], time2\_str[9], message[MAX\_MASSAGE\_LEN]={0};

// Чтение и сохранение отдельных частей строки

if (token != NULL) strcpy(date\_str, token);

token = strtok(NULL, " ");

if (token != NULL) strcpy(time1\_str, token);

token = strtok(NULL, " ");

if (token != NULL) strcpy(time2\_str, token);

token = strtok(NULL, "\n");

if (token != NULL) strcpy(message, token);

// запись финальных значений

unsigned int htonl\_len = htonl(strlen(message));

unsigned int htonl\_date = htonl(date\_convert(date\_str));

unsigned int htonl\_time1 = htonl(time\_convert(time1\_str));

unsigned int htonl\_time2 = htonl(time\_convert(time2\_str));

msg\_id=htonl(msg\_id);

// отправка сообщений //

//printf("до отправки\n");

result = send(s, (const void \*)&msg\_id, 4, 0);

result =send(s, (const void \*)&htonl\_date, 4, 0);

result =send(s, (const void \*)&htonl\_time1, 4, 0);

result =send(s, (const void \*)&htonl\_time2, 4, 0);

result =send(s, (const void \*)&htonl\_len, 4, 0);

result =send(s, message, strlen(message), 0);

//printf("отправка сообщений завершена\n");

result = recv(s, ok, 2, 0);

//printf("ОК ещё не пришел, но реквест был\n");

if (ok[0] == 'o' && ok[1] == 'k') {

//printf("ОК пришел\n");

reply++;

}

//printf("\n%d\n", strlen(message));

}

int ok\_try\_cnt=10;

printf("message cnt: %d ; reply cnt: %d\n", msg\_cnt,reply);

if(msg\_cnt==reply) {

//printf("\nвсе успешно завершилось\n");

s\_close(s);

}

else {

while(msg\_cnt!=reply and ok\_try\_cnt){

result = recv(s, ok, 2, 0);

//printf("ОК ещё не пришел, но реквест был\n");

if (ok[0] == 'o' && ok[1] == 'k') {

//printf("ОК пришел\n");

reply++;

}

if (ok[0] == 'o' && ok[1]!='k'){

usleep(400000);

result = recv(s, ok, 2, 0);

if (ok[0] == 'k' || ok[1] == 'k'){

reply++;

}

}

ok\_try\_cnt--;

usleep(400000);

}

}

///////////////////////////

return 1;

}

**tcpserver:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#define SOCKET\_AMOUNT 1000

#define MAX\_LEN 350000

#include <windows.h>

#include <winsock2.h>

// Директива линковщику: использовать библиотеку сокетов

#pragma comment(lib, "ws2\_32.lib")

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <malloc.h>

int stop = 0;

int soc\_cl\_cnt = 0;

int put\_flag = 0;

int byte\_mode = 0;//1, если по 1 байту

FILE\* message\_file;

void delete\_and\_shift\_array(int\* array, int index) {

// Проверка корректности индекса

if (index < 0 || index >= soc\_cl\_cnt) {

return;

}

// Сдвиг элементов

for (int i = index; i < soc\_cl\_cnt - 1; i++) {

array[i] = array[i + 1];

}

// Уменьшение размера массива

soc\_cl\_cnt--;

}

int get\_client\_data(int socket, int port, int ip) {

char num\_msg[4], date[10] = { 0 }, time\_1[10] = { 0 }, time\_2[10] = { 0 };

unsigned int t1, t2, d, day, month, year, h1, h2, m1, m2, s1, s2;

int j = 0;

char num\_char;

int res;

int res\_num;

while (j != 4) {

Sleep(15);

res\_num = recv(socket, &num\_char, 1, 0);

Sleep(15);

num\_msg[j] = num\_char;

j++;

}

j = 0;

//printf("num recvest\n");

char data\_char = '\0';

while (j != 4) {

res = recv(socket, &data\_char, 1, 0);

Sleep(15);

date[j] = data\_char;

j++;

}

j = 0;

//res = recv(socket, date, 4, 0);

//printf("date recvest\n");

d = ntohl(\*(unsigned int\*)date);

day = d % 100;

month = (d % 10000) / 100;

year = d / 10000;

printf("\date: %02d.%02d.%04d ", day,month, year);

char time1\_char = '\0';

while (j != 4) {

res = recv(socket, &time1\_char, 1, 0);

Sleep(15);

time\_1[j] = time1\_char;

j++;

}

j = 0;

//res = recv(socket, time\_1, 4, 0);

//printf("time1 recvest\n");

t1 = ntohl(\*(unsigned int\*)time\_1);

h1 = t1 / 10000;

m1 = (t1 / 100) % 100;

s1 = t1 % 100;

printf("%02d:%02d:%02d ", h1, m1, s1);

//printf("\ntime 1: %u\n", t1);

//res = recv(socket, time\_2, 4, 0);

char time2\_char = '\0';

while (j != 4) {

res = recv(socket, &time2\_char, 1, 0);

Sleep(15);

time\_2[j] = time2\_char;

j++;

}

j = 0;

//printf("time2 recvest\n");

t2 = ntohl(\*(unsigned int\*)time\_2);

h2 = t2 / 10000;

m2 = (t2 / 100) % 100;

s2 = t2 % 100;

printf("%02d:%02d:%02d \n", h2, m2, s2);

char n[4];

//res = recv(socket, n, 4, 0);

char n\_char = '\0';

while (j != 4) {

res = recv(socket, &n\_char, 1, 0);

Sleep(15);

n[j] = n\_char;

j++;

}

j = 0;

//printf("len bil\n");

//printf("len recvest\n");

char msg[MAX\_LEN]={0};

int msg\_len = ntohl(\*(unsigned int\*)n);

int i = 0;

for (i; i < 350000; i++) {

recv(socket, &msg[i], 1, 0);

//printf("message[i]: %s\n", &msg[i]);

Sleep(15);

if (msg[i] == 0)

break;

}

if (i != msg\_len) {

printf("ne sovpalo\n");

printf("i:%d!=len:%d\n",i,msg\_len);

return 1;

}

printf("message: %s\n", msg);

//printf("%u %u.%u.%u %u:%u:%u %u:%u:%u %s\n", port, day, month, year, h1, m1, s1, h2, m2, s2, msg);

//Sleep(1000000000000);

if (strncmp("stop", msg, 4) == 0 && strlen(msg) == 4)

{

printf("STOP\n");

stop = 1;

put\_flag = 0;

}

fprintf(message\_file, "%d.%d.%d.%d:%u %02u.%02u.%04u %02u:%02u:%02u %02u:%02u:%02u %s\n", (ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, ip & 0xFF, port, day, month, \

year, h1, m1, s1, h2, m2, s2, msg);

return 1;

}

int set\_non\_block\_mode(int s)

{

unsigned long mode = 1;

return ioctlsocket(s, FIONBIO, &mode);

}

int init()

{

// Для Windows следует вызвать WSAStartup перед началом использования сокетов

WSADATA wsa\_data;

return (0 == WSAStartup(MAKEWORD(2, 2), &wsa\_data));

}

void deinit()

{

// Для Windows следует вызвать WSACleanup в конце работы

WSACleanup();

}

int sock\_err(const char\* function, int s)

{

int err;

err = WSAGetLastError();

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

closesocket(s);

}

int main(int argc, char\* argv[])

{

int s;

struct sockaddr\_in addr;

FILE\* std\_out = fopen("stdout.txt", "w+");

// Инициалиазация сетевой библиотеки

init();

// Создание TCP-сокета

int ip = 0;

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

set\_non\_block\_mode(s);

printf("socket created\n");

if (s < 0) return sock\_err("socket", s);

// Заполнение адреса прослушивания

int port = atoi(argv[1]);

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port); // Сервер прослушивает порт 9000

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); // Все адреса

// Связывание сокета и адреса прослушивания

if (bind(s, (struct sockaddr\*)&addr, sizeof(addr)) < 0)

return sock\_err("bind", s);

// Начало прослушивания

if (listen(s, 1) < 0)

return sock\_err("listen", s);

WSAEVENT events[2];

events[0] = WSACreateEvent();

events[1] = WSACreateEvent();

// WSAEventSelect для прослушивающего сокета

WSAEventSelect(s, events[0], FD\_ACCEPT);

int\* socet\_client=(int\*)calloc(SOCKET\_AMOUNT, sizeof(int));

// WSAEventSelect для клиентских сокетов

for (int i = 0; i < SOCKET\_AMOUNT; i++) {

WSAEventSelect(socet\_client[i], events[1], FD\_READ | FD\_WRITE | FD\_CLOSE);

}

int rcv;

char put[4] = { 0 };

//char msg[MAX\_LEN] = { 0 };

message\_file = fopen("msg.txt", "w+");

while (!stop) {

WSANETWORKEVENTS ne;

DWORD dwWaitResult = WSAWaitForMultipleEvents(2, events, FALSE, 1000, FALSE);

//printf("EXIT FTOM MULRIPLE\n");

// Сброс событий

WSAResetEvent(events[0]);

WSAResetEvent(events[1]);

// Обработка события на прослушивающем сокете

if (dwWaitResult == WSA\_WAIT\_FAILED) {

fprintf(std\_out,"WSAWaitForMultipleEvents failed\n");

break;

}

else if (0 == WSAEnumNetworkEvents(s, events[0], &ne) && (ne.lNetworkEvents & FD\_ACCEPT)) {

sockaddr\_in new\_s\_addr;

int new\_s\_addr\_len = sizeof(new\_s\_addr);

int len\_s = sizeof(addr);

printf("wait for accept\n");

int new\_s = accept(s, (struct sockaddr\*)&addr, &len\_s);

if (new\_s < 0) {

fprintf(std\_out, "ERROR with accept\n");

return sock\_err("accept", s);

break;

}

set\_non\_block\_mode(new\_s);

ip = ntohl(addr.sin\_addr.s\_addr);

printf("Client #%d connected: %d.%d.%d.%d:%i\n", soc\_cl\_cnt + 1,(ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, (ip) & 0xFF, port);

// Добавление нового сокета в массив

socet\_client[soc\_cl\_cnt] = new\_s;

//soc\_cl\_cnt++;

WSAEventSelect(socet\_client[soc\_cl\_cnt], events[1], FD\_READ | FD\_WRITE | FD\_CLOSE);

soc\_cl\_cnt++;

}

for (int i = 0; i < soc\_cl\_cnt; i++) {

if (0 == WSAEnumNetworkEvents(socet\_client[i], events[1], &ne)) {

// Обработка событий для сокета клиента

if (ne.lNetworkEvents & FD\_READ) {

// Есть данные для чтения, можно вызывать recv/recvfrom на socet\_client[i]

//printf("WE IN READ\n");

if (put\_flag == 0) {

char put\_char='\0';

rcv = recv(socet\_client[i], put, 3, 0);

Sleep(15);

printf("Put recvest\n");

printf("Put arrey: %s\n", put);

//Sleep(500);

put[3] = '\0';

/\* Проверка на put \*/

if (rcv < 0) s\_close(socet\_client[i]);

if (strcmp("put", put) == 0 || put[0]=='t') {

if (put[0] == 't') byte\_mode = 1;

printf("Put comlete\n");

put\_flag = 1;

//send(socet\_client[i], "ok", 2, 0);//ok

}

//continue;

}

else {

int try\_recv = get\_client\_data(socet\_client[i], port, ip);

if (stop == 1) {

put\_flag = 0;

send(socet\_client[i], "ok", 2, 0);//ok

s\_close(socet\_client[i]);

s\_close(s);

delete\_and\_shift\_array(socet\_client, i);

break;

}

else {

printf("Sending OK \n");

send(socet\_client[i], "ok", 2, 0);

}

}

}

if (ne.lNetworkEvents & FD\_CLOSE) {

//printf("WE IN CLOSE\n");

// Удаленная сторона закрыла соединение

put\_flag = 0;

stop = 0;

s\_close(socet\_client[i]);

delete\_and\_shift\_array(socet\_client, i);

//break;

}

}

}

}

fclose(message\_file);

//Sleep(10000000);

s\_close(s);

deinit();

return 0;

}

**udpclient:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#define MAX\_RECV\_SIZE 20 \* sizeof(int)

#define MAX\_MSG\_SIZE 35000

#include <WS2tcpip.h>

#include <errno.h>

#include <windows.h>

#include <winsock2.h>

// Директива линковщику: использовать библиотеку сокетов

#pragma comment(lib, "ws2\_32.lib")

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdint.h>

int max\_nums = 20;

int lines\_cnt = 0;

int success\_lines\_cnt = 0;

int sent\_count = 0;

int sent\_messages[MAX\_MSG\_SIZE];

int last\_recived, num\_received = 0;

struct Message {

uint32\_t msg\_id;

uint32\_t date;

uint32\_t time1;

uint32\_t time2;

uint32\_t text\_len;

char text[35000 - 20];

};

struct Message\* file\_lines = (struct Message\*)malloc(sizeof(struct Message)\*MAX\_MSG\_SIZE);

int init()

{

// Для Windows следует вызвать WSAStartup перед началом использования сокетов

WSADATA wsa\_data;

return (0 == WSAStartup(MAKEWORD(2, 2), &wsa\_data));

}

void deinit()

{

// Для Windows следует вызвать WSACleanup в конце работы

WSACleanup();

}

int sock\_err(const char\* function, int s)

{

int err;

err = WSAGetLastError();

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

closesocket(s);

}

unsigned int time\_convert(char\* time) {

char\* token\_time = strtok(time, ":");

unsigned int hour = atoi(token\_time) \* 10000;

token\_time = strtok(NULL, ":");

unsigned int minute = atoi(token\_time) \* 100;

token\_time = strtok(NULL, ":");

unsigned int seconds = atoi(token\_time);

return hour + minute + seconds;

}

unsigned int date\_convert(char\* date) {

char\* token\_date = strtok(date, ".");

unsigned int day = atoi(token\_date);

token\_date = strtok(NULL, ".");

unsigned int month = atoi(token\_date) \* 100;

token\_date = strtok(NULL, ".");

unsigned int year = atoi(token\_date) \* 10000;

return day + month + year;

}

void send\_message(int s, struct Message\* msg,struct sockaddr\_in\* addr)

{

if (sendto(s, (const char\*)msg, 20+ sizeof(msg->text), 0, (struct sockaddr\*)addr, sizeof(struct sockaddr\_in))>0) {

printf("sendto complete for message #%d\n", ntohl(msg->msg\_id));

}

else {

printf("ERROR: Sending error.\n");

sock\_err("sendto", s);

}

// char str[20]; // предполагаем, что число не будет занимать больше 20 символов

// sprintf(str, "%d", msg->msg\_id);

// int res=sendto(s, (char\*)&msg, sizeof(struct Message), 0, (struct sockaddr\*)addr, sizeof(struct sockaddr\_in\*));

}

void get\_lines(FILE\* file) {

int msg\_num = 0;

char line[34100];

while (fgets(line, sizeof(line), file)) {

// Разбор строки сообщения

if (line[0] == '\n') continue;

char date[12], time1[12], time2[12], message[34100] = { 0 };

sscanf(line, "%[^ ] %[^ ] %[^ ] %[^\n]", date, time1, time2, message);

msg\_num++;

//парсинг данных в структуру

char line[MAX\_MSG\_SIZE];

struct Message msg;

unsigned int len = strlen(message);

unsigned int date\_int = date\_convert(date);

unsigned int time1\_int = time\_convert(time1);

unsigned int time2\_int = time\_convert(time2);

msg.msg\_id = htonl(msg\_num);

msg.date = htonl(date\_int);

msg.time1 = htonl(time1\_int);

msg.time2 = htonl(time2\_int);

msg.text\_len = htonl(len);

strcpy(msg.text, message);

file\_lines[lines\_cnt] = msg;

lines\_cnt++;

}

}

// Функция принимает дейтаграмму от удаленной стороны.

// Возвращает 0, если в течение 100 миллисекунд не было получено ни одной дейтаграммы

unsigned int recv\_response(int s)

{

char datagram[1024];

struct timeval tv = { 0, 100 \* 1000 }; // 100 msec

int res;

fd\_set fds;

FD\_ZERO(&fds); FD\_SET(s, &fds);

// Проверка - если в сокете входящие дейтаграммы

// (ожидание в течение tv)

int received;

res = select(s + 1, &fds, 0, 0, &tv);

printf("res: %d\n", res);

if (res > 0)

{

// Данные есть, считывание их

struct sockaddr\_in addr;

int addrlen = sizeof(addr);

int recv\_ids[20];

memset(recv\_ids, 0, 20\*sizeof(int));

int received = recvfrom(s, (char\*)recv\_ids, sizeof(recv\_ids)\*lines\_cnt, 0, (struct sockaddr\*)&addr, &addrlen);

printf("recv good\n");

if (received <= 0)

{

// Ошибка считывания полученной дейтаграммы

sock\_err("recvfrom", s);

//return 0;

}

else {

last\_recived = num\_received;

num\_received = received / sizeof(int);

if (num\_received!=last\_recived) {

sent\_count++;

}

for (int i = 0; i < num\_received; i++) {

printf("index of success: %u\n", ntohl(recv\_ids[i]));

printf("Semt message counter = %d\n", sent\_count);

sent\_messages[ntohl(recv\_ids[i])] = 1;

}

}

}

else

{

sock\_err("select", s);

//return 0;

}

return 1;

}

unsigned int get\_host\_ipn(const char\* name)

{

struct addrinfo\* addr = 0;

unsigned int ip4addr = 0;

// Функция возвращает все адреса указанного хоста

// в виде динамического однонаправленного списка

if (0 == getaddrinfo(name, 0, 0, &addr))

{

struct addrinfo\* cur = addr;

while (cur)

{

// Интересует только IPv4 адрес, если их несколько - то первый

if (cur->ai\_family == AF\_INET)

{

ip4addr = ((struct sockaddr\_in\*)cur->ai\_addr)->sin\_addr.s\_addr;

break;

}

cur = cur->ai\_next;

}

freeaddrinfo(addr);

}

return ip4addr;

}

int main(int argc, char\* argv[])

{

int s;

struct sockaddr\_in addr;

// Инициалиазация сетевой библиотеки

init();

// Создание UDP-сокета

s = socket(AF\_INET, SOCK\_DGRAM, 0);

if (s < 0)

return sock\_err("socket", s);

char\* server\_ip = strtok(argv[1], ":");

char\* server\_port = strtok(NULL, ":");

int port = atoi(server\_port);

char\* filename = argv[2];

// Заполнение структуры с адресом удаленного узла

memset(&addr, 0, sizeof(addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port); // Порт DNS - 53

addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

// Открытие файла сообщений

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file: %s\n", filename);

return 1;

}

// Массив для хранения номеров уже отправленных сообщений (макс. 20)

int received\_nums[20];

memset(received\_nums, -1, sizeof(received\_nums));

int num\_received = 0; // Количество полученных подтверждений

// Чтение и отправка сообщений из файла

get\_lines(file);

int i = 0;

memset(sent\_messages, 0, sizeof(sent\_messages)); // Инициализация нулями

while (1) {

while (i < lines\_cnt) {

if (!sent\_messages[i + 1]) {

// 8. Отправка сообщения на сервер

send\_message(s, &file\_lines[i], &addr);

//sent\_messages[i+1] = 0; // Отмечаем сообщение как отправленное

}

i++;

}

int recv\_resp=recv\_response(s);

for (int i = 1; i <= lines\_cnt; i++) {

if (sent\_messages[i] != 1) {

send\_message(s, &file\_lines[i-1], &addr);

}

}

// 12. Выход, если отправлено 20 сообщений

if (sent\_count >= 20 || sent\_count== lines\_cnt) {

printf("All messages successfully sent\n");

break;

}

}

// Закрытие сокета

s\_close(s);

deinit();

//Sleep(10000000000);

return 0;

}

**udpserver:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/select.h>

#include <unistd.h>

#include <sys/types.h>

#include <netdb.h>

#include <errno.h>

#define MAX\_CLIENTS 500

#define MAX\_MSG\_SIZE 4096

#define MSG\_FILE "msg.txt"

int server\_sockfd[50]={0};

//long acceptable\_msg\_table[MAX\_CLIENTS][MAX\_CLIENTS]={-1};

int STOP=0;

int start\_port;

int end\_port;

int MAX\_SIZE=500;

FILE\* file;

int set\_non\_block\_mode(int s)

{

int fl = fcntl(s, F\_GETFL, 0);

return fcntl(s, F\_SETFL, fl | O\_NONBLOCK);

}

int sock\_err(const char\* function, int s)

{

int err;

err = errno;

fprintf(stderr, "%s: socket error: %d\n", function, err);

return -1;

}

void s\_close(int s)

{

close(s);

}

void send\_to(int client\_ind, long acceptable\_msg\_table[500][500], sockaddr\_in\* addr) {

int datagram\_size=0;

char datagram[1000];

char num\_msg\_str[5] = { 0 };

int num;

for (int j = 0; j < MAX\_CLIENTS; j++) {

if (acceptable\_msg\_table[client\_ind][j] != -1) {

num=htonl(acceptable\_msg\_table[client\_ind][j]);

//printf("num: %d\n", ntohl(num));

memcpy(num\_msg\_str, &num, 4);

for (int i = 0; i < 4; i++) {

datagram[datagram\_size + i] = num\_msg\_str[i];

}

datagram\_size += 4;

}

}

//printf(datagram);

int res=sendto(server\_sockfd[client\_ind], datagram, datagram\_size, 0, (struct sockaddr\*)&addr[client\_ind], sizeof(struct sockaddr\_in));

printf("sendto: %d bytes = %d\n", res, datagram\_size);

}

// Функция для обработки сообщения от клиента

void recv\_msg(struct sockaddr\_in\* addr, char\* buf, int ind, long acceptable\_msg\_table[500][500]) {

uint32\_t msg\_id, date, time1, time2, msg\_len;

memcpy(&msg\_id, buf, 4);

memcpy(&date, buf + 4, 4);

memcpy(&time1, buf + 8, 4);

memcpy(&time2, buf + 12, 4);

memcpy(&msg\_len, buf + 16, 4);

msg\_id=ntohl(msg\_id);

date=ntohl(date);

msg\_id=ntohl(msg\_id);

time1=ntohl(time1);

time2=ntohl(time2);

msg\_len=ntohl(msg\_len);

unsigned int num;

char\* p = (char\*)&num;

for (int j = 0; j < 4; j++, ++p)

\*p = buf[j];

num = ntohl(num);

int message\_id=num;

//printf("Now we have #%d message\n",message\_id);

// Проверка на повтор

printf("Message\_id :%d\n",message\_id);

for(int i=0;i<25;i++){

//printf("Acceptable array: %d ",acceptable\_msg\_table[ind][i]);

if (acceptable\_msg\_table[ind][i]==message\_id){

printf("duplicate message #%d\n",message\_id);

return;

}

else if (acceptable\_msg\_table[ind][i]==-1) {

acceptable\_msg\_table[ind][i] = message\_id;

break;

//printf("Now we have #%d message\n",acceptable\_msg\_table[ind][i]);

}

}

//для айпишки в файл

struct sockaddr\_in tmp\_addr;

socklen\_t addr\_len = sizeof(tmp\_addr);

getsockname(server\_sockfd[ind], (struct sockaddr\*)&tmp\_addr, &addr\_len);

printf("indx: %d\n", ind);

int port = ntohs(tmp\_addr.sin\_port);

//printf("addr[ind]: %d\n",addr[ind]);

int ip=ntohl(addr[ind].sin\_addr.s\_addr);

printf("%d.%d.%d.%d:%d %02d.%02d.%04d %02d:%02d:%02d %02d:%02d:%02d %s\n",(ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, ip & 0xFF,port, date % 100, (date % 10000) / 100, date / 10000,time1 / 10000, (time1 % 10000) / 100, time1 % 100,time2 / 10000, (time2 % 10000) / 100, time2 % 100, buf + 20);

// Запись сообщения в файл

if (file) {

fprintf(file, "%d.%d.%d.%d:%d %02d.%02d.%04d %02d:%02d:%02d %02d:%02d:%02d %s\n",(ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, ip & 0xFF,port, date % 100, (date % 10000) / 100, date / 10000,time1 / 10000, (time1 % 10000) / 100, time1 % 100,time2 / 10000, (time2 % 10000) / 100, time2 % 100, buf + 20);

}

}

int main(int argc, char\* argv[]) {

long acceptable\_msg\_table[MAX\_CLIENTS][MAX\_CLIENTS]={-1};

struct sockaddr\_in addr[10000];

//printf("%u\n", ntohl(13015052));

start\_port = atoi(argv[1]);

end\_port = atoi(argv[2]);

// Инициализация сокетов

int n\_clients = 0;

for (int port = 0; port < end\_port - start\_port + 1; port++) {

server\_sockfd[port] = socket(AF\_INET, SOCK\_DGRAM, 0);

//printf("socket: %d\n", server\_sockfd[port - start\_port]);

if (server\_sockfd[port] == -1) {

perror("socket");

return 1;

}

set\_non\_block\_mode(server\_sockfd[port]);

memset(&addr, 0, sizeof(addr));

addr[port].sin\_family = AF\_INET;

addr[port].sin\_port = htons(start\_port+port);

addr[port].sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(server\_sockfd[port], (struct sockaddr\*)&addr[port], sizeof(addr[port])) == -1) {

perror("bind");

return 1;

}

}

fd\_set rfd;

fd\_set wfd;

int nfds = server\_sockfd[0];

//acceptable\_msg\_table=(int\*\*)malloc(MAX\_CLIENTS\*sizeof(int\*));

for (int i = 0; i < MAX\_CLIENTS; i++) {

for (int j = 0; j < MAX\_CLIENTS; j++){

acceptable\_msg\_table[i][j]=-1;

}

}

file = fopen(MSG\_FILE, "w+");

while (!STOP) {

FD\_ZERO(&rfd);

FD\_ZERO(&wfd);

// Добавление сокетов в множество для select

for (int i = 0; i < end\_port - start\_port + 1; i++) {

FD\_SET(server\_sockfd[i], &wfd);

FD\_SET(server\_sockfd[i], &rfd);

if (server\_sockfd[i] > nfds) {

nfds = server\_sockfd[i];

}

}

struct timeval tv = {0, 1000};

//printf("nfds: %d\n", nfds);

int ret = select(nfds+1, &rfd, &wfd, 0, &tv);

if (ret == -1) {

perror("select");

continue;

}

else if (ret == 0) {

perror("select\_time\_out");

continue;

}

//struct Client tmp\_client;

// Обработка событий

for (int i = 0; i < end\_port - start\_port+1; i++) {

if (FD\_ISSET(server\_sockfd[i], &rfd)) {

//printf("IS SET\n");

//char buf[MAX\_MSG\_SIZE]={0};

char buf[MAX\_MSG\_SIZE]={0};

//struct sockaddr\_in from;

socklen\_t len\_s=sizeof(addr[i]);

//memset(&buf,0,MAX\_MSG\_SIZE);

int recv\_len = recvfrom(server\_sockfd[i], buf, MAX\_MSG\_SIZE, 0,(struct sockaddr\*)&addr[i], &len\_s );

if (recv\_len == -1) {

perror("recvfrom");

continue;

}

else{

printf("recv %d bytes\n", recv\_len);

//printf(buf);

}

recv\_msg(addr,buf,i, acceptable\_msg\_table);

if (strncmp(&buf[20], "stop", 4) == 0) {

send\_to(i,acceptable\_msg\_table,addr);

printf("STOP\n");

STOP=1;

//usleep(1000000000);

for(int h=0;h<end\_port - start\_port + 1;h++){

s\_close(server\_sockfd[h]);

}

fclose(file);

return 1;

}

else{

send\_to(i,acceptable\_msg\_table,addr);

}

}

else{

for (int k = 0; k < MAX\_CLIENTS; k++){

acceptable\_msg\_table[i][k] = -1;

}

}

//usleep(10000);

}

}

for(int h=0;h<end\_port - start\_port + 1;h++){

s\_close(server\_sockfd[h]);

}

fclose(file);

return 0;

}