Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Системное программировнаие

Студент: Немкович А.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В

Минск 2025

**Лабораторная работа №4**

Взять готовый проект с сайта «diskstation.belstu.by:5001» по пути «Для\_студентов\_ФИТ\_БГТУ/ПРЕПОДАВАТЕЛИ/Бернацкий/Системное программирование - 3 курс ИСиТ/Лекции/Примеры для лекций/5/COM+CMake.7z».

Изучить проект и полностью разобраться со структурой и содержимым подпроектов.

**ComSampleClient**

**.clang.d**

Файл конфигурации для clangd, языка-сервера C++. Добавляют пути к заголовочным файлам из других частей COM-проекта, чтобы clangd мог правильно находить #include-ы и анализировать интерфейсы.

std=c++17 - Указывает использовать стандарт C++17

**Main.cpp**

* Настраивает среду COM для текущего потока с помощью CoInitializeEx, указывая желаемую потоковую модель (MTA или STA).
* CoCreateInstance для создания экземпляров двух разных COM-классов (CLSID\_CComServerTest и CLSID\_CComServiceTest) в различных контекстах активации (CLSCTX\_INPROC\_SERVER для DLL и CLSCTX\_LOCAL\_SERVER для EXE).
* Получает указатель на интерфейс IComTest (определенный в IComTest.idl) при создании объекта. Вызывает метод WhoAmI этого интерфейса и выводит результат.
* Использует Release() для указателей на интерфейсы, чтобы COM мог управлять временем жизни серверных объектов (уменьшать счетчик ссылок).
* Освобождает память, выделенную сервером для выходных параметров (CoTaskMemFree), и деинициализирует COM с помощью CoUninitialize.
* Он не реализует COM-объекты, а только использует их.

CoInitializeEx

CoCreateInstance

QueryInterface

Release

CoTaskMemFree

CoUninitialize

Execute

Execute(CLSID\_CComServerTest, COINIT\_MULTITHREADED, CLSCTX\_INPROC\_SERVER);

Execute(CLSID\_CComServerTest, COINIT\_APARTMENTTHREADED, CLSCTX\_INPROC\_SERVER);

Execute(CLSID\_CComServiceTest, COINIT\_MULTITHREADED, CLSCTX\_LOCAL\_SERVER);

Execute(CLSID\_CComServiceTest, COINIT\_APARTMENTTHREADED, CLSCTX\_LOCAL\_SERVER);

* CLSID\_CComServerTest: Тестируется COM-класс CComServerTest.
* COINIT\_MULTITHREADED: Поток клиента, вызывающий Execute, инициализируется для работы в многопоточном апартаменте (MTA). Это значит, что вызовы методов COM-объекта из этого потока могут происходить одновременно с вызовами из других MTA-потоков (если они есть), и синхронизация доступа к данным объекта (если он не потокобезопасен) ложится на сам объект или на клиента.
* COINIT\_APARTMENTTHREADED: Поток клиента инициализируется для работы в однопоточном апартаменте (STA). COM гарантирует, что все вызовы методов объекта, связанного с этим STA, будут выполняться последовательно только в этом потоке. Если другие потоки (из MTA или других STA) попытаются вызвать методы этого объекта, COM будет маршалить вызовы и ставить их в очередь сообщений потока STA.
* CLSCTX\_INPROC\_SERVER: COM ищет реализацию CComServerTest в DLL-библиотеке, которая будет загружена в тот же процесс, что и клиент (ComClient.exe). Взаимодействие происходит напрямую через указатели (с возможным маршалингом между апартаментами, если потоковая модель сервера не совпадает с MTA).
* CLSCTX\_LOCAL\_SERVER: COM ищет реализацию CComServiceTest в отдельном EXE-файле. Этот EXE-сервер будет запущен (если еще не запущен) в собственном процессе. Взаимодействие между клиентом (ComClient.exe) и сервером (ComService.exe, например) происходит через межпроцессное взаимодействие (IPC), управляемое COM с помощью маршалинга (упаковки/распаковки вызовов и данных через прокси и заглушки).

**ComSampleProxy**

**dlldata.c**

Содержит данные и код, необходимые для сборки DLL прокси/заглушки из \_p.c и \_i.c. Реализация стандартных экспортных функций COM DLL (DllMain, DllGetClassObject, DllCanUnloadNow, DllRegisterServer, DllUnregisterServer), но адаптированных для регистрации прокси/заглушек, а не самих COM-объектов. Компилируется и линкуется вместе с IComTest\_p.c и IComTest\_i.c для создания окончательной Proxy/Stub DLL.

**IComTest.def**

Файл определения модуля. Используется линковщиком при сборке DLL (как основной DLL сервера, так и DLL прокси/заглушки). Указывает имя DLL и, самое главное, перечисляет функции, которые DLL экспортирует. Для Proxy/Stub DLL это будут DllGetClassObject, DllCanUnloadNow и т.д. Для основной DLL COM-сервера это будут те же функции, но их реализация будет связана с созданием экземпляров ваших COM-классов.

**IComTest.idl**

(Interface Definition Language) Это исходный файл, написанный на языке описания интерфейсов (IDL). Он является контрактом или спецификацией для COM-интерфейса IComTest.

Определяет сам интерфейс IComTest, его уникальный идентификатор (uuid), и то, что он наследуется от IUnknown.

Определяет методы интерфейса, их параметры и тип возвращаемого значения. Этот файл обрабатывается компилятором MIDL (Microsoft IDL Compiler) для генерации нескольких других файлов, необходимых для реализации и использования интерфейса.

**IComTest.tlb**

(Type Library) Библиотека типов. Это скомпилированная, двоичная версия информации из IDL-файла. Содержит метаданные об интерфейсе IComTest, его методах, параметрах, UUID и т. д. в формате, понятном другим приложениям и инструментам разработки. Используется клиентами, написанными на языках высокого уровня, для вызова методов COM-объекта.

**Comtest\_h.h**

Заголовочный файл C/C++. Содержит объявления для использования интерфейса в C++ коде. Определение структуры интерфейса IComTest в стиле C++ (на основе таблицы виртуальных функций - vtable).

Объявление extern const IID IID\_IComTest; (идентификатор интерфейса).

Если в IDL был определен coclass, здесь будет объявление extern const CLSID CLSID\_MyComObject; (идентификатор класса).

Включается (#include) как в код сервера (для реализации интерфейса), так и в код клиента (для вызова методов интерфейса). Гарантирует, что и клиент, и сервер "понимают" структуру интерфейса одинаково.

**IComTest\_i.c**

Файл C, содержащий определения идентификаторов GUID (UUID). Этот файл должен быть скомпилирован и прилинкован к проекту (обычно к серверу и/или клиенту) один раз. Это предоставляет фактическую память для хранения констант GUID. Включение заголовочного файла \_h.h дает только объявления, а линковка этого .c файла дает определения, чтобы избежать ошибок линковщика "unresolved external symbol".

**IComTest\_p.c**

Содержит исходный код на C для прокси и заглушки для интерфейса IComTest. Это код маршалинга.

Функции, которые упаковывают параметры методов IComTest для передачи между процессами и распаковывают их на принимающей стороне.

Компилируется и линкуется для создания отдельной DLL прокси/заглушки (Proxy/Stub DLL). Эта DLL используется средой COM автоматически, когда клиент в одном процессе вызывает метод объекта, находящегося в другом процессе. Прокси работает в процессе клиента, заглушка - в процессе сервера.

**ComSampleServer**

**CComServerTest.cpp**

* Класс CComServerTest наследует от IComTest. Это означает, что он предоставляет конкретную реализацию для всех методов, объявленных в этих интерфейсах (а также для методов базового интерфейса IUnknown).
* Класс реализует методы AddRef и Release интерфейса IUnknown. Он использует внутренний счетчик ссылок (\_cRef), который увеличивается при вызове AddRef и уменьшается при вызове Release. Когда счетчик ссылок достигает нуля в методе Release, объект удаляет сам себя (delete this).
* Используются потокобезопасные функции InterlockedIncrement и InterlockedDecrement для управления счетчиком ссылок, что делает объект безопасным для использования из разных потоков (если его внутреннее состояние также защищено при необходимости).
* Реализация QueryInterface позволяет клиенту запросить у экземпляра CComServerTest указатель на любой из поддерживаемых им интерфейсов. Если запрошенный интерфейс поддерживается, QueryInterface возвращает указатель на него и увеличивает счетчик ссылок объекта (AddRef). Если интерфейс не поддерживается, возвращается ошибка E\_NOINTERFACE.
* Конструктор и деструктор класса управляют глобальным счетчиком ссылок DLL (g\_cRefDll). Это необходимо для того, чтобы COM-сервер в виде DLL мог определить, можно ли его безопасно выгрузить из памяти (с помощью функции DllCanUnloadNow, которая проверяет этот счетчик и счетчик блокировок).
* Фабричная Функция (CComServerTest\_CreateInstance): которая инкапсулирует процесс создания нового экземпляра CComServerTest с помощью оператора new. Она выполняет необходимый начальный вызов QueryInterface и Release, чтобы корректно передать владение объектом (одну ссылку) вызывающей стороне (обычно это метод CreateInstance фабрики классов COM).

**ClassFactory.cpp**

* Этот файл реализует универсальную фабрику классов COM (CClassFactory). В COM каждый регистрируемый класс должен предоставлять "фабрику" (объект, реализующий IClassFactory), которая умеет создавать экземпляры этого класса по запросу (IClassFactory::CreateInstance). Вместо того чтобы писать отдельный класс фабрики для каждого COM-класса в сервере, этот CClassFactory спроектирован так, чтобы его можно было использовать для создания экземпляров любого COM-класса.
* Фабрика CClassFactory при своем создании получает в конструкторе указатель на функцию (PFNCREATEINSTANCE \_pfnCreateInstance). Этот указатель (\_pfnCreateInstance) указывает на реальную функцию, которая знает, как создать экземпляр конкретного COM-класса (например, CComServerTest\_CreateInstance). Когда COM вызывает метод CClassFactory::CreateInstance, эта фабрика просто передает вызов той функции, указатель на которую она хранит (hr = \_pfnCreateInstance(riid, ppv);).
* Класс реализует все методы IClassFactory: AddRef, Release, QueryInterface.
* LockServer: Управляет глобальным счетчиком блокировок сервера (g\_cServerLocks), позволяя клиентам удерживать сервер в памяти.
* CreateInstance: Делегирует создание экземпляра целевого COM-класса внешней функции. Также содержит проверку на неподдерживаемую агрегацию (CLASS\_E\_NOAGGREGATION).
* Конструктор и деструктор CClassFactory управляют глобальным счетчиком ссылок DLL (g\_cRefDll), как и сами COM-объекты. Это гарантирует, что DLL не выгрузится, пока существуют объекты фабрик.
* Фабрика для Фабрики (CClassFactory\_CreateInstance): Это вспомогательная функция для создания экземпляров самой CClassFactory. Она обычно вызывается из экспортной функции DllGetClassObject COM-сервера (DLL). Когда COM запрашивает у DLL фабрику для определенного CLSID, DllGetClassObject вызывает CClassFactory\_CreateInstance, передавая ей указатель на функцию создания именно того COM-класса, который соответствует запрошенному CLSID.

**ClassFactory.h**

* Определяет Тип Указателя на Функцию (PFNCREATEINSTANCE):Он стандартизирует сигнатуру для всех функций, отвечающих за создание экземпляров конкретных COM-классов.
* Объявляет Функцию Создания Фабрики (CClassFactory\_CreateInstance)

**ComSampleServer.def**

* Этот файл является файлом определения модуля для компоновщика. Его основная цель — указать, какие функции должны быть экспортированы из создаваемой DLL (ComSampleServer.dll), чтобы они были доступны для вызова внешним кодом.
* Перечисляет четыре стандартные функции (DllGetClassObject, DllCanUnloadNow, DllRegisterServer, DllUnregisterServer), которые требуются COM и утилитам регистрации для взаимодействия с DLL-сервером.
* Использование ключевого слова PRIVATE для всех экспортов предотвращает включение этих имен функций в файл .lib, генерируемый вместе с .dll.

**ComSampleServerCreateinstances.h**

Он предоставляет необходимый прототип функции создания экземпляра (CComServerTest\_CreateInstance), чтобы универсальная фабрика могла быть настроена для создания объектов именно этого класса.

**ComSampleServerGuids.h**

Определение CLSID\_CComServerTest. CLSID — это уникальный идентификатор, который COM использует для нахождения и активации конкретного COM-класса.

**Dll. Cpp**

Этот файл является основным модулем COM DLL сервера. Он реализует все стандартные точки входа, необходимые для взаимодействия DLL с COM-подсистемой, операционной системой и утилитами регистрации (regsvr32.exe).

* DllGetClassObject: Предоставляет COM объект фабрики классов (IClassFactory) для указанного CLSID. Это функция для создания экземпляров COM-объектов из этой DLL. Ищет запрошенный CLSID в глобальной таблице g\_Classes. Если находит, использует универсальную фабрику (CClassFactory\_CreateInstance), передавая ей указатель на функцию создания конкретного класса из таблицы.
* DllCanUnloadNow: Позволяет COM определить, можно ли безопасно выгрузить DLL из памяти процесса. Проверяет два глобальных счетчика: g\_cServerLocks (счетчик блокировок IClassFactory::LockServer) и g\_cRefDll (общий счетчик ссылок на все объекты и фабрики в DLL). Возвращает S\_OK (можно выгружать) только если оба счетчика равны нулю.
* DllRegisterServer: Выполняет саморегистрацию COM-сервера в системном реестре Windows. Вызывается regsvr32.exe. Итерирует по таблице g\_Classes. Для каждого класса вызывает вспомогательную функцию \_RegisterClass, которая создает необходимые записи под HKEY\_CLASSES\_ROOT\CLSID\{...} (включая InprocServer32, ThreadingModel, AppID). Затем вызывает \_RegisterAppId для создания записей HKEY\_CLASSES\_ROOT\AppID\{...}, необходимых для активации через суррогат (DllHost.exe).
* DllUnregisterServer: Удаляет все записи из реестра, созданные DllRegisterServer. Вызывается regsvr32.exe /u. Итерирует по таблице g\_Classes, вызывая \_UnregisterClass для удаления соответствующего ключа CLSID. Затем вызывает \_UnregisterAppId для удаления ключа AppID. Использует SHDeleteKeyW для рекурсивного удаления ключей.
* DllMain: Вызывается ОС при загрузке/выгрузке DLL и при создании/завершении потоков. При загрузке DLL (DLL\_PROCESS\_ATTACH) сохраняет хэндл модуля (HINSTANCE) в глобальную переменную g\_hInstance (нужен для GetModuleFileNameW) и отключает ненужные уведомления о потоках с помощью DisableThreadLibraryCalls.

**DIl. H**

Файл объявляет две глобальные переменные, важные для управления жизненным циклом COM DLL:

g\_cRefDll: Общий счетчик ссылок для всей DLL. Он увеличивается при создании любого COM-объекта или фабрики классов в этой DLL и уменьшается при их освобождении (Release). Функция DllCanUnloadNow проверяет этот счетчик.

g\_cServerLocks: Счетчик блокировок сервера. Он увеличивается/уменьшается вызовами IClassFactory::LockServer. Функция DllCanUnloadNow также проверяет этот счетчик.

**pch.h**

Общий заголовочный файл.

* windows.h: Базовые типы и функции Windows.
* ole2.h: Фундаментальные определения COM (интерфейсы, функции API, HRESULT).
* olectl.h: Содержит определения, полезные для COM, включая коды ошибок регистрации.
* specstrings.h: Аннотации SAL для статического анализа кода.
* intsafe.h: Безопасные операции с целыми числами.
* strsafe.h: Безопасные строковые функции.
* shlwapi.h: Вспомогательные функции оболочки, включая работу с реестром и QISearch.

**ComSampleService**

**CComServiceTest.cpp**

Реализация COM-класса CComServiceTest. Ключевые отличия от CComServerTest.cpp:

* Нет управления g\_cRefDll: Конструктор и деструктор CComServiceTest не изменяют глобальный счетчик ссылок DLL.
* Используется CComServiceTest\_CreateInstance, объявленная в ComSampleServiceCreateInstances.h.

**ClassFactory.cpp**

Реализует универсальную фабрику классов (CClassFactory), но с ключевыми адаптациями для использования в out-of-process сервере.

LockServer: Реализация этого метода возвращает S\_OK и ничего не делает. В отличие от DLL, где LockServer управляет глобальным счетчиком g\_cServerLocks, влияющим на DllCanUnloadNow, в EXE-сервере время жизни процесса обычно определяется наличием активных объектов и фабрик и внешние вызовы LockServer не должны на это влиять стандартным образом.

Конструктор и деструктор CClassFactory не вызывает InterlockedIncrement(&g\_cRefDll). В контексте EXE-сервера нет необходимости управлять глобальным счетчиком ссылок DLL для функции DllCanUnloadNow.

**ClassFactory.h**

• Определяет Тип Указателя на Функцию (PFNCREATEINSTANCE)

• Объявляет Функцию Создания Фабрики (CClassFactory\_CreateInstance)

**ComSampleServiceCreateinstances.h**

Предоставляет необходимый прототип функции создания экземпляра

**Com Sample ServiceGuids.h**

Определение CLSID\_CComServiceTest

**Infrastructure.h**

Этот заголовочный файл определяет интерфейс и константы для реализации Windows-службы, которая одновременно является out-of-process COM-сервером. Он объявляет функции, необходимые для:

* Регистрация/разрегистрация самой службы в Service Control Manager (SCM) (RegisterService, UnregisterService).
* Регистрация/разрегистрация COM-классов (RegisterClass, UnregisterClass).
* Регистрация/разрегистрация AppID (RegisterAppID, UnregisterAppID), которая связывает COM-активацию с запуском этой конкретной службы Windows (через параметр pszServiceName в RegisterAppID, который записывается в значение LocalService под ключом HKCR\AppID\{...}).
* Запуск (StartFactories) и остановка (StopFactories) фабрик классов. Это необходимо, чтобы COM-сервер мог принимать запросы на создание объектов только тогда, когда служба полностью инициализирована и готова к работе, и прекращал принимать новые запросы перед остановкой.

**Main. Cpp**

Анализирует аргументы, с которыми был запущен EXE и определяет режим работы.

1. /RegisterServer: Вызывает ExeRegisterServer для полной регистрации службы и COM-компонентов.
2. /UnregisterServer: Вызывает ExeUnregisterServer для полного удаления регистрации.
3. /Embedding: Сигнализирует, что запуск инициирован COM для активации объекта. Переходит к запуску службы.
4. Без аргументов: Предполагается запуск службой (SCM) или пользователем напрямую. Переходит к запуску службы.

Инициация Запуска Службы (wmain -> StartServiceCtrlDispatcher): Если определено, что нужно запускать службу, он подготавливает SERVICE\_TABLE\_ENTRY. Вызывает StartServiceCtrlDispatcher, передавая управление Service Control Manager (SCM). Эта функция не возвращается до тех пор, пока служба не остановится. SCM затем вызовет функцию ServiceMain, которая содержит основную логику работы службы.

StartFactories: Создает экземпляры универсальной фабрики классов (CClassFactory) для каждого COM-класса, определенного в таблице g\_Classes, и регистрирует их в COM с помощью CoRegisterClassObject. Это делает классы доступными для создания клиентами (CoCreateInstance). Указатели на фабрики и cookie регистрации сохраняются в g\_Classes. Эта функция должна вызываться из ServiceMain после успешной инициализации службы.

StopFactories: Отзывает регистрацию фабрик с помощью CoRevokeClassObject (используя сохраненные cookie) и освобождает сами объекты фабрик (Release). Это прекращает возможность создания новых объектов. Эта функция должна вызываться из ServiceMain перед остановкой службы.

ExeRegisterServer, ExeUnregisterServer: Эти функции действуют как высокоуровневые обертки, вызывая соответствующие функции для регистрации/разрегистрации службы (RegisterService/UnregisterService) и COM-компонентов (RegisterAppID/UnregisterAppID, RegisterClass/UnregisterClass). Они обеспечивают правильный порядок операций и базовую обработку ошибок с попыткой отката при сбое регистрации.

**Register.cpp**

Реализации функций, отвечающих за взаимодействие с системным реестром Windows и Service Control Manager (SCM) для регистрации и разрегистрации COM-сервера, работающего как служба Windows под учетной записью LocalSystem.

RegisterClass(clsid, name, appid) / UnregisterClass(clsid): Создают/удаляют записи в HKEY\_CLASSES\_ROOT\CLSID\{...}. Для службы не создаются ключи InprocServer32 или LocalServer32. Вместо этого в ключе CLSID\{...} создается значение AppID, которое ссылается на AppID службы.

RegisterAppID(appid, name, serviceName) / UnregisterAppID(appid): Создают/удаляют записи в HKEY\_CLASSES\_ROOT\AppID\{...}. При регистрации AppID создается строковое значение LocalService, в которое записывается внутреннее имя службы (serviceName).

RegisterService(serviceName, displayName) / UnregisterService(serviceName): Используют API SCM (OpenSCManager, CreateService, OpenService, DeleteService, CloseServiceHandle) для создания или удаления регистрации самой службы Windows. При регистрации (CreateService) явно указывается путь к текущему EXE-файлу и учетная запись LocalSystem (путем передачи nullptr в качестве lpServiceStartName).

Обрабатывают случай, когда служба уже существует при регистрации (ERROR\_SERVICE\_EXISTS -> S\_FALSE) или не существует при удалении (ERROR\_SERVICE\_DOES\_NOT\_EXIST -> S\_OK).

**Service.cpp**

Логика реализации Windows-службы, которая функционирует как COM-сервер. Он использует класс CMpcService для инкапсуляции состояния и поведения службы.

CMpcService Класс:

* Хранит состояние службы (\_serviceStatus, \_hServiceStatus).
* Управляет событием остановки (\_hServiceStoppingEvent).
* Предоставляет методы для запуска (\_StartService), остановки (\_StopService) и обработки команд (ServiceHandler).
* Использует критическую секцию (\_csLock) для защиты своего состояния от одновременного доступа из потока ServiceMain и потока(ов) SCM, вызывающих ServiceHandler.

ServiceMain (глобальная функция):

* Точка входа для SCM. Вызывается SCM после StartServiceCtrlDispatcher.
* Создает единственный экземпляр CMpcService (g\_pSvc).
* Вызывает метод g\_pSvc->ServiceMain(), который содержит основной цикл работы службы.
* Удаляет экземпляр CMpcService после завершения g\_pSvc->ServiceMain().

CMpcService::s\_ServiceHandler: Точка входа для обработчика команд SCM. Адрес этой функции регистрируется с помощью RegisterServiceCtrlHandlerEx. Просто перенаправляет вызов в метод ServiceHandler текущего экземпляра g\_pSvc.

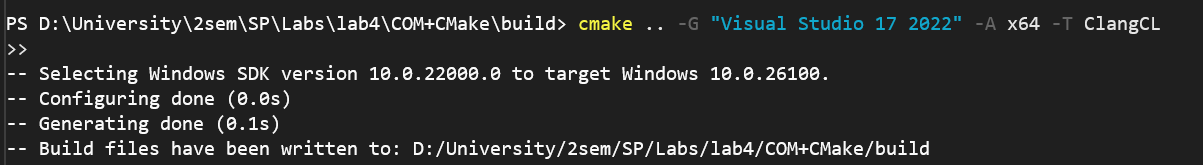
CMpcService::ServiceMain (метод экземпляра): Инициализирует COM (CoInitializeEx, CoInitializeSecurity). Вызывает \_StartService для выполнения начальной настройки (регистрация обработчика, запуск COM-фабрик через StartFactories). Блокируется на WaitForSingleObject, ожидая события \_hServiceStoppingEvent. После получения сигнала об остановке вызывает \_StopService для очистки (остановка COM-фабрик через StopFactories). Деинициализирует COM (CoUninitialize).

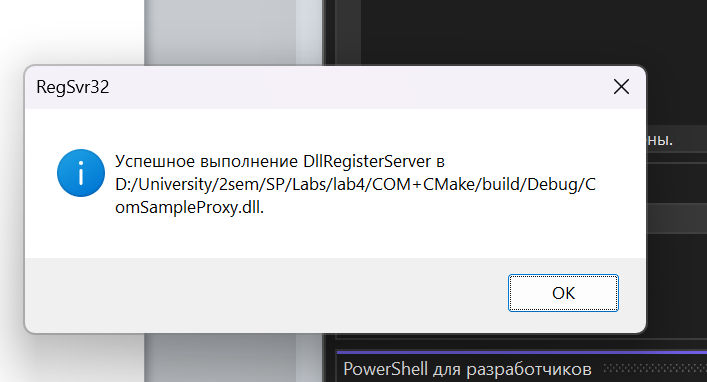
CMpcService::ServiceHandler (метод экземпляра): Обрабатывает команды от SCM. При получении SERVICE\_CONTROL\_STOP, устанавливает статус SERVICE\_STOP\_PENDING и сигнализирует событие \_hServiceStoppingEvent, что разблокирует основной поток в CMpcService::ServiceMain.

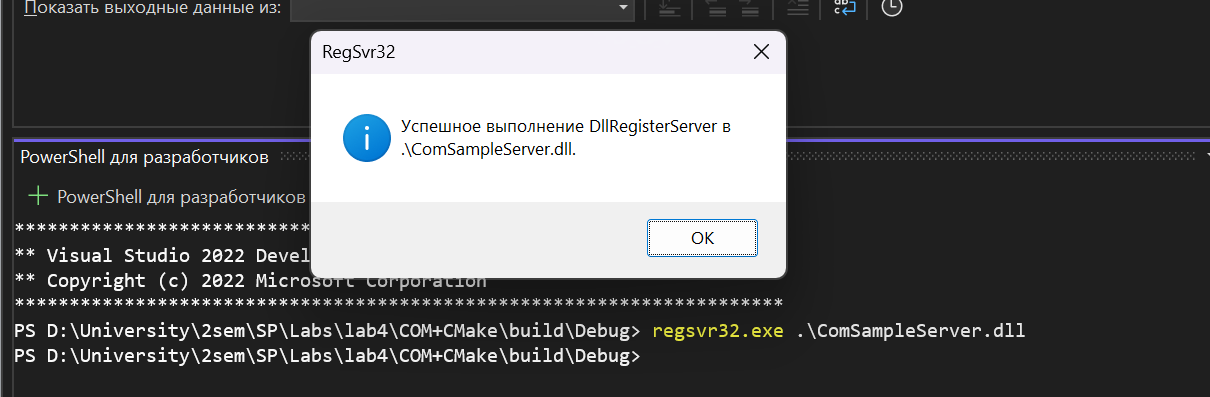
Изменить код всех проектов в необходимом объёме, а именно добавить новый интерфейс I<Ваши инициалы> который будет содержать вашу функцию из лабораторной работы №1. Если функция была написана без использования параметров (т. е., например, данные читались с консоли напрямую в функции), то обязательно параметризировать вашу функцию.

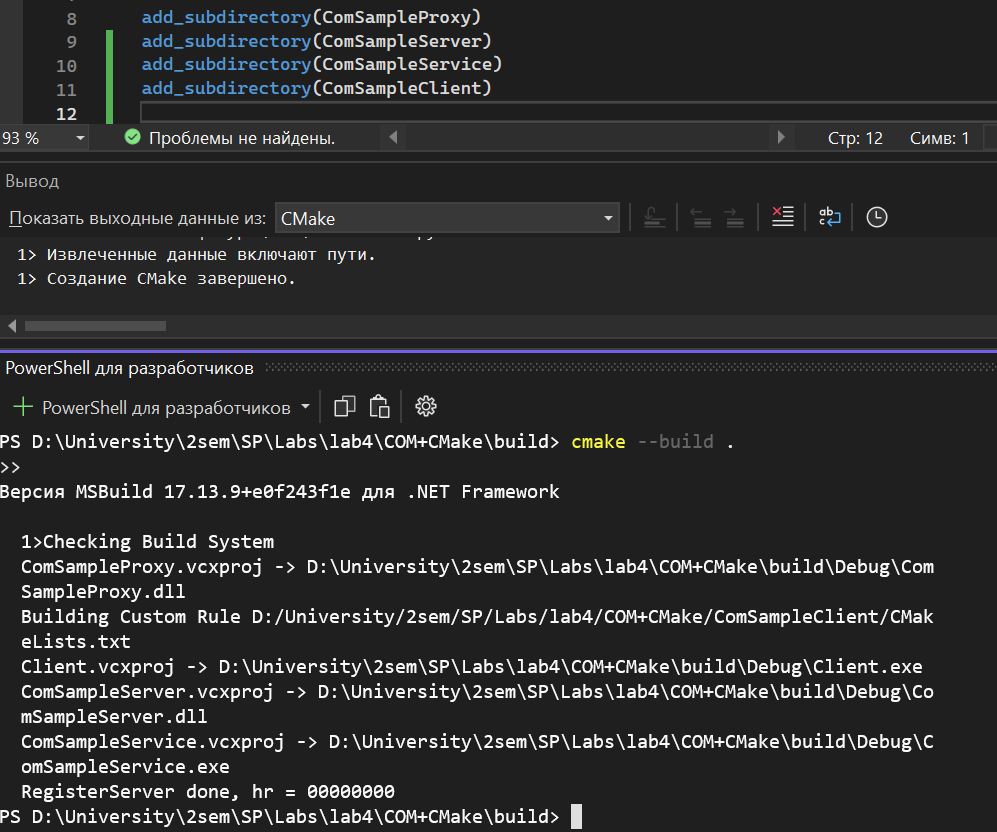
Собрать проект с помощью CMake в связке с Clang++ и продемонстрировать работоспособность получившегося проекта.

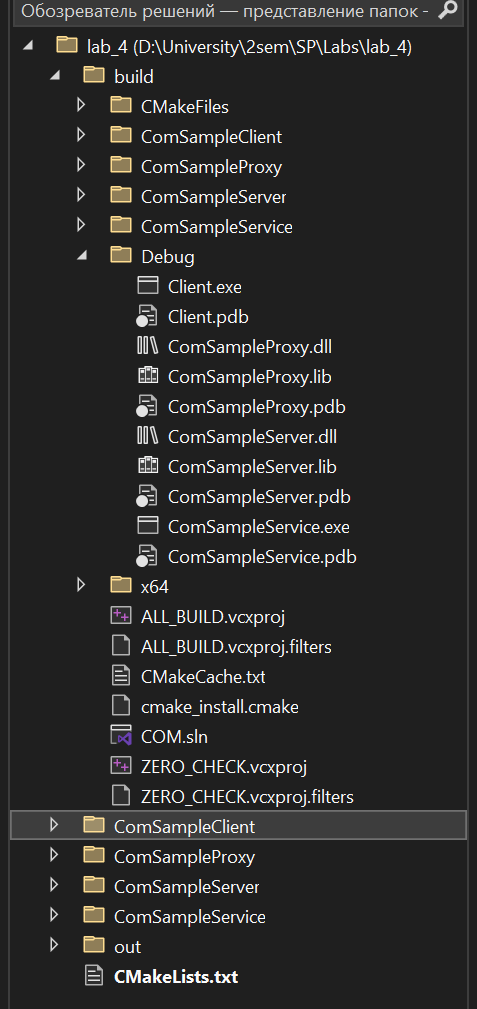
Запуск от имени администратора

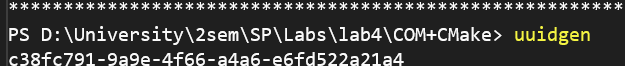


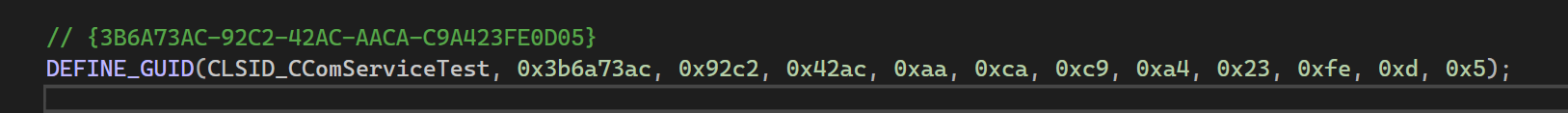


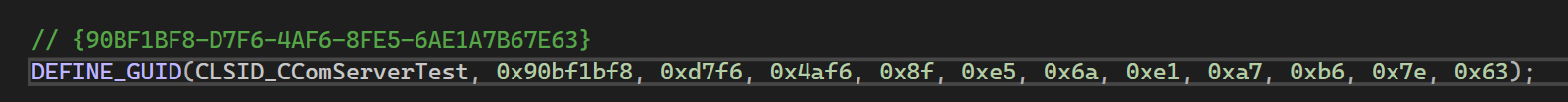






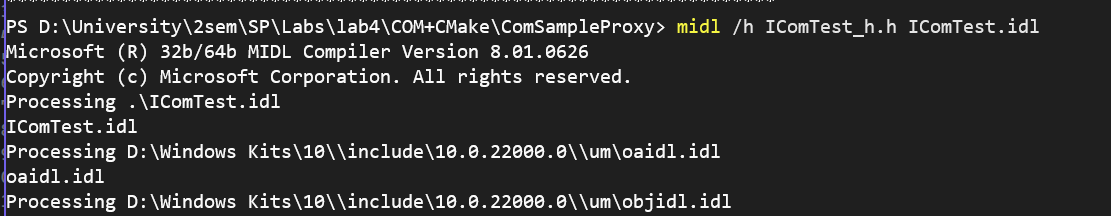


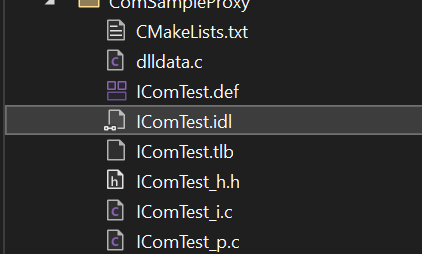


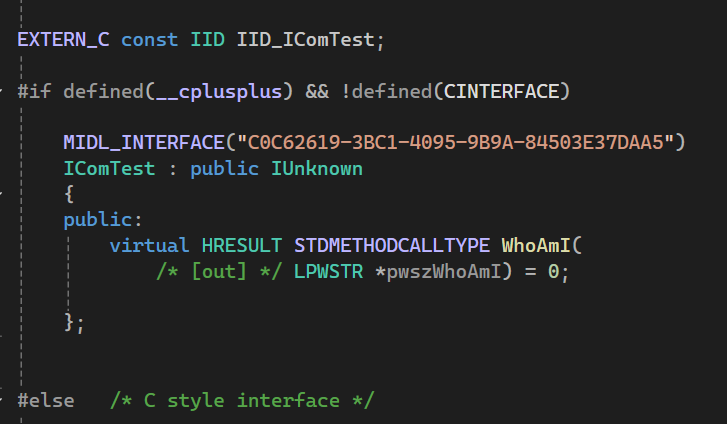


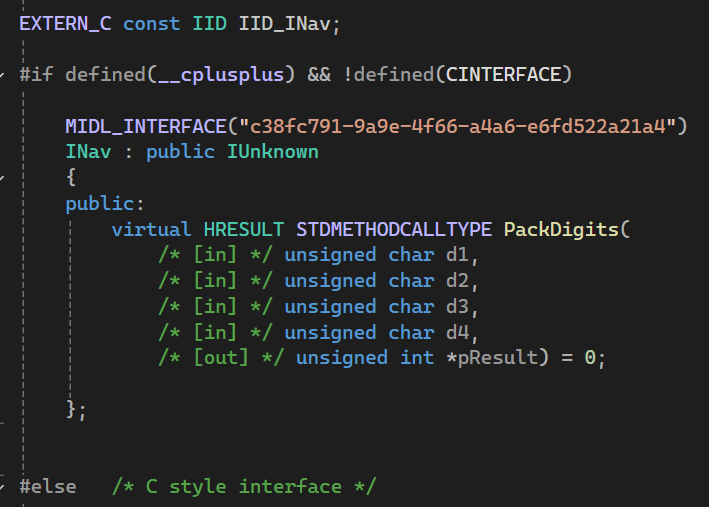
**IComTest.idl**

|  |
| --- |
| import "oaidl.idl";  import "ocidl.idl";  [  object,  uuid(C0C62619-3BC1-4095-9B9A-84503E37DAA5),  oleautomation, pointer\_default(unique),  helpstring("IComTest interface")  ]  interface IComTest : IUnknown  {  HRESULT WhoAmI([out] LPWSTR\* pwszWhoAmI);  };  [  object,  uuid(c38fc791-9a9e-4f66-a4a6-e6fd522a21a4),  oleautomation, pointer\_default(unique),  helpstring("INav interface")  ]  interface INav : IUnknown  {  HRESULT PackDigits(  [in] unsigned char d1, [in] unsigned char d2,  [in] unsigned char d3, [in] unsigned char d4,  [out] unsigned int\* pResult  );  }; |









**CComServiceTest.cpp**

|  |
| --- |
| #include "pch.h"  #pragma hdrstop  #include <objbase.h>  #include <shlwapi.h>  #include <assert.h>  #include <IComTest\_h.h>  #include "ComSampleServiceGuids.h"  #include "ComSampleServiceCreateInstances.h"  class CComServiceTest : public IComTest, public INav  {  public:  // IUnknown  IFACEMETHODIMP\_(ULONG) AddRef()  {  return InterlockedIncrement(&\_cRef);  }  IFACEMETHODIMP\_(ULONG) Release()  {  assert(\_cRef > 0);  ULONG cRef = InterlockedDecrement(&\_cRef);  if (0 == cRef)  {  delete this;  }  return cRef;  }  IFACEMETHODIMP CComServiceTest::QueryInterface(\_\_in REFIID riid, \_\_out void\*\* ppv)  {  static const QITAB qit[] =  {  QITABENT(CComServiceTest, IComTest),  QITABENT(CComServiceTest, INav),  { 0 },  };  return QISearch(this, qit, riid, ppv);  }  IFACEMETHODIMP WhoAmI(\_Out\_ LPWSTR\* ppwszWhoAmI)  {  HRESULT hr = (ppwszWhoAmI != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr))  {  // Create a message string the contains the name of the current process.  wchar\_t wszProcessName[MAX\_PATH] = {};  DWORD dwResult = GetModuleFileNameW(nullptr, wszProcessName, ARRAYSIZE(wszProcessName));  hr = ((dwResult < ARRAYSIZE(wszProcessName)) && (dwResult != 0)) ? S\_OK : HRESULT\_FROM\_WIN32(GetLastError());  if (SUCCEEDED(hr))  {  LPCWSTR pwszMessagePreface = L"Com Test Service running in process ";  // Prepare the message.  size\_t cchMessagePlusTerminatingNul = wcslen(pwszMessagePreface) + wcslen(wszProcessName) + 1 /\* For the terminating NUL character \*/;  wchar\_t\* pwszMessage = (wchar\_t\*)CoTaskMemAlloc(cchMessagePlusTerminatingNul \* sizeof(wchar\_t));  hr = (pwszMessage != nullptr) ? S\_OK : E\_OUTOFMEMORY;  if (SUCCEEDED(hr))  {  hr = StringCchPrintfW(pwszMessage, cchMessagePlusTerminatingNul, L"%s%s", pwszMessagePreface, wszProcessName);  if (SUCCEEDED(hr))  {  hr = SHStrDupW(pwszMessage, ppwszWhoAmI);  }  CoTaskMemFree(pwszMessage);  }  }  }  return hr;  }  IFACEMETHODIMP PackDigits(unsigned char d1, unsigned char d2,  unsigned char d3, unsigned char d4,  unsigned int\* pResult)  {  if (pResult == nullptr) { return E\_POINTER; }  \*pResult = 0;  if (d1 > 9 || d2 > 9 || d3 > 9 || d4 > 9) { return E\_INVALIDARG; }  \*pResult = static\_cast<unsigned int>(d1) \* 1000 +  static\_cast<unsigned int>(d2) \* 100 +  static\_cast<unsigned int>(d3) \* 10 +  static\_cast<unsigned int>(d4);  return S\_OK;  }  public:  CComServiceTest() : \_cRef(1)  {  }  private:  LONG \_cRef;  ~CComServiceTest(void)  {  }  };  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  // Creation function  HRESULT CComServiceTest\_CreateInstance(\_\_in REFIID riid, \_\_out void\*\* ppv)  {  HRESULT hr = E\_OUTOFMEMORY;  CComServiceTest\* pInstance = new CComServiceTest();  if (pInstance != nullptr)  {  hr = pInstance->QueryInterface(riid, ppv);  pInstance->Release();  }  return hr;  } |

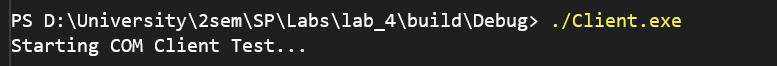
**CComServerTest.cpp**

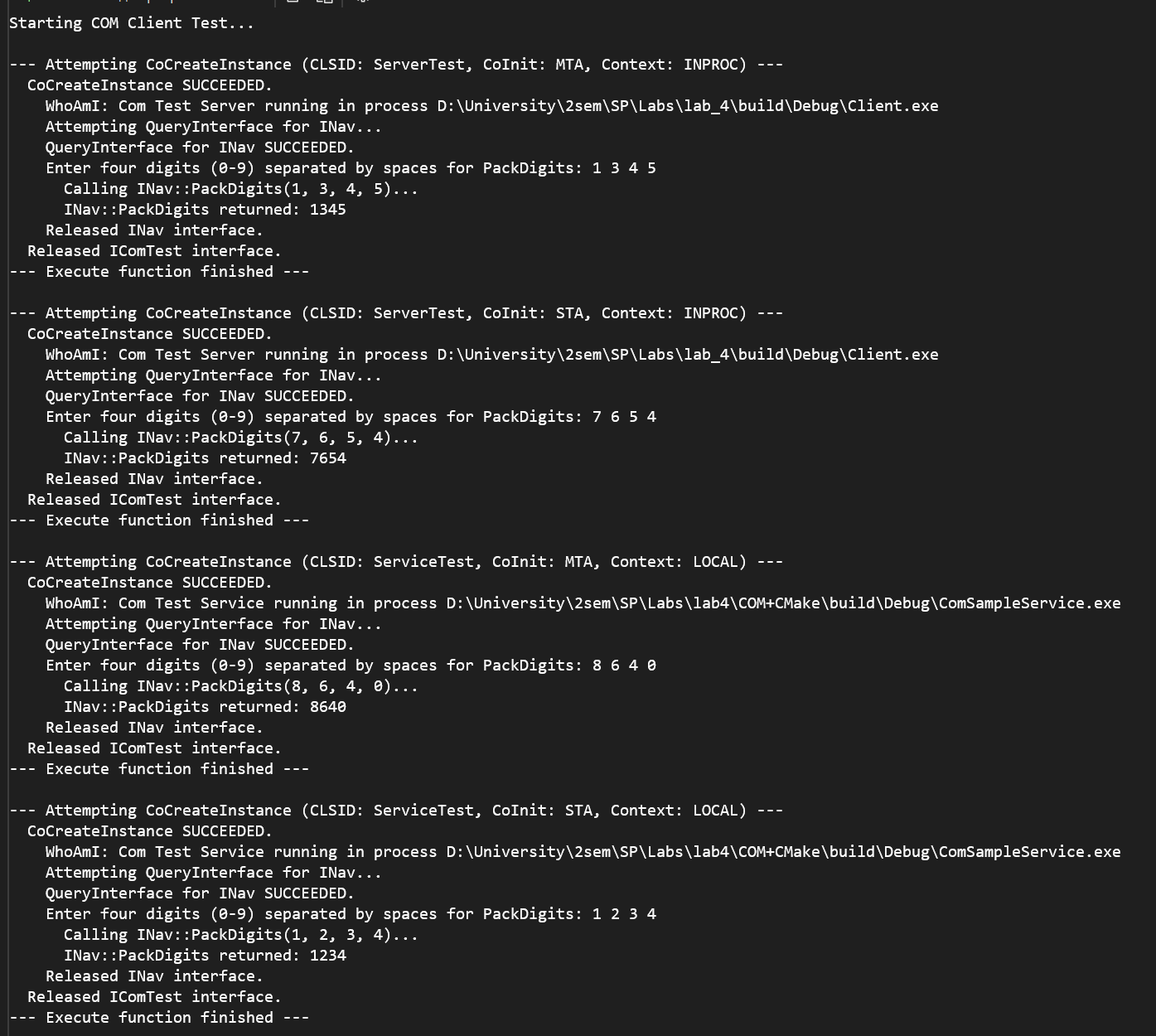
|  |
| --- |
| #include "pch.h"  #pragma hdrstop  #include "ComSampleServerCreateInstances.h"  #include "ComSampleServerGuids.h"  #include "Dll.h"  #include <IComTest\_h.h>  #include <assert.h>  #include <objbase.h>  #include <shlwapi.h>  class CComServerTest : public IComTest, public INav {  public:  // IUnknown  IFACEMETHODIMP\_(ULONG) AddRef() { return InterlockedIncrement(&\_cRef); }  IFACEMETHODIMP\_(ULONG) Release() {  assert(\_cRef > 0);  ULONG cRef = InterlockedDecrement(&\_cRef);  if (0 == cRef) {  delete this;  }  return cRef;  }  IFACEMETHODIMP QueryInterface(\_\_in REFIID riid, \_\_out void\*\* ppv) {  // Проверка ppv на NULL  if (ppv == nullptr) {  return E\_POINTER;  }  \*ppv = nullptr;  if (riid == \_\_uuidof(IUnknown)) {  \*ppv = static\_cast<IUnknown\*>(static\_cast<IComTest\*>(this));  }  else if (riid == \_\_uuidof(IComTest)) {  \*ppv = static\_cast<IComTest\*>(this);  }  else if (riid == \_\_uuidof(INav)) {  \*ppv = static\_cast<INav\*>(this);  }  else {  return E\_NOINTERFACE;  }  reinterpret\_cast<IUnknown\*>(\*ppv)->AddRef();  return S\_OK;  }  IFACEMETHODIMP WhoAmI(\_Out\_ LPWSTR\* ppwszWhoAmI) {  HRESULT hr = (ppwszWhoAmI != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr)) {  // Create a message string the contains the name of the current process.  wchar\_t wszProcessName[MAX\_PATH] = {};  DWORD dwResult = GetModuleFileNameW(nullptr, wszProcessName,  ARRAYSIZE(wszProcessName));  hr = ((dwResult < ARRAYSIZE(wszProcessName)) && (dwResult != 0))  ? S\_OK  : HRESULT\_FROM\_WIN32(GetLastError());  if (SUCCEEDED(hr)) {  LPCWSTR pwszMessagePreface = L"Com Test Server running in process ";  // Prepare the message.  size\_t cchMessagePlusTerminatingNul =  wcslen(pwszMessagePreface) + wcslen(wszProcessName) +  1 /\* For the terminating NUL character \*/;  wchar\_t\* pwszMessage = (wchar\_t\*)CoTaskMemAlloc(  cchMessagePlusTerminatingNul \* sizeof(wchar\_t));  hr = (pwszMessage != nullptr) ? S\_OK : E\_OUTOFMEMORY;  if (SUCCEEDED(hr)) {  hr = StringCchPrintfW(pwszMessage, cchMessagePlusTerminatingNul,  L"%s%s", pwszMessagePreface, wszProcessName);  if (SUCCEEDED(hr)) {  hr = SHStrDupW(pwszMessage, ppwszWhoAmI);  }  CoTaskMemFree(pwszMessage);  }  }  }  return hr;  }  IFACEMETHODIMP PackDigits(unsigned char d1, unsigned char d2,  unsigned char d3, unsigned char d4,  unsigned int\* pResult)  {  if (pResult == nullptr) {  return E\_POINTER;  }  \*pResult = 0;  if (d1 > 9 || d2 > 9 || d3 > 9 || d4 > 9) {  return E\_INVALIDARG;  }  \*pResult = static\_cast<unsigned int>(d1) \* 1000 +  static\_cast<unsigned int>(d2) \* 100 +  static\_cast<unsigned int>(d3) \* 10 +  static\_cast<unsigned int>(d4);  return S\_OK;  }  public:  CComServerTest() : \_cRef(1) {  InterlockedIncrement(&g\_cRefDll); // g\_cRefDll++;  }  private:  LONG \_cRef;  ~CComServerTest(void) {  InterlockedDecrement(&g\_cRefDll); // g\_cRefDll--;  }  };  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  // Creation function  HRESULT CComServerTest\_CreateInstance(\_\_in REFIID riid, \_\_out void\*\* ppv) {  HRESULT hr = E\_OUTOFMEMORY;  CComServerTest\* pInstance = new CComServerTest();  if (pInstance != nullptr) {  hr = pInstance->QueryInterface(riid, ppv);  pInstance->Release();  }  return hr;  }  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

QueryInterface — это способ "переключиться" на другой интерфейс того же COM-объекта.

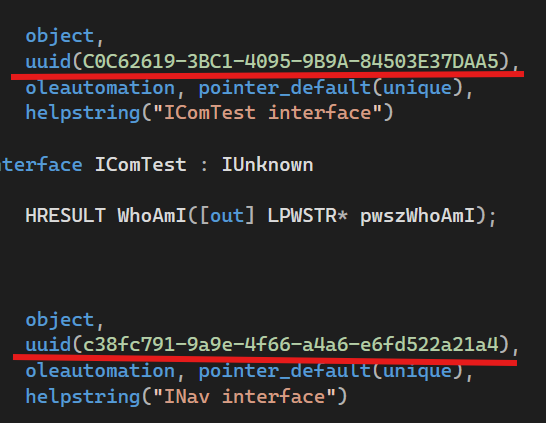
**Main.cpp**

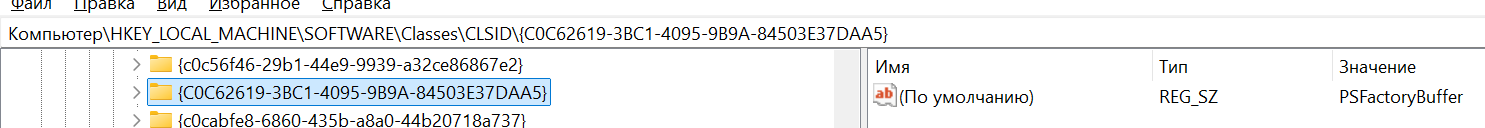
|  |
| --- |
| #include <windows.h>  #include <combaseapi.h> // Для CoInitializeEx, CoCreateInstance и т.д.  #include <strsafe.h> // Для StringCchPrintfW  #include <initguid.h> /  #include "IComTest\_h.h"  #include <ComSampleServerGuids.h>  #include <ComSampleServiceGuids.h>  #include <winerror.h>  #include <stdio.h>  HRESULT Execute(\_In\_ const IID& rclsid, \_In\_ DWORD dwCoInit, \_In\_ DWORD dwClsContext)  {  HRESULT hr = CoInitializeEx(NULL, dwCoInit);  if (SUCCEEDED(hr))  {  IComTest\* pComTest = nullptr;  INav\* pNav = nullptr;  wprintf(L"\n--- Attempting CoCreateInstance (CLSID: %s, CoInit: %s, Context: %s) ---\n",  (rclsid == CLSID\_CComServerTest) ? L"ServerTest" : L"ServiceTest",  (dwCoInit == COINIT\_MULTITHREADED) ? L"MTA" : L"STA",  (dwClsContext == CLSCTX\_INPROC\_SERVER) ? L"INPROC" : L"LOCAL");  hr = CoCreateInstance(rclsid, NULL, dwClsContext, IID\_PPV\_ARGS(&pComTest));  if (SUCCEEDED(hr))  {  wprintf(L" CoCreateInstance SUCCEEDED.\n");  LPWSTR pwszWhoAmI = nullptr;  HRESULT hrWhoAmI = pComTest->WhoAmI(&pwszWhoAmI);  if (SUCCEEDED(hrWhoAmI)) {  wprintf(L" WhoAmI: %s\n", pwszWhoAmI);  CoTaskMemFree(pwszWhoAmI);  }  else {  wprintf(L" WhoAmI call FAILED, hr=0x%x\n", hrWhoAmI);  }  wprintf(L" Attempting QueryInterface for INav...\n");  HRESULT hrNav = pComTest->QueryInterface(IID\_INav, (void\*\*)&pNav);  if (SUCCEEDED(hrNav))  {  wprintf(L" QueryInterface for INav SUCCEEDED.\n");  int d1\_int = -1, d2\_int = -1, d3\_int = -1, d4\_int = -1;  int scan\_result = 0;  printf(" Enter four digits (0-9) separated by spaces for PackDigits: ");  scan\_result = scanf\_s("%d %d %d %d", &d1\_int, &d2\_int, &d3\_int, &d4\_int);  int c;  while ((c = getchar()) != '\n' && c != EOF);  if (scan\_result == 4 &&  d1\_int >= 0 && d1\_int <= 9 &&  d2\_int >= 0 && d2\_int <= 9 &&  d3\_int >= 0 && d3\_int <= 9 &&  d4\_int >= 0 && d4\_int <= 9)  {  unsigned char d1 = static\_cast<unsigned char>(d1\_int);  unsigned char d2 = static\_cast<unsigned char>(d2\_int);  unsigned char d3 = static\_cast<unsigned char>(d3\_int);  unsigned char d4 = static\_cast<unsigned char>(d4\_int);  unsigned int packedResult = 0;  wprintf(L" Calling INav::PackDigits(%u, %u, %u, %u)...\n", d1, d2, d3, d4);  hrNav = pNav->PackDigits(d1, d2, d3, d4, &packedResult);  if (SUCCEEDED(hrNav)) {  wprintf(L" INav::PackDigits returned: %u\n", packedResult);  }  else {  wprintf(L" INav::PackDigits call FAILED, hr=0x%x\n", hrNav);  }  }  else  {  wprintf(L" Invalid input. Skipping PackDigits call for this instance.\n");  }  pNav->Release();  pNav = nullptr;  wprintf(L" Released INav interface.\n");  }  else  {  wprintf(L" QueryInterface for INav FAILED, hr=0x%x\n", hrNav);  }  pComTest->Release();  pComTest = nullptr;  wprintf(L" Released IComTest interface.\n");  }  else {  wprintf(L" CoCreateInstance FAILED, hr=0x%x (Is server registered? Check CLSID/Context combination.)\n", hr);  }  CoUninitialize();  }  else {  wprintf(L"CoInitializeEx FAILED, hr=0x%x\n", hr);  }  wprintf(L"--- Execute function finished ---\n");  return hr;  }  int \_\_cdecl main()  {  wprintf(L"Starting COM Client Test...\n");  Execute(CLSID\_CComServerTest, COINIT\_MULTITHREADED, CLSCTX\_INPROC\_SERVER);  Execute(CLSID\_CComServerTest, COINIT\_APARTMENTTHREADED, CLSCTX\_INPROC\_SERVER);  Execute(CLSID\_CComServiceTest, COINIT\_MULTITHREADED, CLSCTX\_LOCAL\_SERVER);  Execute(CLSID\_CComServiceTest, COINIT\_APARTMENTTHREADED, CLSCTX\_LOCAL\_SERVER);  wprintf(L"\nClient finished execution.\n");  return 0;  } |

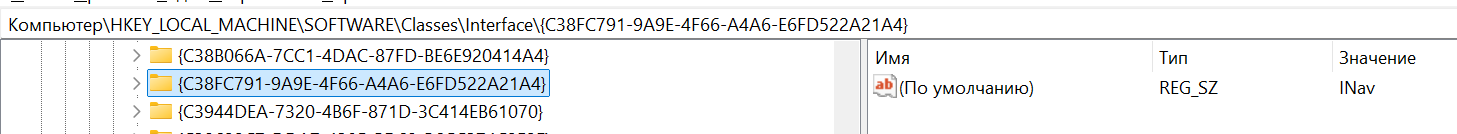




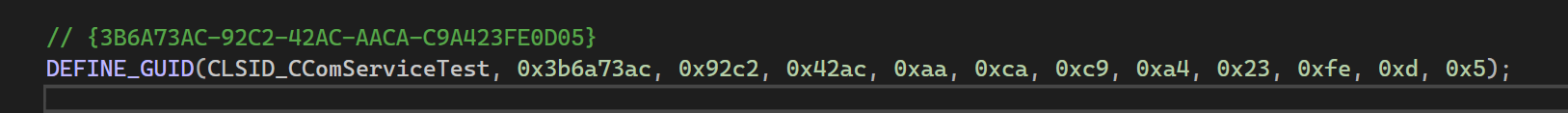
Изучить информацию из реестра о зарегистрированных COM-компонентах. (Обязательно найти оба: из проекта и ваш)



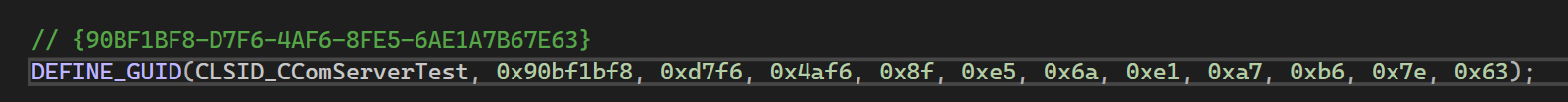


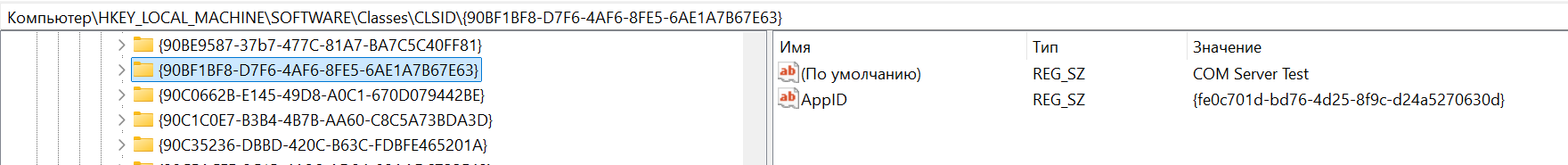


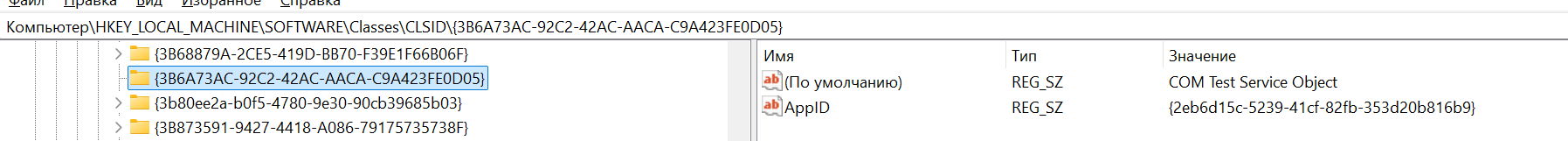
ComSampleServiceGuids.h



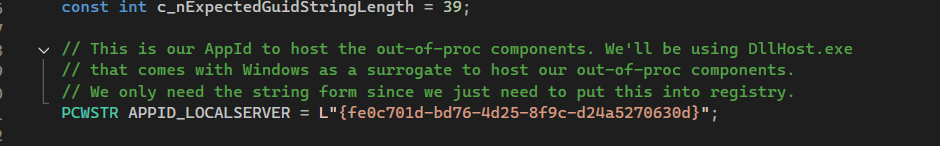
ComSampleServerGuids.h

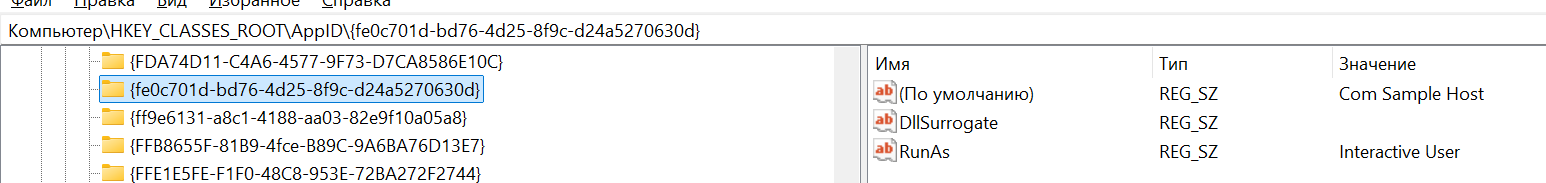




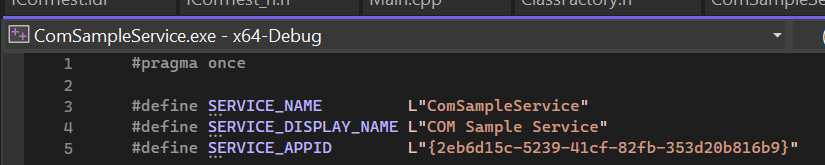


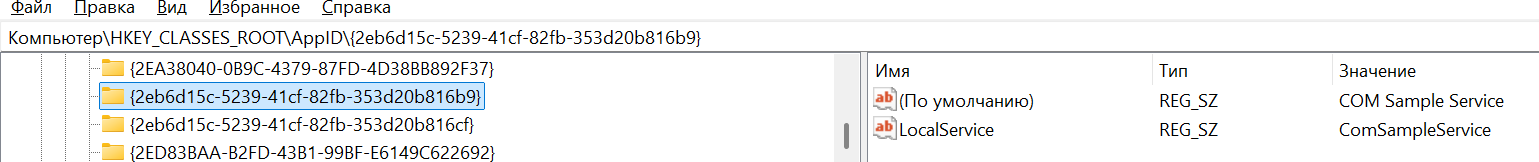
Dll.cpp





Infrastructure.h





**Вопросы для контроля:**

1. Что такое Component Object Model (далее - COM)?
2. Что такое COM-компонент?
3. Что такое COM-интерфейс?
4. Что такое COM-контейнер? Какие бывают?
5. Что такое COM-сервер и COM-клиент? Какие бывают сервера?
6. Что такое GUID? Для чего он используется?
7. Какие бывают интерфейсы?
8. Поясните функции интерфейса IUnknown.
9. Объясните «счётчик ссылок на интерфейс». Когда увеличивается/уменьшается?
10. Поясните функции интерфейса IClassFactory.
11. Объясните «счётчик ссылок на компонент». Когда увеличивается/уменьшается?
12. Какие соглашения о вызове и возврате должны соблюдаться для COM-функций?
13. Опишите структуру HRESULT.
14. Что должен знать COM-клиент чтобы взаимодействовать с COM-сервером?
15. Что такое regsvr32? Принцип работы?
16. Где найти информацию о компоненте в реестре?
17. Перечислите и объясните 5 COM-функций экспортируемых из DLL.
18. Какая системная библиотека отвечает за COM-инфраструктуру?