Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

УДАЛЕННЫЙ ФАЙЛОВЫЙ МЕНЕДЖЕР С ИМПЕРСОНАЦИЕЙ КЛИЕНТА.
по дисциплине «Безопасность современных информационных технологий»

Выполнил
студент гр. 33508/3 Проценко Е.Г.
_____

Руководитель Иванов Д.В.
_____

Санкт-Петербург
2016

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Написать программу-сервер и программу-клиент, работающие под Windows XP и 7. 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.

В ходе работы должны быть проведены эксперименты с запуском сервера и клиентов под различными учетными записями с демонстрацией работы механизмов контроля доступа. Эксперименты должны показывать, что работает как повышение, так и понижение прав потока по отношению к процессу сервера.

2 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). Обычно, реализация RPC технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур, для необъектных RPC). Различные реализации RPC имеют очень отличающуюся друг от друга архитектуру и разнятся в своих возможностях.

Реализуемый сервер RPC может быть statefull сервером– это 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 инсценирующей разрыв стороной не будет являться клиент. Таким образом, сеанс связи между клиентом и сервером не прерывается. Если же описанного выше функционал сервера не наблюдается, сервер называется stateless сервером.

Имперсонализация — это концепт безопасности присущий Windows, что позволяет серверному приложению временно «быть» клиентом для доступа к охраняемому объекту. Имперсонализация состоит из трёх уровней: идентификация, позволяющая серверу проверять подлинность клиента, имперсонализация, позволяющая серверу работать от имени клиента, и делегация, то же, что и имперсонализация, только расширена на работу с удалёнными системами, с которыми связывается сервер.

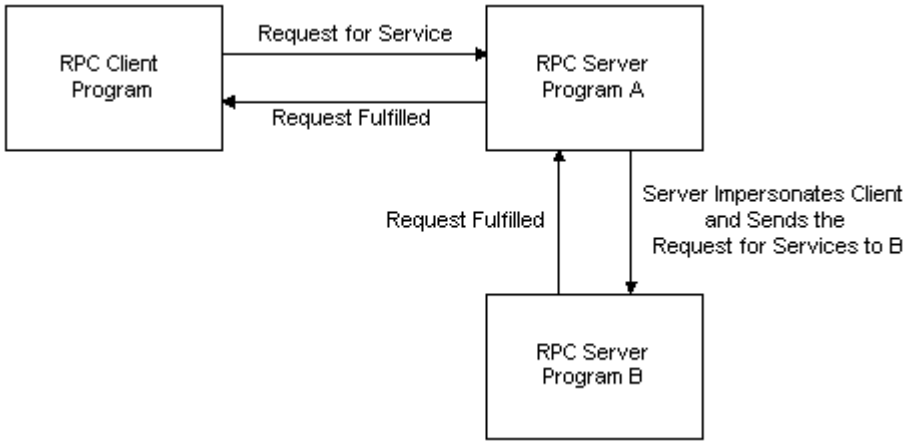

Рисунок 1 – Имперсонация в общем случае

3 РЕЗУЛЬТАТЫ РАБОТЫ

3.1 Разработка и компилирование интерфейса

Каркас интерфейса был получен выполнением команды:

```
uuidgen /i /ohello.idl
```

Получили следующий интерфейс:

```
[
    uuid(7a98c250-6808-11cf-b73b-00aa00b677a7),
    version(1.0)
]
interface INTERFACENAME
{

}
```

Рисунок 1 – Схема компиляции midl

Код конечного интерфейса находится в Приложении А.

Для компиляции интерфейса используется следующая команда midl:

```
midl /char ascii7 /app_config /nocpp Example1.idl
```

На выходе получили 3 файла.

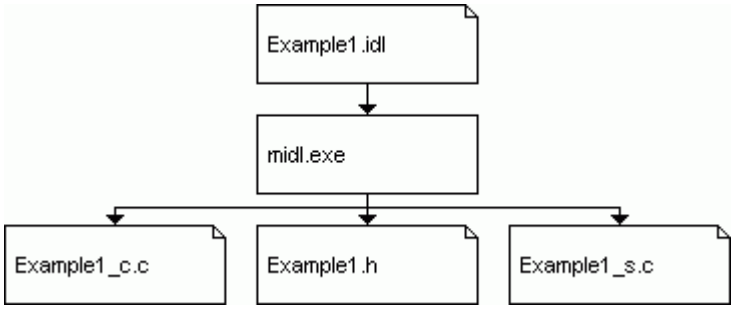

Рисунок 2 – Схема компиляции midl

3.2 Разработка сервера и клиента

При запуске, сервер сначала должен зарегистрировать поддерживаемые интерфейсы, затем настроиться на прослушку сети.

В заголовочном файле, полученном в результате компиляции midl, определены функции, поддерживаемое интерфейсом. Наша задача – описать их. Код сервера находится в Приложении Б.

Для того, чтобы начало работу клентское приложение, нужно настроить протокол, сеть, порт для соединения с сервером. Также нужно настроить(зарегистрировать) интерфейс, которым будет пользоваться приложение.

При вызове функций интерфейса на клиенте вызывается функция-заглушка, а при помощи RPC на сервере вызывается соответствующая реальная функция. Код клиента находится в Приложении В.

3.3 Компиляция

В ходе работы я решил компилировать все исходные файлы вручную из командной строки разработчика (Visual Studio Command Prompt). Для линковки нужно было подключать дополнительные библиотеки. Поскольку очень часто приходилось перекомпилировать код, я решил написать небольшой “Makefile”, который компилирует все от интерфейса с помощью midl, до линковки и создания исполняемых файлов. Исходный код в Приложении Г.

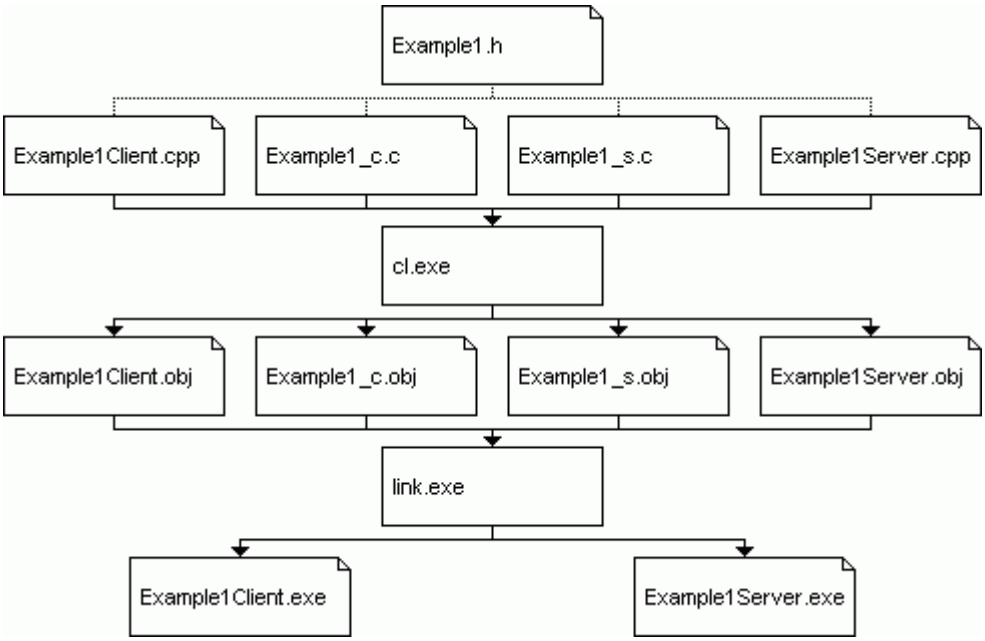

Рисунок 3 – Полная схема компилирования и линковки

4 ВЫВОД

Я познакомился с реализацией клиент-серверных программ с использованием RPC. Познакомился с механизмом имперсонации.

В основном примеры кода пришлось искать на github’e, а также смотреть спецификации на MSDN.

5 ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 В чем различия между statefull и stateless серверами?

Stateless сервер не сохранят состояния сессии. Т.е. каждый запрос к серверу обрабатывается независимо и не является частью старой или новой сессии. Statefull сервер же сохраняет состояния сессии, т.е. такой сервер может хранить данные о том, с какими данными был авторизирован пользователь, какие файлы открыты для чтения.

Главным плюсом stateless сервера является то, что если он упадет, то это не так сильно скажется на сервисе, предоставляемом пользователю, поскольку он не хранит состояний сессии и соответственно ему нечего терять.

5.2 Что такое имперсонация?

Имперсонация – это возможность процесса использовать контекст безопасности отличный от того, которым на самом деле владеет процесс. Обычно серверное приложение имперсонирует клиента. Это позволяет процессам действовать от имени клиента, чтобы получить доступ к объектам на сервере или подтвердить, что клиент является владельцем объекта.

5.3 Что такое LPC? чем этот механизм отличается от RPC?

LPC – недокументированный механизм, который предоставляет быстрое взаимодействие. RPC может использовать LPC как транспорт, когда сервер и клиент находятся на одном компьютере. Многие сервисы, работающие на одном компьютере используют только LPC для взаимодействия при работе с

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. http://www.codeproject.com/Articles/4837/Introduction-to-RPC-Part
2. http://stackoverflow.com/
3. https://github.com/
4. https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa376391(v=vs.85).aspx

ПРИЛОЖЕНИЕ А

```
[
        uuid(f7084abd-6f07-43a0-9ca8-544db30fc176),
        // This is version 1.0 of this interface.
        version(1.0),

        // This interface will use an implicit binding
        // handle named hExample1Binding.
        implicit_handle(handle_t hExample1Binding)
]
interface Example1 // The interface is named Example1
{
        // A function that takes a zero-terminated string.
        int LogIn(
                [in, string] const char * login,
                [in, string] const char * password);
        int LogOut();
        int CreateFileJP(
                [in, string] const char * file_name);
        int OpenFileJP(
                [in, string] const char * file_name);
        int Delete(
                [in, string] const char * file_name);
        int Download(
                [out] int * buffer);
        int Upload(
                [in] int buffer,
                [in] int type);
}
```

ПРИЛОЖЕНИЕ Б

```
// File Example1Server.cpp
#include <iostream>
#include "Example1.h"
#include <Windows.h>

#include <stdio.h>
#include <windows.h>
#include <tchar.h>
#include "accctrl.h"
#include "aclapi.h"

char * get_owner(const char * file_name);

bool login_status = false;
char login_actual[256] = "";
char * opened_file[256];
FILE *filik = NULL;

int download_size;

int LogIn(const char * login, const char * password)
{
    std::cout << login << " logged in" <<login_status << std::endl;
    if (login_status == true) return -1;

    BOOL status;

    HANDLE token;

    status = LogonUser(login, //
        ".",
        password, //
        LOGON32_LOGON_NETWORK,
        LOGON32_PROVIDER_DEFAULT,
        &token);

    if (!status)
    {
        std::cout << "LogonUser error: " << GetLastError() << std::endl;
        return GetLastError();
    }

    status = ImpersonateLoggedOnUser(token);
    if (!status)
    {
        std::cout << "Impersonation error: " << GetLastError() << std::endl;
        return GetLastError();
    }

    login_status = true;
    strcpy(login_actual,login);

    return 0;
}

int LogOut()
{
    BOOL status;
    status = RevertToSelf();
    if (!status)
    {
        std::cout << "RevertingToSelf error: " << GetLastError() << std::endl;
        return GetLastError();
    }
    std::cout << login_actual << " logged out!" << std::endl;
    login_status = false;
    strcpy(login_actual,"");
    return 0;
}

int CreateFileJP(const char * file_name)
{
    std::cout << "Creating file JP" << std::endl;
    if (login_status == false) return -1;

    FILE *f;
    f = fopen( file_name, "rb" );
```

```
    if ( f != 0 )
    {
        std::cout << "file exists" << std::endl;
        fclose( f );
        return -1;
    }else
    {
        std::cout << "file_not_exist" << std::endl;
        //fclose( f );
    }

    filik = fopen(file_name, "wb");
    if (!filik) // если есть доступ к файлу,
    {
      return -1;
    }
    //fclose(opened_file);
    return 0;
}

int OpenFileJP(const char * file_name)
{
    if (login_status == false) return -1;

    filik = fopen( file_name, "rb" );

    if ( filik != 0 )
    {
        std::cout << "file exists" << std::endl;
        fseek(filik, 0, SEEK_END);//перемещает указатель, соответствующий потоку hFile, на
новое место расположения отстоящее от SEEK_END на 0 байтов.
        download_size = ftell(filik);
        //if (file_size == -1) return -4;
        fseek(filik, 0, SEEK_SET);
    }
    else
    {
        std::cout << "file_not_exist" << std::endl;
        fclose( filik );
        return -1;
        //fclose( f );
    }
    //fclose(opened_file);
    return 0;
}

int Upload(int buffer,int type)
{
  /*FILE *out;
  out = fopen((char*)file_name,"ab");
  fwrite(buffer,sizeof(char),size,out);
  fclose(out);*/
  //FILE *hFile;
  //fopen_s(&filik, (char *)buffer, "ab");
  //printf("%x", buffer);

  /*for (int i = 0; i < size; i++)
    fprintf(hFile, "%c", buffer[i]);
  fclose(hFile); */

    if (type == 2)
        fclose(filik);
    else
        fputc(buffer, filik);
    return 0;

}

char * get_owner(const char * file_name)
{
    DWORD dwRtnCode = 0;
    PSID pSidOwner = NULL;
    BOOL bRtnBool = TRUE;
    LPTSTR AcctName = NULL;
    LPTSTR DomainName = NULL;
    DWORD dwAcctName = 1, dwDomainName = 1;
    SID_NAME_USE eUse = SidTypeUnknown;
    HANDLE hFile;
    PSECURITY_DESCRIPTOR pSD = NULL;
```

```
// Get the handle of the file object.
hFile = CreateFile(
                  TEXT(file_name),
                  GENERIC_READ,
                  FILE_SHARE_READ,
                  NULL,
                  OPEN_EXISTING,
                  SECURITY_ANONYMOUS,
                  NULL);

// Check GetLastError for CreateFile error code.
if (hFile == INVALID_HANDLE_VALUE) {
          DWORD dwErrorCode = 0;

          dwErrorCode = GetLastError();
          _tprintf(TEXT("CreateFile error = %d\n"), dwErrorCode);
          return NULL;
}



// Get the owner SID of the file.
dwRtnCode = GetSecurityInfo(
                  hFile,
                  SE_FILE_OBJECT,
                  OWNER_SECURITY_INFORMATION,
                  &pSidOwner,
                  NULL,
                  NULL,
                  NULL,
                  &pSD);

CloseHandle(hFile);
// Check GetLastError for GetSecurityInfo error condition.
if (dwRtnCode != ERROR_SUCCESS) {
          DWORD dwErrorCode = 0;

          dwErrorCode = GetLastError();
          _tprintf(TEXT("GetSecurityInfo error = %d\n"), dwErrorCode);
          return NULL;
}

// First call to LookupAccountSid to get the buffer sizes.
bRtnBool = LookupAccountSid(
                  NULL,           // local computer
                  pSidOwner,
                  AcctName,
                  (LPDWORD)&dwAcctName,
                  DomainName,
                  (LPDWORD)&dwDomainName,
                  &eUse);

// Reallocate memory for the buffers.
AcctName = (LPTSTR)GlobalAlloc(
          GMEM_FIXED,
          dwAcctName);

// Check GetLastError for GlobalAlloc error condition.
if (AcctName == NULL) {
          DWORD dwErrorCode = 0;

          dwErrorCode = GetLastError();
          _tprintf(TEXT("GlobalAlloc error = %d\n"), dwErrorCode);
          return NULL;
}

DomainName = (LPTSTR)GlobalAlloc(
       GMEM_FIXED,
       dwDomainName);

// Check GetLastError for GlobalAlloc error condition.
if (DomainName == NULL) {
      DWORD dwErrorCode = 0;

      dwErrorCode = GetLastError();
      _tprintf(TEXT("GlobalAlloc error = %d\n"), dwErrorCode);
      return NULL;
```

```
    }

    // Second call to LookupAccountSid to get the account name.
    bRtnBool = LookupAccountSid(
          NULL,                  // name of local or remote computer
          pSidOwner,             // security identifier
          AcctName,              // account name buffer
          (LPDWORD)&dwAcctName,  // size of account name buffer
          DomainName,            // domain name
          (LPDWORD)&dwDomainName, // size of domain name buffer
          &eUse);                // SID type

    // Check GetLastError for LookupAccountSid error condition.
    if (bRtnBool == FALSE) {
          DWORD dwErrorCode = 0;

          dwErrorCode = GetLastError();

          if (dwErrorCode == ERROR_NONE_MAPPED)
              _tprintf(TEXT
                   ("Account owner not found for specified SID.\n"));
          else
              _tprintf(TEXT("Error in LookupAccountSid.\n"));
          return NULL;

    } else if (bRtnBool == TRUE)

        // Print the account name.
        _tprintf(TEXT("Account owner = %s\n"), AcctName);

    return AcctName;
}

int Delete(const char * file_name)
{
    if (login_status == false) return -1;

    char * owner = get_owner(file_name);
    std::cout << "owner: " << owner << std::endl;
    std::cout << "login_actual: " << login_actual << std::endl;
    if (strcmp(owner, login_actual) == 0)
    {
        if(remove(file_name)) {
            return -2;
        }
        else return 0;
    }
    else return -3;

}

int Download(int * buffer)
{
    if (download_size == 0){
        fclose(filik);
        filik = NULL;
        return 1;
    }
    download_size--;
    *buffer = fgetc(filik);
    //std::cout << download_size << " ";
    if (*buffer != EOF) return 0;
    else
    {
        fclose(filik);
        filik = NULL;
        return 1;
    }
}

// Naive security callback.
RPC_STATUS CALLBACK SecurityCallback(RPC_IF_HANDLE /*hInterface*/, void* /*pBindingHandle*/)
{
    return RPC_S_OK; // Always allow anyone.
}

int main()
{
   RPC_STATUS status;
```

```
    // Uses the protocol combined with the endpoint for receiving
    // remote procedure calls.
    status = RpcServerUseProtseqEp(
       reinterpret_cast<unsigned char*>("ncacn_ip_tcp"), // Use TCP/IP protocol.
       RPC_C_PROTSEQ_MAX_REQS_DEFAULT, // Backlog queue length for TCP/IP.
       reinterpret_cast<unsigned char*>("4747"), // TCP/IP port to use.
       NULL); // No security.

    if (status)
       exit(status);

    // Registers the Example1 interface.
    status = RpcServerRegisterIf2(
       Example1_v1_0_s_ifspec, // Interface to register.
       NULL, // Use the MIDL generated entry-point vector.
       NULL, // Use the MIDL generated entry-point vector.
       RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH, // Forces use of security callback.
       RPC_C_LISTEN_MAX_CALLS_DEFAULT, // Use default number of concurrent calls.
       (unsigned)-1, // Infinite max size of incoming data blocks.
       SecurityCallback); // Naive security callback.

    if (status)
       exit(status);

    // Start to listen for remote procedure
    // calls for all registered interfaces.
    // This call will not return until
    // RpcMgmtStopServerListening is called.
    status = RpcServerListen(
      1, // Recommended minimum number of threads.
      RPC_C_LISTEN_MAX_CALLS_DEFAULT, // Recommended maximum number of threads.
      FALSE); // Start listening now.

    if (status)
       exit(status);
}

// Memory allocation function for RPC.
// The runtime uses these two functions for allocating/deallocating
// enough memory to pass the string to the server.
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

```
// File Example1Client.cpp
#include <iostream>
#include "Example1.h"
#include <string>
#include <cstdlib>

void main_activity();

int main()
{
   RPC_STATUS status;
   unsigned char* szStringBinding = NULL;

   // Creates a string binding handle.
   // This function is nothing more than a printf.
   // Connection is not done here.
   status = RpcStringBindingCompose(
      NULL, // UUID to bind to.
      reinterpret_cast<unsigned char*>("ncacn_ip_tcp"), // Use TCP/IP protocol.
      reinterpret_cast<unsigned char*>("192.168.116.140"), // TCP/IP network address to use.
      reinterpret_cast<unsigned char*>("4747"), // TCP/IP port to use.
      NULL, // Protocol dependent network options to use.
      &szStringBinding); // String binding output.

   if (status)
      exit(status);

   // Validates the format of the string binding handle and converts
   // it to a binding handle.
   // Connection is not done here either.
   status = RpcBindingFromStringBinding(
      szStringBinding, // The string binding to validate.
      &hExample1Binding); // Put the result in the implicit binding
                          // handle defined in the IDL file.

   if (status)
      exit(status);

   main_activity();

   // Free the memory allocated by a string.
   status = RpcStringFree(
      &szStringBinding); // String to be freed.

   if (status)
      exit(status);

   // Releases binding handle resources and disconnects from the server.
   status = RpcBindingFree(
      &hExample1Binding); // Frees the implicit binding handle defined in the IDL file.

   if (status)
      exit(status);
}

void _Upload(const char * file_name)
{
    if (CreateFileJP(file_name) != 0)
    {
        std::cout << "<<< Uploading Error!!! >>>" << std::endl;
        return;
    }

    FILE* hFile;
    if (fopen_s(&hFile, file_name, "rb"))
    {
        return;
    }

    fseek(hFile, 0, SEEK_END);//перемещает указатель, соответствующий потоку hFile, на новое
место расположения отстоящее от SEEK_END на 0 байтов.
    unsigned int file_size = ftell(hFile);
    //if (file_size == -1) return -4;
    fseek(hFile, 0, SEEK_SET);
    /*std::ifstream infile((char*)file_name, std::ios::binary);

    infile.seekg (0, std::ios::end);
```

```
    int file_size = infile.tellg();
    int blocks = file_size/block_size+1;
    infile.seekg (0, std::ios::beg);*/
    int s = 0;
    int cnt = 0;
    do
    {
        if (cnt == (int)(0.2 * file_size))
            std::cout << "20%" << std::endl;
        if (cnt == (int)(0.4 * file_size))
            std::cout << "40%" << std::endl;
        if (cnt == (int)(0.6 * file_size))
            std::cout << "60%" << std::endl;
        if (cnt == (int)(0.8 * file_size))
            std::cout << "80%" << std::endl;
        if (cnt == file_size)
            std::cout << "100%" << std::endl;
        cnt++;
        s = fgetc(hFile);
        Upload(s, 1);
    } while (cnt != file_size &&  s!= EOF);
    Upload(0, 2);

}

void _Login(const char * login, const char * password)
{
   RpcTryExcept
   {
      // std::cout << LogIn("japroc", "japroc") << std::endl;
      // std::cout << LogIn("acc_name", "acc_pass") << std::endl;
      if (LogIn(login, password) == 0)
         std::cout << "<<< Logged In >>>" << std::endl;
      else
         std::cout << "<<< LogIn Error!!! >>>" << std::endl;
   }
   RpcExcept(1)
   {
      std::cerr << "Runtime reported exception " << RpcExceptionCode()
                << std::endl;
   }
   RpcEndExcept
}

void _Logout()
{
   RpcTryExcept
   {
      // std::cout << LogIn("japroc", "japroc") << std::endl;
      // std::cout << LogIn("acc_name", "acc_pass") << std::endl;
      if (LogOut() == 0)
         std::cout << "<<< Logged Out >>>" << std::endl;
      else
         std::cout << "<<< Logging Out Error!!! >>>" << std::endl;
   }
   RpcExcept(1)
   {
      std::cerr << "Runtime reported exception " << RpcExceptionCode()
                << std::endl;
   }
   RpcEndExcept
}

void _Delete(const char * file_name)
{
   RpcTryExcept
   {
      // std::cout << LogIn("japroc", "japroc") << std::endl;
      // std::cout << LogIn("acc_name", "acc_pass") << std::endl;
      int x;
      x = Delete(file_name);
      //if ((int x = Delete(file_name)) == 0)
      if (x == 0)
         std::cout << "<<< File Deleted >>>" << std::endl;
      else if (x == -3)
         std::cout << "<<< Deleting Error!!! You are not owner of this file!!! >>>" << x <<
std::endl;
      else
         std::cout << "<<< File Deleting Error!!! >>>" << x << std::endl;
   }
```

```
   RpcExcept(1)
   {
      std::cerr << "Runtime reported exception " << RpcExceptionCode()
                << std::endl;
   }
   RpcEndExcept
}

void _Download(const char *file_name)
{

    if (OpenFileJP(file_name) != 0)
    {
        std::cout << "<<< Not authorized!!! >>>" << std::endl;
        return;
    }

    std::string nfn;
    std::cout << "Enter new file name: ";
    std::cin >> nfn;

    //return;
    FILE * file = fopen(nfn.c_str(), "wb");
    if (file) // если есть доступ к файлу,
    {
     // инициализируем строку
        int buffer;
        while (Download(&buffer) == 0)
            fputc(buffer, file);

        fclose(file);
    }
    else
    {
     std::cout << "<<< Cant Create File!!! Error!!! >>>" << std::endl;
    }

}

void main_activity()
{
   std::cout << "Valid commands: login, logout, upload, download, delete, exit" << std::endl;
   while(1)
   {
      std::string cmd;
      std::cout << ">>> ";
      std::cin >> cmd;
      //std::cout << std::endl;
      if (cmd.compare("login") == 0)
      {
         std::string login;
         std::string pass;
         std::cout << "Enter login: ";
         std::cin >> login;
         std::cout << "Enter password: ";
         std::cin >> pass;
         _Login(login.c_str(), pass.c_str());
      }
      else if (cmd.compare("logout") == 0)
      {
         _Logout();
      }
      else if (cmd.compare("upload") == 0)
      {
         std::string file_name;
         std::cout << "Enter file name: ";
         std::cin >> file_name;
         _Upload(file_name.c_str());
      }
      else if (cmd.compare("delete") == 0)
      {
         std::string file_name;
         std::cout << "Enter file name: ";
         std::cin >> file_name;
         _Delete(file_name.c_str());
      }
      else if (cmd.compare("download") == 0)
      {
         std::string file_name;
         std::cout << "Enter file name: ";
```

```
            std::cin >> file_name;
            _Download(file_name.c_str());
        }
        else if (cmd.compare("exit") == 0)
        {
            return;
        }
        else
        {
            std::cout << "Wrong cmd" << std::endl;
        }
    }
}

// Memory allocation function for RPC.
// The runtime uses these two functions for allocating/deallocating
// enough memory to pass the string to the server.
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 Г

```
cls
midl /char ascii7 /app_config /nocpp Example1.idl
cl /c Example1_c.c
cl /c Example1_s.c
cl /c /EHsc Example1Client.cpp
cl /c /EHsc Example1Server.cpp
LINK /OUT:client.exe Example1_c.obj Example1Client.obj Rpcrt4.lib
LINK /OUT:server.exe Example1_s.obj Example1Server.obj Rpcrt4.lib Advapi32.lib
```