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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**«Удаленный файловый менеджер с имперсонацией клиента»**

1. по дисциплине «Безопасность современных информационных технологий»
2. Выполнил
3. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. Дахнович А.Д.

<*подпись*>

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

Получить навыки работы с механизмом удаленного вызова процедур (RPC)

1. **Задачи**

Написать программу-сервер и программу-клиент, работающие под Windows 7-10. Сервер должен предоставлять доступ локальным и удаленным клиентам к файлам в своей файловой системе.

Требования:

* Statefull сервер;
* Сервер не должен быть интерактивным (интерфейс командной строки);
* Взаимодействие с клиентами должно осуществляться с помощью механизма RPC;
* При обслуживании клиента должна осуществляться его имперсонация;
* Пользователю должны предоставляться следующие операции: копирование указанного файла с клиента на сервер, загрузка указанного файла с сервера на клиента, удаление указанного файла на сервере;
* Имя файла передается в формате UNC.

1. **Теоретические сведения**

Удаленный вызов процедур (англ. Remote Procedure Call (RPC)) – класс технологий, позволяющий компьютерным программам вызывать функции и процедуры в другом адресном пространстве (как правило, на удаленных компьютерах).

Statefull сервер значит, что клиент сначала должен идентифицироваться, после чего сервер его «запоминает» и клиент способен выполнять доступные ему действия на сервере до тех пор, пока сеанс связи между клиентом и сервером не будет разорван и инсцинирующей разрыв стороной не будет являться клиент.

Stateless сервер значит, что функционала statefull сервера не предусмотрено.

Для разработки RPC клиент-серверного приложения сначала разрабатывается интерфейс. Интерфейс состоит из названия интерфейса, некоторых атрибутов, необязательных определений типов и констант и набора объявлений процедур-прототипов.

Интерфейсы определяются с использованием MIDL (Microsoft Interface Definition Language = Языка Определения Интерфейсов Microsoft). В интерфейсе в атрибутах необходимо указать UUID интерфейса. Его можно сгенерировать программой Uuidgen.

1. **Ход работы**
   1. **Разработка интерфейса**

Был написан интерфейс в файле «myapp.idl» (листинг в приложении А). Атрибут UUID был сгенерирован с помощь программы «uuidgen-1.1.jar». Функции интерфейса описаны в таблице 1.

Таблица 1 – Функции интерфейса

|  |  |
| --- | --- |
| Функция | Описание |
| login |  |
| logout |  |
| getFileSize | Получение размера файла |
| downloadFile | Скачивание файла |
| uploadFile | Загрузка файла |
| deleteFile | Удаление файла |

Далее файл был скомпилирован с помощью команды «midl InterfaceRPC.idl» из командной строки Visual Studio. На выходе были получены файлы «InterfaceRPC \_c.c», «InterfaceRPC.h», «InterfaceRPC\_s.c».

* 1. **Разработка сервера**

В проект сервера были добавлены скомпилированные файлы «InterfaceRPC\_s.c» и «InterfaceRPC.h» (рисунок 1).

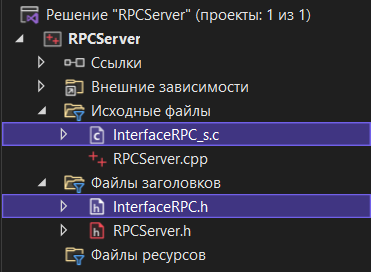


Рисунок 1 – Добавление файлов интерфейса в проект сервера

Заголовочный файл интерфейса был добавлен в файл «RPCServer.cpp». На рисунке 1 также видны файлы «RPCServer.cpp» и «RPCServer.h» – это основной функционал программы – скачивание, удаление и загрузка файлов.

Также был создан файл «main.cpp», в которой были прописаны функции интерфейса, которые вызывают методы разработанного класса.

* 1. **Разработка клиента**

На рисунке 2 представлены файлы проекта клиентского приложения.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Файлы проекта клиентского приложения

Как видно из картинки, были добавлены скомпилированные файлы интерфейса, а также разработан класс меню для пользователя и файл «main.cpp» для запуска меню.

* 1. **Тестирование**

Были созданы 2 пользователя User1 и User2 (рисунки 3 и 4).

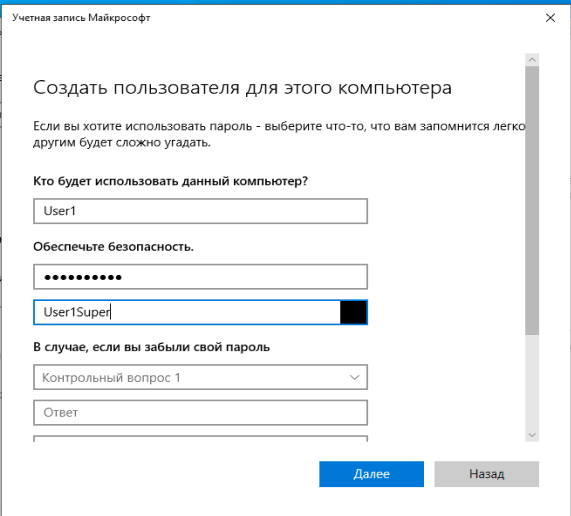


Рисунок 3 – Создание пользователя 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Создание пользователя 2

На рисунке 5 представлен интерфейс работы клиентского приложения. В нем предлагается ввести IP-адрес сервера для установления связи. Используемый порт не выбирается – используется 4444. Далее предлагается ввести логин и пароль пользователя. При успешном входе будет выведено сообщение «[\*] Success login», в противном случае будет предложено попробовать ввести логин и пароль снова.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Интерфейс клиентского приложения

На рисунке 6 представлены логи с сервера – показана попытка войти и ее результат.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Логи с серверного приложения

Далее пользователю 1 был выдан полный доступ к папке (рисунок 7), чтобы продемонстрировать операции удаления, скачивания и загрузки файлов.

Изображение выглядит как текст, снимок экрана, внутренний, ноутбук

Автоматически созданное описание

Рисунок 7 – Настройки безопасности директории

На рисунке 8 представлены логи с сервера при скачивании файла.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – Логи серверного приложения при скачивании файла

На рисунке 9 представлен интерфейс пользователя при скачивании и удалении файла с сервера.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – Интерфейс пользователя при скачивании и удалении файла с сервера

На рисунке 10 представлено содержимое скачанного файла.

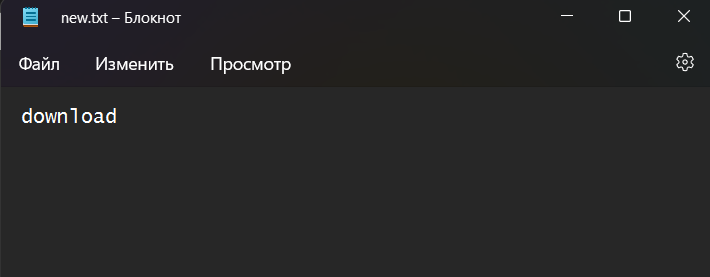


Рисунок 10 – Содержимое файла «new.txt»

На рисунке 11 показаны логи серверного приложения при скачивании, удалении и загрузке файла. Также показано, что в результате там появился новый файл, и показано содержимое нового файла.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Скачивание, удаление и загрузка нового файла на сервер

Далее была попытка произвести все эти операции пользователем 2, которому не было выдано никаких прав. В таком случае результат всех действий – в доступе отказано. Результат представлен на рисунке 12.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Попытка выполнить действия удаленно на сервере без прав доступа

1. **Выводы**

В ходе выполнения данной лабораторной работы были изучены основы работы с MIDL, с RPC-сервером и клиентом. Изученный материал был применен на практике – разработаны сервер и клиент. Клиент имеет возможность удалять, скачивать и загружать файлы от имени пользователя сервера – если тот имеет надлежащие права.

**Приложение А**

Листинг «InterfaceRPC.idl»

//атрибуты

[

uuid(7ac9967a-6d7b-44c5-9764-3359219a9e4f), // сгенерирован с помощью утилиты uuidgen

version(1.0)

]

interface InterfaceRPC {

unsigned char login(

[in, string] const unsigned char\* username, // [in, string] - атрибут string указывает, что приходит строка

[in, string] const unsigned char\* password

);

void logout(

);

unsigned char getFileSize(

[in, string] const unsigned char\* file\_name,

[out] unsigned int\* file\_size // указатель для передачи в метод NdrClientCall3

);

unsigned char downloadFile(

[in, string] const unsigned char\* file\_name,

[in] unsigned int file\_size, // клиент получает заранее с помощь метода getFileSize

[out, size\_is(file\_size + 1)] unsigned char\* data // size\_is показывает размер выделяемого буфера

);

unsigned char uploadFile(

[in, string] const unsigned char\* file\_name,

[in] unsigned int file\_size,

[in, size\_is(file\_size + 1)] unsigned char\* data

);

unsigned char deleteFile(

[in, string] const unsigned char\* file\_name

);

}

**Приложение Б**

Листинг «menu.cpp»

#pragma comment(lib, "rpcrt4.lib")

#include "Menu.h"

#include "InterfaceRPC\_h.h"

#include <iostream>

#include <string>

#include <sstream>

#include <fstream>

#include <vector>

int Menu::start() {

std::string ip, port;

std::cout << "[\*] Set IP-address of server" << std::endl <<

"\tIP-address: ";

//std::cin >> ip;

//\_ip = ip;

//std::cout << "\tPort: ";

//std::cin >> port;

\_ip = ip = "192.168.159.131";

port = "4444";

RPC\_STATUS status;

RPC\_CSTR binding\_string;

status = RpcStringBindingComposeA(

NULL,

(RPC\_CSTR)("ncacn\_ip\_tcp"),

(RPC\_CSTR)(ip.c\_str()),

(RPC\_CSTR)(port.c\_str()),

NULL,

&binding\_string);

if (status) {

std::cout << "[!] RpcStringBindingComposeA error, code: " << status << std::endl;

return 0;

}

std::cout << "[\*] RpcStringBindingComposeA success" << std::endl;

RPC\_BINDING\_HANDLE handler;

status = RpcBindingFromStringBindingA(binding\_string, &handler);

if (status) {

std::cout << "[!] RpcBindingFromStringBindingA error, code: " << status << std::endl;

return 0;

}

std::cout << "[\*] RpcBindingFromStringBindingA success" << std::endl;

\_handler = handler;

std::string username, password;

while (1) {

std::cout << "[\*] Type username and password" << std::endl <<

"\tUsername: ";

//getchar();

//std::getline(std::cin, username);

std::cout << "\tPassword: ";

//std::getline(std::cin, password);

username = "User2";

password = "User2Super";

if (!login(handler, (unsigned char\*)username.c\_str(), (unsigned char\*)password.c\_str())) {

std::cout << "[!] Unsuccess login, try again" << std::endl;

}

else {

std::cout << "[\*] Success login" << std::endl;

break;

}

}

help();

return 1;

}

void Menu::help() {

std::cout << "[\*] Usage:\n\t-d <FILE\_NAME>\tdelete file <FILE\_NAME>\n\t" <<

"-dl <FILE\_NAME> <NEW\_FILE>\tdownload file <FILE\_NAME> and save as <NEW\_FILE>\n\t" <<

"-u <FILE\_NAME>\tupload file <FILE\_NAME>\n\t" <<

"-exit" << std::endl;

}

void Menu::typeCommand() {

std::string command;

std::cout << "[\*] Type a command: ";

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

doCommand(command);

}

void Menu::doCommand(std::string cmd) {

std::istringstream buf(cmd);

std::string temp, temp1;

buf >> temp;

if (temp == "-d") {

buf >> temp;

doDelete(temp);

}

else if (temp == "-dl") {

buf >> temp;

buf >> temp1;

doDownload(temp1, temp);

}

else if (temp == "-u") {

buf >> temp;

doUpload(temp);

}

else if (temp == "-exit") {

exit(0);

}

else {

std::cout << "[!] Unknown command" << std::endl;

help();

}

}

void Menu::doDelete(std::string path) {

//std::string path = getFilePath();

path = "\\\\" + \_ip + "\\" + path;

if (!deleteFile(\_handler, (unsigned char\*)path.c\_str())) {

std::cout << "[!] File did not delete" << std::endl;

return;

}

std::cout << "[\*] File is deleted" << std::endl;

}

void Menu::doDownload(std::string newFileName, std::string path) {

path = "\\\\" + \_ip + "\\" + path;

unsigned int res = fileSize(path);

if (res < 0) {

return;

}

unsigned int file\_size = res;

unsigned char\* data = new unsigned char[file\_size + 1];

if (!downloadFile(\_handler, (unsigned char\*)path.c\_str(), file\_size, data))

std::cout << "[!] Cannot dowload file" << std::endl;

else {

std::ofstream out(newFileName);

out << data;

std::cout << "[\*] Successful dowload file" << std::endl;

out.close();

}

}

void Menu::doUpload(std::string path) {

std::ifstream in(path, std::ifstream::binary);

if (!in.good()) {

std::cout << "[!] Cannot open file" << std::endl;

return;

}

std::vector<unsigned char> data;

std::istreambuf\_iterator<char> begin = std::istreambuf\_iterator<char>(in);

auto end = std::istreambuf\_iterator<char>();

std::copy(begin, end, std::back\_inserter(data));

unsigned int file\_size = data.size();

path = "\\\\" + \_ip + "\\" + path;

if (!uploadFile(\_handler, (unsigned char\*)path.c\_str(), file\_size, (unsigned char\*)data.data()))

std::cout << "[!] Cannot upload file" << std::endl;

else

std::cout << "[\*] Successful upload file" << std::endl;

}

int Menu::fileSize(std::string path) {

unsigned int fileSize;

if (!getFileSize(\_handler, (unsigned char\*)path.c\_str(), &fileSize)) {

std::cout << "[!] Cannot get size of this file" << std::endl;

return -1;

}

std::cout << "[\*] Success got size of this file: " << fileSize << std::endl;

return fileSize;

}

void\* \_\_RPC\_USER midl\_user\_allocate(size\_t size)

{

return malloc(size);

}

// Memory deallocation function for RPC.

void \_\_RPC\_USER midl\_user\_free(void\* p)

{

free(p);

}

**Приложение В**

Листинг «RPCServer.cpp»

#pragma comment(lib, "rpcrt4.lib")

#include "RPCServer.h"

#include "InterfaceRPC\_h.h"

#include <iostream>

#include <Windows.h>

#include <unordered\_map>

#include <future>

#include <fstream>

#include <filesystem>

#include <cstdio>

RPCServer::RPCServer() {

\_server\_name = "192.168.159.131";

\_error\_code = 0; // no error

RPC\_STATUS status;

status = RpcServerRegisterIfEx(InterfaceRPC\_v1\_0\_s\_ifspec, NULL, NULL,

RPC\_IF\_ALLOW\_CALLBACKS\_WITH\_NO\_AUTH, RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, NULL);

if (status != RPC\_S\_OK) {

std::cout << "[!] RpcServerRegisterIf failed, error code: " << (int)status;

\_error\_code = (int)status;

return;

}

status = RpcServerUseProtseqEpA(

(RPC\_CSTR)("ncacn\_ip\_tcp"),

RPC\_C\_PROTSEQ\_MAX\_REQS\_DEFAULT,

(RPC\_CSTR)("4444"),

NULL

);

if (status != RPC\_S\_OK) { // RPC\_S\_OK return value

std::cout << "[!] RpcServerUseProtseqEpA failed, error code: " << (int)status;

\_error\_code = (int)status;

return;

}

RPC\_BINDING\_VECTOR\* rpcBindingVector;

status = RpcServerInqBindings(&rpcBindingVector);

if (status != RPC\_S\_OK) { // RPC\_S\_OK return value

std::cout << "[!] RpcServerInqBindings failed, error code: " << (int)status;

\_error\_code = (int)status;

return;

}

status = RpcEpRegister(InterfaceRPC\_v1\_0\_s\_ifspec, rpcBindingVector, NULL, NULL);

if (status != RPC\_S\_OK) { // RPC\_S\_OK return value

std::cout << "[!] RpcEpRegister failed, error code: " << (int)status;

\_error\_code = (int)status;

return;

}

}

bool RPCServer::listen() {

RPC\_STATUS status;

std::cout << "[\*] Listening" << std::endl;

status = RpcServerListen(1, RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, false);

if (status != RPC\_S\_OK) {

std::cout << "[!] RpcServerListen failed, error code: " << (int)status;

\_error\_code = (int)status;

return false;

}

std::cout << "[\*] Listen success" << std::endl;

return true;

}

bool RPCServer::login(handle\_t binding\_handle, const unsigned char\* username,

const unsigned char\* password) {

std::cout << "[\*] Client tries to login, username: " << username << " password: " << password << std::endl;

HANDLE user\_token;

// attempt to log on on local computer as user and get token handle for impersonalisation

int success = LogonUserA((LPCSTR)username, NULL,

(LPCSTR)password, LOGON32\_LOGON\_INTERACTIVE, LOGON32\_PROVIDER\_DEFAULT,

&user\_token);

if (!success) // no zero if success

{

int error = GetLastError();

if (error == ERROR\_LOGON\_FAILURE) {

std::cout << "[!] Invalid username or password" << std::endl;

return false;

}

else // runtime error

std::cout << "[!] LogonUser failed, error code: " + error;

}

\_clients.insert(std::make\_pair(binding\_handle, user\_token)); // for many users in one time

std::cout << "[\*] Client logged in" << std::endl;

return true;

}

void RPCServer::logout(handle\_t binding\_handle) {

std::cout << "[\*] Client logged out" << std::endl;

\_clients.erase(binding\_handle);

}

bool RPCServer::downloadFile(handle\_t binding\_handle, const unsigned char\* file\_name,

const unsigned int file\_size, unsigned char\* data) {

unsigned int copy\_file\_size = file\_size;

return workWithFile(binding\_handle, file\_name, &copy\_file\_size, data, 1);

}

bool RPCServer::uploadFile(handle\_t binding\_handle, const unsigned char\* file\_name,

const unsigned int file\_size, unsigned char\* data) {

unsigned int copy\_file\_size = file\_size;

return workWithFile(binding\_handle, file\_name, &copy\_file\_size, data, 0);

}

bool RPCServer::deleteFile(handle\_t binding\_handle, const unsigned char\* file\_name) {

unsigned int file\_size = 0;

return workWithFile(binding\_handle, file\_name, &file\_size, nullptr, 2);

}

bool RPCServer::getFileSize(handle\_t binding\_handle, const unsigned char\* file\_name, unsigned int\* file\_size) {

return workWithFile(binding\_handle, file\_name, file\_size, nullptr, 3);

}

bool RPCServer::workWithFile(handle\_t binding\_handle, const unsigned char\* file\_name, unsigned int\* file\_size,

unsigned char\* data, int action) {

auto client = \_clients.find(binding\_handle);

if (client == \_clients.end()) // not founded -> ignore

return false;

std::string local\_file\_name;

try {

if ((local\_file\_name = convertUNCToLocal(file\_name)) == "") {

return false;

}

std::cout << "[\*] Client tries to upload file" << std::endl;

// create thread which will impersonate client

std::future<bool> future = std::async(std::launch::async, [&]() {

if (!ImpersonateLoggedOnUser(client->second)) { // action from user face

std::cout << "[!] ImpersonateLoggedOnUser failed, error code: " << GetLastError() << std::endl;

return false;

}

bool res = true;

switch (action) {

case 0: { // upload

std::ofstream output\_stream(local\_file\_name, std::ifstream::trunc | std::ifstream::binary);

if (!output\_stream.good()) {

std::cout << "[!] File cannot be opened for writing" << std::endl;

res = false;

}

else {

std::copy(data, &data[\*file\_size], std::ostreambuf\_iterator<char>(output\_stream));

std::cout << "[\*] File is uploaded" << std::endl;

}

break;

}

case 1: { // download

std::ifstream input\_stream(local\_file\_name, std::ifstream::binary);

if (!input\_stream.good()) {

std::cout << "[!] File cannot be opened" << std::endl;

res = false;

}

else {

std::istreambuf\_iterator<char> end;

unsigned int cnt = 0;

for (std::istreambuf\_iterator<char> ptr(input\_stream); ptr != end; ++ptr) {

data[cnt++] = \*ptr;

}

data[cnt] = 0;

std::cout << "[\*] File is downloaded" << std::endl;

}

break;

}

case 2: { // delete

int result = remove(local\_file\_name.c\_str());

if (result == 0) {

std::cout << "[\*] File is deleted" << std::endl;

res = true;

}

else {

std::cout << "[!] File cannot be deleted, error code: " << result << std::endl;

res = false;

}

break;

}

case 3: {

std::ifstream input\_stream(local\_file\_name, std::ifstream::binary);

int size = input\_stream.tellg();

\*file\_size = size;

std::cout << "[\*] File size is got" << std::endl;

res = true;

break;

}

}

if (!RevertToSelf()) {

std::cout << "[!] RevertToSelf failed, error code: " << GetLastError() << std::endl;

return false;

}

return res;

});

return future.get();

}

catch (std::exception& e)

{

std::cout << "[!] " << e.what() << std::endl;

return false;

}

}

std::string RPCServer::convertUNCToLocal(const unsigned char\* file\_name)

{

// check begin of UNC name

std::string string\_file\_name((char\*)file\_name);

if (string\_file\_name[0] != '\\' || string\_file\_name[1] != '\\') {

std::cout << "[!] Not UNC format" << std::endl;

return "";

}

// check server name before first '\'

auto server\_end = string\_file\_name.find('\\', 2);

std::string received\_server\_name(string\_file\_name.begin() + 2, string\_file\_name.begin() + server\_end);

if (received\_server\_name != \_server\_name) {

std::cout << "[!] Wrong server name" << std::endl;

return "";

}

auto local\_path = string\_file\_name.substr(server\_end + 1);

return local\_path;

}

void\* \_\_RPC\_USER midl\_user\_allocate(size\_t size)

{

return malloc(size);

}

// Memory deallocation function for RPC.

void \_\_RPC\_USER midl\_user\_free(void\* p)

{

free(p);

}