*haЛабораторная работа 1*

**УДАЛЕННЫЙ ФАЙЛОВЫЙ МЕНЕДЖЕР С ИМПЕРСОНАЦИЕЙ КЛИЕНТА**

**Цель работы:**

Получить навыки работы с механизмом удаленного вызова процедур (RPC)

**Задача:**

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.

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 механизм имперсонации работает как при повышении, так и при понижении прав потока по отношению к процессу сервера.

**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 инсцинирующей разрыв стороной не будет являться клиент. Таким образом, сеанс связи между клиентом и сервером не прерывается. Если же описанного выше функционала сервера не наблюдается, сервер называется stateless сервером.

Для разработки RPC клиент-серверного приложения с использованием языков C/C++ нужно сначала разработать интерфейс, а после этого реализовать этот интерфейс с помощью сначала сервера, а потом клиента. Это необходимо для того, чтобы клиент и сервер взаимодействовали по одним и тем же наборам правил. Распространенной практикой является разработка серверного и клиентского проектов параллельно, но интерфейс всегда разрабатывается первым.

Упомянутый интерфейс RPC должен описывать функции, которые будет реализовывать клиент-серверное приложение. Интерфейс состоит из названия интерфейса, некоторых атрибутов, необязательных определений типов и констант и набора объявлений процедур-прототипов. Все параметры объявляемых процедур должны быть снабжены указателями направления передачи данных [in], [out] или [in,out] (данные передаются для удаленной процедуры, возвращаются удаленной процедурой, либо данные как передаются, так и возвращаются).

Интерфейсы определяются с использованием MIDL (Microsoft Interface Definition Language = Языка Определения Интерфейсов Microsoft). MIDL во многом напоминает C и С++. Определение интерфейса содержится в текстовом файле с расширением .idl – компилятор MIDL (который встроен в Visual Studio) генерирует файл заголовка, который программа будет использовать в исходных кодах клиентского и серверного приложения. Компилятор MIDL также создает два файла на языке C – заглушки, одна для сервера, другая для клиента (необходимо для компиляции файлов). Обычно в дополнение к компилятору MIDLнеобходимо использовать UUID (Universal Unique Identifier = Универсальный Уникальный Идентификатор) – это необходимо для того, чтобы каждый созданный интерфейс мог быть определен уникальным образом. Для генерации UUID обычно используется утилита Uuidgen. Выглядит UUID следующим образом:

ef349889-0b3c-12d1-79be-01244e8eabc3

При запуске сервер сначала должен зарегистрировать поддерживаемые интерфейсы –они обеспечивает диспетчеризацию запросов от клиентов к соответствующим процедурам сервера, однако, прежде чем серверное приложение сможет принимать запросы удаленных вызовов процедур, необходимо обеспечить доступ к нему через сеть. Чтобы это сделать, необходимо известить run-time RPC о своем желании принимать вызовы по одной или нескольким последовательностям протоколов. Выбор поддерживаемых серверным приложением последовательностей протоколов является важным выбором: различные последовательности протоколов обладают различными характеристиками. Сервера, которые ожидают локальные вызовы, должны использовать ncalrpc. Сервера, которые принимают вызовы по сети, должны использовать ncacn\_ip\_tcp. Большинство серверных программ используют все возможные последовательности протоколов, поддерживаемые сетью. После того, как серверное приложение зарегистрирует как минимум одну последовательность протокола, сервера, которые используют динамические конечные точки, должны создать информацию для связывания для каждой последовательности протокола, которую они используют. Сервер хранит информацию о связывании в векторе связывания, который он затем может экспортировать службе сопоставления конечных точек. Регистрация серверной программы в таблице соответствия конечных точек на хост-компьютере сервера позволяет клиентам определить, на какой конечной точке (обычно порт TCP/IP или именованный канал) "слушает" серверная программа. Помимо регистрации имен интерфейсов, серверное приложение может также зарегистрировать UUID-ы объектов в карте соответствия конечных точек. После того, как серверное приложение зарегистрирует свои интерфейсы, создаст необходимую информацию связывания и зарегистрирует свои конечные точки, оно может начать ожидать запросы от клиентских приложений. Сервер RPC использует один или несколько потоков, которые принимают клиентские запросы и доставляют их к процедурам, зарегистрированных интерфейсов. Чтобы установить сессию взаимодействия с сервером клиенту, который использует явные дескрипторы, необходимо создать дескриптор связывания.

Вызов удаленной процедуры осуществляется в исходном коде точно так же, как и вызов обычной локальной функции. Однако клиент не может сразу взять и вызвать функцию удаления файла – требуется регистрация клиента в системе. Для регистрации обычно создается login-функция, вызовом которой начинается «общение» клиента и сервера. При вызове этой функции клиент должен передать серверу свои учетные данные пользователя системы для имперсонизации.

Имперсонализация — это концепт безопасности присущий Windows, что позволяет серверному приложению временно «быть» [клиентом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) 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.

***Указания к выполнению***

В начале разработки RPC приложения необходимо разработать интерфейс взаимодействия сервера и клиента, а также присвоить ему UUID. После реализации всех вышеуказанных действий .idl файл интерфейса должен принять следующий вид:

[

uuid(ef349889-0b3c-12d1-79be-01244e8eabc3),

version(1.0)

]

interface InterfaceRPC

{

const unsigned short INT\_ARRAY\_LEN = 100;

void ProcedureOne(

[in] int agr1,

[out] int outArray[INT\_ARRAY\_LEN]

);

//Далее список других процедур

}

После удачного определения интерфейса и компиляции, на выходе будет 3 файла: InterfaceRPC\_s.c, InterfaceRPC\_c.c и InterfaceRPC.h – данные файлы должны быть подключены к проектам сервера и клиента соответственно (InterfaceRPC.h должен быть подключен к обоим проектам). После этого необходимо использовать регистрацию функций уже непосредственно у сервера. Для этого можно использовать функцию RpcServerRegisterIf:

RPC\_STATUS status;

status = RpcServerRegisterIf(MyInterface\_v1\_0\_s\_ifspec, NULL, NULL);

Первый аргумент функции RpcServerRegisterIf – это структура, которую генерирует компилятор MIDL на основе IDL файла и которая определяет интерфейс к серверу. Второй и третий аргументы – это UUID и вектор точек входа соответственно, они используются при множественной реализации одних и тех же процедур интерфейса. Однако лучше использовать функцию RpcServerRegisterIfEx – преимуществом этой функции является возможность указания функции обратного вызова (call-back) для обеспечения авторизации.

Для регистрации на всех последовательностях обычно используется функция RpcServerUseAllProtseqs, кроме этого могут быть использованы функции RpcServerUseProtseqEx, RpcServerUseProtseqEp или RpcServerUseProtseqEpEx. В данном примере показано использование ncacn\_ip\_tcp:

RPC\_STATUS status;

status = RpcServerUseProtseq(

L"ncacn\_ip\_tcp",

RPC\_C\_PROTSEQ\_MAX\_REQS\_DEFAULT, // зависит от выбранной последовательности протокола

NULL); // всегда NULL

Для получения вектора связывания для серверного приложения необходима функция RpcServerInqBindings, а для того, чтобы зарегистрировать себя в таблице соответствия конечных точек на хост-компьютере сервера, серверной программе необходимо использовать функцию RpcEpRegister. Примеры использования этих двух функций:

RPC\_STATUS status;

RPC\_BINDING\_VECTOR \*rpcBindingVector;

status = RpcServerInqBindings(&rpcBindingVector);

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

Для того, чтобы получать запросы к удаленным процедурам, серверная программа должна использовать функцию RpcServerListen:

RPC\_STATUS status;

status = RpcServerListen(1,RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT,0);

Для регистрации минимального числа потоков, максимального числа одновременно обслуживаемых клиентов используются первый и второй аргумент этой функции. После этого сервер должен начать функционировать.

Функции RpcStringBindingCompose, RpcBindingFromStringBinding и RpcStringFree используется клиентом для создания дескриптора связывания, после их вызовов run-time RPC найдет компьютер, на котором размещена серверная программа, и находит на нем конечные точки, на которых слушает серверная программа, направляя вызовы к ним.

Так как сервер осуществляет имперсонизацию клиента и работает в своем адресном пространстве от имени клиента, то он обладает правами, которыми обладает данный клиент. Применительно к файловому менеджеру это выглядит следующим образом:

- Клиент подключился с учетными данными User1 и паролем User1Super;

- Сервер проверил существование пользователя User1 с паролем User1Super, если пользователь существует, то с клиентом начинается работа, иначе клиент игнорируется;

- Клиент хочет скачать с сервера файл ForUserFile. Для этого файла указано, что User1 имеет к нему доступ – тогда клиент вызывает функцию на сервере для скачивания файла. Сервер, запомнивший клиента, начинает действовать от имени этого клиента, получает доступ к файлу и отправляет клиенту этот файл;

- Клиент хочет удалить файл SecretFileForAdmin.У пользователя User1 нет доступа к этому файлу, поэтому сервер, когда начнет действовать от имени клиента (от имени пользователя User1), получит запрет на доступ к файлу от операционной системы и вернет клиенту сообщение об ошибке.

После завершения сеанса связи клиент должен отключиться, для чего вызывается функция logout.

**Контрольные вопросы:**

1) В чем различия между statefull и stateless серверами?

2) Что такое имперсонация?

3) Что такое LPC? чем этот механизм отличается от RPC?