Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра информационных систем и технологий

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

**Методические указания к выполнению контрольной работы
для студентов специальности
1-40 01 02-03 «Информационные системы
и технологии (издательско-полиграфический комплекс)»
заочной формы обучения**

Минск 2014

УДК 004.415.25(075.8)
ББК 32.965.7я75
С40

Рассмотрены и рекомендованы редакционно-издательским советом Белорусского государственного технологического университета

С о с т а в и т е л ь
В. В. Смелов

Р е ц е н з е н т
кандидат технических наук, доцент,
заведующий кафедрой систем обработки информации
и полиграфического оборудования Белорусского
государственного технологического университета
М. С. Шмаков

По тематическому плану изданий учебно-методической литературы университета на 2013 год. Поз. 181.

Предназначены для студентов специальности 1-40 01 02-03 «Информационные системы и технологии (издательско-полиграфический комплекс)» заочной формы обучения

ПРЕДИСЛОВИЕ

Предлагаемое пособие предназначено для студентов специальности «Информационные системы и технологии (издательско-полиграфический комплекс)» заочной формы обучения, изучающих дисциплину «Системное программирование».

Учебный план студентов заочной формы обучения предусматривает контрольную работу. Выполнение и защита контрольной работы являются допуском к сдаче зачета по дисциплине.

Контрольная работа состоит из девяти практических работ, выполняемых студентом самостоятельно, каждая практическая работа включает несколько заданий. Задания имеют сквозную нумерацию. При выполнении практических работ следует придерживаться последовательности, предложенной в указаниях, так как формулировка некоторых заданий может опираться на результаты предыдущих. Последнее задание каждой практической работы содержит перечень вопросов, на которые студент должен будет ответить при защите работы.

Методические указания состоят из девяти разделов, соответствующих практическим работам контрольной. Каждый раздел содержит тезисное описание теоретического материала, необходимого для выполнения заданий работы. Описание сопровождается ссылками на литературу, рекомендуемую студенту для самостоятельного изучения.

Материал пособия предполагает наличие базовых знаний студента в области программирования и теории операционных систем, а также навыков программирования на языке C++ в объеме [1].

Для самостоятельной теоретической подготовки студентам рекомендуется книга [2], а для выполнения практических заданий источники [3] и [4]. Кроме того, дополнительно при подготовке к зачету следует ознакомиться с литературой [5] и [6].

При выполнении практических работ студенту понадобится компьютер с оперативной памятью не менее 1 ГБ и операционной системой не ниже Windows XP SP3 с установленной интегрированной средой разработки Visual Studio версии не ниже 2008. Задания практических работ сводятся к разработке консольных приложений на языке C++.

Первые четыре практические работы, предложенные ниже, посвящены знакомству со спецификацией COM. Следует отметить, что материал охватывает весьма незначительную часть этой спецификации. Для более полного ее изучения рекомендуется источник [3]. Кроме того, спецификация COM претерпела несколько этапов развития:

спецификации DCOM (Distributed COM – распределенная COM) и COM+. Ознакомиться с ними можно в книге [7].

Пятая практическая работа посвящена изучению структурной обработки исключений – механизму, позволяющему обрабатывать события, произошедшие во время выполнения программы и влекущие ненормальное завершение программы.

В шестой практической студент получит навыки работы с программным интерфейсом, позволяющим работать с пользователями и группами пользователей операционной системы Windows.

Седьмая и восьмая практические работы ознакомят студента с принципами асинхронного ввода-вывода, а также помогут освоить специальный механизм синхронизации – порт завершения ввода-вывода, позволяющий оповещать параллельно работающие потоки о завершении асинхронных операций ввода-вывода.

Последняя, девятая практическая работа посвящена разработке Windows-сервисов – специальных приложений, встраиваемых в операционную систему и загружаемых в память компьютера при загрузке Windows.

Для выполнения практических работ с пятой по девятую рекомендуется ознакомиться с соответствующим материалом из книги [4].

Пособие не содержит практических работ, посвященных потокам и процессам операционной системы, средствам синхронизации потоков, механизмам межпроцессного взаимодействия, работе с оперативной памятью и другим традиционным механизмам операционных систем. Предполагается, что навыки работы с ними студенты получили при выполнении практических работ курса «Операционные системы».

ВВЕДЕНИЕ

Когда говорят о системном программировании, в большинстве случаев подразумевают разработку программ, имеющих один из трех признаков. Первый признак: пользователем разработанного программного обеспечения является программист. Иными словами, системный программист разрабатывает программное обеспечение для других программистов, часто называемых «проблемными». Второй признак: разрабатываемое программное обеспечение является повторно используемым и, как правило, оформляется в виде библиотек функций и применяется в нескольких прикладных приложениях. Можно сказать, что системный программист разрабатывает общее, универсальное программное обеспечение. Третий признак: системное программное обеспечение напрямую использует системные вызовы операционной системы.

Может показаться, что выше перечислено три класса различного программного обеспечения. Но при разработке общего, повторно используемого программного обеспечения, неминуемо приходится применять системные вызовы. Поэтому, как правило, системное программирование подразумевает разработку библиотек универсальных функций, использующих системные вызовы операционной системы.

При разработке повторно используемого программного обеспечения, системный программист берет уже существующую или предлагает новую систему соглашений, которой оно должно соответствовать. Соглашения могут быть оформлены в виде спецификаций или корпоративных стандартов. В этих документах, как правило, оговариваются принципы именования объектов (имена функций, параметров, переменных), структуры и типы используемых данных, система интерфейсов (группы функций, классифицированных по каким-то признакам) и т. д. Примером такой спецификации может служить COM (Component Object Model – объектная модель компоненты) компании Microsoft. Практически вся прикладная часть программного обеспечения (например, Microsoft Office), входящая в операционную систему Windows, разработана в соответствии с этой спецификации. Именно благодаря такому подходу компании Microsoft удалось обеспечить эффективное взаимодействие и совместимость всех своих прикладных продуктов. Программист, освоивший эту спецификацию, получает возможность писать программное обеспечение, которое может использовать открытые интерфейсы для программного взаимодействия с продуктами Microsoft, а также организовать взаимодействие между загрузочными модулями собственного программного обеспечения.

Практическая работа № 1.
ИЗУЧЕНИЕ МОДЕЛИ COM НА ПРИМЕРЕ ПРОСТЕЙШЕГО ПРИЛОЖЕНИЯ

1.1. Теоретические сведения

1.1.1. Общие сведения о спецификации COM

Спецификация COM (Component Object Model – объектная модель компоненты) – это технологический стандарт компании Microsoft, первая версия которого разработана в 1993 году. Стандарт предназначен для создания программного обеспечения на основе взаимодействующих компонентов.

Первым основным понятием, которым оперирует стандарт COM, является COM-компонент, представляющий собой программный модуль. Каждый компонент имеет свой уникальный128-битный идентификатор в формате GUID (Global Unique Identifier – глобальный уникальный идентификатор).

Вторым основным понятием стандарта является COM-интерфейс. Интерфейс представляет собой набор абстрактных функций, имеющий аналогично COM-компонентам свой GUID-идентификатор. Интерфейсы бывают двух типов: стандартные и произвольные. За стандартными интерфейсами закреплены предопределенные GUID-идентификаторы. Важнейшим среди стандартных интерфейсов является интерфейс IUnknown. Все остальные интерфейсы являются производными (наследуют все методы) от IUnknown. Каждый компонент должен поддерживать (часто говорят «реализовывать») как минимум стандартный интерфейс IUnknown.

Для размещения компонентов могут быть применены два вида контейнеров: DLL-файл и EXE-файл. Приложения, использующие COM-компоненты (вызывающие функции интерфейсов, реализованных COM-компонентами), называют COM-клиентами, а контейнеры с расположенными в них компонентами – COM-серверами. В зависимости от типа контейнера и места его расположения (локальное или удаленное) различают несколько типов серверов: INPROC (DLL, локальный), LOCAL (EXE, локальный), REMOTE (EXE, удаленный). При этом COM-сервер сам может выступать в виде клиента, если он вызывает методы интерфейсов, реализованные другими компонентами.

Каждый COM-компонент должен быть зарегистрирован в Windows-реестре. Для регистрации компонента применяется специальная

утилита **regsvr32**. При работе с COM-компонентом клиент должен «знать» только GUID-идентификатор этого компонента, GUID-идентификаторы и структуры (сигнатуры соответствующих методов) произвольных интерфейсов компонента, которые он предполагает применять.

В данной работе будет рассматриваться только реализация компонентов, размещенных в DLL-файлах.

1.1.2. Принципы взаимодействия клиента и сервера

Поддержка программ соответствующих COM-модели в операционной системе Windows обеспечивается с помощью динамически подключаемой библиотеки OLE32.DLL и соответствующей ей библиотеки экспорта функций OLE32.LIB. На рис. 1.1 изображена схема взаимодействия COM-клиента и COM-сервера.

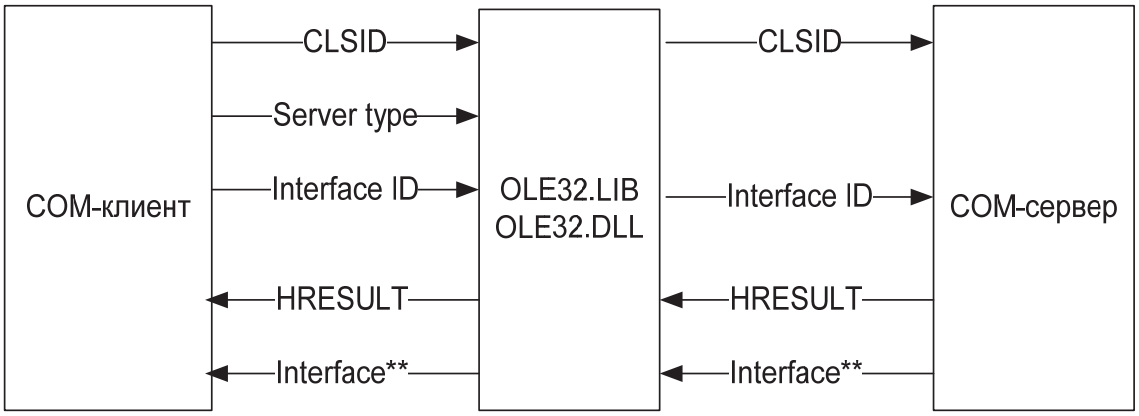

Рис. 1.1. Взаимодействие клиента и сервера в модели COM

OLE32.DLL выполнят роль посредника между клиентом и сервером. С помощью экспортируемых функций библиотеки клиент передает информацию об идентификаторе компонента (на схеме CLSID), типе сервера (Server type), идентификаторах интерфейсов (Interface ID). Именно OLE32.DLL по идентификатору CLSID через реестр операционной системы определяет место расположения контейнера компонента, загружает и инициализирует его. Как правило, поступающая в OLE32.DLL в виде значений параметров функций, информация передается функциям компонента. Компонент выполняет вызываемые OLE32.DLL стандартные функции или методы интерфейса, адрес которого (Interface**) он должен предварительно сообщить клиенту. За небольшим исключением все функции компонента должны возвращать результат в виде структуры HRESULT.

1.1.3. Структура однокомпонентного COM-сервера

В общем случае один COM-сервер может обеспечивать работу нескольких компонентов, в этом случае говорят от двух-трех и т. д. компонентных серверах. Однокомпонентный сервер поддерживает работу одного COM-компонента. В любом случае любой COM/DLL-сервер представляет собой DLL-библиотеку, экспортирующую стандартный набор функций. На рис. 1.2 приведен пример DEF-файла DLL-библиотеки (контейнера), обеспечивающей работу одного или нескольких компонентов. В таблице дается краткое пояснение для каждой стандартной экспортируемой функции.

```
LIBRARY "COMDLL"
EXPORTS
        DllCanUnloadNow     PRIVATE
        DllGetClassObject   PRIVATE
        DllInstall          PRIVATE
        DllRegisterServer   PRIVATE
        DllUnregisterServer PRIVATE
```

Рис. 1.2. Пример DEF-файла DLL-контейнера

Стандартные функции DLL-контейнера

Имя функции	Назначение
DllCanUnloadNow	Функция автоматически вызывается OLE32.DLL перед попыткой клиентом выгрузить COM-сервер. В зависимости от результата работы функции OLE32.DLL выгружает или не выгружает COM-сервер.
DllGetClassObject	Первая функция компонента, вызываемая OLE32.DLL при работе с клиентом. Функция проверяет идентификатор компонента, создает фабрику классов компонента и через параметры возвращает OLE32.DLL указатель на стандартный интерфейс IClassFactory.
DllInstal	Функция вызывается утилитой **regsvr32** при наличии соответствующего параметра, применяется для выполнения дополнительных действий при регистрации и удаления регистрации компонентов.
DllRegisterServer	Функция вызывается утилитой **regsvr32** при наличии соответствующего параметра, применяется для регистрации компонентов сервера в реестре операционной системы.
DllUnregisterServer	Функция вызывается утилитой **regsvr32** при наличии соответствующего параметра, применяется для удаления информации о компонентах сервера из реестра операционной системы.

Следует обратить внимание на то, что библиотека OLE32.DLL «ничего не знает» кроме двух функций (DllCanUnloadNow и DllGetClassObject). Поэтому разработка компоненты в простейшем случае сводится к двум проблемам: разработке этих функций и класса, обеспечивающего работу компонента в соответствии со спецификацией COM.

Две другие функции предназначены для записи (DllRegisterServer) или удаления (DllUnregisterServer) информации о компоненте из реестра операционной системы. Функция DllInstal носит вспомогательный характер и может быть вызвана совместно с функциями DllRegisterServer и DllUnregisterServer.

С точки зрения разработчика, компонент представляет собой класс C++, реализующий один или несколько интерфейсов. На рис. 1.3 приведен пример кода описания интерфейса.

```
#pragma once
#include <objbase.h>                // поддержка COM

static const GUID IID_IY =          // GUID-идентификатор интекфейса IY
{ 0xec10ad8f,0xdd52,0x41a6,{0x99, 0x9d, 0xb6, 0xf7, 0x74, 0x66, 0xb6, 0x11 }};

interface IY:IUnknown               // интерфейс IY наследует  IUnknown
{
  virtual HRESULT  STDMETHODCALLTYPE  Fy1() = 0;  // Метод Fy1 интерфейса IY
  virtual HRESULT  STDMETHODCALLTYPE  Fy2() = 0;  // Метод Fy2 интерфейса IY
};
```

Рис. 1.3. Пример кода описания интерфейса

Интерфейс в программе на C++ определяется с помощью ключевого слова **interface** (можно считать аналогом ключевого слова **structure**). Следует обратить внимание, что интерфейс является производным от стандартного интерфейса IUnknown, а методы возвращают значение HRESULT и используют стандартное соглашение вызова функций API-интерфейса операционной системы Windows. Для объявления идентификатора интерфейса применяется структура GUID.

На рис. 1.4 приведен пример объявления класса COM-компонента. Компонент представляет собой класс C++, написанный в соответствии со стандартом COM. Представленный на рисунке класс компонента CA реализует (с помощью механизма множественного наследования) три интерфейса: два произвольных IX и IY и обязательный стандартный IUnknown (унаследованный от произвольных интерфейсов).

```
#pragma once
#include <objbase.h>      // поддержка COM
#include "IX.h"           // интерфейс IX
#include "IY.h"           // интерфейс IY
class CA:public IX, public IY   // компонент CA реализует интерфесы IX и  IY
{
 public:
   CA();         //конструктор
   ~CA();        //деструктор
   // методы интерфейса IUnknown
   virtual HRESULT STDMETHODCALLTYPE QueryInterface(REFIID riid,void  **ppv);
   virtual ULONG   STDMETHODCALLTYPE AddRef(void);
   virtual ULONG   STDMETHODCALLTYPE Release(void);
   // методы интерфейса IX
   virtual HRESULT STDMETHODCALLTYPE  Fx1();
   virtual HRESULT STDMETHODCALLTYPE  Fx2();
   // методы интерфейса IX
   virtual HRESULT STDMETHODCALLTYPE  Fy1();
   virtual HRESULT STDMETHODCALLTYPE  Fy2();
 private:
   volatile ULONG m_Ref;  // счетчик ссылок на интерфейсы компонента
};
```

Рис. 1.4. Пример объявления класса COM-компонента

Следует обратить внимание на методы стандартного интерфейса IUnknown. Метод QueryInterface имеет два параметра. Первый параметр используется для передачи идентификатора интерфейса, второй – для возврата указателя на запрашиваемый интерфейс. Два других метода (AddRef и Realese) используются для подсчета ссылок на интерфейсы компонента. Метод AddRef увеличивает счетчик ссылок (переменная m_Ref) на 1, а метод Realese уменьшает на 1. Кроме того, метод Release уничтожает экземпляр компонента, если обнаруживает, что после уменьшения счетчика ссылок на 1, его значение стало равным 0.

Следует также отметить, что приведенный на рис. 1.4 пример класса компонента не во всем соответствует спецификации COM. Это сделано с целью упрощения. В практической работе № 4 будет приведен и разобран пример класса, полностью соответствующего спецификации.

1.1.4. Структура простейшего COM-клиента

На рис. 1.5 представлен пример программного кода простейшего COM-клиента, работающего с единственным компонентом.

Работа клиента должна начинаться с инициализации библиотеки OLE32.DLL (вызов функции CoInitialize).

```
#include "stdafx.h"
#include <iostream>
#include <objbase.h>  // поддержка COM
#include "IX.h"       // интерфейс IX
#include "COM.h"      // CLSID_CA
int _tmain(int argc, _TCHAR* argv[])
{

 HRESULT hr0;
 IX* pIX = NULL;                // указатель на интерфейс IX
 IUnknown* pIUnknown = NULL;    // указатель на интерфейс IUnknown
 CoInitialize(NULL);            // инициализация библиотеки OLE32
 hr0 = CoCreateInstance(CLSID_CA, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (void**)&pIUnknown);
 if (SUCCEEDED(hr0))
 {
     if (SUCCEEDED(pIUnknown->QueryInterface(IID_IX,(void**)&pIX))) // получить указатель на IX
     {
      // выполнение методов интерфейса IX
      if (!SUCCEEDED(pIX->Fx1())) std::cout << "error IX::Fx1"<<std::endl ;
      if (!SUCCEEDED(pIX->Fx2())) std::cout << "error IX::Fx2"<<std::endl ;
      pIX->Release();
     }
 }
 else std::cout << "error IX"<<std::endl ;
 pIUnknown->Release();
 CoFreeUnusedLibraries();    // завершение работы с библиотекой
 return 0;
}
```

Рис. 1.5. Пример программного кода COM-клиента

Для создания экземпляра компонента необходимо вызвать функцию CoCreanteInstace. Функция принимает пять параметров, причем первые четыре являются входными, последний – выходной. Первый параметр (CLSID_CA) передает значение GUID-идентификатора компонента, а третий – тип сервера (в примере INPROC-сервер). Назначение второго параметра (в примере его значение равно NULL) будет пояснено в четвертой практической работе. В четвертом параметре передается значение GUID-интерфейса (в примере передается идентификатор интерфейса IUnknown), указатель на который возвращается с помощью последнего, пятого параметра. Функция CoCreateInstance возвращает значение в формате HRESULT. Проверка результата осуществляется с помощью макроса SUCCEEDED, который формирует истинное значение в случае возврата значения, интерпретируемое как успешное завершение. В функции CoCreanteInstace можно запрашивать указатель на любой интерфейс (не обязательно IUnknown, как в примере).

Для получения указателя на другие интерфейсы можно применить метод QueryInterface стандартного интерфейса IUnknown. Метод принимает два параметра. Первый параметр является входным и используется для передачи значения GUID-идентификатора запрашиваемого интерфейса, а второй – выходной, для возврата указателя.

Метод QueryInterface увеличивает счетчик ссылок на интерфейсы на 1 (выполняет метод AddRef). После того, как в COM-клиенте предполагается, что указатель на интерфейс больше не потребуется, следует уменьшить счетчик на 1 (выполнить метод Release). Следует помнить, что если счетчик достигнет значения 0, то компонент будет разрушен и потребует, в случае необходимости, повторной инициализации (функция CoCreanteInstace).

Перед своим завершением COM-клиент обязан корректно закончить работу с библиотекой OLE32.DLL. Для этого он должен выполнить функцию CoFreeUnusedLibrary. Надо помнить, что вызов этой функции приведет к вызову функции DllCanUnloadNow.

1.2. Задания

Задание 1. Разработка COM/DLL-сервера:

1) разработайте COM/DLL-сервер, реализующий два интерфейса (IX и IY), приведенных на рис. 1.6 и 1.7;

```
#pragma once
#include <objbase.h>           // поддержка COM

static const GUID IID_IX =     // GUID-идентификатор интекфейса IX
{ 0xe04d1098, 0x2f6f, 0x4413, {0x88, 0xa9, 0x17, 0xaa, 0x9b, 0x97, 0x9d, 0x95}};

interface IX:IUnknown          // интерфейс IX наследует  IUnknown
{
 virtual HRESULT  __stdcall Fx1() = 0;  // Метод Fx1 интерфейса IX
 virtual HRESULT  __stdcall Fx2() = 0;  // Метод Fx2 интерфейса IX
};
```

Рис. 1.6. Интерфейс IX

```
#pragma once
#include <objbase.h>             // поддержка COM

static const GUID IID_IY =       // GUID-идентификатор интекфейса IY
{ 0xec10ad8f,0xdd52,0x41a6,{0x99, 0x9d, 0xb6, 0xf7, 0x74, 0x66, 0xb6, 0x11 }};

interface IY:IUnknown            // интерфейс IY наследует  IUnknown
{
  virtual HRESULT  STDMETHODCALLTYPE  Fy1() = 0;  // Метод Fy1 интерфейса IY
  virtual HRESULT  STDMETHODCALLTYPE  Fy2() = 0;  // Метод Fy2 интерфейса IY
};
```

Рис. 1.7. Интерфейс IY

2) оба метода интерфейса IX должны выводить в текстовый файл собственное имя;

3) оба метода интерфейса IY должны выводить в текстовый файл собственное имя и через двоеточие текущую дату;

4) поместите в отчет по контрольной работе распечатку программного кода разработанного сервера.

Задание 2. Регистрация COM/DLL-сервера:

1) запустите в командной строке утилиту **regsvr32** без параметров и прочитайте появившуюся в окне инструкцию;

2) зарегистрируйте COM-компонент, разработанный в задании 1;

3) используя утилиту **oleview** убедитесь, что регистрация выполнена успешно;

4) используя утилиту **regedit** убедитесь, что регистрация COM-компонента выполнена успешно;

5) поместите в отчет по контрольной работе скриншоты **oleview** и **regedit**, подтверждающие успешную регистрацию COM-компонента.

Задание 3. Разработка COM-клиента:

1) разработайте COM-клиент, вызывающий последовательно все методы интерфейсов IX и IY, разработанного в задании 1 сервера;

2) поместите в отчет по контрольной работе распечатку программного кода разработанной программы;

3) выполните программу COM-клиента;

4) поместите в отчет распечатку содержимого текстового файла, сформированного в результате вызова COM-клиентом методов COM-сервера.

Задание 4. Контрольные вопросы:

1) поясните понятия «COM-компонент», «COM-интерфейс», GUID;

2) перечислите основные требования к методам COM-интерфейса;

3) поясните назначение полей структуры HRESULT;

4) перечислите все стандартные функции, экспортируемые COM/DLL-сервером, поясните их назначение;

5) перечислите стандартные COM-интерфейсы, поясните назначение методов этих интерфейсов;

6) поясните назначение фабрики классов в составе COM-сервера;

7) поясните назначение утилит **regsvr32**, **oleview** и **regedit**;

8) поясните структуру COM-клиента, разработанного в задании 3.

Практическая работа № 2
РАЗРАБОТКА ДВУХКОМПОМПОНЕНТНОГО COM/DLL-СЕРВЕРА

2.1. Теоретические сведения

2.1.1. Основные отличия многокомпонентного COM/DLL-сервера от однокомпонентного

Принципиальных отличий в структурах многокомпонентного и однокомпонентного сервера нет. Два компонента (два класса) помещаются в один контейнер – DLL-файл и существуют независимо друг от друга. Каждый из них реализует свой набор интерфейсов.

Многокомпонентный сервер должен иметь тот же набор стандартных экспортируемых функций, что и однокомпонентный, но они должны учитывать наличие нескольких компонент. На рис. 2.1 представлен пример программного кода стандартной функции DllGetClassObject, предназначенной для двухкомпонентного сервера.

```
#include "stdafx.h"
#include "CFACTORY_DEF.h"          // шаблон фабрики классов

extern  GUID CLSID_CA, CLSID_CB;  // GUID-идентификаторы компонент CA и CB

STDAPI DllGetClassObject(const CLSID& clsid, const IID& iid, void**ppv)
{
 HRESULT rc = E_UNEXPECTED;
 if    (clsid == CLSID_CA)     // компонента CA?
  {
   CFactory<CA>  *pCA = new CFactory<CA>();  //  фабрика классов для CA
   rc = pCA->QueryInterface(iid, ppv);       //  запросить интерфейс
   pCA->Release();
  }
 else if  (clsid == CLSID_CB)  // компонента CB?
  {
   CFactory<CB>  *pCB = new CFactory<CB>(); //  фабрика классов для CB
   rc = pCB->QueryInterface(iid, ppv);      //  запросить интерфейс
   pCB->Release();
  }
 else  rc = CLASS_E_CLASSNOTAVAILABLE;      // нет такого копонента
 return rc;
};
```

Рис. 2.1. Пример реализации функции DllGetClassObject
для двухкомпонентного сервера

Не сложно заметить, что представленная функция состоит из двух аналогичных блоков кода, соответствующих двум компонентам. Значением первого параметра функции является GUID-идентификатор компонента, который задан первым параметром функции CoCreateInstance, вызываемой COM-клиентом. После проверки идентификатора функция создает объект фабрики классов. Фабрика классов – это вспомогательный компонент сервера, реализующий стандартный интерфейс IClassFactory и предназначенный для создания экземпляров соответствующего компонента. Для каждого компонента должна существовать своя фабрика классов. Программные коды всех фабрик классов сервера отличаются друг от друга незначительно, поэтому при реализации кода целесообразно применить шаблон класса. В [3] подробно рассмотрен пример создания такого шаблона.

Аналогично DllGetClassObject функция DllRegisterServer должна регистрировать два компонента в реестре сервера, а функция DllUnregisterServer удалять информацию о двух компонентах из реестра. Функция DllCanUnloadNow должна проверять наличие экземпляров также для двух компонент.

2.1.2. Принципы взаимодействия COM-клиента с несколькими COM/DLL-компонентами

COM-клиент, работающий с несколькими компонентами принципиально ничем не отличается от клиента, работающего с одним компонентом. На рис. 2.2. представлен пример COM-клиента, выполняющего методы интерфейсов, реализованных двумя различными компонентами.

Следует обратить внимание, что при работе с несколькими компонентами, требуется создание каждой по отдельности с помощью функции CoCreateInstance, а также на то, что клиент не знает, где располагаются компоненты – в разных серверах или в одном.

2.2. Задания

Задание 5. Разработка двухкомпонентного COM/DLL-сервера:

1) разработайте COM/DLL-сервер, содержащий два компонента: CA и CB;

2) компонент CA должен реализовывать два интерфейса: ICA_summer и ICA_multiplier (рис. 2.3); назначение методов пояснено в комментариях;

```
#include "stdafx.h"
#include "../CACB/ICA.h"
#include "../CACB/ICB.h"
#include "../CACB/CACB.h"

int _tmain(int argc, _TCHAR* argv[])
{
 double zA = 0, zS = 0, zM = 0, zD = 0, zP = 0, zL = 0;
 CoInitialize(NULL);
 double *pzA = NULL;
 ICA_summer* ps = NULL;
 HRESULT hrA = CoCreateInstance(CLSID_CA,NULL,CLSCTX_INPROC_SERVER,IID_ICA_summer,(void**)&ps);
 if (SUCCEEDED(hrA))
 {
  ps->Add(2,3,zA);    // вызов методов интефеса IID_ICA_summer комонента CA
  ps->Sub(16,5, zS);
 };
 ICA_multiplier* pm = NULL;
 HRESULT hrA_m = ps->QueryInterface(IID_ICA_multiplier, (void**)&pm);
 if (SUCCEEDED(hrA_m))
 {
  pm->Mul(3,6, zM); // вызов методов интефеса IID_ICA_multiplier комонента CA
  pm->Div(16,0, zD);
  pm->Release();
 };
 ps->Release();
 ICB_power* pp = NULL;
 HRESULT hrB  = CoCreateInstance(CLSID_CB,NULL,CLSCTX_INPROC_SERVER,IID_ICB_power,(void**)&pp);
 if (SUCCEEDED(hrB))
 {
  pp->Pow(2,3,zP);   // вызов методов интефеса IID_ICB_power комонента CB
  pp->Log(-16,2, zL);
  ps->Release();
 };
CoFreeUnusedLibraries();
return 0;
}
```

Рис. 2.2. Пример COM-клиента, работающего
с двумя компонентами

```
#pragma once
#include <objbase.h>
extern  GUID CLSID_CA,                          // GUID-идентификатор компонета
        IID_ICA_summer, IID_ICA_multiplier;  // GUID-идентификаторы интерфесов

interface ICA_summer:IUnknown // сумматор
{
 virtual HRESULT  __stdcall Add(const double x,    // [in]слагаемое x
                                const double y,    // [in]слагаемое y
                                double& z) = 0;    // [out] результат z = x+y
 virtual HRESULT  __stdcall Sub(const double x,    // [in] уменьшаемое
                                const double y,    // [in] вычитаемое y
                                double& z) = 0;    // [out] результат z = x-y
};

interface ICA_multiplier:IUnknown // множитель
{
 virtual HRESULT  __stdcall Mul(const double x,    // [in]сомножитель x
                                const double y,    // [in]сомножитель y
                                double& z) = 0;    // [out] результат z = x*y
 virtual HRESULT  __stdcall Div(const double x,    // [in] делимое x
                                const double y,    // [in] делитель y
                                double& z) = 0;    // [out] результат z = x/y
};
```

Рис. 2.3. Интерфейсы компонента CA

3) компонент CB должен реализовывать интерфейс ICB_power (рис. 2.4); назначение методов пояснено в комментариях;

```
#pragma once
#include <objbase.h>

extern  GUID CLSID_CB,   // GUID-идентификатор компонета
        IID_ICB_power;   // GUID-идентификатор интерфеса

interface ICB_power:IUnknown // степень
{
 virtual HRESULT  __stdcall Pow(const double x,   // [in]основание  x
                                const double y,   // [in]степень y
                                double& z) = 0;   // [out] результат z = x^y
 virtual HRESULT  __stdcall Log(const double x,   // [in] основание log
                                const double y,   // [in] аргумент log
                                double& z) = 0;   // [out] результат z = log_y(x)
};
```

Рис. 2.4. Интерфейс компонента CB

4) зарегистрируйте компоненты в реестре операционной системы с помощью утилиты **regsvr32;**

5) используя утилиты **oleview** и **regedit** убедитесь, что регистрация выполнена успешно;

6) поместите в отчет по контрольной работе скриншот **oleview**, подтверждающий успешную регистрацию COM-компонентов.

7) поместите в отчет по контрольной работе распечатку программного кода всех стандартных функций и фабрик классов.

Задание 6. Разработка COM-клиента, взаимодействующего с двумя COM/DLL-компонентами:

1) разработайте COM-клиент, работающий с COM/DLL-сервером, разработанным в задании 5; программа должна выполнять все методы каждого компонента сервера; результаты выполнения методов должны выводиться в окно консоли;

2) выполните программу COM-клиента;

3) поместите в отчет по контрольной работе скриншот окна консоли с результатами выполнения программы COM-клиента.

Задание 7. Контрольные вопросы:

1) поясните отличия многокомпонентного COM/DLL-сервер от однокомпонентного;

2) поясните назначение фабрики классов в составе COM-сервера;

3) поясните назначение стандартного интерфейса ICalssFactory и его методов;

4) поясните отличия COM-клиента, работающего с одним компонентом, от COM-клиента, работающего с несколькими компонентами.

Практическая работа № 3
ПОВТОРНАЯ ПРИМЕНИМОСТЬ COM-КОМПОНЕНТОВ: ВКЛЮЧЕНИЕ

3.1. Теоретические сведения

3.1.1. Повторное применение COM-компонентов

Одной из основных парадигм COM-программирования является устойчивость интерфейсов. Под устойчивостью понимается неизменность. Один раз разработанный интерфейс должен таким оставаться всегда. Это связано с тем, что работающие с компонентом клиенты зависимы от используемых ими интерфейсов: изменение интерфейса повлечет за собой изменение кода клиента. Следует обратить внимание на то, что речь идет о неизменности именно интерфейсов, а не от их реализации, которая от клиента скрыта и его код никак от реализации методов интерфейса не зависит. COM-компонент может изменяться только путем увеличения количества поддерживаемых им интерфейсов. Устойчивость интерфейсов позволяет создавать COM-компоненты сложной архитектуры, которая строится на взаимодействии нескольких COM-компонентов. В таких случаях говорят о повторном применении компонентов.

Спецификация COM предлагает два способа взаимодействия COM-компонентов: включение и агрегация. В любом случае все сводится к взаимодействию двух компонентов, один из которых является клиентом (вызывает методы серверного компонента), а другой – сервером (предоставляет методы клиенту).

При взаимодействии двух COM-компонентов, компонент, выступающий в качестве сервера, называется внутренним, а выступающий в качестве клиента – внешним. Серверы, служащие контейнерами для компонент тоже называют соответственно внутренними и внешними.

В соответствии со спецификацией COM, каждый компонент должен иметь открытый метод с именем Init, предназначенный для создания экземпляров внутренних компонентов (вызова функций CoCreateInstance). Если компонент не использует внутренние компоненты, то этот метод просто ничего не должен выполнять (рис. 3.1).

3.1.2. Включение COM-компонентов

Если COM-компонент включает интерфейсы другого компонента, то реализация метода Init, должна быть примерно такой же, как на рис. 3.2.

```
class CA:public ICA_summer
{
 public:
    CA();
    ~CA();
    IUNCNOWN_DEF;
    // ICA_summer
    virtual HRESULT  __stdcall Add(double x,   // [in]слагаемое x
                                   double y,   // [in]слагаемое y
                                   double* z);       // [out] результат z = x+y
    virtual HRESULT  __stdcall Sub(double x,   // [in] уменьшаемое
                                   double y,   // [in] вычитаемое y
                                   double* z); // [out] результат z = x-y
    HRESULT Init(){return S_OK;};              // нет внутренних компонент
 private:
    volatile ULONG m_Ref;
};
```

Рис. 3.1. Пример класса компонента, не использующего внутренний компонент

```
extern GUID CLSID_CD;
extern GUID IID_ICD_log;
HRESULT CC::Init()
{
return (CoCreateInstance(CLSID_CD, NULL,
                         CLSCTX_INPROC_SERVER, IID_ICD_log, (void**)&this->log));
};
```

Рис. 3.2. Пример реализации метода Init, компонента, применяющего включение методов другого компонента

Метод Init в этом случае должен, выполнить функцию CoCreateInstance (см. п. 1.1.4), создающую экземпляр внутреннего компонента. Другими словами внешний компонент по отношению к внутреннему выступает в роли клиента.

На рис. 3.3 приведен пример внешнего компонента, реализующего два интерфейса: ICC_summer и ICC_power. При этом первый интерфейс реализуется собственным кодом, а выполнение второго сводится к вызову соответствующих методов внутреннего компонента (рис. 3.4).

Следует обратить внимание, что при вызове функции CoCreateInstance заказывается указатель на требуемый интерфейс внутреннего компонента. Для хранения указателя используется переменная, объявленная в классе внешнего компонента.

```
class CC:public ICC_summer, public ICC_power
{
 public:
     CC();
    ~CC();
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(REFIID iid,void **ppv);
    virtual ULONG   STDMETHODCALLTYPE AddRef(void);
    virtual ULONG   STDMETHODCALLTYPE Release(void);

    // ICC_summer
    virtual HRESULT  __stdcall Add(const double x,   // [in]слагаемое x
                                   const double y,   // [in]слагаемое y
                                   double& z);       // [out] результат z = x+y
    virtual HRESULT  __stdcall Sub(const double x,   // [in] уменьшаемое
                                   const double y,   // [in] вычитаемое y
                                   double& z);       // [out] результат z = x-y
    //ICC_power использует методы интерфейса ICD_log комппонента CD (включение)

    virtual HRESULT  __stdcall Pow(const double x,   // [in]основание  x
                                   const double y,   // [in]степень y
                                   double& z);       // [out] результат z = x^y
    virtual HRESULT  __stdcall Log(const double x,   // [in] основание log
                                   const double y,   // [in] аргумент log
                                   double& z);       // [out] результат z = log_y(x)
    HRESULT Init();
 private:
   volatile ULONG m_Ref;
   ICD_log *log;
};
```

Рис. 3.3. Пример класса, применяющего включение другого компонента

```
HRESULT  STDAPICALLTYPE CC::Add(const double x, const double y, double& z)
 { z = x+y; return S_OK;};
HRESULT  STDAPICALLTYPE CC::Sub(const double x, const double y, double& z)
 { z = x-y; return S_OK;};

HRESULT  STDAPICALLTYPE CC::Pow(const double x, const double y, double& z)
 { this->log->Pow(x,y,z); return S_OK;};
HRESULT  STDAPICALLTYPE CC::Log(const double x, const double y, double& z)
 { this->log->Log(x,y,z); return S_OK;};
```

Рис. 3.4. Реализация методов интерфейсов компонента,
применяющего включение другого компонента

Должно быть также понятно, что если для реализации методов внешнего компонента потребуется еще один интерфейс того же внутреннего компонента, то указатель на него может быть получен с помощью метода QueryInterface. Если же необходимы интерфейсы иного компонента, то в Init следует сделать еще один вызов функции CoCreateInstance.

Таким образом, включение – это способ повторного применения COM-компонента, при котором некоторые методы внешнего компо-

нента имитируют выполнение методов внутреннего компонента. Имитация сводится к прямому вызову методов интерфейсов созданного экземпляра внутреннего компонента. При этом COM-клиент ничего не знает о внутреннем компоненте и его структура остается точно такой как при работе с одним компонентом.

3.2. Задания

Задание 8. Разработка внутреннего COM/DLL-сервера:

1) разработайте COM/DLL-сервер с компонентом CD, реализующим интерфейс, представленный на рис. 3.5.

```
interface ICD_log:IUnknown  // возведение в степень
{
 virtual HRESULT STDAPICALLTYPE Pow(const double x,   // [in]основание  x
                                    const double y,   // [in]степень y
                                    double& z) = 0;   // [out] результат z = x^y
 virtual HRESULT  STDAPICALLTYPE  Log(const double x,   // [in] основание log
                                    const double y,   // [in] аргумент log
                                    double& z) = 0;   // [out] результат z = log_y(x)
};
```

Рис. 3.5. Интерфейсы внутреннего компонента CD

2) зарегистрируйте компонент CD в реестре операционной системы с помощью утилиты **regsvr32;**

3) используя утилиты **oleview** и **regedit** убедитесь, что регистрация выполнена успешно;

4) поместите в отчет по контрольной работе скриншот **oleview**, подтверждающий успешную регистрацию COM-компонента.

Задание 9. Разработка внешнего COM/DLL-сервера:

1) разработайте COM/DLL-сервер с компонентом CC, реализующим интерфейсы, представленные на рис. 3.6 и 3.7; обратите внимание, что методы интерфейсов ICD_log (рис. 3.5) и ICC_power (рис. 3.7) совпадают;.

```
interface ICC_summer:IUnknown // сумматор
 {
  virtual HRESULT  STDAPICALLTYPE Add(const double x,   // [in]слагаемое x
                                      const double y,   // [in]слагаемое y
                                      double& z) = 0;   // [out] результат z = x+y
  virtual HRESULT  STDAPICALLTYPE Sub(const double x,   // [in] уменьшаемое
                                      const double y,   // [in] вычитаемое y
                                      double& z) = 0;   // [out] результат z = x-y
};
```

Рис. 3.6. Интерфейсы внешнего компонента CC

```
interface ICC_power:IUnknown  // возведение в степень
 {
  virtual HRESULT STDAPICALLTYPE Pow(const double x,   // [in]основание  x
                                     const double y,   // [in]степень y
                                     double& z) = 0;   // [out] результат z = x^y
  virtual HRESULT STDAPICALLTYPE Log(const double x,   // [in] основание log
                                     const double y,   // [in] аргумент log
                                     double& z) = 0;   // [out] результат z = log_y(x)
 };
```

Рис. 3.7. Интерфейсы внешнего компонента CC

2) реализуйте методы интерфейса ICC_power таким образом, чтобы они использовали вызов соответствующих методов интерфейса ICD_log компонента CD;

3) зарегистрируйте компонент CC в реестре операционной системы с помощью утилиты **regsvr32;**

4) используя утилиты **oleview** и **regedit** убедитесь, что регистрация выполнена успешно;

5) поместите в отчет по контрольной работе скриншот **oleview**, подтверждающий успешную регистрацию COM-компонента.

Задание 10. Разработка клиента, взаимодействующего с внутренним COM/DLL-сервером:

1) разработайте COM-клиент, работающий с COM/DLL-сервером, разработанным в задании 8; программа должна выполнять все методы всех интерфейсов сервера; результаты выполнения методов должны выводиться в окно консоли;

2) выполните программу COM-клиента;

3) поместите в отчет по контрольной работе скриншот окна консоли с результатами выполнения программы COM-клиента.

Задание 11. Разработка клиента, взаимодействующего с внешним COM/DLL-сервером:

1) разработайте COM-клиент, работающий с COM/DLL-сервером, разработанным в задании 9; программа должна выполнять все методы всех интерфейсов сервера; результаты выполнения методов должны выводиться в окно консоли;

2) выполните программу COM-клиента;

3) поместите в отчет по контрольной работе скриншот окна консоли с результатами выполнения программы COM-клиента.

Задание 12. Контрольные вопросы:

1) поясните понятие «повторная применимость компонента»;

2) поясните понятия «внутренний компонент», «внешний компонент»;

3) «внутренний сервер», «внешний сервер»;

4) поясните суть метода включения COM-компонента;

5) поясните назначение метода Init класса COM-компонента;

6) поясните принцип включения нескольких компонентов.

Практическая работа № 4
ПОВТОРНАЯ ПРИМЕНИМОСТЬ COM-КОМПОНЕНТОВ: АГРЕГИРОВАНИЕ

4.1. Теоретические сведения

4.1.1. Принципиальное отличие агрегирования от включения

Включение COM-компонента предполагает наличие у внешнего компонента собственных интерфейсов, представляющих наборы методов, которые являются функциональным наполнением компонента. Принципиальным моментом включения является то, что реализация некоторых методов сводится к вызову методов внутреннего компонента.

Агрегирование, в отличие от включения, не требует от внешнего компонента наличия собственных интерфейсов, вызывающих методы внутреннего, а осуществляет прямое подключение к интерфейсам внутреннего компонента. Другими словами, внешний компонент ничего не знает о структуре интерфейсов внутреннего компонента, а знает только их идентификаторы. При запросе клиентом указателя на интерфейс, который реализован внутренним компонентом, внешний компонент возвращает указатель на этот интерфейс. После этого клиент работает напрямую с внутренним компонентом.

Сложность реализации агрегирования заключается в необходимости обеспечить требование спецификации COM, заключающееся в том, что метод QueryInterface интерфейса IUnknown, вызванный через любой интерфейс, должен возвращать указатель на любой запрашиваемый интерфейс. Если запрашивается указатель на некоторый интерфейс через интерфейс, реализованный тем же компонентом, проблемы не возникает. Сложность вызывает получение указателя на интерфейс из другого компонента.

4.1.2. Агрегирование COM-компонентов

Спецификация COM предусматривает знание компонента о том, что он агрегируется другим (внешним) компонентом. Признаком того, что компонент подвержен агрегации, является ненулевое значение параметра конструктора класса в реализации компонента. На рис. 4.1 представлен пример класса компонента CD, который может быть агрегирован другим компонентом. Следует обратить внимание на два момента: конструктор класса имеет параметр, класс реализует дополнительный интерфейс ND_IUnknown.

Параметр конструктора может иметь значение NULL, что соответствует применению компонента не через агрегирование, а может иметь другое значение – в этом случае оно будет интерпретироваться как указатель на стандартный интерфейс IUnknown внешнего компонента.

Если значение параметра конструктора содержит адрес интерфейса IUnknown, то именно этот интерфейс будет применяться агрегируемым компонентом. В этом случае IUnknown называется делегирующим.

```
interface ND_IUnknown
{
 virtual HRESULT STDAPICALLTYPE ND_QueryInterface(REFIID iid,void** ppv) = 0;
 virtual ULONG   STDAPICALLTYPE ND_AddRef(void) = 0;
 virtual ULONG   STDAPICALLTYPE ND_Release(void) = 0;
};
class CD:public ICD_log, public ND_IUnknown
{
 public:
    CD(IUnknown* pUO); // pUO != NULL - признак агрегирования
    ~CD();
    // IUnknown - делегирующий IUnknown
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(REFIID iid,void **ppv);
    virtual ULONG   STDMETHODCALLTYPE AddRef(void);
    virtual ULONG   STDMETHODCALLTYPE Release(void);
    // ND_IUnknown - неделегирующий IUnknown
    virtual HRESULT STDAPICALLTYPE ND_QueryInterface(REFIID iid,void** ppv);
    virtual ULONG   STDAPICALLTYPE ND_AddRef(void);
    virtual ULONG   STDAPICALLTYPE ND_Release(void);
    // ICD_log
    virtual HRESULT  __stdcall Pow(const double x,   // [in]основание  x
                                   const double y,   // [in]степень y
                                   double& z);       // [out] результат z = x^y
    virtual HRESULT  __stdcall Log(const double x,   // [in] основание log
                                   const double y,   // [in] аргумент log
                                   double& z);       // [out] результат z = log_y(x)
    HRESULT Init();
 private:
    volatile ULONG m_Ref;
    IUnknown* pUO;       // текущее значение указателя на  IUnknown
};
```

Рис. 4.1. Пример класса компонента, который может быть агрегирован

В том случае, если компонент используется без агрегирования (параметр конструктора равен NULL), то компонент должен применять обычный интерфейс IUnknown, который называется неделегирующим. Для обеспечения работы неделегирующего IUnknown-интерфейса служит объявления ND_IUnknown. Примеры кодов конструктора класса (рис. 4.2) и реализации методов QueryInterface (рис. 4.3) поясняют сказанное.

```
extern ULONG CD_ComponentsCount;
CD::CD(IUnknown* pUO):pUO(pUO),m_Ref(1)
{
   InterlockedIncrement((LONG*)&CD_ComponentsCount);
   if (this->pUO == NULL) this->pUO = (IUnknown*)((ND_IUnknown*)this);
};
```

Рис. 4.2. Пример конструктора класса компонента

```
HRESULT STDMETHODCALLTYPE CD::QueryInterface(REFIID iid, void  **ppv)
 {return this->pUO->QueryInterface(iid, ppv);};
ULONG  STDMETHODCALLTYPE CD::AddRef(void)
 {return this->pUO->AddRef();};
ULONG  STDMETHODCALLTYPE CD::Release(void)
 {return this->pUO->Release();};
```

Рис. 4.3. Пример реализации методов интерфейса IUnknown

Параметр, принимаемый конструктором класса внутреннего компонента передается методом Init внешнего компонента. Значение параметра передается с помощью функции CoCreateInstance. На рис. 4.4 представлен пример метода Init, вызывающий функцию CoCreateInstance. Второй параметр функции – указатель на интерфейс IUnknown внешнего компонента.

```
extern GUID CLSID_CD;
extern GUID IID_ICD_log;
HRESULT CC::Init()
{
  HRESULT hr = CoCreateInstance(CLSID_CD, (IUnknown*)this,
                                CLSCTX_INPROC_SERVER, IID_IUnknown, (void**)&this->pIUInner);
  if (SUCCEEDED(hr))
  {
    hr = this->pIUInner->QueryInterface(IID_ICD_log, (void**)&this->log);
    this->pIUInner->Release();
  }
  return (hr);
};
```

Рис. 4.4. Пример реализации метода Init внешнего компонента

Для полного понимания на рис. 4.5 приведен пример реализации метода QueryInterface внешнего компонента.

Следует обратить внимание на то, что метод QueryInterface возвращает указатели как на собственные интерфейсы, так и на поддерживаемые интерфейсы внутреннего (агрегируемого) компонента.

```
HRESULT STDMETHODCALLTYPE CC::QueryInterface(REFIID riid, void  **ppv)
 {
   HRESULT rc = S_OK;                                        // COM
   *ppv = NULL;                                              // COM
   if       (riid == IID_IUnknown)     *ppv = (ICC_summer*)this;
   else if  (riid == IID_ICC_summer)   *ppv = (ICC_summer*)this;
   else if  (riid == IID_ICD_log)      *ppv = this->log; // внешний компонент
   else rc = E_NOINTERFACE;                                  // COM
   if (rc == S_OK) this->AddRef();
   return rc;
 };
```

Рис. 4.5. Пример реализации метода QueryInterface внешнего компонента

4.2. Задания

Задание 13. Задание 13. Разработка внутреннего COM/DLL-сервера:

1) разработайте COM/DLL-сервер с компонентом CD, реализующим интерфейс, представленный на рис. 4.6;

```
interface ICD_log:IUnknown  // возведение в степень
{
 virtual HRESULT STDAPICALLTYPE Pow(const double x,   // [in]основание  x
                                const double y,   // [in]степень y
                                double& z) = 0;   // [out] результат z = x^y
 virtual HRESULT  STDAPICALLTYPE  Log(const double x,   // [in] основание log
                                const double y,   // [in] аргумент log
                                double& z) = 0;   // [out] результат z = log_y(x)
};
```

Рис. 4.6. Интерфейсы внутреннего компонента CD

2) зарегистрируйте компонент CD в реестре операционной системы с помощью утилиты **regsvr32;**

3) используя утилиты **oleview** и **regedit**, убедитесь, что регистрация выполнена успешно;

4) поместите в отчет по контрольной работе скриншот **oleview**, подтверждающий успешную регистрацию COM-компонента.

Задание 14. Разработка внешнего COM/DLL-сервера:

1) разработайте COM/DLL-сервер с компонентом CC, реализующий интерфейс, представленные на рис. 4.7;

2) обеспечьте поддержку компонентом CC интерфейса ICD_log компонента CD (рис. 4.6) методом агрегирования;

3) зарегистрируйте компонент CC в реестре операционной системы с помощью утилиты **regsvr32;**

```
interface ICC_summer:IUnknown // сумматор
 {
  virtual HRESULT  STDAPICALLTYPE Add(const double x,   // [in]слагаемое x
                                      const double y,   // [in]слагаемое y
                                      double& z) = 0;   // [out] результат z = x+y
  virtual HRESULT  STDAPICALLTYPE Sub(const double x,   // [in] уменьшаемое
                                      const double y,   // [in] вычитаемое y
                                      double& z) = 0;   // [out] результат z = x-y
};
```

Рис. 4.7. Интерфейсы внешнего компонента CC

4) используя утилиты **oleview** и **regedit**, убедитесь, что регистрация выполнена успешно;

5) поместите в отчет по контрольной работе скриншот **oleview**, подтверждающий успешную регистрацию COM-компонента.

Задание 15. Разработка клиента, взаимодействующего с внутренним COM/DLL-сервером:

1) разработайте COM-клиент, работающий с COM/DLL-сервером, созданным в задании 13; программа должна выполнять все методы всех интерфейсов сервера; результаты выполнения методов должны выводиться в окно консоли;

2) выполните программу COM-клиента;

3) поместите в отчет по контрольной работе скриншот окна консоли с результатами выполнения программы COM-клиента.

Задание 16. Разработка клиента, взаимодействующего с внешним COM/DLL-сервером:

1) разработайте COM-клиент, работающий с COM/DLL-сервером, созданным в задании 15; программа должна выполнять все методы всех интерфейсов сервера; результаты выполнения методов должны выводиться в окно консоли;

2) выполните программу COM-клиента;

3) поместите в отчет по контрольной работе скриншот окна консоли с результатами выполнения программы COM-клиента.

Задание 17. Контрольные вопросы:

1) поясните суть метода агрегирования COM-компонента;

2) поясните назначение метода Init класса COM-компонента при агрегировании;

3) поясните понятия «делегирующий» и «неделегирующий» IUnknown-интерфейсы;

4) поясните принцип агрегирования нескольких компонентов.

Практическая работа № 5
СТРУКТУРНАЯ ОБРАБОТКА ИСКЛЮЧЕНИЙ

5.1. Теоретические сведения

5.1.1. Общие принципы обработки исключений

Стандартный способ обработки исключений, существующий в языке C++ основан на действии операторов try, catch, throw. Microsoft в компилятор C++ для операционной системы Windows встроил еще один механизм структурной обработки исключений (SEH – Structured exception handling), основанный на операторах __try, __except, __finally. SEH является низкоуровневым механизмом операционной системой Windows в том смысле, что все ошибки (аппаратные и программные сбои, ошибки исполнения программы), возникающие при выполнении программ в Windows, обрабатываются именно по этой схеме. Все остальные способы обработки ошибок, предоставляемые языками программирования, в конце концов, сводятся к SEH. На рис. 5.1 представлен пример программы, применяющей структурную обработку исключений.

```
#include "stdafx.h"
#include "windows.h"
#include <iostream>
int _tmain(int argc, _TCHAR* argv[])
{
    int  x = 0, y = 140;
    __try
    {
     // охраняемый код
     std::cout << "Hello from try-code 1"  << std::endl;
     y /= x;
     std::cout << "y = " << y << std::endl;
     std::cout << "Hello from try-code 2"<< std::endl;
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
     // код обработки исключения
     std::cout << "Hello from handler code" <<  std::endl;
    }
    system( "pause");
    return 0;
}
```

Рис. 5.1. Пример программы, применяющей структурную обработку исключений

Блок кода, ограниченный фигурными скобками оператора __try, обычно называют охраняемым кодом. Предполагается, что в этом блоке может возникнуть исключение, которое следует обработать.

Блок кода оператора __except предназначен для обработки возникшего в блоке __try исключения. Управление передается в этот блок только в том случае, если в охраняемом коде произошло исключение и в круглых скобках установлено соответствующее значение, называемое фильтром.

В общем случае в круглых скобках оператора __except может быть выражение, формирующее одно из трех значений. На рис. 5.1 константа EXCEPTION_EXECUTE_HANDLER, установленная в __except указывает на необходимость обработки возникшего исключения в данном блоке. Другие значения, которые можно указать в __except, позволяют повторить выполнение кода вызвавшего исключение, или обработать исключение в блоке __except на внешнем уровне вложенности.

Блоки кода механизма SEH могут быть вложенными. На рис. 5.2 приведен пример программы, применяющей вложенные блоки SEH-кода. Следует обратить внимание на применение оператора __finally, который определяет блок кода, всегда выполняющийся после кода блока __try независимо от результата его выполнения.

Следует отметить, что иногда для обработки исключений механизм SEH встраивают в стандартный механизм try/throw/catch языка C++. Такой подход позволяет упростить код. Подробно это описано в литературе [4].

5.1.2. Обработка исключений операций с плавающей точкой

По умолчанию операционная система Windows блокирует все исключения, возникающие при операциях с плавающей точкой. При возникновении исключений все равно формируется результат, но он должен интерпретироваться программистом как ошибка.

Для того, чтобы включить SEH-механизм, необходимо воспользоваться функцией _controlfp, которая позволяет изменить маску, управляющую обработкой исключений операций с плавающей точкой. На рис. 5.3 приведен пример фрагмента программы, изменяющей маску таким образом, чтобы стало возможным обработать исключения переполнения и деления на ноль.

```
#include "stdafx.h"
#include "windows.h"
#include <iostream>
int _tmain(int argc, _TCHAR* argv[])
{
 int  x = 0, y = 140;
 __try
 {
    __try
    {
     // охраняемый код
     std::cout << "Hello from try-code 1"  << std::endl;
     y /= x;
     std::cout << "y = " << y << std::endl;
     std::cout << "Hello from try-code 2"<< std::endl;
    }
    __finally
    {
     if (AbnormalTermination())
     {std::cout << "Abnormale Termination" <<  std::endl;}
     else
     {std::cout << "Normale Termination" <<  std::endl;};
    }
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    // код обработки исключения
    std::cout << "Hello from handler code" <<  std::endl;
  }
  system( "pause");
  return 0;
}
```

Рис. 5.2. Пример программы, применяющей финальный блок обработки исключения (__finally)

```
int cw = _controlfp(0,0);
cw&=~(EM_OVERFLOW|EM_ZERODIVIDE);
_controlfp(cw,_MCW_EM);
```

Рис. 5.3. Применение функции _controlfp для формирования маски SEH-исключений для операций с плавающей точкой

После того как маска SEH-исключений для операций с плавающей точкой будет установлена, обработка такого рода исключений может быть выполнена в обычном порядке.

5.2. Задания

Задание 18. **Обработка исключения «защита памяти»:**

1) разработайте программу, которая аварийно завершается с исключением «защита памяти» (EXEPTION_ACCESS_VIOLATION);

2) выполните программу и получите скриншот с ошибкой, сформированной операционной системой;

3) поместите в отчет по контрольной работе полученный скриншот и текст кода вашей программы;

4) скорректируйте программу таким образом, чтобы исключение обрабатывалось в блоке __except; обеспечьте трассировку хода выполнения программы с помощью сообщений, выводимых на консоль;

5) выполните программу, сделайте скриншот окна консоли;

6) поместите в отчет по контрольной работе полученный скриншот и откорректированный текст кода программы.

Задание 19. **Обработка исключений операций с плавающей точкой:**

1) разработайте программу, которая выполняет деление на ноль (делимое и делитель должны иметь тип double);

2) выполните программу пошагово с помощью отладчика;

3) с помощью отладчика исследуйте результат выполненной операции; получите скриншот, подтверждающий обнаруженный результат;

4) поместите в отчет по контрольной работе полученный скриншот и текст кода вашей программы;

5) скорректируйте программу таким образом, чтобы исключение «деление на ноль» для чисел с плавающей точкой обрабатывалось в блоке __except; обеспечьте трассировку хода выполнения программы с помощью сообщений, выводимых на консоль;

6) выполните программу, сделайте скриншот окна консоли;

7) поместите в отчет по контрольной работе полученный скриншот и откорректированный текст кода программы.

Задание 20. **Обработка SEH-исключений с помощью стандартного механизма try/catch обработки исключений C++:**

1) скорректируйте программу разработанную в задании 18, таким образом, чтобы обработка исключений осуществлялась с помощью стандартного механизма C++; обеспечьте трассировку хода выполнения программы с помощью сообщений, выводимых на консоль;

2) выполните программу, сделайте скриншот окна консоли;

3) поместите в отчет по контрольной работе полученный скриншот и откорректированный текст кода программы.

Задание 21. **Контрольные вопросы:**

1) объясните принцип работы стандартного механизма обработки исключений языка C++;

2) объясните принцип работы механизма SEH;

3) поясните принцип применения SHE-оператора __finally;

4) поясните особенности обработки исключений для операций с плавающей точкой;

5) поясните принцип встраивания механизма SEH в стандартный механизм обработки исключений языка C++.

Практическая работа № 6
УПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯМИ И ГРУППАМИ

6.1.Теоретические сведения

6.1.1. Пользователи и группы пользователей Windows

Любые действия, осуществляемые в операционной системе (выполнение программы, создание, изменение файла и т. п.), всегда выполняются от чьего-то лица. С другой стороны, все объекты операционной системы (потоки, процессы, объекты синхронизации, файлы и т. п.) имеют своего владельца. На этом принципе построена дискреционная защита – контроль доступа, используемый в операционных системах.

Прежде чем любой пользователь сможет работать в среде операционной системы Windows, он должен быть зарегистрирован администратором системы. Администратор системы – пользователь, обладающий специальными возможностями. При установке операционной системы всегда создается первый администратор (задается его имя и пароль), который в последствии может создавать (регистрировать) других пользователей, в том числе и обладающих возможностями администратора.

Каждому пользователю операционной системы Windows соответствует учетная запись, которая хранится в базе данных менеджера учетных записей. Эта база данных является частью системного реестра Windows.

Кроме учетных записей администратора и пользователей, созданных администратором, есть несколько специальных учетных записей, используемых операционной системой для внутренних целей и сервисной поддержки программного и аппаратного обеспечения.

Каждый пользователь Windows имеет имя и может иметь пароль. Эти данные позволяют нему подключиться к операционной системе. С каждым объектом безопасности операционной системы (объекты, имеющие имя) связаны специальные списки, характеризующие права доступа пользователей к этим объектам.

Для того чтобы просмотреть список учетных записей пользователей, создать, изменить свойства или удалить учетные записи, можно воспользоваться оснасткой «Управление компьютером» mmc-консоли, доступ к которой можно получить через опцию «Компьютер/Управление» меню «Пуск». На рис. 6.1 представлен внешний вид оснастки «Управление компьютером».

Другим важным объектом операционной системы, для которого тоже создается учетная запись в базе данных менеджера учетных записей, является группа пользователей. Группа пользователей также имеет имя, может быть создана, изменена или удалена с помощью оснастки «Управление компьютером» (рис. 6.1). Основное назначение группы – объединять пользователей, имеющих одинаковые характеристики доступа к объектам операционной системы.

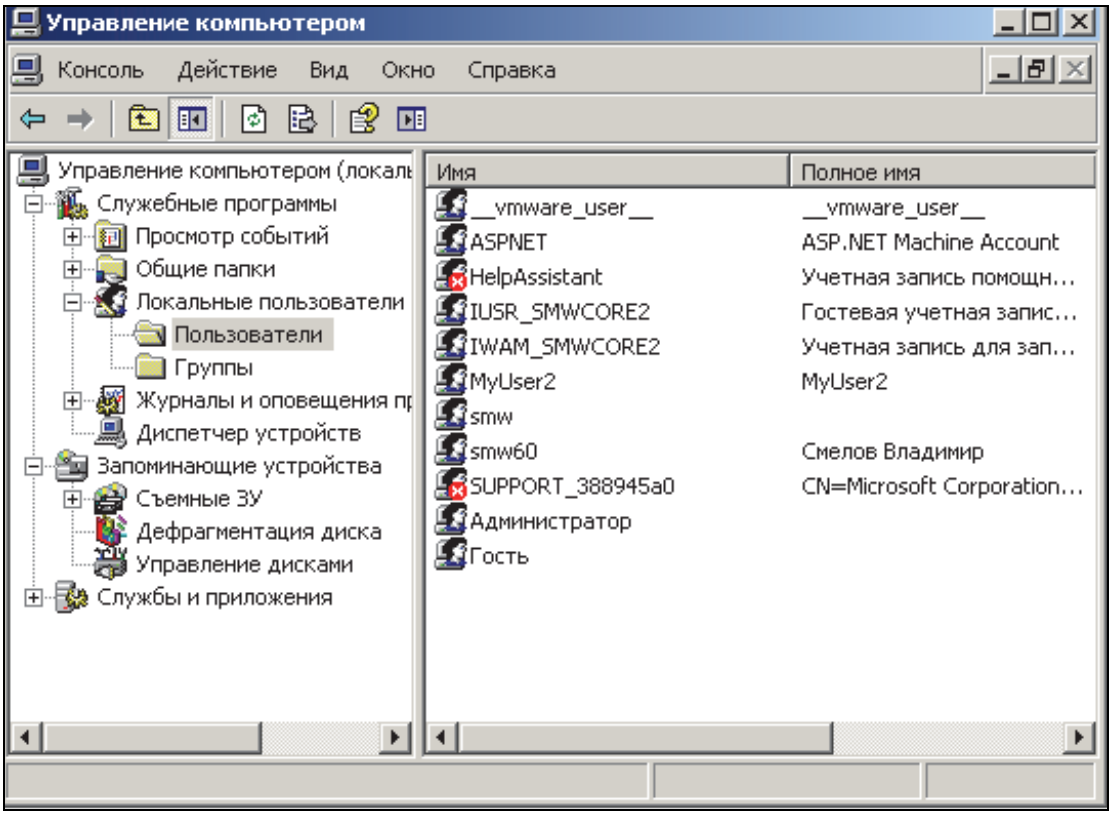

Рис. 6.1. Оснастка «Управление компьютером» mmc-консоли

Разрешение на доступ и выполнение операций может быть назначено как пользователю, так и группе пользователей. В последнем случае назначенное разрешение распространяется на всех пользователей группы. Один пользователь может состоять в нескольких группах, что позволяет ему унаследовать разрешения, назначенные всем эти группам. Другими словами, группу пользователей можно рассматривать как поименованный набор разрешений. Как и в случае с пользователями, существуют предопределенные группы, имеющие специальное назначение.

Следует отметить, что в рамках данной практической работы рассматриваются только локальные пользователи и группы. Информация

об этих пользователях и группах хранится на локальном компьютере и служит для управления доступом к объектам операционной системы локального компьютера. Операционная система Windows позволяет создавать доменные учетные записи пользователей и групп, действие которых распространяется на несколько связанных сетью компьютеров (домен). Подробнее об этом можно прочитать в других источниках[4].

6.1.2. Программный интерфейс для управления пользователями

В состав API операционной системы Windows входит богатый перечень функций, позволяющих программисту управлять пользователями и группами пользователей операционной системы. Наличие такого программного интерфейса позволяет программистам использовать систему безопасности Windows в своих программах.

Все функции управления пользователями можно разбить на две группы: функции для работы с пользователями и функции для работы с группами.

Функции для работы с пользователями позволяют: прочитать, создать, удалить или изменить учетную запись, проверить или изменить пароль пользователя, а также получить перечень групп, к которым принадлежит пользователь. На рис. 6.2 приведен пример программы, вызывающей функцию создания учетной записи пользователя.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#include <lm.h>                        // для функций netapi32.lib
#pragma comment(lib,"netapi32.lib") // подключить библиотеку netapi32.lib
int _tmain(int argc, _TCHAR* argv[])
{
  setlocale(LC_ALL, "rus");
  USER_INFO_1 buf4;       // структура для информации о пользователе
  buf4.usri1_name = L"MyUser2";      // имя пользователя
  buf4.usri1_comment =L"Comment for MyUser2";
  buf4.usri1_password = L"123456";  //  пароль
  buf4.usri1_home_dir = NULL;
  buf4.usri1_priv = USER_PRIV_USER;
  buf4.usri1_flags = UF_SCRIPT;
  buf4.usri1_script_path = NULL;
  NET_API_STATUS ns4 = NetUserAdd(NULL,1,LPBYTE)&buf4,NULL);
  // если NET_API_STATUS = 0, то пользователь создан
  std::cout<<"NET_API_STATUS = "<<ns4<<std::endl;
  system("pause");
  return 0;
}
```

Рис. 6.2. Пример программы, создающей учетную запись пользователя

Функции для работы с группами пользователей позволяют: получить информацию о группах пользователей и перечнях пользователей входящих в группы, создать или удалить группу, изменить информацию о группе, добавить или удалить пользователей группы. На рис. 6.3 приведен пример программы, получающий перечень групп к которым принадлежит заданный пользователь.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#include <lm.h>                        // для функций netapi32.lib
#pragma comment(lib,"netapi32.lib") // подключить библиотеку netapi32.lib
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    GROUP_USERS_INFO_0  *buf3;     // структура для группы
    DWORD uc3 = 0, tc3 = 0;
    NET_API_STATUS ns3 = NetUserGetLocalGroups(NULL,
                                               L"smw60",  // имя пользователя
                                               0, LG_INCLUDE_INDIRECT,
                                               (LPBYTE*)&buf3,
                                               MAX_PREFERRED_LENGTH,
                                               &uc3,&tc3
                                               );
    if (ns3 ==  NERR_Success)
      for (int i = 0; i < uc3; i++)
      {std::wcout<<buf3[i].grui0_name<<std::endl;}
    NET_API_STATUS ns3x = NetApiBufferFree((LPVOID)buf3);
    system("pause");
    return 0;
}
```

Рис. 6.3. Пример программы, получающей перечень групп,
к которым принадлежит заданный пользователь

6.2. Задания

Задание 22. Разработка приложения с применением интерфейса управления пользователями:

1) создайте с помощью mmc-консоли группы пользователей с именами: Server и MyUsers;

2) создайте пользователей U1, U2 и U3;

3) в группу Server добавьте пользователей U1 и U2;

4) в группу MyUser добавьте пользователя U3;

5) включите пользователей U1 и U2 в стандартную группу с именем Администраторы;

6) разработайте программу SimpleServer;

7) программа циклически должна выводить меню:

- подключиться к серверу;
- создать пользователя;
- удалить пользователя;
- вывести пользователей группы;
- вывести группы пользователя;
- отключиться от сервера;
- завершить работу;

8) при выборе пункта меню «подключиться к серверу» программа SimpleServer должна запросить имя и пароль пользователя и осуществить подключение к операционной системе от имени заданного пользователя; в случае успешного подключения программа должна выдать сообщение «подключение выполнено», в случае ошибки – «ошибка подключения, код xxx», где xxx – код ошибки Windows;

9) при выборе пункта меню «создать пользователя» программа SimpleServer должна убедиться, что текущий пользователь (от лица которого выполняется программа) одновременно принадлежит группам Администраторы и Servers; если пользователь не удовлетворяет этим условиям, то выдается сообщение «в доступе отказано», иначе запрашивается информация о пользователе (имя, пароль, комментарии) и создается пользователь, входящий в группы MyUsers и Server; в случае успешного создания выдается сообщение «пользователь создан», иначе «ошибка создания пользователя, код xxx», где xxx – код ошибки Windows;

10) при выборе пункта меню «удалить пользователя» программа SimpleServer должна убедиться, что текущий пользователь (от лица которого выполняется программа) одновременно принадлежит группам Администраторы и Servers; если пользователь не удовлетворяет этим условиям, то выдается сообщение «в доступе отказано», иначе запрашивается информация о пользователе (имя, пароль); если такой пользователь существует в группе MyUsers – он удаляется, иначе выдается сообщение «пользователь не найден»; в случае успешного удаления выдается сообщение «пользователь удален», иначе «ошибка удаления пользователя, код xxx», где xxx – код ошибки Windows;

11) при выборе пункта меню «вывести пользователей группы» программа SimpleServer должна убедиться, что текущий пользователь (от лица которого выполняется программа) принадлежит хотя бы одной из групп Servers или MyUsers; если пользователь не удовлетворяет этим условиям, то выдается сообщение «в доступе отказано», иначе выдается список всех пользователей принадлежащих к тем же группам (Server и/или MyUsers), что и текущий пользователь;

12) при выборе пункта меню «вывести пользователей группы» программа SimpleServer должна убедиться, что текущий пользователь (от лица которого выполняется программа) принадлежит хотя бы одной из групп Servers или MyUsers; если пользователь не удовлетворяет этим условиям то выдается сообщение «в доступе отказано» иначе выдается список всех групп, к которым принадлежит текущий пользователь;

13) при выборе пункта меню «отключиться от сервера» программа SimpleServer должна подключиться к серверу от имени пользователя, который запустил программу (до первого подключения);

14) при выборе пункта меню «завершить работу» программа SimpleServer должна завершить свою работу.

15) поместите скриншоты, демонстрирующие создание пользователей и групп, в отчет по контрольной работе;

16) поместите текст кода программы SimpleServer в отчет по контрольной работе;

17) поместите скриншоты, демонстрирующие работу программы SimpleServer, в отчет по контрольной работе.

Задание 23. **Контрольные вопросы:**

1) поясните понятия «учетная запись», «пользователь», «группа пользователей»;

2) опишите с помощью какого инструментария можно создавать пользователей и группы пользователей?

3) приведите функцию Windows API, с помощью которой можно проверить пароль пользователя;

4) какие структуры используются для хранения информации о пользователях и группах пользователей Windows?

Практическая работа № 7
АСИНХРОННЫЙ ВВОД-ВЫВОД

7.1. Теоретические сведения

7.1.1. Системные вызовы файлового ввода-вывода

В состав практически любой операционной системы входит файловая система, определяющая порядок хранения, именования, способы организации на физических носителях, формат хранения и защиты данных. Для доступа к данным, расположенным на физических носителях, в состав операционной системы входят специальные программные компоненты называемые драйверами файловой системы.

Существует несколько видов операционных систем: FAT16, FAT32, NTFS, HFS, HPFS и др. Современные версии операционной системы Windows поддерживают файловые системы FAT32 и NTFS. Именно эти операционные системы рассматриваются в данной практической работе.

С точки зрения программиста файловая система – это наборы данных, организованные в файлы (поименованный набор данных в определенном формате), которые могут располагаться в иерархии каталогов, представляющих собой особый тип файлов, предназначенных для хранения ссылки на другие каталоги или файлы данных.

Для доступа к файлам программист использует набор функций (часто называемый API файловой системы), обеспечивающих системные вызовы, позволяющие создавать, удалять, читать, писать или изменять файлы.

На рис. 7.1 представлен пример программы, которая создает файл на диске и заполняет его данными.

Операции ввода-вывода являются относительно медленными. За время выполнения операции ввода-вывода могут быть выполнены миллионы инструкций процессора. Дисковые контроллеры (устройства управления дисковыми устройствами) представляют собой специализированные процессоры, работа которых может выполняться параллельно с работой центрального процессора.

Цикл заполнения файла данными на рис. 7.1 последовательно вызывает функцию WriteFile, которая записывает на диск данные порциями длинной 128 байт. Следует обратить внимание на то, что функция WriteFile, получившая управление не возвращает его до тех пор, пока не закончится операция ввода-вывода. Во время операции ввода-вывода поток (последовательность инструкций процессора) находится

в состояние ожидания завершения операции ввода-вывода. Операции, ожидающие завершения ввода-вывода, называются синхронными операциями.

API файловой системы Windows предоставляет программисту возможность использовать другой режим ввода-вывода, называемый асинхронным. Выполнение операции ввода-вывода в этом режиме программист может распараллелить с работой центрального процессора, что в некоторых случаях может существенно повлиять на производительность разработанного кода.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>
#define NF  128
#define NB (NF/sizeof(long))
int _tmain(int argc, _TCHAR* argv[])
{
  BOOL b  = FALSE;
  DWORD wb = 0;
  long buf[NB];
  DeleteFile(L"D:\exp.txt");                // удаление файла
  HANDLE f =  CreateFile(                   // создание и открытие файла
                         L"D:\exp.txt",     // полное имя
                         GENERIC_WRITE,
                         NULL,              // режим совместного использования
                         NULL,              // атрибуты защиты
                         OPEN_ALWAYS,
                         NULL,              // флаги и атрибуты
                         NULL);             // файл атрибутов
  for (int i = 0; i < NB; i++) buf[i] = (long)i;
  for (int i = 0; i < 1000; i++)
  {
     b =  WriteFile(f,                     // файл
                    (LPCVOID)buf,          // буфер
                     NF,                   // писать байт
                    &wb,
                    NULL);
   }
  CloseHandle(f);
  return 0;
}
```

Рис. 7.1. Пример программы, создающей и заполняющей файл на дисковом устройстве

7.1.2. Понятие асинхронного ввода-вывода

На рис. 7.2 приведен пример программы, выполняющей асинхронное чтение данных с дискового устройства. Следует обратить внимание на следующие моменты.

1. При открытии файла (функция CreateFile) специальный параметр (FILE_FLAG_OVERLAPPED) указывает на то, что операции будут осуществляться в асинхронном режиме.

2. В функции чтения (ReadFile) используется последний параметр (типа OVERLAPPED), применяемый для продвижения указателя позиции в файле.

3. Применяется функция WaitForSingleObject для ожидания завершения ввода.

4. В программе выполняется сдвиг значения текущей позиции.

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>
int _tmain(int argc, _TCHAR* argv[])
{
    char bufx[30];
    BOOL b = TRUE;
    DWORD realreadx = 0;
    OVERLAPPED ovlx; ovlx.Offset = ovlx.OffsetHigh = 0; ovlx.hEvent = NULL;
    HANDLE hfx  = CreateFile(L"D:\\exp.txt", GENERIC_READ,   // открыть для чтения
                             FILE_SHARE_READ,NULL,OPEN_EXISTING,
                             FILE_FLAG_OVERLAPPED,           // асинхронный режим
                             NULL);
    for (int k = 0; k < 10 && b; k++)
    {
      if ((b = ReadFile(hfx,(LPVOID)bufx,12,&realreadx, &ovlx))) // читать асинхронно
      {
         // здесь может быть код,
         // который  будет выполняться
         // параллельно с чтением из файла
         WaitForSingleObject(hfx,INFINITE);   // ждать завершения ввода
         ovlx.Offset+=realreadx;              // сдвинуть указатель позиции
      };
    };
    CloseHandle(hfx);
    return 0;
}
```

Рис. 7.2. Пример программы, выполняющей асинхронное чтение файла с дискового устройства

На рис. 7.2 комментарием указано место в программе, в котором может быть помещен код, который будет выполняться параллельно выводом данных из файла.

7.2. Задания

Задание 24. Исследование производительности асинхронного файлового ввода-вывода:

1) разработайте программу, которая в асинхронном режиме создает и заполняет файл (любой информацией), состоящий из 1 млн. строк длинной по 100 байт (цикл, состоящий из 1млн. итераций);

2) в рамках созданного в п. 1. приложения, разработайте функцию, которая выполняет 100 тыс. операций сложения (цикл, состоящий из 100 тыс. итераций);

3) осуществите в цикле вывода программы вызов функции (п. 2) таким образом, чтобы функция выполнялась параллельно с операцией вывода данных в файл;

4) с помощью системной функции clock из стандартной библиотеки C++ выполните измерение времени выполнения каждых 100 тыс. операций вывода данных; выведите эти значения результатов замера в окно консоли;

5) выполните программу;

6) поместите скриншот с результатами замера и текст программы в отчет по контрольной работе;

7) выполните п. п. 1–6, но вывод информации в файл осуществите в синхронном режиме;

8) с помощью Excel постройте диаграмму, отображающую два графика: зависимость времени выполнения асинхронных операций вывода от количества операций и зависимость времени выполнения синхронных операций вывода от количества операций;

9) поместите диаграмму в отчет по контрольной работе.

Задание 25. Контрольные вопросы:

1) поясните понятия «файловая система», «файл», «каталог», «операция ввода-вывода»;

2) перечислите известные вам типы файловых систем;

3) поясните понятие «синхронная операция ввода-вывода»;

4) поясните понятие «асинхронная операция ввода-вывода».

Практическая работа № 8
ПРИМЕНЕНИЕ ПОРТОВ ЗАВЕРШЕНИЯ

8.1.Теоретические сведения

8.1.1. Концепция портов завершения Windows

Порт завершения – это объект операционной системы, предназначенный для синхронизации работы параллельно работающих потоков с операциями асинхронного ввода-вывода.

На рис. 8.1 изображена схема приложения, использующего порт завершения.

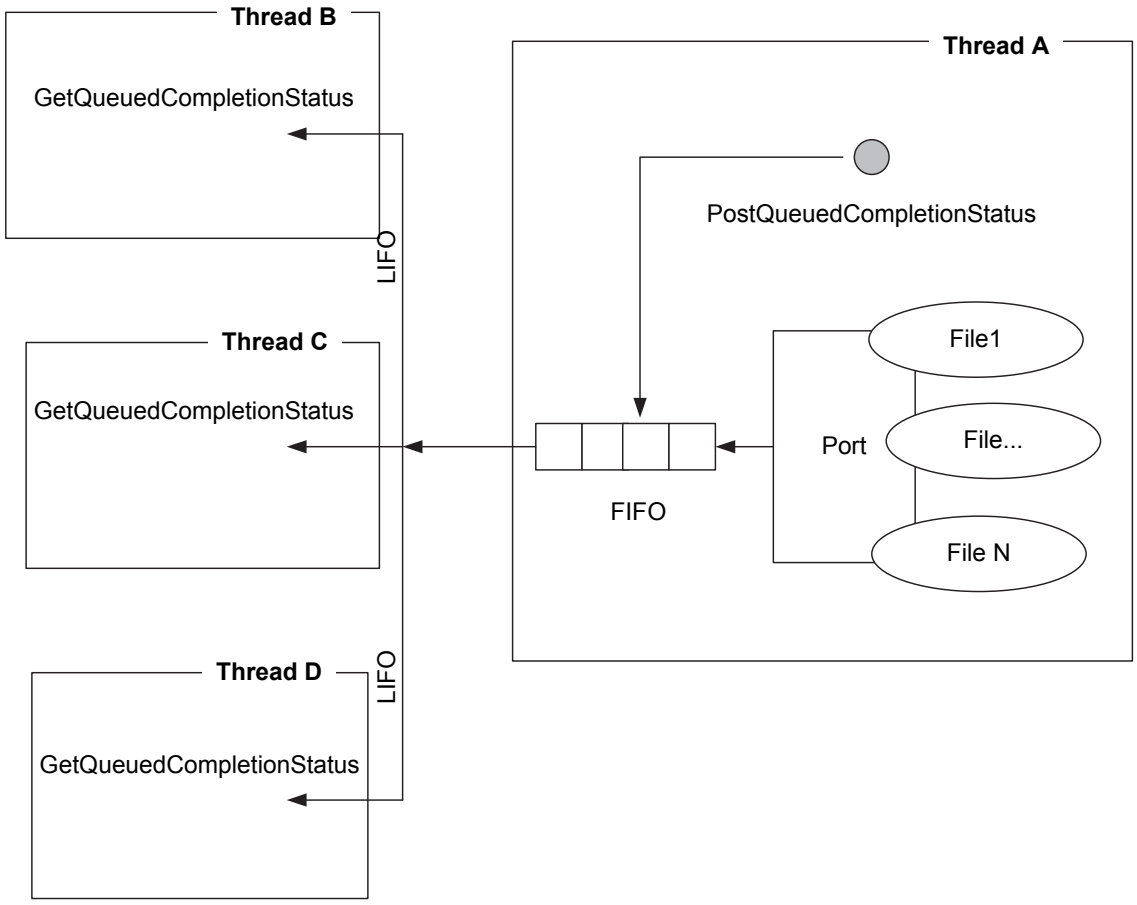

Рис. 8.1. Схема приложения, использующего порт завершения ввода-вывода

На рисунке изображены четыре потока (A, B. C, D) некоторого Windows-процесса. Поток A использует порт завершения, связанный с дескрипторами N файлов. Поток в определенном порядке выполняет операции асинхронного ввода или вывода для этих файлов. Порт

отслеживает завершение операций ввода или вывода для связанных с ним файлов. При завершении каждой операции порт формирует очередной элемент выходной FIFO-очереди. Элемент очереди содержит информацию о завершенной операции.

Потоки B, C и D предназначены для циклического считывания элементов выходной очереди порта. При получении из очереди элемента поток может определить файл, для которого была выполнена операция, значение текущей позиции файла и другую информацию.

Таким образом, порты завершения ввода-вывода используются в тех случаях, когда формирование блока выводимых данных или обработка блока входных данных требует значительных затрат процессорного времени. Причем обработка каждого блока может выполняться независимо друг от друга. В этом случае достигается с одной стороны распараллеливание операций ввода-вывода с вычислительными операциями процессора, с другой стороны – распараллеливание обработки нескольких блоков данных (это эффективно, если компьютер имеет несколько процессоров).

8.1.2. Принципы разработки приложениий с применением порта завершения

На рис. 8.2 приведен фрагмент программы, демонстрирующий создание порта завершения ввода-вывода и связывания его с файлами.

```
HANDLE hfx  = CreateFile(L"D:\\XXX.txt", GENERIC_READ,FILE_SHARE_READ,
                         NULL,OPEN_EXISTING,FILE_FLAG_OVERLAPPED,NULL);
HANDLE hfy  = CreateFile(L"D:\\YYY.txt", GENERIC_READ,FILE_SHARE_READ,
                        NULL,OPEN_EXISTING,FILE_FLAG_OVERLAPPED,NULL);

hp = CreateIoCompletionPort(hfx, NULL, (ULONG)77,2);  // создание порта и
     CreateIoCompletionPort(hfy, hp,   (ULONG)88,3);  // и связвание  с файлами
```

Рис. 8.2. Фрагмент программы, создающей порт завершения

Следует обратить внимание на функцию CreateIOComplectionPort, предназначенную для создания порта, первый параметр которой – дескриптор файла, второй – дескриптор порта (используется, если с портом связывается более одного файла), а предпоследний параметр – ключ (идентификатор) файла, который считывается из очереди в рабочих потоках.

На рис. 8.3 представлен пример потоковой функции, взаимодействующей с портом ввода-вывода. Функция в цикле вызывает функцию GetQueuedComplectionStatus, считывающую первый элемент очереди.

В том случае, если очередь пуста, поток переводится в состояние ожидания до появления элемента, указывающего на завершение операции асинхронного ввода-вывода. Параметры функции позволяют получить значение ключа файла, указатель на структуру OVERLAPPED, количество обработанных (считанных или записанных) байт.

```
DWORD WINAPI mythread(LPVOID)
{
    int k = 0;
    DWORD nb =  0;
    ULONG key = 0;
    LPOVERLAPPED lpovl;
    while(GetQueuedCompletionStatus(hp,&nb, &key, &lpovl,INFINITE))
    {
       std::cout<<std::endl<<k++<<": "
                <<"ThreadID: "<< GetCurrentThreadId()<< " key: "<< key;
       // обработка очередного прочитанного блока
    }
    return 0;
}
```

Рис. 8.3. Пример потоковой функции, взаимодействующей с портом

Таким образом, приложение, применяющее порт завершения, должно выполнить следующие действия:

1. Открыть один или несколько файлов в асинхронном режиме.
2. Создать порт и связать его с открытыми файлами.
3. Создать несколько потоков, которые считывают очередь порта.
4. Запустить цикл операций асинхронного ввода или вывода для связанных файлов.

8.2. Задания

Задание 26. Разработка приложения с применением порта завершения:

1) разработайте программу, формирующую три файла и заполняющую каждый из них 10 блоками по 100 байт;

2) разработайте программу с применением порта завершения, которая считывает данные из трех файлов, подготовленных на предыдущем шаге задания, и использует пять потоков, обрабатывающих считанные блоки;

3) обеспечьте трассировку работы программы с помощью сообщений, выводимых в консоль;

4) выполните программу;

5) поместить скриншот трассировки и текст кода программы в отчет по контрольной работе.

Задание 27. **Контрольные вопросы:**

1) поясните понятия «порт завершения», «потоковая функция», «очередь FIFO», «ключ файла»;

2) перечислите порядок действий приложения, применяющего порт завершения;

3) опишите, в каких случаях целесообразно применять порт завершения ввода-вывода.

Практическая работа № 9
РАЗРАБОТКА WINDOWS-СЕРВИСА

9.1.Теоретические сведения

9.1.1. Назначение Windows-сервисов

Windows-сервис – это процесс операционной системы, выполняющий служебные функции. Как правило, сервис стартует при загрузке операционной системы и заканчивает свою работу при завершении работы операционной системы. Часто сервисы называют службами.

Обычно сервисы выполняют служебные функции, необходимые для какого-нибудь приложения. Иногда сервисы используются для доступа к внешним устройствам, они называются драйверами. В виде сервисов реализованы сетевые службы, такие как DNS, FTP, Telnet, DHCP и другие. Сам сервис может быть как консольным приложением, так и приложением с графическим интерфейсом.

Для просмотра действующих на локальном компьютере Windows-сервисов может быть использована mmc-оснастка «Службы» (рис. 9.1).

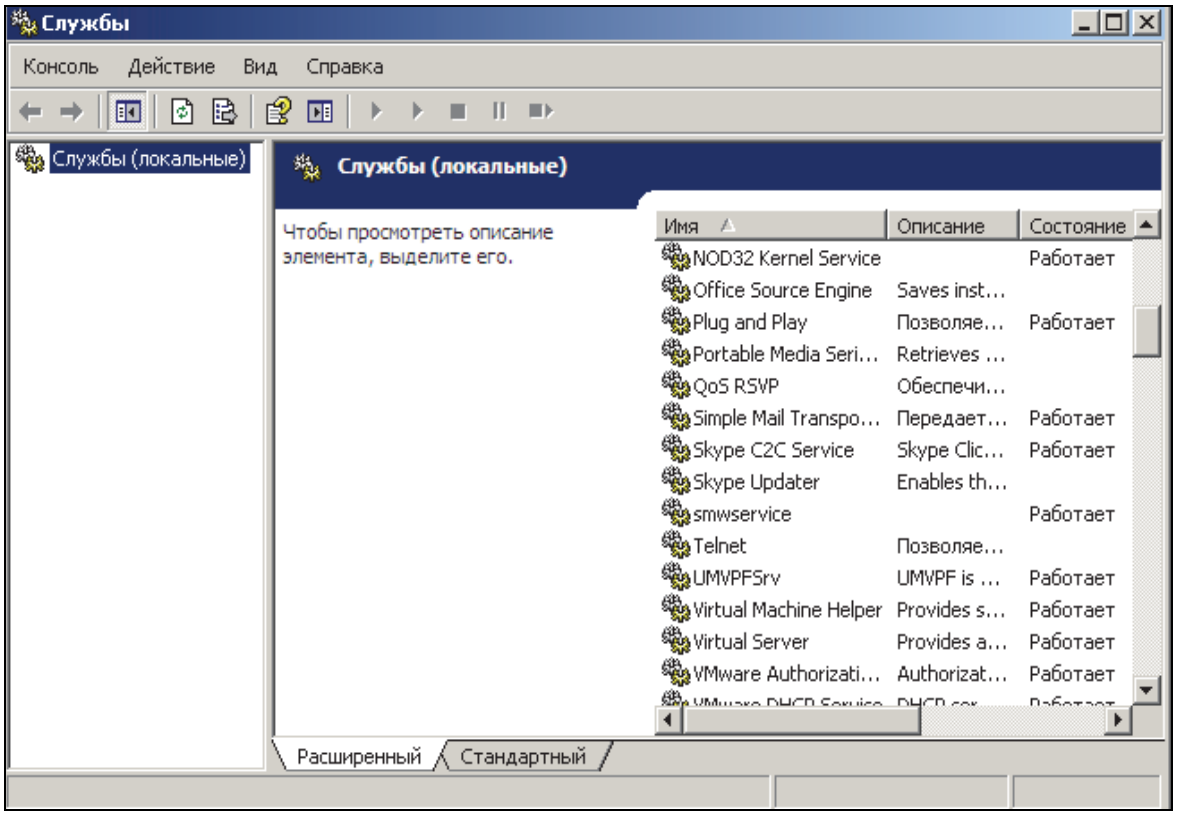

Рис. 9.1. Внешний вид mmc-оснастки «Службы»

Оснастка позволяет остановить или запустить работу службы, просмотреть и поменять некоторые свойства.

9.1.2. Принципы разработки Windows-сервиса

Обычно windows-сервисы (далее просто сервисы) разрабатываются как консольное приложение. Приложение, как правило содержит несколько функций предназначенных для: установки (регистрации) сервиса, запуска и остановки сервиса, для реализации сервиса.

Обычно точка входа сервиса имеет имя ServiceMain. На рис. 9.2 представлен пример реализации функции ServiceMain.

```
VOID WINAPI ServiceMain(DWORD dwArg,  LPWSTR  *lpszArgv)
{
    DWORD   status = 0, DWORD   specificError = 0xffffffff;
    ServiceStatus.dwServiceType         = SERVICE_WIN32;
    ServiceStatus.dwCurrentState        = SERVICE_START_PENDING; //Service State
    ServiceStatus.dwControlsAccepted    = SERVICE_ACCEPT_STOP |
                                          SERVICE_ACCEPT_SHUTDOWN |
                                          SERVICE_ACCEPT_PAUSE_CONTINUE;
    ServiceStatus.dwWin32ExitCode       =    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwCheckPoint          =    ServiceStatus.dwWaitHint           = 0;
    // регистрация обработчика управляющих команд
    hServiceStatusHandle = RegisterServiceCtrlHandler(L"smwservice", ServiceHandler);
    if (hServiceStatusHandle==0)
    {
        long nError = GetLastError(); char pTemp[121];
        sprintf_s(pTemp, "RegisterServiceCtrlHandler failed, error code = %d\n", nError);
        return;
    }
    ServiceStatus.dwCurrentState        = SERVICE_RUNNING;
    ServiceStatus.dwCheckPoint          =  ServiceStatus.dwWaitHint     = 0;
    if(!SetServiceStatus(hServiceStatusHandle, &ServiceStatus))
    {
        long nError = GetLastError();char pTemp[121];
        sprintf_s(pTemp, "\nSetServiceStatus failed, error code = %d\n", nError);
    }
}
```

Рис. 9.2. Пример реализации функции ServiceMain

Основное назначение функции ServiceMain – определить текущее состояние сервиса (структура ServiceStatus и функция SetServiceStatus), а также зарегистрировать обработчик управляющих команд (функция SetServiceStatus).

Обработчик управляющих команд – это функция, имеющая обычно имя ServiceHandler и предназначенная для обработки команд, поступающих сервису, например, через mmc-консоль. Именно эта функция определяет перечень команд, на которые реагирует сервис. На рис. 9.3 представлен фрагмент этой функции. Функция принимает

единственный параметр, который интерпретируется как команда, которую следует выполнить. В зависимости от команды устанавливается новый статус сервиса.

```
VOID WINAPI ServiceHandler(DWORD dwControl)
{
   switch(dwControl)
    {
        case SERVICE_CONTROL_STOP:
        case SERVICE_CONTROL_SHUTDOWN:
            ServiceStatus.dwWin32ExitCode = 0;
            ServiceStatus.dwCurrentState  = SERVICE_STOPPED;
            ServiceStatus.dwCheckPoint    = 0;
            ServiceStatus.dwWaitHint      = 0;
            break;
        case SERVICE_CONTROL_PAUSE:
            ServiceStatus.dwCurrentState = SERVICE_PAUSED;
            stopcycle = false;
            break;
        case SERVICE_CONTROL_CONTINUE:
            ServiceStatus.dwCurrentState = SERVICE_RUNNING;
            stopcycle = true;
            break;

   }        // ........
   if (!SetServiceStatus(hServiceStatusHandle,  &ServiceStatus))
   {
            long nError = GetLastError();
            char pTemp[121];
            sprintf_s(pTemp, "SetServiceStatus failed, error code = %d\n", nError);
  }
}
```

Рис. 9.3. Фрагмент кода функции обработчика управляющих команд

9.2. Задания

Задание 28. Разработка простейшего Windows-сервис:

1) разработайте приложение, которое регистрирует Windows-сервис или удаляет его регистрацию в зависимости от входного параметра;

2) windows-сервис должен формировать и записывать каждые 10 секунд строку, содержащую текущее время в текстовый файл;

3) windows-сервис должен реагировать на команды «Стоп», «Пауза», «Перезапустить», введенные через mmc-консоль;

4) после регистрации сервиса получите скриншот mmc-консоли, демонстрирующий наличие сервиса среди служб; поместите скриншот в отчет по лабораторной работе;

5) обеспечьте трассировку выполнения команд с помощью сообщений, записанных в текстовый файл;

6) поместите текст кода разработанного сервиса в отчет по лабораторной работе.

Задание 29. **Контрольные вопросы:**

1) поясните понятие «windows-сервис»;

2) поясните назначение windows-сервисов;

3) опишите, каким способом можно получить перечень действующих на локальном компьютере windows-сервисов;

4) поясните структуру windows-сервиса.

ЛИТЕРАТУРА

1. Пацей, Н. В. Основы алгоритмизации и программирования / Н. В. Пацей. – Минск: БГТУ, 2010. – 285 с.

2. Таненбаум, Э. Операционные системы: разработка и реализация / Э. Таненбаум, А. Вудхал. – СПб.: Питер, 2006. – 576 с.

3. Роджерсон, Д. Основы COM / Д. Роджерсон. – М.: Русская Редакция, 2001. – 400 с.

4. Побегайло, А. Н. Системное программирование в Windows / А. Н. Побегайло. – СПб.: БХВ-Петербург, 2006. – 1056 с.

5. Мэтью, Н. Основы программирования в Linux / Н. Мэтью, Р. Стоунс. – СПб.: БХВ-Петербург, 2009. – 896 с.

6. Лав, Р. Linux. Системное программирование / Р. Лав. – СПб.: Питер, 2008. – 416 с.

7. Лейнкер, Р. COM+. Энциклопедия программиста / Р. Лейнкер. – СПб.: ДиаСофтЮП, 2002. – 656 с.

ОГЛАВЛЕНИЕ

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Методические указания к выполнению контрольной работы

Составитель: **Смелов** Владимир Владиславович

Редактор *К. В. Великода*
Компьютерная верстка *К. В. Великода*
Корректор *К. В. Великода*

Издатель:
УО «Белорусский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя
распространителя печатных изданий
№ 1/227 от 20.03.2014
Ул. Свердлова, 13а, 220006, г. Минск