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

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**«Сетевые взаимодействия»**

1. по дисциплине «Операционные системы»
2. Выполнил
3. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. Крундышев В.М.

<*подпись*>

1. Санкт-Петербург
2. 2022
3. **Цель работы**

Цель работы – изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

1. **Задачи**

Реализовать следующие программы:

* *tcpclient*: ОС – Windows, реализовать TCP клиент
* *udpclient*: ОС – Linux, реализовать UDP клиент
* *udpserver*: ОС – Windows, реализовать UDP сервер, функционирующий в неблокирующем режиме для нескольких пользователей. Механизм параллельного обслуживания сокетов: select
* *tcpserver*: ОС – Linux, релизовать TCP сервер, функционирующий в неблокирующем режиме для нескольких пользователей. Механизм параллельного обслуживания: poll

1. **Ход работы**
   1. **Алгоритм работы TCP-сервера**

На рисунке 1 представлена блок-схема алгоритма работы TCP-сервера.

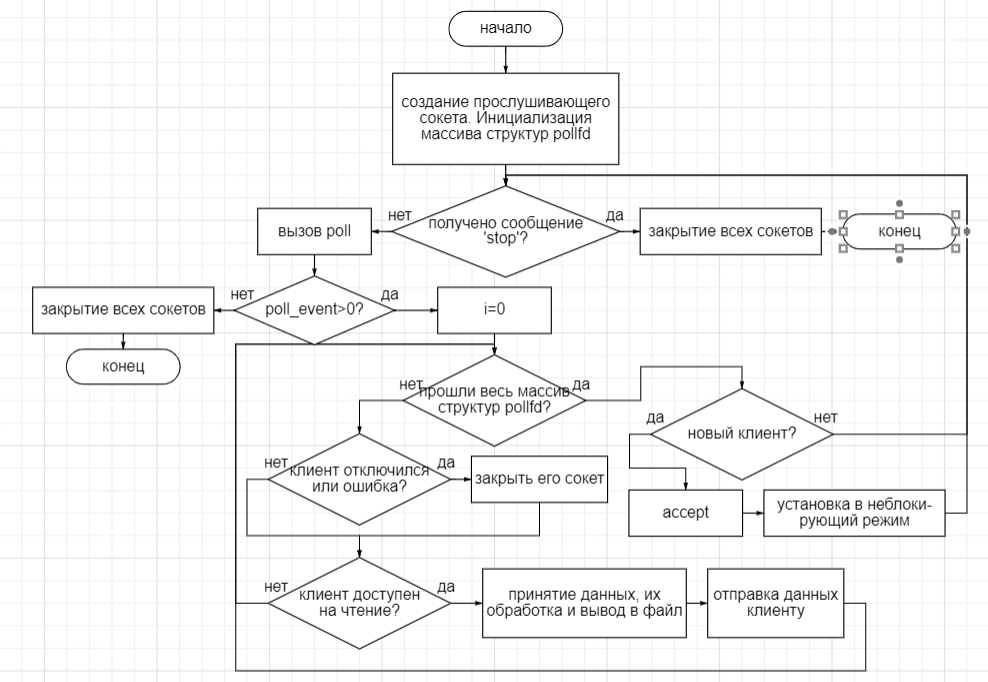


Рисунок 1 – Блок-схема алгоритма работы TCP-сервера

Перед началом работы с клиентами заполняется массив структур pollfd – в каждом элементе хранится сокет, запрошенные события и возникшие события. Запрошенные и возникшие события – это битовые маски – каждый бит говорит о событии.

Далее вызывается функция poll – функция просматривает сколько структур в массиве имеют ненулевое поле revents (т.е. с обнаруженными событиями). Если таких структур не ноль, то проверяется массив, находятся эти структуры и обрабатываются в соответствии с событием.

Для написания сервера не использовались дополнительные структуры данных. В таблице 1 приведены разработанные для сервера функции.

Таблица 1 – Разработанные для сервера функции

|  |  |  |  |
| --- | --- | --- | --- |
| *Имя функции* | *Аргументы* | *Возвращаемое значение* | *Описание* |
| void byte\_to\_num(char\*, unsigned char); | Указатель на число,  Входной байт | - | Перевод байта в двухразрядное число десятичной системы счисления |
| void data\_processing(int, char\*); | Сокет, строка | - | Обработка полученной информации от клиента. Форматирование строки для вывода в файл |
| void reverse(char s[]); | Строка | - | Разворачивает строку |
| void itoa(int n, char s[]); | Число, строка | - | Переводит число в строку |
| int print\_error(const char\*, int); | Тест ошибки, сокет | Код ошибки | Вывод сообщения о случившейся ошибке |
| int set\_unblocked\_mode(int); | Сокет | Результат установки | Перевод в неблокирующий режим |

* 1. **Алгоритм работы TCP-клиента**

Создается сокет, далее обрабатывается файл – сохраняются все непустые строки в обработанном виде. После этого происходит 10 попыток подключиться к серверу. Если попытка неудачная и не 10-ая, то происходит ожидание 100 мс, после чего происходит новая попытка. Если с 10-ой попытки не получилось соединиться, то клиент закрывается и программа завершается. В противном случае происходит отправка всех сообщений на сервер, после чего ожидание принятия сообщений от сервера. Клиент не завершит работу до тех пор, пока не получит столько же сообщений от сервера, сколько сам ему отправил.

* 1. **Алгоритм работы UDP-сервера**

На рисунке 2 представлена блок-схема работы алгоритма UDP-сервера.

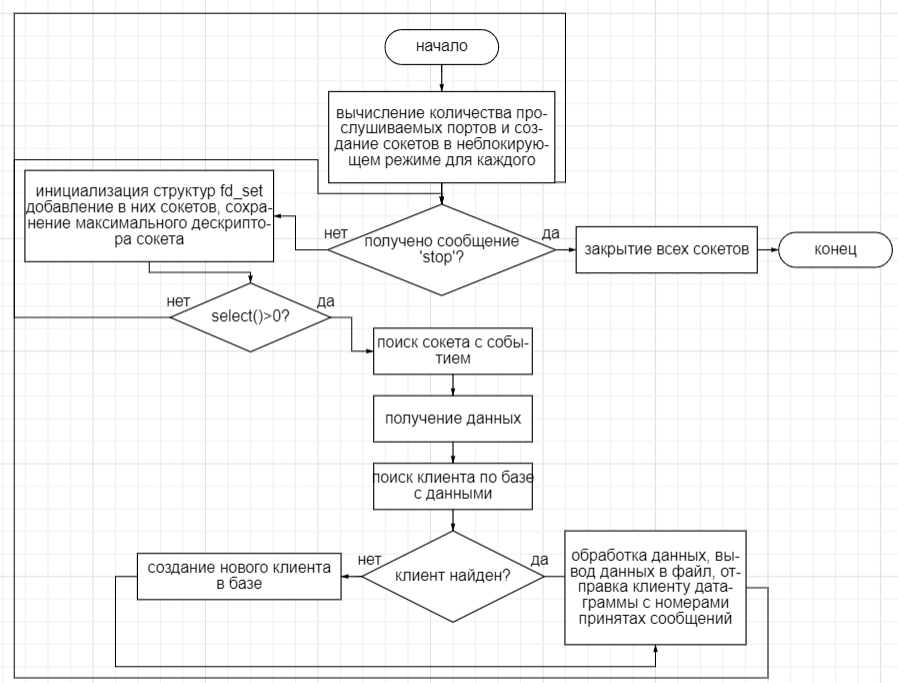


Рисунок 2 – Блок-схема алгоритма работы UDP-сервера

UDP-сервер подготавливает сокеты на все прослушиваемые порты, после чего запускается цикл, в котором инициализируются структуры fd\_set – после вызова select они будут содержать дескрипторы сокетов, на которых произошли события. Если происходит событие, то ищется сокет с этим событием, после этого принимаются данные. Далее по базе с клиентами ищется, была ли уже работа с данным клиентом. Если нет, то в базу добавляется клиент. Далее с найденным (новым) клиентом происходит работа – обрабатывается полученная информация, выводится в файл и клиенту отправляется датаграмма с номерами полученных от него сообщений.

Для написания сервера были написаны следующие структуры:

typedef struct

{

unsigned int get\_msg; // количество принятых сервером сообщений

int msg\_num[256]; // Массив для хранения номеров принятых сообщений

int last\_n\_msg[20];

sockaddr\_in addr;

} profile;

Данная структура используется для хранения информации о работе с клиентами – с какими клиентами уже была проведена работа, сколько сообщений от них было получено, какие номера у этих сообщений.

В таблице 2 приведены написанные функции для работы сервера.

Таблица 2 – Написанные функции для UDP-сервера

|  |  |  |  |
| --- | --- | --- | --- |
| *Имя функции* | *Аргументы* | *Возвращаемое значение* | *Описание* |
| void byte\_to\_num(char\*, unsigned char); | Строка, байт | - | Перевод байта в двухразрядное число десятичной системы счисления |
| int find\_profile(sockaddr\_in addr); | ip-адрес клиента | Индекс клиента в массиве | Поиск клиента по базе данных |
| int format\_msg(unsigned int clienumb, char\* msg, int msglen, int port); | Индекс клиента, сообщение, длина сообщения, порт клиента | 1 – получено сообщение «stop», 0 – не получено | Обработка сообщения и запись его в файл |

* 1. **Алгоритм работы UDP-клиента**

UDP-клиент создает сокет, обрабатывает информацию из файла – если там не менее 20 непустых строк, то для завершения работы будет достаточно, чтобы отправилось 20. Если там менее 20 строк, то будет проверяться, чтобы отправились все. Далее обрабатываются и отправляются все неотправленные сообщения и после каждой отправки происходит попытка получения сообщения от сервера с номерами принятых сообщений. Если пришло сообщение, то оно обрабатывается и увеличивается счетчик сообщений, дошедших до сервера, а также удаляются сообщения, которые дошли (чтобы не пытаться отправить их снова).

* 1. **Временные диаграммы взаимодействия сервера с клиентом**

На рисунках 3-4 представлены временные диаграммы взаимодействия клиента с сервером.

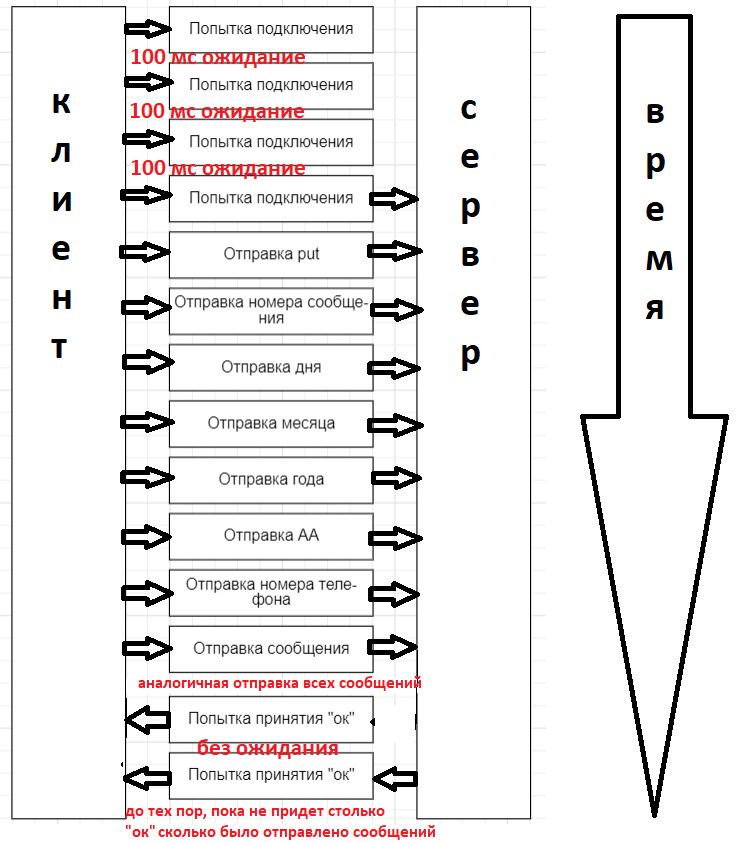


Рисунок 3 – Временная диаграмма взаимодействия TCP-клиента с -сервером

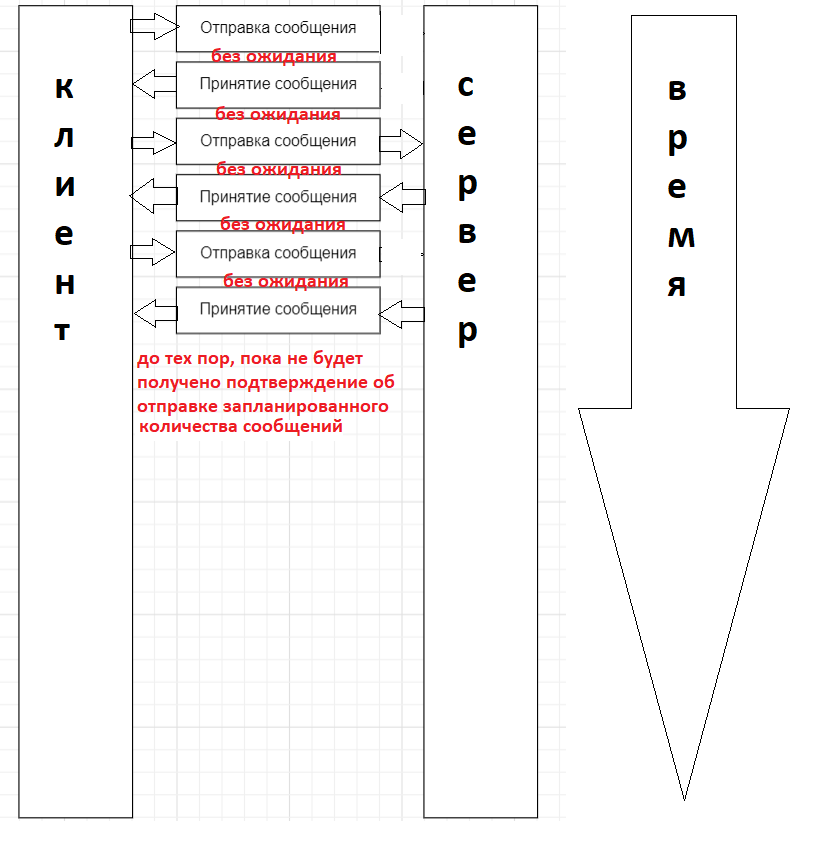


Рисунок 4 – Временная диаграмма работы UDP-сервера и -клиента

Как видно из диаграммы, TCP сервер не начнет отправлять и сообщения до тех пор, пока сервер не откликнется, после этого он передает много сообщений, каждое из которых доходит (стрелочки на рисунке), после чего ждет сообщений от сервера до тех пор, пока не придут все подтверждения. Если произойдет ошибка отправки одного из сообщений, то это сбой и клиент перестанет работать.

В UDP клиент отправляет все свои сообщения и ему не важно, дойдут они или нет. После чего принимает сообщения и ему также не важно, придет оно или нет (это видно по стрелочкам).

1. **Результаты**

В результате выполнения данной лабораторной работы не удалось добиться прохождения разработанными программами всех тестов в системе тестирования на сайте <http://ibks.ftk.spbstu.ru:3508/labs/18995/attempt?attempt%5Bkind%5D=0>. В результате прошло 14/50 тестов. В таблице 3 представлены тесты, которые не были пройдены системой с причинами.

Таблица 3 – Описание причин непрохождения тестов

|  |  |
| --- | --- |
| Тесты | Причина |
| Tcpserver | |
| Test1-test4, test12 | ожидаемый и полученный файлы одинаковые, но тест не пройден. |
| Test5,test8,test13 | в самом начале от клиента пришло лишнее сообщение с нулями – это могло произойти, если после отправки "put" клиент не начнет отправлять сообщения сразу, а будет делать какие-то доп действия – сервер работает в неблокирующей режиме, поэтому он пропустит все сообщения и выведет зануленный массив. Далее все сообщения пришли корректно. Для избежания этой проблемы можно при принятии сообщения put не принимать сообщение в этом проходе. |
| Test6,test7,test9 | длина сообщения 332 КБ, сервер настраивался по коду ruby – там размер был меньше. Для решения проблемы можно увеличить размер буфера. |
| Test10 | выходной файл клиента пустой, в других тестах в выходной файле писалось куда пытается подключиться клиент. Следовательно, клиент не попробовал подключиться. В run.log написана места --slowput, скорее всего эта метка задерживает отправку put серверу. Так как клиент должен отправить всего 3 сообщения, то работать должно быстро, но из-за slowput происходит задержка. Если на сайте стоит ограничение по времени, то отсюда и ошибка timeout. |
| Test11 | при запуске файла стоит места --slowsent. Строка входного файла отправляется серверу в несколько этапов, из-за метки между этапами есть задержки. При обнаружении события сервер принимает все части сообщения подряд. Так как клиент поставил между ними задержки, а сервер работает  в неблокирующем режиме, то все мини сообщения будут первым принятым, а остальные не примутся. Поэтому, при получении сообщения 'stop' сервер его примет за номер очередного сообщения (первое, что отправляет клиент), а не за последнее поле строки (само сообщение). |
| Test14 | клиенты запускаются с метками --nosend. Получается, что при подключении отправляется put, но не отправляется сообщение. Здесь многократно происходит ошибка как и в 5-ом тесте – приходит put, но не приходит сообщение – выводится зануленное сообщение в файл. |
| Test15 | запускается 2 клиента, 1-ый клиент с меткой --nowait, в сообщении второго содержится stop. То есть первый файл запускается раньше и отправляет сообщение - сервер точно примет хотя бы 1 сообщение от первого клиента. Но в выходном файле ожидаются сообщения только от второго клиента, что невозможно. |
| Test16 | клиенты запускаются с меткой --cndrop, что скорее всего подразумевает connection drop, то есть скидывание соединения через n единиц времени. Сервер принимает от клиента сообщения, если клиент сбросится, то сообщение просто дозаполнится неккоректно, из-за чего в файле и присутствуют некорректные сообщения. Для исправления можно ввести проверку на принятое сообщение и не выводить его в файл, если оно пришло не полностью. |
| Tcpclient | |
| Системой не был создан файл msg.txt для сравнения с ожидаемым результатом. | |
| Udpserver | |
| Test6-test8 | в файле run.log можно видеть, что не запускается клиент с файлом, который содержит stop – отсюда и timeout. |
| Udpclient | |
| Test8,test9 | ожидаемый файл содержит те же самые сообщения, но в другом порядке. Ошибки нет – сообщения отправляются по кругу до тех пор, сервер после каждого принятия обрабатывает сообщение, в то же время клиент пытается отправить другие. Поэтому порядок пришедших сообщений другой. |
| Test12 | пришло на 1 сообщение меньше – так как клиент отправляет все сообщения подряд, не проверяя дошли ли они, то он мог отправить датаграмму с этим сообщением, которая не дошла, а после отправить датаграмму со словом stop, которая дошла. |
| Test13 | запускаются 2 клиента – сообщение stop от второго клиента приходит раньше, чем первый отправит все свои сообщения, из-за чего и не совпадают файлы msg.txt и expected\_msg.txt. |
| Test14 | запуск с метками --skipeach=3 --skipfirst=29, из-за которых может каждый раз пропускаться датаграмма от второго клиента, который должен отправить stop. Когда пройдет разумное время, а датаграмма не будет отправлена, выйдет ошибка timeout. Для решения этой проблемы можно увеличить на сайте допустимое время работы, но это лишь сделает ее более редкой. |
| Test15 | сообщение stop содержится в файле cli11.txt. В файле run.log не происходит его отправки, из-за чего и происходит timeout. Ошибку можно исправить, изменив работу теста (добавить запуск клиента с отправкой содержимого файла cli11.txt) |

1. **Выводы**

В ходе выполнения данной лабораторной работы был изучен программный интерфейс сетевых сокетов, получены навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

Реализованы tcp-сервер и -клиент, udp-сервер и -клиент. При реализации серверов были изучены и применены механизмы многопользовательского обслуживания select и poll.

**Приложение А**

Листинг «tcpserver.cpp»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/poll.h>

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

const int SOCKET\_AMOUNT = 64;

int put\_flag = 0xFF;

int stop\_flag = 0xFF;

void byte\_to\_num(char\*, unsigned char); // Перевод байта в двухразрядное число десятичной системы счисления

int set\_unblocked\_mode(int); // Перевод в неблокирующий режим

int print\_error(const char\*, int); // Вывод сообщения о случившейся ошибке

void data\_processing(int, char\*); // Обработка полученной информации от клиента. Форматирование строки для вывода в файл

void reverse(char s[]);

void itoa(int n, char s[]);

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

{

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

if (ls < 0)

return print\_error("Socket creation", ls);

set\_unblocked\_mode(ls);

struct sockaddr\_in addr;

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(atoi(argv[1]));

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

int last\_s = 0;

int cs[SOCKET\_AMOUNT] = { 0 };

struct pollfd pfd[SOCKET\_AMOUNT + 1];

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

{

pfd[i].fd = cs[i];

pfd[i].events = POLLIN | POLLOUT;

}

pfd[SOCKET\_AMOUNT].fd = ls;

pfd[SOCKET\_AMOUNT].events = POLLIN;

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

return print\_error("Binding", ls);

if (listen(ls, 1) < 0)

return print\_error("Listening", ls);

int c=0;

//printf("Ready to connect\n");

while (stop\_flag)

{

int ev\_cnt = poll(pfd, sizeof(pfd) / sizeof(struct pollfd), 100);

if (ev\_cnt > 0)

{

//printf("ev\_cnt=%d\n",ev\_cnt);

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

{

//POLLHUP - клиент отключился, POLLERR - произошла ошибка\

в обоиз случаях можно закрывать сокет

if (pfd[i].revents & POLLHUP || pfd[i].revents & POLLERR)

{

//printf("Disconnected\n");

close(pfd[i].fd);

}

//cs[i] доступен на чтение, можно вызывать recv/recvfrom

if (pfd[i].revents & POLLIN)

{

//printf("Reading...\n");

char recvbuf[65536];// check\_buf[1000];

memset(recvbuf, '\0', 65536 \* sizeof(char));

//memset(check\_buf, '\0', 1000 \* sizeof(char));

data\_processing(cs[i], recvbuf);

//if (strncmp(recvbuf,check\_buf, 1000))

//{

//for (int kk=0;kk<100;kk++)

//printf("%c",recvbuf[kk]);

//printf("\n");

//if (recvbuf[4] == '\0')//

//{

FILE\* output = fopen("msg.txt", "a+");

if (output==NULL)

{

printf("Fopen error\n");

continue;

}

//else

//printf("File is created\n");

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

//IP-адрес, двоеточие, порт клиента, пробел

//printf("c=%d\n",++c);

//printf("%s\n",recvbuf+5);

char checker[]="00.00";

if (!strncmp(recvbuf+5,checker,5))

{

printf("continue...\n");

continue;

}

fprintf(output, "%d.%d.%d.%d:%d ", (ip >> 24) & 0xff, (ip >> 16) & 0xff, (ip >> 8) & 0xff, ip & 0xff, atoi(argv[1]));

unsigned message\_num = 0;

for (int j = 3; j >= 0; j--)

{

message\_num <<= 8;

message\_num += recvbuf[j];

}

//номер сообщения

//fprintf(output,"%u", message\_num);

//day and month

for (int j=5;j<11;j++)

fprintf(output,"%c", recvbuf[j]);

//год

unsigned short year = 0;

for (int j =12; j >= 11; j--)

{

year <<= 8;

year += (unsigned char)recvbuf[j];

}

year = ntohs(year);

fprintf(output, "%hu", year);

fprintf(output, "%c", ' ');

//AA

short AA = 0;

/\*for (int j = 15; j >= 14; j--)

{

AA <<= 8;

AA += recvbuf[j];

}\*/

AA += recvbuf[15];

AA<<=8;

AA+=(unsigned char)recvbuf[14];

AA = ntohs(AA);

fprintf(output, "%hi", AA);

fprintf(output, "%c", ' ');

//номер телефона

char mobile\_number[13];

strncpy(mobile\_number, recvbuf + 17, 13);

//"+7\*\*\*\*\*\*\*\*\*\* "

for (int j = 0; j < 13; j++)

fprintf(output, "%c", mobile\_number[j]);

//сообщение и перенос строки для следующего сообщения

fprintf(output, "%s\n", recvbuf+30);

fclose(output);

//}

//else printf("Empty message\n");

//}

}

}

//printf("ev\_cnt\_outside=%d\n",ev\_cnt);

// Сокет ls доступен на чтение - можно вызывать accept, принимать

// новое подключение

if (pfd[SOCKET\_AMOUNT].revents & POLLIN)

{

socklen\_t addrlen = sizeof(addr);

int new\_s = accept(ls, (struct sockaddr\*)&addr, &addrlen);

if (new\_s < 0)

{

print\_error("Acception", ls);

break;

}

set\_unblocked\_mode(new\_s);

cs[last\_s] = new\_s;

pfd[last\_s].fd = new\_s;

last\_s++;

//printf("Connect success\n");

}

}

}

//закрытие всех сокетов

for (int i = 0; i < last\_s; i++)

close(cs[i]);

close(ls);

return 0;

}

int print\_error(const char\* error, int s)

{

int err = errno;

fprintf(stderr, "Error type: %s. Socket #%d:\n", error, err);

return -1;

}

void data\_processing(int s, char\* buffer)

{

//принятие put

//sprintf("proseccing\n");

if (put\_flag)

{

char putBuffer[4];

memset(putBuffer, '\0', 4 \* sizeof(char));

recv(s, putBuffer, 3 \* sizeof(char), MSG\_NOSIGNAL);

if ((put\_flag = strncmp(putBuffer, "put", 3)))

{

printf("Not put\n");

return;

}

}

//номер сообщения - 4 байта

recv(s, buffer, 4 \* sizeof(char), MSG\_NOSIGNAL);//0-3

buffer[4] = ' ';//4

//значение дня

recv(s, buffer+5, sizeof(char), MSG\_NOSIGNAL);

byte\_to\_num(buffer + 5, buffer[5]);//5-6

buffer[7] = '.';//7

//значение месяца

recv(s, buffer + 8, sizeof(char), MSG\_NOSIGNAL);

byte\_to\_num(buffer + 8, buffer[8]);//8-9

buffer[10] = '.';//10

//год

recv(s, buffer + 11, 2\*sizeof(char), MSG\_NOSIGNAL);//11-12

buffer[13] = ' ';//13

//AA

recv(s, buffer + 14, 2 \* sizeof(char), MSG\_NOSIGNAL);//14-15

buffer[16] = ' ';//16

//номер телефона

recv(s, buffer + 17, 12\*sizeof(char), MSG\_NOSIGNAL);//17-28

buffer[29] = ' ';//29

//принятие сообщения

for (unsigned short i = 30;; i++)

{

recv(s, buffer + i, sizeof(char), MSG\_NOSIGNAL);

if (buffer[i] == '\0')

break;

}

stop\_flag = strcmp(buffer + 30, "stop");

send(s, "ok", 2 \* sizeof(char), MSG\_NOSIGNAL);

}

void byte\_to\_num(char\* res\_num, unsigned char num)

{

res\_num[0] = num / 10 + 48;

res\_num[1] = num % 10 + 48;

}

int set\_unblocked\_mode(int s)

{

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

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

}

void reverse(char s[])

{

int i, j;

char c;

for (i = 0, j = strlen(s)-1; i<j; i++, j--) {

c = s[i];

s[i] = s[j];

s[j] = c;

}

}

void itoa(int n, char s[])

{

int i, sign;

if ((sign = n) < 0) /\* записываем знак \*/

n = -n; /\* делаем n положительным числом \*/

i = 0;

do { /\* генерируем цифры в обратном порядке \*/

s[i++] = n % 10 + '0'; /\* берем следующую цифру \*/

} while ((n /= 10) > 0); /\* удаляем \*/

if (sign < 0)

s[i++] = '-';

s[i] = '\0';

reverse(s);

}

**Приложение Б**

Листинг «tcpclient.cpp»

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#pragma comment(lib, "ws2\_32.lib") // Директива линковщику: использовать библиотеку сокетов

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <conio.h>

#define MAX\_SIZE (200000)

typedef struct

{

unsigned int notempty;//номер непустой строки

//dd.mm.yyyy

char day;

char month;

unsigned short year;

//unsigned int date;

//AA - -32768 <= AA <= 32767

short AA;

//+7xxxxxxxxxx

char mobile[13];

//размер собщения

unsigned int volume\_of\_data;

//message

char\* message;

} profile;

profile\* packet\_profile; //массив содержимого строк файла

unsigned int count = 0; //кол-во строк файла

int parse\_file(char const\* f\_name);//обработка содержимого файлов

int init(); // init Winsock library

void deinit(); // deinit Winsock

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

unsigned int get\_host\_ipn(const char\* name); // Функция определяет IP-адрес узла по его имени. Адрес возвращается в сетевом порядке байтов.

int sending(int s);

//int main()

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

{

//127.0.0.1:9000 file1.txt

//int argc = 3;

//char argv[3][25] = { "","192.168.159.129:9000","sample.txt" };

if (argc != 3)

{

printf("Incorrect number of arguments\n");

return 0;

}

char\* ip = strtok(argv[1], ":");//считываем ip

int port = atoi(strtok(NULL, " "));//считываем порт и переводим из строкового вида в числовой

printf("%s %d\n", ip, port);//выводим ip и порт

//обработка файла

int res=parse\_file(argv[2]);

if (res == 0)

{

if (packet\_profile != NULL) free(packet\_profile);

printf("parse\_file error\n");

return 0;

}

//инициализация сетевой библиотеки

init();

//создание TCP-сокета

struct sockaddr\_in addr;

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

if (s < 0)

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

//Заполнение структуры с адресом удаленного узла

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

addr.sin\_addr.s\_addr = get\_host\_ipn(ip);

//10 попыток подключения (по условию задания)

for (int i = 0; i < 10; i++)

{

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

{

printf("Bad connect\n");

if (i == 9)//10 неудачных попыток, печатаем ошибку и завершаем работу

{

closesocket(s);

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

}

Sleep(100);//спим и снова пытаемся подключиться

//continue;

}

else//подключение произошло

{

printf("Success connect\n");

char put[] = "put";

res = send(s, put, 3, 0);//размер - 3, 0 - флаги

//printf("i=%d\n", i);

if (res < 0)

{

closesocket(s);

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

}

break;//все успешно, поэтому выходим из цикла отправки

}

}

int fail = sending(s);

if (fail == 1)

printf("success");

else

{

printf("error sending\n");

closesocket(s);

return 0;

}

char buffer[2];

int cnt=0;

//ждем подтверждения принятия всех сообщений от сервера

while (cnt != count)

{

res = recv(s, buffer, sizeof(buffer), 0);

if (res < 0)

{

closesocket(s);

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

}

cnt++;

}

//отправляем сообщение серверу для завершения работы

char end[] = "stop";

res = send(s, end, 4, 0);//размер - 1, 0 - флаги

if (res < 0)

{

closesocket(s);

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

}

//все сообщения приняты, можем завершать работу клиента

closesocket(s);

/\*for (int i = 0; i < count; i++)

{

char \*c=packet\_profile[i].message;

if (c) free(c);

}\*/

free(packet\_profile);

return 0;

}

int init()

{

WSADATA wsa\_data;

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

}

void deinit()

{

WSACleanup();

}

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

{

int err;

err = WSAGetLastError();

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

return -1;

}

int parse\_file(char const\* f\_name)

{

char\* string\_buf, \*mobile, \*message;

packet\_profile = (profile\*)malloc(sizeof(profile));

if (packet\_profile == NULL)

{

printf("malloc error, line 124\n");

return 0;

}

FILE\* pfile = fopen(f\_name, "r");

if (pfile == NULL)

{

printf("fopen error, line 130\n");

return 0;

}

string\_buf = (char\*)malloc(MAX\_SIZE \* sizeof(char));

if (string\_buf == NULL)

{

fclose(pfile);

printf("malloc error, line 136\n");

return 0;

}

//обработка всех строк, результат будет в массиве packet\_profile

while (fgets(string\_buf, MAX\_SIZE, pfile))//считываем строку и обрабатываем

{

if (string\_buf[0]!='\n')//если строка пустая, то сохранится единственный символ - символ переноса строки

{

packet\_profile = (profile\*)realloc(packet\_profile, (count + 1) \* sizeof(profile));//увеличиваем размер массива на 1

if (packet\_profile == NULL)

{

printf("realloc error: line 162\n");

fclose(pfile);

free(string\_buf);

return 0;

}

int size = strlen(string\_buf) - 1;

string\_buf[strlen(string\_buf)-1] = '\0';//избавляемся от символа переноса строки

packet\_profile[count].notempty = count;//номер

//yyyymmdd - в число int сохраняем в таком формате

packet\_profile[count].day = (char)(atoi(strtok(string\_buf, ".")));//считываем дату

packet\_profile[count].month = (char)(atoi(strtok(NULL, ".")));//считываем месяц

packet\_profile[count].year=(unsigned short)(atoi(strtok(NULL, " ")));//считываем год

//AA

packet\_profile[count].AA = (short)atoi(strtok(NULL, " "));//считываем число

//+7хххххххххх

mobile = strtok(NULL, " ");

strcpy(packet\_profile[count].mobile, mobile);

//копирование сообщения в структуру

message = strtok(NULL, "\0");

packet\_profile[count].volume\_of\_data = (unsigned int)(strlen(message));

packet\_profile[count].message = (char\*)malloc(packet\_profile[count].volume\_of\_data \* sizeof(char));

if (packet\_profile[count].message == NULL)

{

printf("malloc error: line 179\n");

fclose(pfile);

free(string\_buf);

return 0;

}

strcpy(packet\_profile[count].message, message);

count++;

}

}

//вывод всей считанной информации

/\*for (int i = 0; i < count; i++)

{

printf("%d ", packet\_profile[i].day);

printf("%d ", packet\_profile[i].month);

printf("%d ", packet\_profile[i].year);

printf("%hi ", packet\_profile[i].AA);

printf("%s ", packet\_profile[i].mobile);

printf("%s\n", packet\_profile[i].message);

}\*/

//char\* ptr = &packet\_profile[0].day;

//printf("%d\n", \*ptr);

fclose(pfile);

free(string\_buf);

return 1;

}

/\*union number

{

unsigned int n;

char string[4];

};\*/

int sending(int s)//s - сокет

{

//union number string\_number;

int res;

for (int i = 0; i < count; i++)

{

//printf("Sending #%d\n", i);

//4 байта в сетевом порядке байт - номер непустой строки файла

//string\_number.n = packet\_profile[count].notempty;

//char\* ptr = string\_number.string;

char\* ptr = (char\*)&packet\_profile[i].notempty;

res = send(s, ptr, 4, 0);//размер - 4, 0 - флаги

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//1 байт - номер дня (1-31)

ptr = &packet\_profile[i].day;

res = send(s, ptr, 1, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//1 байт - номер месяца (1-12)

ptr = &packet\_profile[i].month;

res = send(s, ptr, 1, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//2 байта - год (0-9999)

unsigned short year=htons(packet\_profile[i].year);

res = send(s, (char\*) & year, 2, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//2 байта - значение АА (-32768-32767)

unsigned short AA = htons(packet\_profile[i].AA);

ptr = (char\*)&AA;

res = send(s, ptr, 2, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//+7xxxxxxxxxx - номер телефона, длина 12, x - цифра 0-9

ptr = packet\_profile[i].mobile;

res = send(s, ptr, 12, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

//N+1 байт - N символов сообщения Message, а в конце \

передается дополнительно нулевой байт (как для конца строки) - маркер конца сообщения

ptr = packet\_profile[i].message;

//printf("message=%s\n", packet\_profile[i].message);

res = send(s, ptr, packet\_profile[i].volume\_of\_data+1, 0);

if (res < 0)

{

//printf("i=%d\n", i);

sock\_err("send", s);

return 0;

}

}

return 1;//успешно

}

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;

}

**Приложение В**

Листинг «udpserver.cpp»

#define \_CRT\_SECURE\_NO\_WARNINGS

//#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#pragma comment (lib,"Ws2\_32.lib")

#define SOCKET\_AMOUNT 64

typedef struct

{

unsigned int get\_msg; // количество принятых сервером сообщений

int msg\_num[256]; // Массив для хранения номеров принятых сообщений

int last\_n\_msg[20];

sockaddr\_in addr;

} profile;

profile\* packet\_profile = (profile\*)malloc(sizeof(profile));

unsigned int cnt = 0;

void byte\_to\_num(char\*, unsigned char); // Перевод байта в двухразрядное число десятичной системы счисления

int find\_profile(sockaddr\_in addr);

int format\_msg(unsigned int clienumb, char\* msg, int msglen, int port);

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

{

int l\_port = atoi(argv[1]);

int r\_port = atoi(argv[2]);

int ports\_amount = r\_port - l\_port + 1;

int\* port\_arr = (int\*)malloc(ports\_amount \* sizeof(int));

for (int i = 0; i < ports\_amount; i++)

port\_arr[i] = l\_port + i;

WSADATA wsa\_data;

WSAStartup(MAKEWORD(2, 2), &wsa\_data);

int\* listen\_sockets = (int\*)calloc(ports\_amount, sizeof(int));

struct sockaddr\_in\* listen\_addresses = (struct sockaddr\_in\*)calloc(ports\_amount, sizeof(struct sockaddr\_in));

printf("Listening ports: ");

for (int i = 0; i < ports\_amount; i++)

{

if ((listen\_sockets[i] = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{//ошибка создания сокета

for (int j = 0; j < i; j++)

closesocket(listen\_sockets[j]);

free(listen\_sockets);

free(listen\_addresses);

WSACleanup();

free(port\_arr);

return 1;

}

unsigned long mode = 0;

ioctlsocket(listen\_sockets[i], FIONBIO, &mode);

memset(&listen\_addresses[i], 0, sizeof(listen\_addresses[i]));

listen\_addresses[i].sin\_family = AF\_INET;

listen\_addresses[i].sin\_port = htons(l\_port + i);

listen\_addresses[i].sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(listen\_sockets[i], (struct sockaddr\*)&listen\_addresses[i], sizeof(listen\_addresses[i])))

{

for (int j = 0; j < i; j++)

closesocket(listen\_sockets[j]);

free(listen\_sockets);

free(listen\_addresses);

WSACleanup();

free(port\_arr);

return 0;

}

printf("%d ", l\_port + i);

}

printf("\n");

fd\_set rfd;

fd\_set wfd;

int nfds = listen\_sockets[0];

struct timeval tv = { 1, 0 };//1 секунда

char buf[1024];

int stop = 0, answer=0;

while (1)

{

FD\_ZERO(&rfd);

FD\_ZERO(&wfd);

for (int i = 0; i < ports\_amount; i++)

{

FD\_SET(listen\_sockets[i], &rfd);

FD\_SET(listen\_sockets[i], &wfd);

if (nfds < listen\_sockets[i])

nfds = listen\_sockets[i];

}

if (select(nfds + 1, &rfd, &wfd, 0, &tv) > 0)

{ // Есть события

for (int i = 0; i < ports\_amount; i++)

{

if (FD\_ISSET(listen\_sockets[i], &rfd))

{

// Сокет cs[i] доступен для чтения. Функция recv вернет данные, recvfrom - дейтаграмму

memset(buf, 0, 1024 \* sizeof(char));

sockaddr\_in from;

int addrlen = sizeof(from);

int recv = recvfrom(listen\_sockets[i], buf, sizeof(buf), 0, (struct sockaddr\*)&from, &addrlen);

if (recv > 0)

{

int clinumb = find\_profile(from);

if (clinumb < 0)

{//новый клиент

packet\_profile = (profile\*)realloc(packet\_profile, sizeof(profile) \* (cnt + 1));

for (int k = 0; k < 20; k++)

{

packet\_profile[cnt].msg\_num[k]=-1;

packet\_profile[cnt].last\_n\_msg[k] = -1;

}

for (int k = 20; k < 101; k++)

packet\_profile[cnt].msg\_num[k] = -1;

packet\_profile[cnt].get\_msg = 0;

packet\_profile[cnt].addr = from;

clinumb = cnt++;

//unsigned int port = ntohs(from.sin\_port);

printf("New client: %d.%d.%d.%d:%d\n", from.sin\_addr.s\_addr & 0xFF, from.sin\_addr.s\_addr >> 8 & 0xFF, from.sin\_addr.s\_addr >> 16 & 0xFF, from.sin\_addr.s\_addr >> 24 & 0xFF, port\_arr[i]);

}

stop=format\_msg(clinumb, buf, 26+strlen(buf+26), port\_arr[i]);

//printf("stop=%d\n", stop);

char\* sendbuf = (char\*)(packet\_profile[clinumb].last\_n\_msg);

//for (int kk = 0; kk < 20; kk++)

// printf("%d ", packet\_profile[clinumb].last\_n\_msg[kk]);

//printf("\n");

sendto(listen\_sockets[i], sendbuf, packet\_profile[clinumb].get\_msg \* sizeof(int), 0, (sockaddr\*)(&(packet\_profile[clinumb].addr)), sizeof(sockaddr));

}

if (stop)

break;

}

}

}

if (stop)

break;

}

for (int i = 0; i < ports\_amount; i++)

closesocket(listen\_sockets[i]);

free(listen\_sockets);

free(listen\_addresses);

free(port\_arr);

WSACleanup();

return 0;

}

void byte\_to\_num(char\* buf, unsigned char c)

{

buf[0] = c / 10 + 48;

buf[1] = c % 10 + 48;

}

int find\_profile(sockaddr\_in addr)

{

for (int i = 0; i < cnt; i++)

if (!memcmp(&packet\_profile[i].addr, &addr, sizeof(sockaddr)))

return i;

return -1;

}

int format\_msg(unsigned int profile\_num, char\* msg, int msglen, int port)

{

FILE\* output = fopen("msg.txt", "a+");

if (output == NULL)

{

printf("Fopen error\n");

return 1;

}

unsigned message\_num = 0;

for (int j = 3; j >= 0; j--)

{

message\_num <<= 8;

message\_num += msg[j];

}

message\_num = ntohl(message\_num);

int new\_clie\_num = 0;

for (;; new\_clie\_num++)

{

//уже принято

if (message\_num == packet\_profile[profile\_num].msg\_num[new\_clie\_num])

{

fclose(output);

return 0;

}

//идем дальше по массиву

if (packet\_profile[profile\_num].msg\_num[new\_clie\_num] != -1)

continue;

//найдено место

packet\_profile[profile\_num].msg\_num[new\_clie\_num] = message\_num;

break;

}

if (packet\_profile[profile\_num].get\_msg < 20)//если получено меньше 20 сообщений, то записываем номер принятого нового

packet\_profile[profile\_num].last\_n\_msg[packet\_profile[profile\_num].get\_msg++] = htonl(message\_num);

else

{//иначе сдвигаем все сообщения на 1 влево и добавляем новое принятое

for (unsigned char i = 19; i > 0; i--)

packet\_profile[profile\_num].last\_n\_msg[i - 1] = packet\_profile[profile\_num].last\_n\_msg[i];

packet\_profile[profile\_num].last\_n\_msg[19] = htonl(message\_num);

}

//IP-адрес, двоеточие, порт клиента, пробел

unsigned int netip = htonl(packet\_profile[profile\_num].addr.sin\_addr.s\_addr);

fprintf(output, "%u.%u.%u.%u:%d ", (netip >> 24) & 0xff, (netip >> 16) & 0xff, (netip >> 8) & 0xff, netip & 0xff, port);

//номер сообщения

//fprintf(output, "%u ", message\_num);

//day and month

char day = msg[4];

char month = msg[5];

//год

unsigned short year = 0;

for (int j = 7; j >= 6; j--)

{

year <<= 8;

year += (unsigned char)msg[j];

}

year = ntohs(year);

char day\_arr[3] = { '0' + day / 10,'0' + day % 10,0 };

char month\_arr[3] = { '0' + month / 10,'0' + month % 10,0 };

fprintf(output, "%s.%s.%hu ", day\_arr, month\_arr, year);

//fprintf(output, "%c", ' ');

//AA

short AA = 0;

AA += msg[9];

AA <<= 8;

AA += (unsigned char)msg[8];

//printf("%hi\n", AA);

AA = ntohs(AA);

fprintf(output, "%hi ", AA);

//номер телефона

char mobile\_number[12];

strncpy(mobile\_number, msg + 10, 12);

//"+7\*\*\*\*\*\*\*\*\*\* "

for (int j = 0; j < 12; j++)

{

fprintf(output, "%c", mobile\_number[j]);

//printf("%c", mobile\_number[j]);

}

//printf("\n");

//сообщение и перенос строки для следующего сообщения

fprintf(output, " %s\n", msg + 22);

//printf("%s\n", msg + 22);

fclose(output);

if (strcmp("stop", msg + 22) == 0)

return 1;

return 0;

}

int find\_cli(struct cli\* clie, sockaddr\_in addr, unsigned int size)

{

for (int i = 0; i < size; i++)

if (!memcmp(&packet\_profile[i].addr, &addr, sizeof(sockaddr)))

return i;

return -1;

}

**Приложение Г**

Листинг «udpclient.cpp»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/time.h>

#include <sys/select.h>

#include <netdb.h>

#include <errno.h>

#include <unistd.h>

#include <fcntl.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int print\_error(const char\*, const int); // Функция для вывода последнего сообщения об ошибке

void get\_strings(FILE\*, char\*\*); // Функция для считывания строк из файла в память

int sending(int, char\*, unsigned int, struct sockaddr\_in); // Функция для форматирования сообщения и его отправки на сервер

int recv\_response(int s, char\*\*, unsigned int, int\*); // Функция для получения ответа от сервера и отметки отправленных сообщений

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

{

if (argc != 3)

{

printf("Incorrect number of arguments\n");

return 0;

}

char\* ip = strtok(argv[1], ":");//считываем ip

int port = atoi(strtok(NULL, " "));//считываем порт и переводим из строкового вида в числовой

printf("%s %d\n", ip, port);//выводим ip и порт

int s;

int i;

struct sockaddr\_in addr;

// Создание UDP-сокета

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

if (s < 0)

return print\_error("socket", s);

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

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

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

if (input == NULL)

return print\_error("File open error", s);

char\* mes\_str[101];

memset(mes\_str, 0, 101 \* sizeof(char\*));

get\_strings(input, mes\_str);

fclose(input);

input = NULL;

int max\_lines = 0;

//нужно отправить хотя бы 20 сообщений, если их меньше, то нужно отправить все\

поэтому узнаем сколько их

for (int i = 0; i < 21 && mes\_str[i] != NULL; i++)

max\_lines++;

//хотя бы 20->20 достаточно, будем отправлять 20

if (max\_lines > 20)

max\_lines--;

//printf("max=%d\n",max\_lines);

for (int l\_cnt = 0;;)

{

for (unsigned int i = 0;; i++)

{

//printf("l\_cnt=%d\n",l\_cnt);

if (i==101) i=0;

if (mes\_str[i])//если сообщение есть, то отправляем

{

int result = 1;

result = sending(s, mes\_str[i], i, addr);

if (result)

{

close(s);

return result;

}

recv\_response(s, mes\_str, i, &l\_cnt);

}

if (l\_cnt >= max\_lines)//если отправили нужное количество сообщений, то завершаем работу

{

close(s);

return 0;

}

}

}

}

int print\_error(const char\* name, const int s)

{

int error\_code = errno;

fprintf(stderr, "%s: socket #%d error: %d\n", name, s, error\_code);

return -1;

}

void get\_strings(FILE\* file, char\*\* str)

{

fseek(file, 0, SEEK\_SET);

for (unsigned char i = 0; i < 101;)

{

char buf[65536];

fgets(buf, 65536, file);

if (\*buf == '\n')//строка пустая

continue;

if (buf[strlen(buf) - 1] == '\n')

buf[strlen(buf) - 1] = '\0';//fgets сохраняет в массив и символ переноса строки, так что удаляем его

str[i] = (char\*)calloc(strlen(buf) + 1, sizeof(char));

strncpy(str[i++], buf, strlen(buf)+1);

printf("%s\n",str[i-1]);

int check = fgetc(file);

if (check == EOF)//проверка на окончание файла

break;

fseek(file, -1, SEEK\_CUR);//если все в порядке, то возвращаем указатель и продолжаем считывать строки

}

}

union num\_bytes

{

//int signed\_dig;

unsigned int dig;

char bytes[4];

};

union year\_AA\_bytes

{

unsigned short year;

short AA;

char bytes[2];

};

int sending(int s, char\* str, unsigned int number, struct sockaddr\_in addr)

{

//printf("%s\n",str);

char buf[65536];

memset(buf, '\0', 65536 \* sizeof(char));

union num\_bytes message\_num;

message\_num.dig = htonl(number);

strncpy(buf, message\_num.bytes, 4);//номер сообщения

//день

char day[3];

strncpy(day, str, 2);

day[2] = 0;

char day\_n = (char)atoi(day);

//printf("day=%d\n",day\_n);

buf[4] = day\_n;

//месяц

char month[3];

strncpy(month, str+3, 2);

month[2] = 0;

char month\_n = (char)atoi(month);

buf[5] = month\_n;

//год

union year\_AA\_bytes year;

char year\_array[5];

strncpy(year\_array, str+6, 4);

year\_array[4] = 0;

unsigned short year\_array\_n = (unsigned short)atoi(year\_array);

year.year = htons(year\_array\_n);

strncpy(buf+6, year.bytes, 2);//год

//AA

union year\_AA\_bytes AA;

char AA\_array[7]={'\0','\0','\0','\0','\0','\0','\0'};

//strncpy(AA\_array, str + 11, 4);

for (int k=0;k<6;k++)

if (str[11+k]!=' ')

AA\_array[k]=str[11+k];

else break;

//AA\_array[4] = 0;

//printf("AA\_array=%s\n",AA\_array);

short AA\_array\_n = (short)atoi(AA\_array);

AA.AA = htons(AA\_array\_n);

//printf("%d\n",AA.AA);

//printf("%d\n",AA.bytes[0]);

//printf("%d\n",AA.bytes[1]);

strncpy(buf + 8, AA.bytes, 2);//AA

int AA\_len=strlen(AA\_array);

//номер телефона

strncpy(buf + 10, str+11+AA\_len+1, 12);

//сообщение

strncpy(buf + 22, str+25+AA\_len, strlen(str));

socklen\_t addrlen = sizeof(addr);

//printf("for send: %s\n",buf+14);

int res = sendto(s, buf, (22+strlen(buf+22))\*sizeof(char), MSG\_NOSIGNAL, (struct sockaddr\*)&addr, sizeof(struct sockaddr\_in));

if (res < 0)

return print\_error("Sendto error", s);

printf("Send success\n");

return 0;

}

int recv\_response(int s, char\*\* str, unsigned int n, int\* l)

{

char datagram[80];

memset(datagram, 0, 80 \* sizeof(char));

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

fd\_set fds;

FD\_ZERO(&fds);

FD\_SET(s, &fds);

// Проверка - если в сокете входящие дейтаграммы\

(ожидание в течение tv)

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

if (res > 0)

{

struct sockaddr\_in addr;

int addrlen = sizeof(addr);

int received = recvfrom(s, datagram, 80\*sizeof(char), 0, (struct sockaddr\*)&addr, (socklen\_t\*)&addrlen);

if (received <= 0)

return print\_error("recvfrom", s);

//printf("recvfrom success\n");

//printf("%d%d%d%d\n",datagram[0],datagram[1],datagram[2],datagram[3]);

int dig\_buf[20];

memcpy(dig\_buf, datagram, 80);

//for (int j=0;j<20;j++)

// printf("%d ",dig\_buf[j]);

//printf("\n");

//смотрим по полученным номерам, какие сообщения больше не надо отправлять\

и увеличиваем количество отправленных сообщений

for (int i = 0; i < 20; i++)

{

if (ntohl(dig\_buf[i])==-1) break;

if (str[ntohl(dig\_buf[i])] != NULL)

{

free(str[ntohl(dig\_buf[i])]);

str[ntohl(dig\_buf[i])] = NULL;

(\*l)++;

}

}

return 0;

}

else

return 1;

}