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

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**«МЕХАНИЗМЫ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ»**

по дисциплине «Операционные системы»

1. Выполнил
2. студент гр. 4851003/10002 Галкин К. К.

1. Руководитель
2. К. н. т Крундышев В. М.

Санкт-Петербург

2023

1. **ЦЕЛЬ РАБОТЫ**

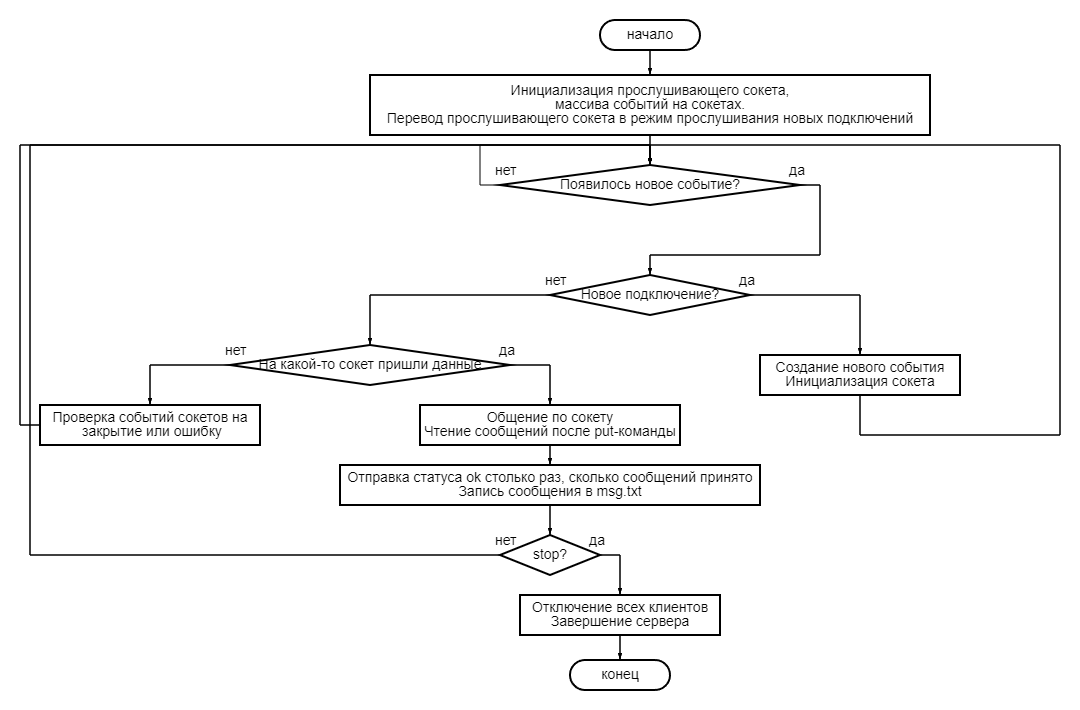
Цель работы — изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сервисов.

1. **ХОД РАБОТЫ**

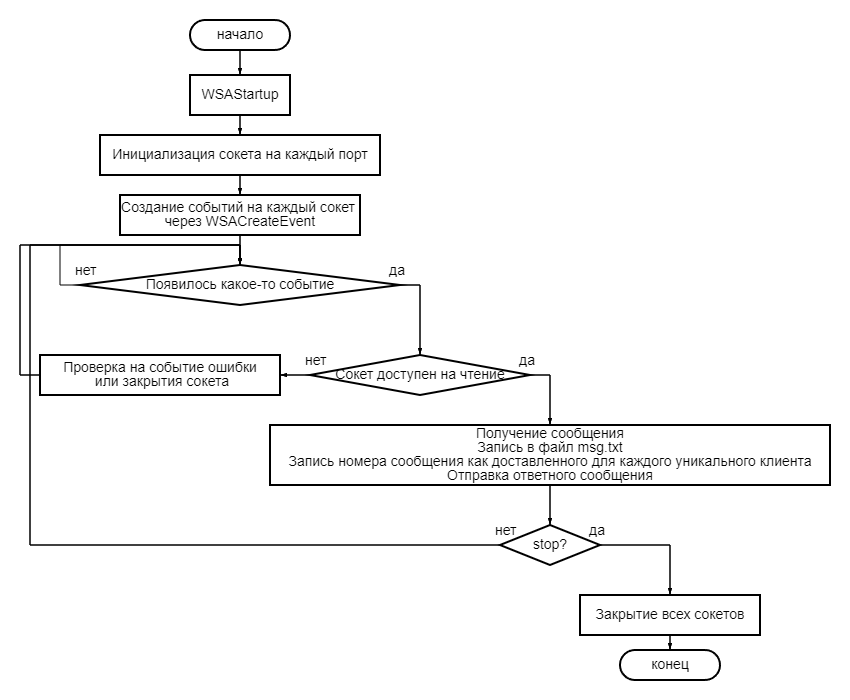
Перед выполнением работы были поставлены следующие задачи:

* Написать TCP-клиент под ОС Windows и TCP-сервер под ОС Linux с механизмом параллельного обслуживания poll.
* Написать UDP-клиент под ОС Linux и UDP-сервер под ОС Windows с использование механизма WSAEvent().
* Продумать тесты для программ, рассматривающие “нормальные” и граничные случаи.
* Протестировать разработанное решение.

Блок – схемы серверов представлены ниже



**Рисунок 1 Блок-схема работы TCP-сервера**



**Рисунок 2 Блок-схема работы UDP-сервера**

Таблица для описания структур и используемых функций представлена ниже:

|  |  |
| --- | --- |
| Функция/Структура | Назначение |
| pollfd pfd[MAX\_SOCKETS + 1] | Массив структур с описанием состояний сокетов. Нужна для обнаружения новых событий |
| union convert, short\_ints | Объединения для конвертации чисел с обратным порядком байтов. Используются два поля: одно – число с необходимым размером в байтах, второе – буфер определенного размера байтов. |
| string findClient(struct sockaddr\_in \*addr) | Функция для возврата строки вида IP\_SOURCE:PORT из структуры addr |
| int writeToFile(string buffer, string client) | Функция запись полученного сообщения в файл. Возвращает -1, если обнаружено сообщение stop |
| void disconnectClient(int index, sockaddr\_in \*addr) | Функция отключения клиента, освобождение сокета |
| static int readPacket(int sock, sockaddr\_in \*addr, int index) | Функция последовательного чтения данных из пакета. Возвращает значение функии writeToFile() |

Таблица с описанием структур и функций UDP-сервера:

|  |  |
| --- | --- |
| Функция/Структура | Назначение |
| map<string, set<uint32\_t, greater<uint32\_t>>> clients; | Словарь всех клиентов и их номеров сообщений. Номера сообщений отсортированы. |
| SOCKET \*sockets\_desc | Массив всех сокетов |
| struct sockaddr\_in \*addr | Массив структур адресов для каждого сокета |
| union convert, short\_ints | Объединения для конвертации чисел с обратным порядком байтов. Используются два поля: одно – число с необходимым размером в байтах, второе – буфер определенного размера байтов. |
| string findClient(struct sockaddr\_in \*addr) | Функция для возврата строки вида IP\_SOURCE:PORT из структуры addr |
| int writeToFile(string buffer, string client) | Функция запись полученного сообщения в файл. Возвращает -1, если обнаружено сообщение stop |
| int analyseDatagram(SOCKET sock, int index) | Функция анализа полученного сообщения. Проверка на то, если клиент уникальный, добавление к множеству всех отправленных сообщений клиента нового значения из полученной датаграмы. |
| void configListeners(uint16\_t port\_start, uint16\_t port\_end) | Функция настройки сокетов для каждого порта. Идет проверка на то, что первый аргумент(порт) должен быть меньше второго, так же если port\_start и port\_end совпадают, то конфигурация идет только одного сокета. |
| void sendToClient(SOCKET s, int index) | Отправка клиенту ответного сообщения, в котором находятся номера отправленных клиентом сообщений. |
| void disconnectClient(int index, sockaddr\_in \*addr) | Функция отключения клиента, освобождение сокета |

Клиенты работают примерно по одному и тому же принципу, независимо от используемой ОС:

* Инициализация соединения. Попытка подключения через connect в течение 10 раз с разницей в 100мс.
* Чтения сообщений из файла в вектор строк.
* Для TCP – отправка команды put
  + После отправляется каждое сообщение из вектора msgs в отформатированном виде
  + Идет получение статуса ok на каждое сообщение
  + Если все сообщения доставлены – сокет закрывается
* Для UDP-клиента
  + Инициализация сокета
  + Чтение данных из файла в словарь строк allMsgs, где каждому сообщению присвоен индекс
  + Создается очередь из неотправленных сообщений.
  + Попытка отправить пачку отформатированных данных
  + Проверка последнего пакета на номера полученных сообщений
  + Полученные номера попадают во множество msgArchive
  + Очередь для отправки обновляется до тех пор, пока не останется сообщений.

Для тестирования такого рода приложений стоит учитывать:

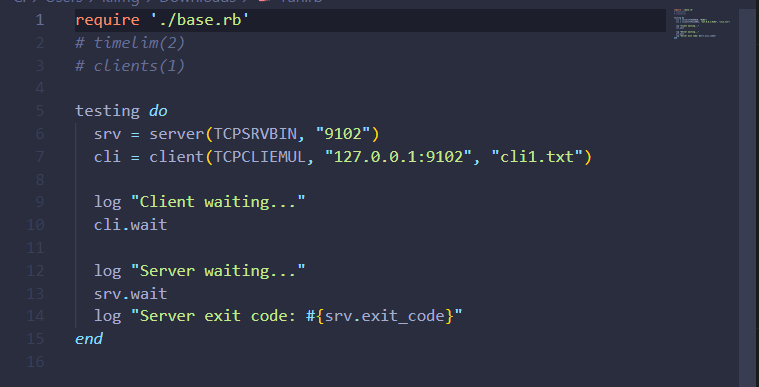
* Максимальный размер тела пакета для UDP(учитывается так же для TCP) – 65507 байтов. То есть нужно проверить отправку строки большого размера
* В данных из файла может быть только одна строка с сообщением stop.
* Проверять последовательность получения сообщений в UDP нет смысла в силу гонки обработки данных на сокетах.
* Стоит проверить набор данных с пустыми строками в файле
* Стандартная проверка с обычным сообщением.
* Для UDP клиента нужно проверить граничные случаи инициализации сокетов на диапазон портов: порт указан на самом деле один, портов слишком много, сначала указан конец диапазона, потом начало.
* Нужно сделать тесты на проверку аварийного завершения программы, если не получилось настроить конфигурацию сокета или поставить сокет на прослушивание.

Вот простой пример входных сообщений для программ-клиентов:



**Рисунок 3 - Набор сообщений для тестирования**

Проверка работоспособности сетевых функций делается путем написания скриптов, которые либо запускают клиентов параллельно, либо последовательно на один сокет.



**Рисунок 4 - Пример теста для проверки сетевых функций**

Тексты всех программ представлены в приложений.

1. **ВЫВОД**

В ходе выполнения лабораторной работы были изучены механизмы сетевого взаимодействия программ, параллельного обслуживания клиентов, спецификации современных протоколов.

Были написаны программы-клиенты и программы-серверы для протоколов TCP и UDP под разные ОС с разными механизмами обслуживания. В ходе написания кода возникали проблемы аварийных завершений программы до закрытия сокета, непонимания технического задания, некоторые функции работали некорректно (как, например, memcpy).

Для разработанных решений были описаны тестовые ситуации, которые позволяют покрыть большую часть функционала программ.

1. **ПРИЛОЖЕНИЕ**
   1. TCP-клиент

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#include <iostream>

#include <fstream>

#include <sstream>

#include <stdio.h>

#include <windows.h>

#include <stdlib.h>

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

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

#include <vector>

#include <stdint.h>

#define TCP\_MAX\_SIZE 65507

using namespace std;

vector<string> msgs;

int init()

{

WSADATA wsaData;

return (0 == WSAStartup(MAKEWORD(2, 2), &wsaData));

}

void readFile(char \*fileName)

{

ifstream fdesc(fileName);

string msg;

while (getline(fdesc, msg))

{

if (!msg.empty())

{

msgs.push\_back(msg);

}

msg.clear();

}

fdesc.close();

}

void append\_chunk(string &s, const uint8\_t \*chunk, size\_t chunk\_num\_bytes)

{

s.append((char \*)chunk, chunk\_num\_bytes);

}

void transferData(SOCKET s)

{

char ok[3] = {0};

int okCount = 0;

int msgCount = msgs.size();

while (okCount != msgCount)

{

string currentMsg = msgs[okCount], send\_msg, token;

vector<string> Tokens;

istringstream \_stream(currentMsg);

while (std::getline(\_stream, token, ' '))

{

if (!token.empty())

{

Tokens.push\_back(token);

}

token.clear();

}

vector<string> dateParts;

stringstream date(Tokens[0]);

string part;

while (std::getline(date, part, '.'))

{

dateParts.push\_back(part);

part.clear();

}

uint32\_t index\_in\_message = htonl(okCount);

uint8\_t day = (uint8\_t)atoi(dateParts[0].c\_str());

uint8\_t month = (uint8\_t)atoi(dateParts[1].c\_str());

uint16\_t year = htons(atoi(string(dateParts[2].begin(), dateParts[2].end()).c\_str()));

int16\_t AA = htons(atoi(Tokens[1].c\_str()));

/\* Phone Number Process \*/

char \*phone = (char \*)Tokens[2].c\_str();

/\* Packing message process \*/

send\_msg.reserve(currentMsg.size() + 4);

append\_chunk(send\_msg, (const uint8\_t \*)&index\_in\_message, sizeof(uint32\_t));

append\_chunk(send\_msg, (const uint8\_t \*)&day, 1);

append\_chunk(send\_msg, (const uint8\_t \*)&month, 1);

append\_chunk(send\_msg, (const uint8\_t \*)&year, sizeof(uint16\_t));

append\_chunk(send\_msg, (const uint8\_t \*)&AA, sizeof(signed short));

append\_chunk(send\_msg, (const uint8\_t \*)phone, strlen(phone));

/\* Pacling all data after phone \*/

size\_t data\_pos = currentMsg.find("+") + 13;

string data = currentMsg.substr(data\_pos);

append\_chunk(send\_msg, (const uint8\_t \*)data.c\_str(), data.length());

send\_msg[send\_msg.size()] = '\0';

int status = send(s, send\_msg.c\_str(), send\_msg.size() + 1, 0);

int r;

while ((r = recv(s, ok, 2, 0)))

{

if (strcmp(ok, "ok") == 0)

{

okCount++;

break;

}

}

}

}

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

{

bool isConnected = false;

if (argc != 3)

{

cout << "Usage ./tcpclient IP:PORT TEXTFILE" << endl;

return 0;

}

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

char \*port = strtok(NULL, "\0");

struct sockaddr\_in addr;

init();

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

if (s < 0)

{

cout << "socket() function error" << endl;

return 0;

}

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(atoi(port));

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

for (int connAttempt = 0; connAttempt < 10 && !isConnected; connAttempt++)

{

if (connect(s, (struct sockaddr \*)&addr, sizeof(addr)) != -1)

{

int startMessaging = send(s, "put", 3, 0);

if (startMessaging < 0)

{

cout << "Sending problems..." << endl;

break;

}

isConnected = true;

cout << "Connection complete!" << endl;

}

else

{

cout << "Attempt to connect..." << endl;

Sleep((DWORD)100);

}

}

if (!isConnected)

{

cout << "Connection refused" << endl;

closesocket(s);

return 0;

}

readFile(argv[2]);

transferData(s);

closesocket(s);

WSACleanup();

return 0;

}

* 1. TCP-сервер.

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/ioctl.h>

#include <sys/poll.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <stdint.h>

#include <stdlib.h>

#include <unistd.h>

#define FILE\_LOGGER "msg.txt"

#define MAX\_SOCKETS 255

#define TCP\_MAX\_SIZE 65507

using namespace std;

int s\_c = 0;

int cs[MAX\_SOCKETS];

pollfd pfd[MAX\_SOCKETS + 1];

union

{

uint32\_t integer;

char buffer[4];

} convert;

union

{

char buffer[2];

uint16\_t integer;

int16\_t ninteger;

} short\_ints;

string findClient(struct sockaddr\_in \*addr)

{

char ip[16];

char port[5];

string client;

inet\_ntop(AF\_INET, &addr->sin\_addr, ip, sizeof(ip));

client.clear();

snprintf(port, 5, "%d", htons(addr->sin\_port));

client.append((char \*)ip, strlen(ip));

client += ':';

client.append((char \*)port, strlen(port));

return client;

}

int writeToFile(string buffer, string client)

{

ofstream file\_desc;

file\_desc.open(FILE\_LOGGER, ios::app);

uint16\_t day, month, year = 0;

day = buffer[0];

month = buffer[1];

short\_ints.buffer[0] = buffer[3];

short\_ints.buffer[1] = buffer[2];

year = short\_ints.integer;

file\_desc << client << " ";

file\_desc << setw(2) << setfill('0') << day << '.' << setw(2) << setfill('0') << month << '.' << year << " ";

short\_ints.buffer[0] = buffer[5];

short\_ints.buffer[1] = buffer[4];

int16\_t AA = short\_ints.ninteger;

file\_desc << AA << " ";

file\_desc << string(buffer.begin() + 6, buffer.begin() + 18) << " " << string(buffer.begin() + 18, buffer.end() - 1) << endl;

file\_desc.close();

if (string(buffer.begin() + 18, buffer.end() - 1) == "stop")

{

return -1;

}

return 0;

}

void disconnectClient(int index, sockaddr\_in \*addr)

{

string client = findClient(addr);

cout << " Peer disconnected: " << client << endl;

cs[index] = -1;

close(pfd[index].fd);

pfd[index].fd = -1;

}

static int readPacket(int sock, sockaddr\_in \*addr, int index)

{

char \*buffer = new char[TCP\_MAX\_SIZE];

int r = 0, write = 0;

string b;

while ((r = recv(sock, buffer, TCP\_MAX\_SIZE, 0)) > 0)

{

string client = findClient(addr);

b.append((char \*)buffer, r);

write = writeToFile(b, client);

int s\_status = send(sock, "ok", 2, 0);

b.clear();

}

//disconnectClient(index, addr);

delete[] buffer;

return write;

}

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

{

if (argc != 2)

{

cout << "Usage: ./tcpserver <port>" << endl;

return 0;

}

int i;

uint32\_t port = atoi(argv[1]);

sockaddr\_in addr;

unsigned long mode = 1, opt = 1;

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

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

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

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(port);

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

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

if (s < 0)

{

cout << "socket() error function" << endl;

return 0;

}

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

{

cout << "bind() function error" << endl;

return 0;

}

if (listen(s, 1) < 0)

{

cout << "listen() function error" << endl;

return -1;

}

for (size\_t i = 0; i < MAX\_SOCKETS; i++)

{

cs[i] = -1;

}

pfd[0].fd = s;

pfd[0].events = POLLIN;

cout << "Listening in " << port << endl;

while (true)

{

int res = poll(pfd, sizeof(pfd) / sizeof(pfd[0]), 100);

if (res > 0)

{

for (size\_t i = 1; i <= s\_c; i++)

{

if (pfd[i].revents & POLLIN)

{

char \*buffer = (char \*)malloc(4);

int r = recv(pfd[i].fd, buffer, 4, 0);

if (strcmp(buffer, "put") == 0)

break;

else

{

int r\_status = readPacket(pfd[i].fd, &addr, i);

if (r\_status == -1)

{

cout << "stop message detected. Powering off..." << endl;

for (size\_t i = 1; i <= s\_c; i++)

{

close(pfd[i].fd);

}

close(s);

exit(0);

}

}

}

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

{

disconnectClient(i, &addr);

}

}

if (pfd[0].revents & POLLIN)

{

unsigned long mode = 1;

unsigned int socklen = sizeof(addr);

int new\_connection = accept(pfd[0].fd, (struct sockaddr \*)&addr, &socklen);

string client = findClient(&addr);

cout << " Peer connected: " << client << endl;

if (new\_connection < 0)

{

break;

}

s\_c++;

for (size\_t i = 1; i <= s\_c; i++)

{

if (cs[i] == -1)

{

if (i == s\_c) s\_c--;

cs[i] = new\_connection;

pfd[i].fd = new\_connection;

pfd[i].events = POLLIN | POLLOUT;

break;

}

}

}

}

}

for (size\_t i = 0; i < s\_c; i++)

{

close(pfd[i].fd);

}

close(s);

return 0;

}

* 1. UDP-клиент

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <map>

#include <vector>

#include <set>

#include <iomanip>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/select.h>

#include <arpa/inet.h>

#include <stdint.h>

#define MAX\_BODY\_SIZE 65507

#define MAX\_CONNECTIONS 65535

using namespace std;

map<uint32\_t, string> allMsgs;

set<uint32\_t> msgArchive;

set<uint32\_t> recvData(int s, int sendCount);

string datagramCreate(uint32\_t index);

static int sendData(int s, sockaddr\_in \*addr, set<uint32\_t> msgNums);

void readFile(char \*file\_name);

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

{

if (argc != 3)

{

cout << "Usage: ./udpclient IP:PORT FILENAME" << endl;

return 0;

}

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

char \*port = strtok(NULL, "\0");

cout << ip << " " << port << endl;

sockaddr\_in addr;

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

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

addr.sin\_family = AF\_INET;

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

addr.sin\_port = htons(atoi(port));

readFile(argv[2]);

set <uint32\_t> startMsgs;

set <uint32\_t> memSave;

for (auto it = allMsgs.begin(); it != allMsgs.end(); it++)

{

startMsgs.insert(it->first);

}

int sendCount = 0;

sendCount = sendData(s, &addr, startMsgs);

memSave = startMsgs;

while ((allMsgs.size() <= 20 && msgArchive.size() != allMsgs.size()) || (allMsgs.size() > 20 && msgArchive.size() != 20))

{

set<uint32\_t> newData = recvData(s, sendCount);

if (newData.size() == 0)

{

newData = memSave;

}

sendCount = sendData(s, &addr, newData);

memSave = newData;

}

}

void readFile(char \*file\_name)

{

ifstream fdesc(file\_name, ios::in);

string fline;

uint32\_t index = 0;

while(getline(fdesc, fline))

{

if (!fline.empty())

{

allMsgs.insert(make\_pair(index, fline));

index++;

}

fline.clear();

}

}

void append\_chunk(string &s, const uint8\_t \*chunk, size\_t chunk\_num\_bytes)

{

s.append((char \*)chunk, chunk\_num\_bytes);

}

string datagramCreate(uint32\_t index)

{

string payload;

string msgInfo = allMsgs[index].substr(0, allMsgs[index].find('+') + 12), token;

string msgData = allMsgs[index].substr(allMsgs[index].find('+') + 13, allMsgs[index].size());

vector <string> Tokens;

istringstream \_stream(msgInfo);

while (getline(\_stream, token, ' '))

{

if (!token.empty())

{

Tokens.push\_back(token);

}

token.clear();

}

string date = Tokens[0];

istringstream \_datestream(date);

vector <string> dateParts;

while(getline(\_datestream, token, '.'))

{

if (!token.empty())

{

dateParts.push\_back(token);

token.clear();

}

}

/\* Data processing \*/

uint32\_t msgIndex = htonl(index);

uint8\_t day = atoi(dateParts[0].c\_str());

uint8\_t month = atoi(dateParts[1].c\_str());

uint16\_t year = htons(atoi(dateParts[2].c\_str()));

int16\_t AA = htons(atoi(Tokens[1].c\_str()));

append\_chunk(payload, (const uint8\_t \*)&msgIndex, sizeof(uint32\_t));

append\_chunk(payload, (const uint8\_t \*)&day, sizeof(uint8\_t));

append\_chunk(payload, (const uint8\_t \*)&month, sizeof(uint8\_t));

append\_chunk(payload, (const uint8\_t \*)&year, sizeof(uint16\_t));

append\_chunk(payload, (const uint8\_t \*)&AA, sizeof(int16\_t));

append\_chunk(payload, (const uint8\_t \*)Tokens[2].c\_str(), 12);

append\_chunk(payload, (const uint8\_t \*)msgData.c\_str(), msgData.size());

return payload;

}

set <uint32\_t> recvData(int s, int sendCount)

{

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

fd\_set fds;

FD\_ZERO(&fds);

FD\_SET(s, &fds);

set <uint32\_t> nums;

int32\_t result = select(s + 1, &fds, 0, 0, &tv);

if (result > 0)

{

sockaddr\_in addr;

socklen\_t addrLen = sizeof(addr);

uint32\_t currentNums[20];

memset(currentNums, 0, 20 \* sizeof(uint32\_t));

int i = 0, r;

while(i != sendCount - 1)

{

r = recvfrom(s, &currentNums, 20 \* sizeof(uint32\_t), 0, (sockaddr \*)&addr, &addrLen);

i++;

}

r = recvfrom(s, &currentNums, 20 \* sizeof(uint32\_t), 0, (sockaddr \*)&addr, &addrLen);

if (r > 0)

{

for (size\_t i = 0; i < (int) r / 4; i++)

{

uint32\_t rawInt = ntohl(currentNums[i]);

if (msgArchive.find(rawInt) == msgArchive.end())

{

msgArchive.insert(rawInt);

}

}

for (size\_t i = 0; i < allMsgs.size(); i++)

{

if (msgArchive.find(i) == msgArchive.end())

{

nums.insert(i);

if (nums.size() == 20) break;

}

}

}

}

return nums;

}

static int sendData(int s, sockaddr\_in \*addr, set <uint32\_t> msgNums)

{

cout << msgNums.size() << " message(s) in queue.." << endl;

for (auto it = msgNums.begin(); it != msgNums.end(); it++)

{

string buffer = datagramCreate(\*it);

buffer.reserve(allMsgs[\*it].size() + 4);

int status = sendto(s, buffer.c\_str(), buffer.size() + 1, 0, (sockaddr \*)addr, sizeof(sockaddr));

}

return msgNums.size();

}

* 1. UDP-сервер

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <windows.h>

#include <winsock2.h>

#include <WS2tcpip.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <map>

#include <set>

#include <iomanip>

#include <string.h>

#include <stdint.h>

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

#define USHORT\_MAX 0xffff

#define UDP\_MAX\_LEN 65507

#define FILE\_LOGGER "msg.txt"

using namespace std;

map<string, set<uint32\_t, greater<uint32\_t>>> clients;

SOCKET \*sockets\_desc = new SOCKET[100];

struct sockaddr\_in \*addr = new struct sockaddr\_in[100];

uint16\_t last\_index;

bool stop\_hook = false;

union

{

uint32\_t integer;

char buffer[4];

} convert;

union

{

char buffer[2];

uint16\_t integer;

int16\_t ninteger;

} short\_ints;

string findClient(struct sockaddr\_in \*addr)

{

char ip[16];

char port[5];

string client;

inet\_ntop(AF\_INET, &addr->sin\_addr, ip, sizeof(ip));

client.clear();

\_itoa(htons(addr->sin\_port), port, 10);

client.append((char \*)ip, strlen(ip));

client += ':';

client.append((char \*)port, strlen(port));

return client;

}

int writeToFile(string buffer, string client)

{

ofstream file\_desc;

file\_desc.open(FILE\_LOGGER, ios::app);

uint16\_t day, month, year = 0;

day = buffer[0];

month = buffer[1];

short\_ints.buffer[0] = buffer[3];

short\_ints.buffer[1] = buffer[2];

year = short\_ints.integer;

file\_desc << client << " ";

file\_desc << setw(2) << setfill('0') << day << '.' << setw(2) << setfill('0') << month << '.' << year << " ";

short\_ints.buffer[0] = buffer[5];

short\_ints.buffer[1] = buffer[4];

int16\_t AA = short\_ints.ninteger;

file\_desc << AA << " ";

file\_desc << string(buffer.begin() + 6, buffer.begin() + 18) << " " << string(buffer.begin() + 18, buffer.end() -1) << endl;

file\_desc.close();

if (string(buffer.begin() + 18, buffer.end() -1) == "stop")

{

return -1;

}

return 0;

}

int analyseDatagram(SOCKET sock, int index)

{

char \*buf\_input = new char[UDP\_MAX\_LEN];

socklen\_t addrLen = sizeof(addr[index]);

int status = recvfrom(sock, buf\_input, UDP\_MAX\_LEN, 0, (struct sockaddr \*)&addr[index], &addrLen);

char num\_bytes[4];

int number = 0, result = 0, i;

string buf;

string client;

buf.append((char \*)buf\_input, status);

for (i = 0; i < 3; i++)

{

number += buf\_input[i] & 0xff;

number = (number << 8);

}

number += buf\_input[i] & 0xff;

if (status > 0)

{

set<uint32\_t, greater<uint32\_t>> msgs;

client = findClient(&addr[index]);

if (clients.find(client) == clients.end())

{

cout << "New client connected: " << client << endl;

clients.insert(make\_pair(client, msgs));

}

if (clients[client].find(number) == clients[client].end())

{

clients[client].insert(number);

result = writeToFile(string(buf.begin() + 4, buf.end()), client);

}

}

buf.clear();

delete[] buf\_input;

return (result == -1 ? -1 : 0);

}

void configListeners(uint16\_t port\_start, uint16\_t port\_end)

{

uint16\_t ports\_count = 0;

uint16\_t step;

if (port\_start > port\_end)

{

uint16\_t tmp = port\_start;

port\_start = port\_end;

port\_end = tmp;

}

ports\_count = port\_end - port\_start;

if (port\_end == port\_start)

{

ports\_count = 1;

}

cout << "Listening on ports: ";

for (step = 0; step <= ports\_count; step++)

{

unsigned long mode = 1;

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

if (s < 0) {

cout << "socket() func error " << WSAGetLastError() << endl;

return;

}

sockets\_desc[step] = s;

ioctlsocket(sockets\_desc[step], FIONBIO, &mode);

memset(&addr[step], 0, sizeof(addr[step]));

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

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

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

if (bind(sockets\_desc[step], (struct sockaddr \*)&addr[step], sizeof(addr[step])) < 0)

{

cout << "bind() func error " << WSAGetLastError() << endl;

return;

}

cout << port\_start + step << ' ';

}

cout << endl;

}

void sendToClient(SOCKET s, int index)

{

string client = findClient(&addr[index]);

char \*buffer = new char[80];

memset(buffer, 0, 80);

int i = 0;

for (auto it = clients[client].begin(); it != clients[client].end(); it++)

{

convert.integer = htonl(\*it);

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

{

buffer[i] = convert.buffer[j];

i++;

}

}

sendto(s, buffer, clients[client].size() \* 4, 0, (struct sockaddr \*)&addr[index], sizeof(addr[index]));

delete[] buffer;

}

void disconnectClient(int index)

{

string client = findClient(&addr[index]);

clients.erase(client);

cout << "Old client disconnected: " << client << endl;

}

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

{

if (argc != 3)

{

cout << "Usage: ./udpserver <port1> <port2>" << endl;

return 0;

}

WSADATA ws;

WSAStartup(MAKEWORD(2, 2), &ws);

uint16\_t port1, port2, ports\_count;

port1 = atoi(argv[1]);

port2 = atoi(argv[2]);

ports\_count = port2 - port1;

configListeners(port1, port2);

WSAEVENT \*event = new WSAEVENT[ports\_count];

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

{

event[i] = WSACreateEvent();

WSAEventSelect(sockets\_desc[i], event[i], FD\_READ | FD\_WRITE | FD\_CLOSE);

}

while (true)

{

WSANETWORKEVENTS ne;

DWORD dw = WSAWaitForMultipleEvents(ports\_count, event, FALSE, 1000, FALSE);

DWORD res = 0;

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

{

if ((res = WSAEnumNetworkEvents(sockets\_desc[i], event[i], &ne)) == 0)

{

if (ne.lNetworkEvents & FD\_READ)

{

int value = analyseDatagram(sockets\_desc[i], i);

sendToClient(sockets\_desc[i], i);

if (value == -1)

{

cout << "stop message detected. Power off...." << endl;

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

{

WSACloseEvent(event[i]);

closesocket(sockets\_desc[i]);

}

WSACleanup();

delete[] sockets\_desc;

return 0;

}

}

if (ne.lNetworkEvents & FD\_CLOSE)

{

disconnectClient(i);

}

}

}

}

return 0;

}