

Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

—
Институт Кибербезопасности и Защиты Информации

ЛАБОРАТОРНАЯ РАБОТА № 1

«РАСПРЕДЕЛЕННАЯ СИСТЕМА СБОРА ИНФОРМАЦИИ»

по дисциплине «Безопасность современных информационных технологий»

Выполнил
студент гр. 3651003/80002

<подпись>

Сошнев М.Д

Преподаватель

<подпись>

Иванов Д.В.

Санкт-Петербург
2020

Оглавление

Цель работы.....	3
Задача.....	4
Ход работы.....	5
Контрольные вопросы	7
Выводы	8
Приложение А – программа-клиент.....	9
Приложение Б – программа-сервер	16

ЦЕЛЬ РАБОТЫ

Получить навыки организации параллельной обработки большого количества клиентских соединений при помощи механизма портов завершения Win32. Получить навыки реализации шифрования передаваемых данных при помощи средств CryptoAPI. Получить навыки извлечения информации о системе.

ЗАДАЧА

Разработать распределенную систему сбора информации о компьютере, состоящую из сервера и клиента, взаимодействующих через сокеты.

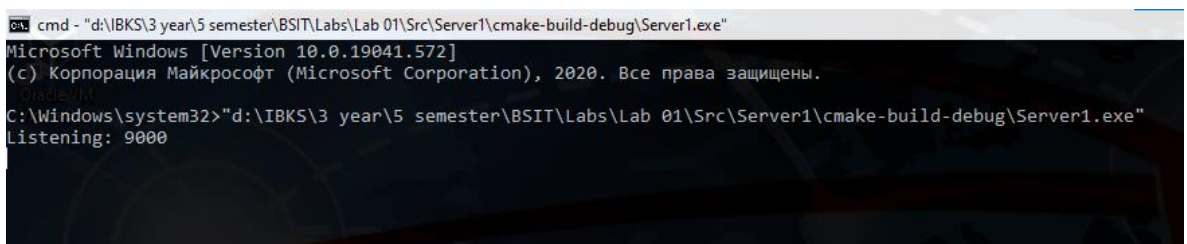
ХОД РАБОТЫ

Архитектура сети выглядит следующим образом: на центральном компьютере работает клиентская часть программы, на остальных компьютерах, подключенных в данную сеть – серверная часть, взаимодействующая с клиентом по протоколу портов завершения.

Серверы работают без остановок, имея возможность в любой момент выслать нужную информацию клиенту. Клиент же запускается только тогда, когда надо получить информацию от какого-либо сервера.

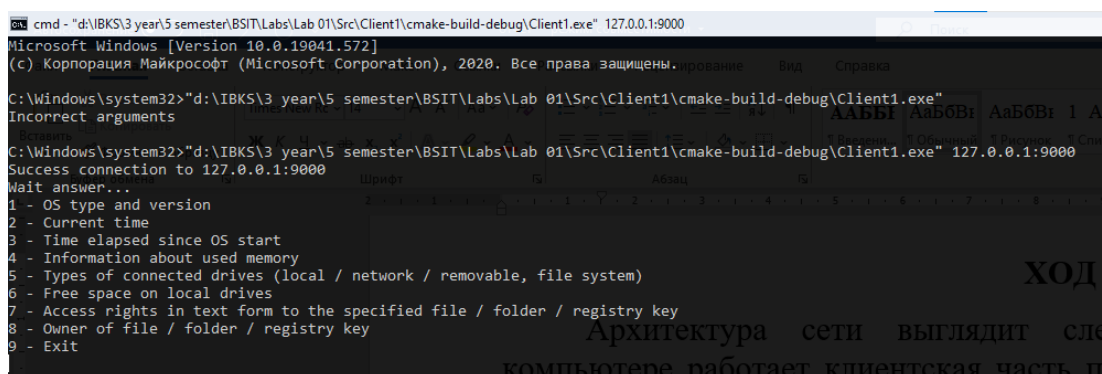
При запуске программы-сервера начинает прослушиваться один порт.

При запуске программы-клиента в качестве аргумента командной строки указывается ipv4-адрес сервера и порт. В случае успешного подключения, клиент генерирует пару ключей – открытый и закрытый, для асимметричного шифрования. Открытый ключ отправляется на сервер. Сервер, поймав открытый ключ генерирует сеансовый ключ, шифрует его полученным открытым ключом и отправляет клиенту. Клиент, поймав пакет, расшифровывает его сгенерированным закрытым ключом и получает сеансовый ключ – на этом ключе будут шифроваться все последующие пакеты. Сгенерированные клиентом открытый и закрытый ключ можно удалить. Данный процесс полностью инкапсулирован от пользователя. Далее высвечивается меню с возможными операциями и пользователю предлагается выбрать одну из них.



```
cmd - "d:\IBKS\3 year\5 semester\BSIT\Labs\Lab 01\Src\Server1\cmake-build-debug\Server1.exe"
Microsoft Windows [Version 10.0.19041.572]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.
C:\Windows\system32>"d:\IBKS\3 year\5 semester\BSIT\Labs\Lab 01\Src\Server1\cmake-build-debug\Server1.exe"
Listening: 9000
```

Рисунок 1 – запуск программы-сервера



```
cmd - "d:\IBKS\3 year\5 semester\BSIT\Labs\Lab 01\Src\Client1\cmake-build-debug\Client1.exe" 127.0.0.1:9000
Microsoft Windows [Version 10.0.19041.572]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.
C:\Windows\system32>"d:\IBKS\3 year\5 semester\BSIT\Labs\Lab 01\Src\Client1\cmake-build-debug\Client1.exe"
Incorrect arguments
C:\Windows\system32>"d:\IBKS\3 year\5 semester\BSIT\Labs\Lab 01\Src\Client1\cmake-build-debug\Client1.exe" 127.0.0.1:9000
Success connection to 127.0.0.1:9000
Wait answer...
1 - OS type and version
2 - Current time
3 - Time elapsed since OS start
4 - Information about used memory
5 - Types of connected drives (local / network / removable, file system)
6 - Free space on local drives
7 - Access rights in text form to the specified file / folder / registry key
8 - Owner of file / folder / registry key
9 - Exit
```

Рисунок 2 – запуск программы-клиента

При выборе операции будет сформирован пакет с номером операции (в случае 7ой и 8ой операции в пакет также добавляется путь к файлу, который также спрашивается у пользователя). Пакет шифруется сеансовым ключом и отправляется на сервер. Как только пакет будет полностью передан на сервер, на сервере сработает завершение по передачи данных – сервер расшифрует пакет, выполнит соответствующую процедуру, сформирует ответный пакет, зашифрует его и передаст клиенту. Клиент, получив ответ, расшифрует его и выведет в консоль. Далее будет предложено выбрать еще одну операцию и так, пока пользователь не выберет exit.

В общем случае структура пакетов выглядит следующим образом:

Клиент-серверу:



*-номер выбранной операции (0, в случае передачи сгенерированного ключа)

* - дополнительный аргумент: путь к файлу в случае 7ой или 8ой операции, сгенерированный ключ в случае 0ой операции. В остальных случаях это поле отсутствует

Сервер-клиенту:



*-номер обработанной операции (0, в случае передачи ответного ключа)

* - результат операции. В случае 0ой операции, зашифрованный сеансовый ключ

Клиент в данном случае является тонким – он просто отправляет номер операции, а при получении ответа выводит расшифрованное сообщения. Основная работа и парсинг выполняется на стороне сервера.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1) *Какова структура списков контроля доступа в ОС Windows?*

```
typedef struct _ACL {  
    BYTE AclRevision;  
    BYTE Sbz1;  
    WORD AclSize;  
    WORD AceCount;  
    WORD Sbz2;  
} ACL;
```

2) *Что такое наследование прав доступа?*

Получение субъектом всех тех прав доступа, которые были у её предка.

3) *Для чего используются well-known SID?*

Для того, чтобы идентифицировать общие группы или общих пользователей.

4) *Как выглядит схема шифрования с использованием сеансового ключа?*

1) Клиент генерирует асимметричный ключ—пару ключей публичный/приватный

2) Клиент посылает публичный ключ серверу

3) Сервер генерирует сеансовый ключ

4) Сервер получает публичный ключ клиента

5) Сервер шифрует сеансовый ключ публичным ключом клиента и отправляет получившееся зашифрованное сообщение клиенту

6) Клиент получает зашифрованное сообщение и расшифровывает его с помощью своего приватного ключа

7) У клиента и сервера есть сеансовый ключ. Теперь можно использовать симметричное шифрование для защищенного обмена сообщениями

5) *В чем преимущества использования сеансового ключа?*

- скорость
- простота реализации (за счёт более простых операций)
- меньшая требуемая длина ключа для сопоставимой стойкости
- изученность (за счёт большего возраста)

ВЫВОДЫ

В ходе выполнения лабораторной работы был получен навык в передачи данных через порты завершения, а также изучено шифрование с использованием интерфейса CryptoApi. Была реализована двухуровневая сеть с шифрованием данных.

ПРИЛОЖЕНИЕ А

Программа-клиент

main.cpp

```
#include "Client.h"

int main(int argc, char** argv)
{
    if (argc != 2) {
        printf("Incorrect arguments\n");
        return -1;
    }

    Client client(argv[1]);
    if (client.try_connect()) {
        printf("Success connection to %s:%d\n", client.get_ip(), client.get_port());
    }
    else {
        printf("No connection...\n");
        return -1;
    }

    // формируем запрос чтобы отправить ключ
    client.request_formation();

    // отправляем запрос
    client.send_request();

    // ждем ответа на отправленный ключ
    printf("Wait answer...\n");
    client.recv_request();

    // На основе ответа устанавливаем ключ для дальнейшей переписки
    client.set_key();

    assert (client.crypt_test(client.get_key()));

    while (true) {
        client.print_dialog();
        if (client.ask_request_type()) {
            return 0;
        }

        client.request_formation();
        client.send_request();

        // ждем ответа на отправленное сообщение
        printf("Wait answer...\n");
        client.recv_request();

        // Расшифровываем и выводим на экран полученное сообщение
        client.decrypt_msg();

        getchar();
        getchar();
        system("cls");

        client.clear_buf();
    }
}
```

client.h

```
#ifndef CLIENT1_CLIENT_H
#define CLIENT1_CLIENT_H

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <wincrypt.h>

#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cassert>
#include <cstdint>

#pragma comment(lib, "ws2_32.lib")

class Client {
public:
    explicit Client(char* address_port);
    ~Client();

    bool try_connect();
    void request_formation();

    int send_request() const;
    int recv_request();

    void set_key();
    int crypt_test(HCRYPTKEY key);

    const char* get_ip() const { return static_cast<const char *>(ip); }
    int get_port() const { return port; }
    int get_key() const { return session_key; }

    void print_dialog();
    int ask_request_type();

    void decrypt_msg();

    void clear_buf();

private:
    short reconnect_tries;

    struct sockaddr_in addr{};
    char ip[16]{}; // xx.xx.xx.xx
    int port;

    SOCKET socket_;

    WSABUF send_buf, recv_buf; // Буфер на прием и передачу

    // data for crypting:
    HCRYPTPROV csp; // Контекст ключей
    HCRYPTKEY session_key; // Ключ для переписки
    HCRYPTKEY key_pair; // Пара для получения нижних двух ключей
    HCRYPTKEY private_key, public_key; // приватный и публичный ключ которые нужны для
    получения SessionKey
};
```

```

uint8_t request_type;
char request_arg[128];

static unsigned int get_host_ipn(const char* name);
};

#endif //CLIENT1_CLIENT_H

```

client.cpp

```

#include "Client.h"

Client::Client(char *address_port) : reconnect_tries(10), request_type(0) {
    strncpy(ip, strtok_s(address_port, ":", &address_port), 16);
    port = atoi(address_port);

    WSADATA wsa_data;
    int check = WSAStartup(MAKEWORD(2, 2), &wsa_data);
    assert(check == 0);

    socket_ = socket(AF_INET, SOCK_STREAM, 0);
    assert(socket_ != 0);

    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = get_host_ipn(ip);

    send_buf.buf = (CHAR*)malloc(1024);
    recv_buf.buf = (CHAR*)malloc(1024);
}

Client::~Client() {
    free(send_buf.buf);
    free(recv_buf.buf);

    closesocket(socket_);
}

unsigned int Client::get_host_ipn(const char* name)
{
    struct addrinfo* addr = nullptr;
    unsigned int ip4addr = 0;

    int check = getaddrinfo(name, nullptr, nullptr, &addr);
    if (check == 0) {
        struct addrinfo* cur = addr;
        while (cur) {
            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;
}

bool Client::try_connect() {

```

```

int check;
for (int i = 0; i < reconnect_tries; ++i) {
    check = connect(socket_, (struct sockaddr*)&addr, sizeof(addr));
    if (check != 0) {
        // connection fail
        printf("Trying to reconnect... (%d)\n", i + 1);
        Sleep(100);
    }
    else {
        break;
    }
}

return check == 0;
}

void Client::request_formation()
{
    if (request_type == 0) {
        // Формируем пакет в виде ключа

        // контекст для ключей
        BOOL check = CryptAcquireContext(&csp, nullptr, MS_ENHANCED_PROV, PROV_RSA_FULL,
CRYPT_NEWKEYSET);
        if (!check) {
            check = CryptAcquireContext(&csp, nullptr, MS_ENHANCED_PROV, PROV_RSA_FULL,
(DWORD) NULL);
            assert(check);
        }

        // генерируем пару ключей
        check = CryptGenKey(csp, AT_KEYEXCHANGE, 1024<<16, &key_pair);
        assert(check);
        check = CryptGetUserKey(csp, AT_KEYEXCHANGE, &public_key);
        assert(check);
        check = CryptGetUserKey(csp, AT_KEYEXCHANGE, &private_key);
        assert(check);

        //printf("Generated public key: %lu\n", public_key);
        //printf("Generated private key: %lu\n", private_key);

        // экспортируем публичный ключ в буфер
        send_buf.len = 1024;

        check = CryptExportKey(public_key, (HCRYPTKEY) NULL, PUBLICKEYBLOB, 0, nullptr,
&send_buf.len);
        assert(check);

        check = CryptExportKey(public_key, (HCRYPTKEY) NULL, PUBLICKEYBLOB, 0,
(BYTE*)send_buf.buf, &send_buf.len);
        assert(check);

        // Добавляем в начало код 0 - т.к. это ключ
        memcpy(send_buf.buf + 1, send_buf.buf, send_buf.len++);
        send_buf.buf[0] = 0;
    }
    else {
        // Формируем пакет в виде сообщения
        // шифруем REQUEST_TYPE на ключе SESSION_KEY и кладем в буфер на отправку SEND_BUF

        memset(send_buf.buf, 0, 1024);
        send_buf.buf[0] = request_type;
        send_buf.len = 1;
    }
}

```

```

        if (request_type == 7 || request_type == 8) {
            memcpy(send_buf.buf + 2, request_arg, strlen(request_arg));
            send_buf.len += strlen(request_arg);
            send_buf.buf[1] = strlen(request_arg);
            send_buf.len++;
        }

        BOOL check = CryptEncrypt(session_key, NULL, TRUE, NULL, (BYTE*)send_buf.buf,
(DWORD*)&send_buf.len, 1024);
        assert(check);
    }
}

// Отправляет запрос из SEND_BUF через сокет SOCKET
int Client::send_request() const
{
    int send_bytes = 0;
    int res;

    while (send_bytes < send_buf.len) {
        res = send(socket_, send_buf.buf + send_bytes, send_buf.len - send_bytes, 0);

        if (res < 0) {
            return -1;
        }

        send_bytes += res;
    }

    return 0;
}

// Принимает запрос в RECV_BUF через сокет SOCKET
int Client::recv_request()
{
    recv_buf.len = recv(socket_, recv_buf.buf, 1024, 0);

    return recv_buf.len;
}

// Устанавливает параметры для ключей из RECV_BUF
void Client::set_key()
{
    // Первый символ - код 0, тк это ответ на ключ. Игнорируем его

    BOOL check = CryptImportKey(csp, (const BYTE*)(recv_buf.buf + 1), recv_buf.len - 1,
private_key, 0, &session_key);
    assert(check);

    // printf("Session key: %lu\n", session_key);
}

int Client::crypt_test(HCRYPTKEY key)
{
    // printf("\n***** Crypt test *****\n");

    int len = 16;
    BYTE test_buf[128] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    BYTE buf_0[128];

    memcpy(buf_0, test_buf, len);

    //printf("Buf          : ");
    // for (int i = 0; i < len; ++i) printf("%d ", buf_0[i]); printf("\n");

```

```

    BOOL check;
    check = CryptEncrypt(key, NULL, TRUE, NULL, (BYTE*)test_buf, (DWORD*)&len, 1024);
    assert(check);

    // printf("Crypt buf  : ");
    // for (int i = 0; i < len; ++i) printf("%d ", test_buf[i]); printf("\n");

    check = CryptDecrypt(key, NULL, TRUE, NULL, (BYTE*)test_buf, (DWORD*)&len);
    assert(check);

    // printf("Decrypt buf: ");
    // for (int i = 0; i < len; ++i) printf("%d ", test_buf[i]); printf("\n");

    // printf("*****\n\n");

    return memcmp(test_buf, buf_0, len) == 0;
}

void Client::print_dialog()
{
    printf("1 - OS type and version\n");
    printf("2 - Current time\n");
    printf("3 - Time elapsed since OS start\n");
    printf("4 - Information about used memory\n");
    printf("5 - Types of connected drives (local / network / removable, file system)\n");
    printf("6 - Free space on local drives\n");
    printf("7 - Access rights in text form to the specified file / folder / registry key\n");
    printf("8 - Owner of file / folder / registry key\n");
    printf("9 - Exit\n");
}

int Client::ask_request_type()
{
    char e;
    do {
        scanf("%d%c", &request_type, &e);
    } while (request_type > 9 || request_type < 1);

    printf("Path: ");

    if (request_type == 7 || request_type == 8) {
        int i = 0;
        do {
            scanf("%c", &request_arg[i]);
            if (request_arg[i] == '\n') {
                request_arg[i] = '\0';
                break;
            }
            ++i;
        } while (true);
    }

    if (request_type == 9) return -1;
    else return 0;
}

// decrypt RECV_BUF and print to stdout
void Client::decrypt_msg()
{
    BOOL check = CryptDecrypt(session_key, NULL, TRUE, NULL, (BYTE*)recv_buf.buf,
(DWORD*)&recv_buf.len);
    assert (check);

    for (int i = 0; i < recv_buf.len; ++i) {

```

```
        printf("%c", recv_buf.buf[i]);
    }
    printf("\n");
}

void Client::clear_buf()
{
    memset(recv_buf.buf, 0, 512);
    memset(send_buf.buf, 0, 512);
    recv_buf.len = 0;
    send_buf.len = 0;
}
```

ПРИЛОЖЕНИЕ Б

Программа-сервер

main.cpp

```
#include <iostream>

#include "Server.h"

int main()
{
    Server server;
    server.exec();

    return 0;
}
```

server.h

```
#ifndef SERVER1_SERVER_H
#define SERVER1_SERVER_H

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <winsock2.h>
#include <mswsock.h>
#include <stdio>
#include <stdlib>
#include <Wincrypt.h>
#include <aclapi.h>

#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "mswsock.lib")

class Server {
public:
    Server();
    ~Server();

    [[noreturn]] void exec();
private:
    static const int max_clients = 100;

    typedef struct client_ctx {
        int socket;
        CHAR buf_recv[512];           // Буфер приема
        CHAR buf_send[512];          // Буфер отправки
        unsigned int sz_recv;         // Принято данных
        unsigned int sz_send_total;   // Данных в буфере отправки
        unsigned int sz_send;        // Данных отправлено

        // Структуры OVERLAPPED для уведомлений о завершении
        OVERLAPPED overlap_recv;
        OVERLAPPED overlap_send;
        OVERLAPPED overlap_cancel;
        DWORD flags_recv;            // Флаги для WSARcv

        HCRYPTKEY key = 0;            // Ключ по которому переписываемся с данным клиентом
    } client_ctx;
};
```



```

HANDLE g_io_port;
SOCKET s; // Прослушивающий сокет

// Crypting data:
HCRYPTPROV csp{}; // контекст
HCRYPTKEY c_public_key{};

// Прослушивающий сокет и все сокет подключения хранятся
// в массиве структур (вместе с overlapped и буферами)
client_ctx g_ctxs[max_clients + 1];
int g_accepted_socket{};

// Функция добавляет новое принятое подключение клиента
void add_accepted_connection();

// Функция запускает операцию приема соединения
void schedule_accept();

// Handlers for overlap
void overlap_recv_handler(int idx, DWORD transferred);
void overlap_send_handler(int idx, DWORD transferred);
void overlap_cancel_handler(int idx);

// Функция запускает операцию чтения из сокета
void schedule_read(DWORD idx);

// Функция запускает операцию отправки подготовленных данных в сокет
void schedule_write(DWORD idx);

// Crypting:
inline bool client_send_crypt_key(int idx) { return g_ctxs[idx].buf_recv[0] == 0; }
void form_answ_key(int idx);

// Функция тестирования шифрования. Принимает на вход дескриптор ключа, формирует тестовый
// буфер, шифрует его и расшифровывает
static bool crypt_test(HCRYPTKEY key);

void msg_handler(int idx);

// Requests handlers:
void os_version_handler(int idx);
void current_time_handler(int idx);
void os_time_handler(int idx);
void memory_status_handler(int idx);
void disks_types_handler(int idx);
void free_space_handler(int idx);
void access_right_handler(int idx);
void owner_handler(int idx);
};

#endif //SERVER1_SERVER_H

```

server.cpp

```
#include "Server.h"

#include <VersionHelpers.h>
#include <cassert>
#include <sddl.h>

Server::Server()
{
    for (int i = 0; i < max_clients; ++i) {
        g_ctxs[i].key = 0;
        g_ctxs[i].socket = 0;
    }

    WSADATA wsa_data;
    if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0) {
        printf("WSAStartup error\n");
    }

    // Создание сокета прослушивания
    s = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);

    // Создание порта завершения
    g_io_port = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
    if (NULL == g_io_port) {
        printf("CreateIoCompletionPort error: %x\n", GetLastError());
        exit(-1);
    }

    // Обнуление структуры данных для хранения входящих соединений
    struct sockaddr_in addr;
    memset(g_ctxs, 0, sizeof(g_ctxs));
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(9000);

    if (bind(s, (struct sockaddr*) &addr, sizeof(addr)) < 0 ||
        listen(s, 1) < 0) {
        printf("error bind() or listen()\n");
        exit(-1);
    }

    printf("Listening: %hu\n", ntohs(addr.sin_port));

    // Присоединение существующего сокета s к порту io_port.
    // В качестве ключа для прослушивающего сокета используется 0
    HANDLE check = CreateIoCompletionPort((HANDLE)s, g_io_port, 0, 0);
    if (check == 0) {
        printf("CreateIoCompletionPort error: %x\n", GetLastError());
        exit(-1);
    }

    g_ctxs[0].socket = s;
    // Старт операции принятия подключения.
    schedule_accept();
}

Server::~~Server()
{
    CryptReleaseContext(csp, NULL);
}
```

```

[[noreturn]] void Server::exec()
{
    // Бесконечный цикл принятия событий о завершенных операциях
    DWORD transferred;
    ULONG_PTR key;
    OVERLAPPED* lp_overlap;

    while (true) {
        // Ожидание событий в течение 1 секунды
        BOOL b = GetQueuedCompletionStatus(g_io_port, &transferred, &key, &lp_overlap, 1000);

        if (!b) {
            // Ни одной операции не было завершено в течение заданного времени, программа
            может
            // выполнить какие-либо другие действия
            // ...
            continue;
        }

        // Поступило уведомление о завершении операции
        if (key == 0) {
            // ключ 0 - для прослушивающего сокета
            g_ctxs[0].sz_recv += transferred;

            // Принятие подключения и начало принятия следующего
            add_accepted_connection();
            schedule_accept();
        }

        // Иначе поступило событие по завершению операции от клиента.
        // Ключ key - индекс в массиве g_ctxs
        else {
            if (&g_ctxs[key].overlap_recv == lp_overlap) {
                overlap_recv_handler(key, transferred);
            }

            else if (&g_ctxs[key].overlap_send == lp_overlap) {
                overlap_send_handler(key, transferred);
            }

            else if (&g_ctxs[key].overlap_cancel == lp_overlap) {
                overlap_cancel_handler(key);
            }
        }
    }
}

// Функция добавляет новое принятое подключение клиента
void Server::add_accepted_connection()
{
    DWORD i; // Поиск места в массиве g_ctxs для вставки нового подключения
    for (i = 0; i < max_clients; i++) {
        if (g_ctxs[i].socket == 0) {
            unsigned int ip = 0;
            struct sockaddr_in* local_addr = 0, *remote_addr = 0;
            int local_addr_sz, remote_addr_sz;

            GetAcceptExSockaddrs(g_ctxs[0].buf_recv, g_ctxs[0].sz_recv, \
                sizeof(struct sockaddr_in) + 16, sizeof(struct sockaddr_in) + 16, \
                (struct sockaddr **) &local_addr, &local_addr_sz, (struct sockaddr **)
            &remote_addr, &remote_addr_sz);

```

```

        if (remote_addr) {
            ip = ntohl(remote_addr->sin_addr.s_addr);
        }

        printf("Connection %u created, remote IP: %u.%u.%u.%u\n", \
            i, (ip >> 24) & 0xff, (ip >> 16) & 0xff, (ip >> 8) & 0xff, (ip) & 0xff);

        g_ctxs[i].socket = g_accepted_socket;

        // Связь сокета с портом IOCP, в качестве key используется индекс массива
        HANDLE check = CreateIoCompletionPort((HANDLE)g_ctxs[i].socket, g_io_port, i, 0);
        if (check == 0) {
            printf("CreateIoCompletionPort error: %lx\n", GetLastError());
            return;
        }

        schedule_read(i);

        return;
    }
}

// Место не найдено => нет ресурсов для принятия соединения
closesocket(g_accepted_socket);
g_accepted_socket = 0;
}

// Функция запускает операцию приема соединения
void Server::schedule_accept()
{
    // Создание сокета для принятия подключения (AcceptEx не создает сокетов)
    g_accepted_socket = WSASocket(AF_INET, SOCK_STREAM, 0, nullptr, 0, WSA_FLAG_OVERLAPPED);
    memset(&g_ctxs[0].overlap_recv, 0, sizeof(OVERLAPPED));

    // Принятие подключения.
    // Как только операция будет завершена - порт завершения пришлет уведомление. // Размеры
    // буферов должны быть на 16 байт больше размера адреса согласно документации разработчика ОС
    AcceptEx(g_ctxs[0].socket, g_accepted_socket, \
        g_ctxs[0].buf_recv, 0, sizeof(struct sockaddr_in) + 16, \
        sizeof(struct sockaddr_in) + 16, nullptr, &g_ctxs[0].overlap_recv);
}

void Server::overlap_recv_handler(int idx, DWORD transferred)
{
    // Данные приняты:
    if (transferred == 0) {
        // Соединение разорвано
        CancelIo((HANDLE)g_ctxs[idx].socket);
        PostQueuedCompletionStatus(g_io_port, 0, idx, &g_ctxs[idx].overlap_cancel);
        return;
    }

    g_ctxs[idx].sz_recv += transferred;

    if (client_send_crypt_key(idx)) {
        form_answ_key(idx);
    }

    else {
        msg_handler(idx);
    }

    schedule_write(idx);
}

```

```

}

void Server::overlap_send_handler(int idx, DWORD transferred)
{
    // Данные отправлены
    g_ctxs[idx].sz_send += transferred;

    if (g_ctxs[idx].sz_send < g_ctxs[idx].sz_send_total && transferred > 0) {
        // Если данные отправлены не полностью - продолжить отправлять
        schedule_write(idx);
    }
    else {
        g_ctxs[idx].sz_recv = 0;
        memset(g_ctxs[idx].buf_send, 0, 512);
        schedule_read(idx);
    }
}

void Server::overlap_cancel_handler(int idx)
{
    // Все коммуникации завершены, сокет может быть закрыт
    closesocket(g_ctxs[idx].socket); memset(&g_ctxs[idx], 0, sizeof(g_ctxs[idx]));
    printf("Connection %u closed\n", idx);

    CryptDestroyKey(g_ctxs[idx].key);
}

//// socket I/O functions

// Функция запускает операцию чтения из сокета
void Server::schedule_read(DWORD idx)
{
    WSABUF buf;
    buf.buf = g_ctxs[idx].buf_recv + g_ctxs[idx].sz_recv;
    buf.len = sizeof(g_ctxs[idx].buf_recv) - g_ctxs[idx].sz_recv;

    memset(&g_ctxs[idx].overlap_recv, 0, sizeof(OVERLAPPED));
    g_ctxs[idx].flags_recv = 0;

    WSAREcv(g_ctxs[idx].socket, &buf, 1, &buf.len, &g_ctxs[idx].flags_recv,
    &g_ctxs[idx].overlap_recv, NULL);
}

// Функция запускает операцию отправки подготовленных данных в сокет
void Server::schedule_write(DWORD idx)
{
    WSABUF buf;
    buf.buf = g_ctxs[idx].buf_send + g_ctxs[idx].sz_send;
    buf.len = g_ctxs[idx].sz_send_total - g_ctxs[idx].sz_send;

    memset(&g_ctxs[idx].overlap_send, 0, sizeof(OVERLAPPED));
    WSASend(g_ctxs[idx].socket, &buf, 1, &buf.len, 0, &g_ctxs[idx].overlap_send, NULL);
}

void Server::form_answ_key(int idx) {
    // strcpy(g_ctxs[idx].buf_send, "KEY_ANSW");
    // g_ctxs[idx].sz_send_total = strlen(g_ctxs[idx].buf_send);

    BOOL check;

```

```

// Создаем контекст для ключей
check = CryptAcquireContext(&csp, NULL, MS_ENHANCED_PROV, PROV_RSA_FULL, NULL);
if (!check) {
    check = CryptAcquireContext(&csp, NULL, MS_ENHANCED_PROV, PROV_RSA_FULL,
CRYPT_NEWKEYSET);
    assert(check);
}

// Достаем полученный от клиента публичный ключ
// Первый символ - 0 - код того, что это ключ. К ключу он отношения не имеет
assert(g_ctxs[idx].buf_recv[0] == 0);
memcpy(g_ctxs[idx].buf_recv, g_ctxs[idx].buf_recv + 1, g_ctxs[idx].sz_recv);
g_ctxs[idx].buf_recv[g_ctxs[idx].sz_recv-- - 1] = 0;

check = CryptImportKey(csp, (BYTE*)g_ctxs[idx].buf_recv, g_ctxs[idx].sz_recv, NULL, NULL,
&c_public_key);
assert(check);

//printf("Import public key: %lu\n", c_public_key);

// Генерация сеансового ключа для данного клиента
check = CryptGenKey(csp, CALG_RC4, CRYPT_EXPORTABLE | CRYPT_ENCRYPT | CRYPT_DECRYPT,
&g_ctxs[idx].key);
assert(check);

//printf("Key generated: %lu\n", g_ctxs[idx].key);

// Экспорт сеансового ключа в буфер на отправку, шифруя полученным публичным
check = CryptExportKey(g_ctxs[idx].key, c_public_key, SIMPLEBLOB, NULL, nullptr,
(DWORD*)&g_ctxs[idx].sz_send_total);
assert(check);
check = CryptExportKey(g_ctxs[idx].key, c_public_key, SIMPLEBLOB, NULL,
(BYTE*)g_ctxs[idx].buf_send, (DWORD*)&g_ctxs[idx].sz_send_total);
assert(check);

memcpy(g_ctxs[idx].buf_send + 1, g_ctxs[idx].buf_send, g_ctxs[idx].sz_send_total++);
g_ctxs[idx].buf_send[0] = 0;

assert (crypt_test(g_ctxs[idx].key));
}

bool Server::crypt_test(HCRYPTKEY key)
{
    // printf("\n***** Crypt test *****\n");

    int len = 16;
    BYTE test_buf[128] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    BYTE buf_0[128];

    memcpy(buf_0, test_buf, len);

    // printf("Buf          : ");
    //for (int i = 0; i < len; ++i) printf("%d ", buf_0[i]); printf("\n");

    BOOL check;
    check = CryptEncrypt(key, NULL, TRUE, NULL, (BYTE*)test_buf, (DWORD*)&len, 1024);
    assert(check);

    // printf("Crypt buf   : ");
    // for (int i = 0; i < len; ++i) printf("%d ", test_buf[i]); printf("\n");

    check = CryptDecrypt(key, NULL, TRUE, NULL, (BYTE*)test_buf, (DWORD*)&len);
    assert(check);

    //printf("Decrypt buf: ");

```

```

    // for (int i = 0; i < len; ++i) printf("%d ", test_buf[i]); printf("\n");

    //printf("*****\n\n");

    return memcmp(test_buf, buf_0, len) == 0;
}

// Расшифровка сообщения из RECV_BUF и формирование ответа в SEND_BUF с зашифровкой
void Server::msg_handler(int idx)
{
    g_ctxs[idx].sz_send = 0;

    // расшифровка
    BOOL check = CryptDecrypt(g_ctxs[idx].key, NULL, TRUE, NULL, (BYTE*)g_ctxs[idx].buf_recv,
(DWORD*)&g_ctxs[idx].sz_recv);
    assert (check);

    int request_type;
    request_type = (int)(g_ctxs[idx].buf_recv[0]);
    assert (request_type <= 8 && request_type >= 1);

    memset(g_ctxs[idx].buf_send, 0, 512);

    switch (request_type) {
        case 1:
            os_version_handler(idx);
            break;
        case 2:
            current_time_handler(idx);
            break;
        case 3:
            os_time_handler(idx);
            break;
        case 4:
            memory_status_handler(idx);
            break;
        case 5:
            disks_types_handler(idx);
            break;
        case 6:
            free_space_handler(idx);
            break;
        case 7:
            access_right_handler(idx);
            break;
        case 8:
            owner_handler(idx);
            break;
        default: break;
        // сюда не попадем из-за assertа выше
    }

    g_ctxs[idx].sz_send_total = strlen(g_ctxs[idx].buf_send);

    // шифруем BUF_SEND
    check = CryptEncrypt(g_ctxs[idx].key, NULL, TRUE, NULL, (BYTE*)g_ctxs[idx].buf_send,
(DWORD*)&g_ctxs[idx].sz_send_total, 512);
    assert(check);

    printf("Request %d processed successfully\n", request_type);
}

// Хендлеры анализируют расшифрованный BUF_RECV и формируют ответ в BUF_SEND

```

```

void Server::os_version_handler(int idx)
{
    NTSTATUS(WINAPI *RtlGetVersion)(LPOSVERSIONINFOEXW);
    OSVERSIONINFOEXW osInfo;

    *(FARPROC*)&RtlGetVersion = GetProcAddress(GetModuleHandleA("ntdll"), "RtlGetVersion");
    int os = 0;
    if (nullptr != RtlGetVersion) {
        osInfo.dwOSVersionInfoSize = sizeof(osInfo);
        RtlGetVersion(&osInfo);
        os = osInfo.dwMajorVersion;
    }

    //    printf("OS=%d", os);

    if (os==10) strncpy(g_ctxs[idx].buf_send, (char*)"win10", 5);
    else if (IsWindows8OrGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"win8", 4);
    else if (IsWindows7SP10rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"win7SP1", 7);
    else if (IsWindows7OrGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"win7", 4);
    else if (IsWindowsVistaSP20rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"vistaSP2",
8);
    else if (IsWindowsVistaSP10rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"vistaSP1",
8);
    else if (IsWindowsVistaOrGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"vista", 5);
    else if (IsWindowsXPSP30rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"XPSP3", 5);
    else if (IsWindowsXPSP20rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"XPSP2", 5);
    else if (IsWindowsXPSP10rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"XPSP1", 5);
    else if (IsWindowsXP0rGreater()) strncpy(g_ctxs[idx].buf_send, (char*)"XP", 2);
}

void Server::current_time_handler(int idx)
{
    SYSTEMTIME sm;
    GetSystemTime(&sm);

    sprintf(g_ctxs[idx].buf_send, "%d%.%d%.%d %d%.%d%.%d%.%d", sm.wDay/10, sm.wDay%10, \
        sm.wMonth/10, sm.wMonth%10, sm.wYear, \
        sm.wHour/10, sm.wHour%10, sm.wMinute/10, sm.wMinute%10, sm.wSecond/10, sm.wSecond%10);
}

void Server::os_time_handler(int idx)
{
    int time_ticks = GetTickCount();
    int hours = time_ticks / (1000 * 60 * 60);
    int minutes = time_ticks / (1000 * 60) - hours * 60;
    int seconds = (time_ticks / 1000) - (hours * 60 * 60) - minutes * 60;

    sprintf(g_ctxs[idx].buf_send, "%d hours, %d minutes, %d seconds", hours, minutes,
seconds);
}

void Server::memory_status_handler(int idx)
{
    MEMORYSTATUS stat;
    GlobalMemoryStatus(&stat);

    const float convert = (const float)1073741824;

    sprintf(g_ctxs[idx].buf_send, \
        "MemoryLoad: %lu%%\n" \
        "TotalPhys: %lu Bytes, %.3f GB\n" \
        "AvailPhys: %lu Bytes, %.3f GB\n" \
        "TotalPageFile: %lu Bytes, %.3f GB\n" \

```



```

        "AvailPageFile: %lu Bytes, %.3f GB\n" \
        "TotalVirtual: %lu Bytes, %.3f GB\n" \
        "AvailVirtual: %lu Bytes, %.3f GB\n", \
        stat.dwMemoryLoad, stat.dwTotalPhys, (float)(stat.dwTotalPhys/convert), \
        stat.dwAvailPhys, (float)(stat.dwAvailPhys / convert), \
        stat.dwTotalPageFile, (float)(stat.dwTotalPageFile / convert), \
        stat.dwAvailPageFile, (float)(stat.dwAvailPageFile / convert), \
        stat.dwTotalVirtual, (float)(stat.dwTotalVirtual / convert), \
        stat.dwAvailVirtual, (float)(stat.dwAvailVirtual / convert));
    }

    void Server::disks_types_handler(int idx)
{
    int n, type, len;
    DWORD dr = GetLogicalDrives();

    char disks[26][3] = { 0 };
    char name[128];
    char name_file_system[128];

    DWORD serial_number;
    DWORD size_tom;
    DWORD type_file_system;

    const static char disks_types[4][16] = {
        "removable",
        "fixed",
        "network",
        "CD-ROM"
    };

    for (int i = 0, j = 0, k = 0; i < 26; i++)
    {
        n = ((dr >> i) & 0x1);
        if (n == 1)
        {
            disks[j][0] = char(65 + i);
            disks[j][1] = ':';
            type = GetDriveTypeA(disks[j]);
            GetVolumeInformation(disks[j], name, sizeof(name), &serial_number, &size_tom,
&type_file_system, name_file_system, sizeof(name_file_system));
            g_ctxs[idx].buf_send[k] = disks[j][0];
            k++;
            g_ctxs[idx].buf_send[k] = disks[j][1];
            k++;

            k += strlen (
                strcpy(g_ctxs[idx].buf_send + k, "\n\t type: ")
            );

            strcpy(g_ctxs[idx].buf_send + k, disks_types[type]);
            k += strlen(disks_types[type]);

            k += strlen (
                strcpy(g_ctxs[idx].buf_send + k, "\n\t file system: ")
            );

            len = strlen(name_file_system);
            memcpy(g_ctxs[idx].buf_send + k, name_file_system, len);
            k += len;

            k += strlen (
                strcpy(g_ctxs[idx].buf_send + k, "\n")
            );
        }
    }
}

```

```

        j++;
    }
}

void Server::free_space_handler(int idx)
{
    int i, n;
    char disks[26][3] = { 0 };
    int s, b, f, c;
    double freeSpace;
    int count = 0;
    DWORD dr = GetLogicalDrives();

    for (i = 0; i < 26; i++)
    {
        n = ((dr >> i) & 0x00000001);
        if (n == 1)
        {
            disks[count][0] = char(65 + i);
            disks[count][1] = ':';
            if (GetDriveTypeA(disks[count]) == DRIVE_FIXED)
            {
                g_ctxs[idx].buf_send[count * 13 + 2] = ' ';

                GetDiskFreeSpaceA(disks[count], (LPDWORD)& s, (LPDWORD)& b, (LPDWORD)& f,
(LPDWORD)& c);
                freeSpace = (double)f * (double)s * (double)b / 1024.0 / 1024.0 / 1024.0;
                g_ctxs[idx].buf_send[count * 13] = disks[count][0];
                g_ctxs[idx].buf_send[count * 13 + 1] = disks[count][1];

                char buf[16];
                memcpy(g_ctxs[idx].buf_send + count * 13 + 3, gcvt(freeSpace, 10, buf), 8);
                memcpy(g_ctxs[idx].buf_send + count * 13 + 9, " Gb\n", 4);
                count++;
            }
        }
    }

    memcpy(g_ctxs[idx].buf_send, g_ctxs[idx].buf_send, count * 10);
}

void Server::owner_handler(int idx)
{
    int path_len = g_ctxs[idx].buf_recv[1];
    char path[128] = "";

    strncpy(path, g_ctxs[idx].buf_recv + 2, path_len);

    // printf("\nPath: %s\n", path);

    PSECURITY_DESCRIPTOR pSD;
    ACL_SIZE_INFORMATION aclInfo;
    PSID pOwnerSid;

    GetNamedSecurityInfo((LPCSTR)path, SE_FILE_OBJECT, OWNER_SECURITY_INFORMATION, &pOwnerSid,
nullptr, nullptr, nullptr, &pSD);

    char user[50] = "", domain[50] = "";
    DWORD user_len, domain_len;
    SID_NAME_USE type;

    BOOL check = LookupAccountSid(nullptr, pOwnerSid, user, &user_len, domain, &domain_len,
&type);

```

```

    if (1 || check) {
        if (strlen(user) == 0) {
            strcpy(user, "Max");
        }
        strcat(g_ctxs[idx].buf_send, "Owner: ");
        strcat(g_ctxs[idx].buf_send, user);
    }
}

void Server::access_right_handler(int idx)
{
    int path_len = g_ctxs[idx].buf_recv[1];
    char path[128] = "";

    strncpy(path, g_ctxs[idx].buf_recv + 2, path_len);

    // printf("\nPath: %s\n", path);

    PACL a;
    PSECURITY_DESCRIPTOR pSD;
    ACL_SIZE_INFORMATION aclInfo;

    int check;
    check = GetNamedSecurityInfo((LPCSTR)path, SE_FILE_OBJECT, DACL_SECURITY_INFORMATION,
    nullptr, nullptr, &a, nullptr, &pSD);

    GetAclInformation(a, &aclInfo, sizeof(aclInfo), AclSizeInformation);
    LPVOID AceInfo;

    SID_NAME_USE type;
    DWORD mask;

    char user[32];
    char domain[32];
    char *sid_str;

    for (int i = 0; i < a->AceCount; ++i) {
        memset(user, 0, 32);
        memset(domain, 0, 32);

        DWORD user_len = 512, domain_len = 512;

        GetAce(a, i, &AceInfo);
        PSID pSID = (PSID)((ACCESS_ALLOWED_ACE*)AceInfo->SidStart);

        BOOL check = LookupAccountSid(nullptr, pSID, (LPSTR) user, &user_len, (LPSTR) domain,
        &domain_len, &type);
        if (check) {
            ConvertSidToStringSidA(pSID, &sid_str);
            mask = (DWORD)((ACCESS_ALLOWED_ACE*)AceInfo->Mask;

            strcat(g_ctxs[idx].buf_send, "User: ");
            strcat(g_ctxs[idx].buf_send, user);
            strcat(g_ctxs[idx].buf_send, "\nSID: ");
            strcat(g_ctxs[idx].buf_send, sid_str);
            strcat(g_ctxs[idx].buf_send, "\nMask: ");

            char mask_str[16];
            sprintf(mask_str, "0x%lx", mask);

            strcat(g_ctxs[idx].buf_send, mask_str);
        }

        break;
    }
}

```

```
    }  
    //delete[] user;  
    // delete[] domain;  
}
```