Министерство науки и высшего образования Российской Федерации
Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева

КОНСТРУИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS

Методические указания к выполнению лабораторных работ для студентов бакалавриата по направлению подготовки 09.03.04 «Программная инженерия» всех форм обучения

Красноярск 2025 г.

УДК 004.4'2 004.43 004.9

Конструирование программного обеспечения в операционной системе Windows: метод. указания к выполнению лаб. работ для студентов напр. 09.03.04 «Программная инженерия» всех форм обучения / сост.: А. Г. Зотин; СибГУ им. М. Ф. Решетнева. – Красноярск, 2025. – 40 с.

ОГЛАВЛЕНИЕ

ОБЩИЕ СВЕДЕНИЯ

В современном мире программные продукты находят все более широкое применение в различных сферах человеческой деятельности. При этом эти продукты обладают большими функциональными возможностями, реализованными с учетом многопоточности обработки данных и заимствования функциональных возможностей из готовых динамически подключаемых библиотек (Dinamic Link Libraries, DLL). Некоторые программные продукты и системы поддерживают механизмы расширения функционала (плагины).

Дисциплина «Конструирование программного обеспечения» нацелена на расширение теоретических знаний в области разработки прикладного программного обеспечения и получение практических навыков по разработке многомодульных систем и программных комплексов.

Данные методические указания предназначены для освоения методов разработки и реализации комплексных программных систем с поддержкой расширения функционала. При этом основной упор делается на получение практических навыков создания многомодульных приложений, основанных на использовании динамически подключаемых библиотек и системы модулей расширения (плагинов). Также рассматриваются использование многопоточной реализации обработки данных, организация межпрограммного/межмодульного взаимодействия, реализация механизма защиты с использованием ключа на основе USB Flash Drive, создание инсталляционного комплекта для собственного программного продукта.

В методических указаниях приведены основные теоретические сведения по конструированию программного обеспечения в операционной системе Windows, описания семи лабораторных работ и библиографический список с рекомендуемой для изучения литературой.

Методические указания предназначены для студентов бакалавриата, обучающихся по направлению подготовки 09.03.04 «Программная инженерия», всех форм обучения. В зависимости от формы обучения объем индивидуального задания на лабораторных занятиях устанавливает преподаватель.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Разработка динамических библиотек

Простые программы обычно состоят из одного исходного файла. Но если программа становится большой, то рекомендуется разбивать ее на части, которые функционально ограничены и закончены, и помещать эти части в библиотеки. Это вызвано тем, что процесс правки при большом исходном тексте становится сложным и поиск даже незначительной ошибки может повлечь за собой вынужденный пересмотр кода заново. К тому же библиотеки являются хорошим способом повторного использования кода: вместо того чтобы каждый раз реализовывать одни и те же подпрограммы (писать одни и те же фрагменты кода) в каждом вновь создаваемом приложении, их можно разработать единожды и затем ссылаться на них из других приложений.

Динамически подключаемые (динамические) библиотеки (Dynamic Link Libraries, DLL) являются хранилищем общедоступных процедур. Механизм DLL-библиотек возник вместе с операционной системой Windows и является ее неотъемлемой частью. Суть этого механизма состоит в том, что в процессе компоновки исполняемого модуля с применением внешних процедур в него помещаются не сами процедуры, а только их названия (номера) вместе с названиями DLL-библиотек, в которых они содержатся.

Использование динамических библиотек позволяет значительно упростить и саму разработку программного обеспечения. Так, вместо того чтобы каждый раз компилировать огромную EXE-программу, достаточно перекомпилировать лишь ее отдельный динамический модуль. Кроме того, доступ к динамической библиотеке возможен сразу из нескольких исполняемых модулей, что делает многозадачность более гибкой. Еще одним достоинством DLL-библиотек является то, что помимо программного кода они могут содержать и ресурсы. При вызовах процедур и функций каждая программа использует определенные соглашения о вызовах, которые определяют следующие параметры вызова:

– порядок передачи параметров, в том числе использование регистров;

– порядок очистки стека;

– преобразование имени подпрограммы.

При создании DLL-библиотек и работе с ними, особенно если библиотека и применяющая ее программа написаны на разных языках

программирования, необходимо следить за тем, чтобы при реализации и вызове подпрограмм использовались одни и те же соглашения о вызовах.

В языке C++ основными соглашениями о вызовах являются **_cdecl**, **_stdcall**, **_fastcall** и **_thiscall** (табл. 1). По умолчанию используется соглашение **_cdecl**. Модификатор, определяющий соглашение о вызовах, записывается после типа результата функции:

```
void _stdcall MyFunction(double x) { … }
```

Таблица 1

Соглашение о вызовах, используемые в языке C++

Соглашение о вызовах	Порядок следования параметров	Очистка стека	Использование регистров
_cdecl	Справа налево	Вызывающая программа	+
_stdcall	Справа налево	Вызываемый метод	+
_fastcall	Справа налево	Вызываемый метод	+
_thiscall	Справа налево	Вызываемый метод	+

В языке C++ соглашения о вызовах подразумевают использование регистров для передачи результата функции. Если размер результата не превышает двойного слова, то результат передается через регистр **EAX**. Если размер результата не превышает восемь байт, то результат передается через регистровую пару **EDX:EAX**. В противном случае через регистр **EAX** передается адрес результата функции.

В языках Паскаль и Delphi для указания соглашения о вызовах используются директивы **register**, **pascal**, **cdecl**, **stdcall** и **safecall** (табл. 2). Директивы размещаются в конце заголовка подпрограммы:

```
function MyFunction(x: real): real; cdecl;
```

Таблица 2

Директивы, применяемые для указания соглашения о вызовах в языках Паскаль и Delphi

Директива	Порядок следования параметров	Очистка стека	Использование регистров
register	Слева направо	Вызываемый метод	+
pascal	Слева направо	Вызываемый метод	–
cdecl	Справа налево	Вызывающая программа	–
stdcall	Справа налево	Вызываемый метод	–
safecall	Справа налево	Вызываемый метод	–

Исходный код DLL-библиотеки имеет схожую структуру во всех языках программирования и содержит следующие компоненты:

– необязательную часть кода, которая отвечает за инициализацию и очистку библиотеки;

– набор подпрограмм библиотеки (процедуры/функции);

– явное указание, какие подпрограммы должны экспортироваться из библиотеки.

Инициализирующая часть кода оформляется в виде процедуры, которая получает ряд параметров, один из которых указывает на причину обращения. Этот параметр может принимать одно из следующих значений:

– **DLL_PROCESS_ATTACH** – библиотека подключена к вызывающему процессу;

– **DLL_PROCESS_DETACH** – библиотека отсоединена от вызывающего процесса;

– **DLL_THREAD_ATTACH** – процесс создает новый поток;

– **DLL_THREAD_DETACH** – процесс завершает поток.

Процедура должна возвращать значение «истина» в случае успешного завершения, и значение «ложь» при возникновении ошибки. В этом случае дальнейшие действия прекращаются.

2. Работа с динамическими библиотеками

Прежде чем вызывать подпрограммы, включённые в DLL-библиотеку, их необходимо импортировать в приложение. Это можно сделать двумя способами: путем объявления внешней процедуры или функции (статическое подключение) или динамически с помощью функций Win32 API. Но в любом случае подпрограммы из DLL-библиотеки будут импортированы только во время работы приложения.

Наиболее просто статическое подключение реализуется на языках Delphi и Pascal, в которых объявление внешней процедуры и указание подключаемой библиотеки осуществляется с помощью директивы **external**:

```
procedure <имя> (<список параметров>); (<Соглашение о вызовах>;
external <строка, задающая имя библиотеки>;
function <имя> (<список параметров>): <тип результата>; <Согла-
шение о вызовах>; external <строка, задающая имя библиотеки>;
```

Если включить в код приложения подобное объявление, то библиотека, имя которой задано после ключевого слова **external**, будет загружаться при запуске приложения. Имя импортируемой процедуры или функции будет означать одну и ту же подпрограмму

из одной и той библиотеки в течение всего времени выполнения приложения.

В языке C# статическое подключение выглядит следующим образом:

```
[DllImport("<строка, задающая имя библиотеки>", опции)]
public static extern <тип результата> <имя>(<список параметров>);
```

В качестве опций могут выступать параметры кодировок и соглашения о вызове. Например, при использовании кодировки Unicode и соглашения **Cdecl** строка с опциями имеет вид

```
CharSet=CharSet.Unicode,
CallingConvention=CallingConvention.Cdecl
```

Для организации взаимодействия языка C# с DLL-библиотекой необходимо использовать пространство имен **System.Runtime.InteropServices**.

В случае если DLL-библиотека создается на языке C++ в Visual Studio, то при ее сборке формируются три файла: **.dll**, **.lib** и **.exp**. Файл **.lib** – это таблица импорта, файл **.exp** – таблица экспорта. Чтобы статически подключить DLL-библиотеку, необходимо:

– подключить **h**-файл DLL к разрабатываемой программе;

– подключить **lib**-файл к разрабатываемой программе.

Так, при наличии **h**-файла DLL нужно скопировать файлы **.h** и **.lib** в папку с программой, затем в начале программы (после блока подключения основных модулей) ввести следующий код:

```
#include "MyDll.h"  //Подключение h-файла библиотеки
#pragma comment(lib, "MyDll.lib") //Подключение lib-файла
```

При необходимости можно записать полный путь к этим файлам. В случае отсутствия **h**-файла для объявления функции используется следующая конструкция:

```
extern "C" __declspec(dllimport) <тип результата> <имя> (<список параметров>)
```

Если DLL-библиотека разработана не на языке C++, то для ее статического подключения в C++ необходимо сгенерировать lib-файл[1], и перед этим – файл определений **.def**, содержащий перечень функций, экспортируемых из DLL-библиотеки. Для окончательной генерации файла **.lib** с использованием **def-**файла в командной строке нужно выполнить команду

```
lib /def:C:\path\some.def /out:C:\path\some.lib /machine:x86
```

[1] Более подробно об этом см. в статье «Генерация .lib из DLL с помощью Visual Studio» (URL: http://xbb.uz/dev/Gjenjeracija-.lib-iz-DLL-s-pomoshhju-Visual- Studio).

При отсутствии **def-**файла его нужно создать, например, в блокноте.

Для того чтобы узнать, какие функции содержатся в DLLбиблиотеке, можно воспользоваться программой Dependency Walker или командой

```
dumpbin /exports C:\path\to\some.dll
```

Динамическое подключение DLL-библиотеки предполагает, что она подключается к приложению в момент его исполнения. Для правильного взаимодействия приложения с DLL-библиотекой необходимо использовать API-функции **LoadLibrary**, **GetProcAddress** и **FreeLibrary**. В общем виде схему работы с DLL-библиотекой можно представить следующим образом.

Шаг 1. Задать в программе тип-определитель функции (делегат для C#), который должен совпадать с прототипом импортируемой из библиотеки функции:

– на языке Delphi/Object pascal:

```
type
FType = function(<список параметров>):<тип результата>;
<Соглашение о вызовах>;
```

– на языке C++:

```
typedef <тип результата> (<Соглашение о вызовах> *FType) (<список параметров>);
```

– на языке C#:

```
[UnmanagedFunctionPointer(<Соглашение о вызовах>)] delegate <тип результата> FType(<список параметров>);
```

Шаг 2. Загрузить DLL-библиотеку с помощью функции **LoadLibrary** или **LoadLibraryEx**:

– на языке Delphi/ Object pascal:

```
Var hLib: THandle = 0;
hLib := LoadLibrary('some.dll');
```

– на языке C++:

```
HINSTANCE hLib = LoadLibrary(TEXT("some.dll"));
```

– на языке C#:

```
int hLib = LoadLibrary(@"some.dll");
```

Примечание. В среде разработки RAD Studio возможно использование функции **Safeload Library**.

Шаг 3. Использовать функцию **GetProcAddress** для получения указателя на интересующую функцию и полученный результат назначить переменной с соответствующим типом определения указателя данной функции:

– на языке Delphi/Object pascal:

```
Var FTName: FType;
FTName:= GetProcAddress(hLib, PChar('Имя_функции'));
```

– на языке C++:

```
FTName = (FType)GetProcAddress(hLib, "Имя_функции");
```

– на языке C#:

```
IntPtr pProc;
pProc = GetProcAddress(hLib, "Имя_функции");
FTName = (FType)Marshal.GetDelegateForFunctionPointer(pProc,
typeof(FType));
```

Шаг 4. Использовать функцию через соответствующую переменную.

Шаг 5. Выгрузить DLL-библиотеку с помощью функции **FreeLibrary** по окончании работы. Код выгрузки библиотеки одинаков для всех языков программирования:

```
FreeLibrary(hLib);
```

Следует помнить о том, что после загрузки DLL-библиотеки нужно проверять переменную, содержащую дескриптор DLL, для того, чтобы удостовериться, что она загружена успешно. Также важно проверять корректность полученного указателя на интересующую функцию. Так, для выявления ошибки на языке Delphi можно проверять адрес на равенство **nil (@FTName = nil**), на языке C++ – на значение **NULL** (**FTName == NULL**), на языке C# – на равенство **IntPtr.Zero (pProc == IntPtr.Zero)**.

В случае языка Python для работы с DLL-библиотеками понадобится использование модуля ctypes, который представляет собой совместимые с языком C типы данных и позволяет вызывать функции в библиотеках DLL:

```
import ctypes
```

В общем виде схему работы с DLL-библиотекой на языке Python можно представить следующим образом

Шаг 1. Загрузить DLL-библиотеку с помощью метода ctypes.CDLL в который передается имя файла.

```
Объект_dll = ctypes.CDLL('some.dll')
```

Следует помнить о том, что после загрузки DLL-библиотеки возможно использование двух подходов при работе с её функциями.

Шаг 2 (*вариант 1*). Используя прямое указание функции для объекта загруженной DLL-библиотеки выполняется определение типов аргументов и возвращаемого значения:

```
Объект_dll.Имя_функции.argtypes = [перечень элементов ctypes.тип]
Объект_dll.Имя_функции.restype = ctypes.тип
```

Шаг 2 (*вариант 2*). Создание объекта функции на основе адреса полученного по имени функции для объекта загруженной DLL-библиотеки и определение типов аргументов и возвращаемого значения:

```
Объект_функции = getattr(Объект_dll, Имя_функции)
Объект_функции.argtypes = [перечень элементов ctypes.тип]
Объект_функции.restype = ctypes.тип
```

Осуществить проверку существования функции с указанным именем в загруженной DLL-библиотеке можно воспользовавшись функцией hasattr:

```
hasattr(Объект_dll, Имя_функции)
```

Шаг 3. Использовать функцию соответственно выбранному варианту использования:

```
# Вариант 1
Результат = Объект_dll.Имя_функции([передаваемые_значения]
# Вариант 2
Результат = Объект_функции([передаваемые_значения]
```

Рекомендуется использовать конструкцию **try…except** (**try…catch**) для определения возможных ошибок входе загрузки DLL-библиотеки и использования функций.

3. Типовой подход к разработке систем, использующих динамически подключаемые модули расширения (плагины)

Требования, предъявляемые к программному обеспечению, постоянно меняются. Зачастую этими требованиями являются изменение или добавление возможностей программы. Конечно, для этого можно каждый раз просто дописывать существующий код программы, но если делать это постоянно, то код программы заметно увеличится, что значительно усложнит её сопровождение и повысит вероятность нежелательных побочных эффектов. Одним из шаблонных способов решения этой проблемы являются плагины.

Плаги́н (от англ. *plug-in*) – независимо компилируемый программный модуль, динамически подключаемый к основной программе (ядру) и предназначенный для расширения и/или использования ее возможностей. По сути плагин – это маленькая программа, которая встраивается в основную программу и тем самым расширяет ее возможности. Чаще всего основной программой является **.exe**-файл, а в качестве плагинов выступают **.dll**-файлы.

Основная программа (ядро) предоставляет сервисы, которые плагин может использовать, например возможность зарегистрировать себя в ядре и протокол обмена данными с другими плагинами.

Плагины являются зависимыми от сервисов, предоставляемых ядром, и отдельно, т. е. сами по себе, не используются. Под *протоколом обмена данными с другими плагинами*, также называемым Application Programming Interface (API), понимается набор правил (функций), которому соглашаются следовать ядро и плагины, чтобы понять друг друга и успешно взаимодействовать. Все сервисы могут предоставляться только в рамках этого протокола.

Как правило, на этапе конструирования архитектуры будущего программного продукта определяются такие функциональные возможности программы, которые впоследствии могут быть модифицированы или дополнены. Например, если программа должна анализировать изображение, то можно сразу предположить, что эту возможность следует реализовать с помощью технологии плагинов. Тогда впоследствии можно будет лишь дописать отдельный модуль программы, а не перекомпилировать весь программный продукт. Кроме того, если после добавления модуля будут возникать нежелательные побочные эффекты, то достаточно будет исправить конкретный модуль, выпустив его новую версию, а не искать место, вызывающее сбой по всему коду приложения.

На стадии разработки системы следует определить, каким образом будут реализованы API к плагинам. В большинстве случаев рекомендуется применять API, независимые от сред разработки, т. е. реализуемые на чистом языке программирования, за счет чего достигается многоплатформенность кода. Однако могут возникнуть ситуации, когда использовать чистый язык программирования либо не удается, либо не возможно. В подобных ситуациях появляется связь со средой разработки и, как следствие, универсальность кода резко сокращается.

Для организации взаимодействия с плагином самым простым способом является создание набора функций (API), которые будут описывать взаимодействие ядра с плагинами, например таких:

– **GetInfo()** – для получения информации о плагине в виде структуры[1];

– **GetGUID()** – для получения информации об уникальном идентификаторе плагина GUID (Globally Unique Identifier)[2];

– **GetPluginType()** – для выявления типа плагина в случае, если предполагается использование плагинов для разных нужд;

– **DoWork(<список аргументов>)** – для выполнения обработки;

[1] В качестве альтернативы для каждого вида информации может быть создана своя функция

[2] Уникальный 128-битный идентификатор, записывается в виде строки из тридцати двух шестнадцатеричных цифр, разбитой на группы дефисами и опционально окружённой фигурными скобками: {6F9619FF-8B86-D011-B42D-00CF4FC964FF}

– **GetGUIinfo()** – для получения параметров настройки интерфейса приложения;

– **ShowGUI()** – для непосредственного вызова диалогового окна настроек параметров работы плагина.

Если для выполнения обработки используются классы внутри и нужно поддерживать независимость от сред разработки, то необходимо предусмотреть функции по созданию и уничтожению объектов классов **CreateP()**, **DestroyP()**, а также передачи указателя на созданный объект.

Помимо такого подхода к реализации системы плагинов существуют и другие подходы, например с использованием интерфейсов[1].

4. Использование процессов и потоков при разработке программ

Часто разработчику приходится сталкиваться с проблемой, когда необходимо одновременное выполнение нескольких задач одного приложения. Для решения этой и других проблем разработчику предоставлены в распоряжение средства, позволяющие реализовать многопоточность. Механизм, реализующий многопоточностъ, основан на понятиях процессов и потоков.

Процесс – это контейнер для набора ресурсов, используемых потоками, которые выполняют экземпляр программы. На самом высоком уровне абстракции процесс включает в себя:

– закрытое виртуальное адресное пространство – диапазон адресов виртуальной памяти, которым может пользоваться процесс;

– исполняемую программу – начальный код и данные, проецируемые на виртуальное адресное пространство процесса;

– список открытых описателей различных системных ресурсов: семафоров, коммуникационных портов, файлов и других объектов, доступных всем потокам в данном процессе;

– контекст защиты, называемый *маркером доступа* (Access Token), который идентифицирует пользователя, группы безопасности и привилегии, сопоставленные с процессом;

– уникальный идентификатор процесса (PID);

– минимум один поток.

Win32 API содержит несколько функций, которые можно использовать для создания и управления процессами. Процесс может

[1] Более подробно об этом см.: Плагины (plug-in’s)в Delphi с использованием интерфейсов. URL: http://www.webdelphi.ru/2010/05/plaginy-plug-ins-v-delphi-s-ispolzovaniem-interfejsov/ ; Создание Plugin-архитектуры с помощью C#. URL: http://msugvnua000.web710.discountasp.net/Posts/Details/3477

быть создан потоком другого процесса путем вызова функции операционной системы **CreateProcess**. Процесс можно завершить четырьмя способами:

– входная функция первичного потока, например **WinMain**, возвращает управление (это предпочтительный способ);

– один из потоков процесса вызывает функцию **ExitProcess** (что также является предпочтительным);

– поток другого процесса вызывает функцию **TerminateProcess** (это нежелательный, аварийный способ);

– все потоки процесса завершаются по своей воле.

Рекомендуется проектировать приложение таким образом, чтобы его процесс завершался только после возврата управления функцией первичного потока. Это единственный способ, который гарантирует корректную очистку всех ресурсов, принадлежащих первичному потоку.

Система Windows поддерживает шесть категорий приоритетов процессов:

– фоновый (**IDLE_PRIORITY_CLASS**);

– нормальный заднего плана (**BELOW_NORMAL_PRIORITY _CLASS**);

– нормальный (**NORMAL_PRIORITY_CLASS**);

– нормальный переднего плана (**ABOVE_NORMAL _PRIORITY_CLASS**);

– высокий (**HIGH_PRIORITY_CLASS**);

– процесс реального времени (**REALTIME_PRIORITY_CLASS**).

По умолчанию при создании нового процесса ему присваивается нормальный приоритет.

Поток (Thread) – это объект ядра, отвечающий за выполнение программного кода в процессе. При инициализации процесса система создает и его первичный поток. Первичный поток начинает свое выполнение с функции **main** (в консольных приложениях) или **WinMain** (в GUI-приложениях) и существует до тех пор, когда функция **main** (**WinMain**) возвращает управление. Большинство процессов обходятся одним потоком, однако существует возможность создания новых потоков, которые будут выполнять программный код параллельно с первичным потоком.

Для программной реализации потоков в различных языках программирования применяются схожие механизмы, однако они имеют свои особенности[1].

[1] Более подробно об этом см.: Работа с потоками в C#. URL:http://rsdn.org/article/dotnet/CSThreading1.xml; Потоки и методы их синхронизации в Delphi. URL: http://www.interface.ru/?artId=6105 .

Различают два основных типа потоков: независимые и взаимодействующие. *Независимые потоки* предназначены для выполнения различных задач, не разделяющих совместные ресурсы. *Взаимодействующие потоки* конкурируют за один и тот же ресурс, например переменную, файл или массив.

При реализации потоку можно задать один из следующих приоритетов работы:

– **tpIdle** – фоновый приоритет. Поток с таким приоритетом выполняется только при отсутствии хоть какой-либо загрузки процессора;

– **tpLowest** – низкий приоритет;

– **tpLower** – пониженный приоритет;

– **tpNormal** – нормальный приоритет, соответствующий поведению обычной программы;

– **tpHigher** – повышенный приоритет. Использование потоков с таким или более высоким приоритетом весьма критично скажется на производительности остальных потоков;

– **tpHighest** – высочайший приоритет. Применяется только для потоков, которые выполняются в кратчайшие промежутки времени;

– **tpTimeCritical** – приоритет реального времени.

Особое внимание при реализации обработки в отдельных потоках следует уделять организации взаимодействия с графическим интерфейсом приложения (Grafical User Interface, GUI) ввиду того, что он является общим ресурсом.

5. Способы организации взаимодействия приложений

Многие приложения не являются самодостаточными, т. е. им требуется информация от других приложений либо они должны эту информацию сообщать. Именно поэтому в операционную систему встраивается множество механизмов, которые обеспечивают *межпроцессное взаимодействие* (Interproccess Communication, IPC), такие как файл, буфер обмена, сообщение **WM_COPYDATA**, динамические библиотеки и сокеты.

Файл – одна из самых примитивных форм IPC, которая заключается в том, что данные и команды передаются при помощи файлов.

Буфер обмена (Clipboard) – наиболее простая и хорошо известная форма IPC, которая появилась еще в самых ранних версиях Windows. Основная его задача состоит в том, чтобы обеспечивать обмен данными между программами по желанию и под контролем

пользователя. Не рекомендуется использовать его для внутренних нужд приложения и помещать в него то, что не предназначено для прямого просмотра пользователем.

Сообщение **WM_COPYDATA** – стандартное сообщение для передачи участка памяти другому процессу. Это сообщение работает однонаправленно, принимающий процесс должен расценивать полученные данные как **read only**. Посылать это сообщение предпочтительно с помощью сообщения **SendMessage**, которое в отличие от **PostMessage** ждет завершения операции. Таким образом, посылающий процесс тормозится на время передачи данных. Это не имеет значения для данных малого размера, но для больших объемов данных или для **real-time**-приложений может быть значимо.

Библиотеки динамической компоновки – механизм обмена данными между процессами, основанный на использовании общей (Shared) переменной, которая занимает одно и то же место в физической памяти. В этом случае важно помнить о синхронизации.

Сокеты (Sockets) – программный интерфейс, с помощью которого процессы могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью (локальной или Интернетом). Сокет – абстрактный объект, представляющий конечную точку соединения. Различают клиентские и серверные сокеты. С этими объектами программа и должна работать, например ждать соединения, посылать данные и т. д. В Windows входит достаточно мощный интерфейс прикладного программирования (Application Programming Interface, API) для работы с сокетами.

6. Организация защиты на основе USB Flash Drive

Для организации системы защиты на основе USB Flash Drive необходимо реализовать получение информации об устройстве. В Windows информацию об устройствах предоставляет стандартная системная библиотека **SetupApi.dll**, в которой для этих целей используются функции **SetupDiGetClassDevsA**, **SetupDiEnumDeviceInfo**, **SetupDiDestroyDeviceInfoList**, **CM_Get_Device_ID_Size**, **CM_Get_ Device_IDA**.

С помощью этих функций можно получить идентификатор (ID) и серийный номер USB Flash Drive, выполнив следующие шаги.

Шаг 1. Получить указатель на набор информации обо всех установленных устройствах, соответствующих заданным параметрам (в нашем случае – это устройства хранения), используя функцию **SetupDiGetClassDevs**.

Шаг 2. Обработать данные каждого элемента из набора, сформированного на шаге 1:

– вызвать функцию **SetupDiEnumDeviceInfo** для получения информации об элементе набора и произвести анализ идентификатора устройства для проверки того, является ли устройство USBустройством;

– при помощи функции **CM_Get_Device_ID_Size** определить размер строки идентификатора устройства, а затем непосредственно саму строку идентификатора, воспользовавшись функцией **CM_Get_Device_ID**;

– в полученной строке идентификатора проверить наличие фрагмента **USBSTOR**, присутствие которого показывает, что данное устройство является USB Flash Drive.

Шаг 3. Получить данные о накопителе (производитель, имя продукта) и серийный номер USB-устройства из строки идентификатора. Строка идентификатора для USB Flash Drive выглядит следующим образом:

```
USBSTOR\DISK&VEN_SAMSUNG&PROD_MIGHTY_DRIVE&REV_PMAP\07521094081F&0
```

Из данной строки можно получить следующую информацию:

– производитель (VEN_): `SAMSUNG`

– наименование продукта (PROD_): `MIGHTY_DRIVE`

– серийный номер (фрагмент, отвечающий за серийный номер типично находится самом конце после &REV_ или еще дополнительной информации): `07521094081F`

Шаг 4. Вызвать функцию **SetupDiDestroyDeviceInfoList** для корректного завершения процесса получения информации об устройствах и освобождения занимаемых ресурсов.

Альтернативе системной библиотеки **SetupApi.dll** является Windows Management Instrumentation (WMI) – одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows. Работа с WMI идет на основе запроса данных подобно конструкции SQL. Рассмотрим типовые шаги:

Шаг 1. Сформировать запрос (обращение к WMI) на получение информации о желаемых устройствах, например о устройствах подключенных по интерфейсу USB:

```
SELECT * FROM Win32_DiskDrive WHERE InterfaceType='USB'
```

Шаг 2. Создать объект для доступа к WMI и выполнить запрос.

– на языке Delphi/ Object pascal:

```
// Создаем OLE объект для доступа к WMI
  searcher:= CreateOleObject('WbemScripting.SWbemLocator');
```

```
// Соединяемся с WMI
  Wmi := searcher.ConnectServer('', '', '', '');
// Формируем обращение к WMI(выполняем запрос)
  Ret := Wmi.ExecQuery('SELECT * FROM Win32_DiskDrive WHERE
InterfaceType=' + QuotedStr('USB'));
```

– на языке C#:

```
// Формируем строку запроса (обращения к WMI)
string query = "SELECT * FROM Win32_DiskDrive WHERE
     InterfaceType='USB'";
// Создаем объект ManagementObjectSearcher на основе запроса
ManagementObjectSearcher searcher = new
     ManagementObjectSearcher(query);
```

Шаг 3. Интерпретировать полученный результат как набор и перебирая все элементы получить строку идентификации устройства.

– на языке Delphi/ Object pascal:

```
// интерпретируем полученный результат как набор перечислений
Enum := IUnknown(Ret._newenum) as ienumvariant;
// Проходим по объектам
while (Enum.next(1, tmp, value) = S_OK) do
Begin // Получаем строку идентификации устройства
  DeviceIDstr:= tmp.pnpdeviceid;
End
```

– на языке C#:

```
// Проходим по объектам
foreach (ManagementObject diskDrive in
searcher.Get().Cast<ManagementObject>()){
// Получаем строку идентификации устройства
  string DeviceIDstr = (string)diskDrive["PNPDeviceID"];
}
```

Шаг 4. Получить данные о накопителе (производитель, имя продукта) и серийный номер USB-устройства из строки идентификатора.

Полученные в ходе выполнения этих шагов сведения включая серийный номер USB Flash Drive могут быть использованы как для привязки данных ключа к устройству, так и в качестве элемента шифрования файла лицензионного ключа.

Механизм защиты можно реализовать в отдельном потоке разрабатываемого приложения или в виде службы Windows. Так, в виде службы Windows можно организовать проверку подключения/отключения USB Flash Drive, которая при наступлении соответствующих событий будет передавать нужные данные защищаемому приложению.

Служба Windows (Windows Service) – это программа, которая может автоматически запускаться во время загрузки системы без необходимости входа кого-либо в систему. Пояснение процессов

разработки служб Wndows с приведением программного кода для языков Delphi и C# показано в ряде статей, доступных в сети Интернет[1].

7. Создание инсталляторов

Если было принято решение о распространении разработанных программ, то возникает необходимость в установочном дистрибутиве. Так, для развертывания программы можно, казалось бы, обойтись обычным самораспаковывающимся SFX-архивом, в котором прописано, какой файл программы и куда должен быть распакован. Однако это будет лишь примитивный инсталлятор, в котором пользователю не предоставляется никаких возможностей выбора.

В случае если проект достаточно сложен и содержит десятки файлов, то необходимо, чтобы пользователь имел выбор способа установки программы: куда устанавливать; нужно ли создавать ярлыки на рабочем столе и в панели быстрого запуска; какую версию устанавливать: полную, выборочную или минимальную; регистрировать ли программу в реестре или создать **INI**-файл. Кроме того, пользователь может когда-нибудь захотеть удалить эту программу со своего ПК, а это значит, что нужно предусмотреть и обратный процесс – деинсталляцию. Все эти операции в обычном архиваторе не выполняются или сопряжены с большими сложностями.

С учетом этих особенностей были разработаны программы, позволяющие создавать инсталляционные комплекты. Одной из таких программ является инсталлятор (скрипт) Inno Setup (URL: http:// www.jrsoftware.org/). Источник (скрипт) Inno Setup – это бесплатное средство для создания пакетов (дистрибутивов) программ, поддерживающее шифрование, установку пароля, выполнение различных задач по завершении установки. Среди других особенностей продукта можно отметить расширенную поддержку 64-битных приложений, настраиваемые типы установки, встроенный препроцессор и мощный язык сценариев на основе языка Паскаль. Наличие мастера генерации сценария работы инсталлятора позволяет достаточно быстро создать заготовку. Сам же инсталлятор имеет

---

[1] Создание служб Windows в Delphi с использованием VCL. URL: http://www.interface.ru/home.asp?artId=16705; Managing and Writing Windows services with Delphi. URL: https://www.freepascal.org/~michael/articles/services/services.pdf; Пошаговое руководство. Создание приложения служб Windows в конструкторе компонентов. URL: https://learn.microsoft.com/ru-ru/dotnet/framework/windows-services/walkthrough-creating-a-windows-service-application-in-the-component-designer; Пишем свой Windows service. URL: https://habrahabr.ru/post/102826

структуру, характерную для конфигурационных файлов **ini**-формата, поэтому код хорошо читаем и удобен в редактировании. Документ разделен на секции, каждая из которых отвечает за определенную задачу инсталлятора. Всего имеется два типа секций: с параметрами и секция «директива – значение».

Более подробную информацию о принципах и особенностях работы с инсталлятором Inno Setup можно получить в Интернете[1]

[1] См., например: Создание дистрибутива Windows-приложения в Inno Setup. URL: http://www.proghouse.ru/programming/40-innosetup; Inno Setup: создание инсталлятора на примере развертывания C#-приложения. URL: https://habrahabr.ru/post/255807/.

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1
ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫЕ БИБЛИОТЕКИ (DLL)
(*6 часов*)

Цель работы: разработать программу для работы с динамически подключаемыми библиотеками.

Задачи работы:

– ознакомиться с принципами разработки динамических библиотек в средах RAD Studio, Lazarus и Visual Studio;

– изучить способы взаимодействия с DLL-библиотеками на языках программирования C++, C#, Delphi, Object-pascal, Python;

– получить навыки разработки приложений, использующих динамические библиотеки.

Порядок выполнения работы

1. Ознакомиться с п. 1, 2 основных теоретических сведений.
2. Получить исходные коды (пример программ и динамических библиотек) у преподавателя или взять с сервера дистанционного образования СибГУ им. М.Ф. Решетнева.
3. Выбрать предметную область.
4. Выполнить задания 1–4.
5. Подготовить отчет по лабораторной работе.
6. Защитить лабораторную работу перед преподавателем.

Задания

Задание 1 Разработать три алгоритма, выполняющих обработку данных для выбранной предметной области, при этом необходимо реализовать обработку/анализ набора данных, представленного:

– в виде одномерных массивов, в качестве входных параметров использовать два и более одномерных массива;

– в виде двумерного массива, например матриц или графической информации, т. е. изображения.

Задание 1.2. Реализовать разработанные алгоритмы в форме динамических библиотек и собрать их используя не менее трех различных компиляторов. Обязательно использовать компиляторы двух семейств языков программирования C/C++ и Delphi/Object pascal. Возможно использование сред разработки Visual Studio (C++), RAD Studio

(C++ Builder и Delphi) и Lazarus и иных. Сборку библиотек осуществлять в режиме **Release**. Пример компиляторов для языков программирования:

– C/C++ (Visual Studio C++, Intel oneAPI DPC++/C++ Compiler, C++ Builder и иные доступные)

– Delphi/Object pascal (компиляторы сред разработки Delphi/Lazarus).

Задание 3. Написать пять демонстрационных программ, в которых подключаются DLL-библиотеки, вызывающие функции обработки согласно разработанным ранее алгоритмам. При этом необходимо измерять время работы алгоритма.

Реализацию трех из пяти программ осуществлять на выбор в средах разработки для двух семейств языков программирования C/C++ и Delphi/Object pascal. Обязательно реализовать два приложения из которых одно на языке C#, а второе на языке Python. Для языков Python и C++ допускается консольное приложение.

Задание 4. Провести экспериментальное исследование по обработке данных с разными реализациями алгоритмов на сборке в режиме **Release**. При выполнении исследования использовать одномерные массивы с размерностью от 100 000 элементов и двумерные массивы с общим числом элементов более 400 000.

В ходе исследования необходимо в каждой программе определить время выполнения функций, вызываемых из различных DLL-библиотек, в миллисекундах (мс). По результатам исследования нужно заполнить таблицу, в которой указать минимальное (Min), максимальное (Max) и среднеарифметическое (Avg) значения, полученные в ходе 50 запусков выполнения функции:

Приложение 1									
Среда разработки DLL	Функция 1			Функция 2			Функция 3		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
DLL Сборка 1	*,****								
DLL Сборка 2									
DLL Сборка 3									
Приложение 2									
……									
Приложение 5									
Среда разработки DLL	Функция 1			Функция 2			Функция 3		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
DLL Сборка 1									
DLL Сборка 2									
DLL Сборка 3									

Для замера времени нужно использовать следующие программные конструкции:

– на языке C# (среда разработки Visual Studio):

```
// Подключение библиотек
[DllImport("Kernel32.dll")]
private static extern bool QueryPerformanceCounter(out long
     lpPerformanceCount);
[DllImport("Kernel32.dll")]
private static extern bool QueryPerformanceFrequency(out long
     lpFrequency);
// Объявление переменных для замера времени
private long FbeginCount, FendCount; Ffrequence;
Double Ftime;
// Замер времени начала (до выполнения действий)
QueryPerformanceFrequency(out Ffrequence);
QueryPerformanceCounter(out FbeginCount);

/// Выполнение некоторых вычислений (вызов функций из dll)

// замер времени окончания (после выполнения действий)
QueryPerformanceCounter(out FendCount);
Ftime = ((FendCount – FbeginCount) / (double)Ffrequence) * 1000;
```

– на языке C++ (среда разработки Visual Studio):

```
include <windows.h>
// Объявление переменных для замера времени
LARGE_INTEGER FbeginCount, FendCount, Ffrequence;
double Ftime;
// Замер времени начала (до выполнения действий)
QueryPerformanceFrequency(&Ffrequence);
QueryPerformanceCounter(&FbeginCount);

/// Выполнение некоторых вычислений (вызов функций из dll)

// Замер времени окончания (после выполнения действий)
QueryPerformanceCounter(&FendCount);
Ftime = ((FendCount.QuadPart – FbeginCount.QuadPart) /
(double)Ffrequence.QuadPart) * 1000;
```

– на языке C++ (среда разработки RAD Studio C++ Builder):

```
// Объявление переменных для замера времени
ULONGLONG FbeginCount, FendCount, Ffrequence;
double Ftime;
// Замер времени начала (до выполнения действий)
QueryPerformanceFrequency((LARGE_INTEGER*)&Ffrequence);
QueryPerformanceCounter((LARGE_INTEGER*)&FbeginCount);

/// Выполнение некоторых вычислений (вызов функций из dll)

// Замер времени окончания (после выполнения действий)
QueryPerformanceCounter((LARGE_INTEGER*)&FendCount);
Ftime = ((FendCount – FbeginCount) / (double)Ffrequence) * 1000;
```

– на языке Delphi/Object pascal (среда разработки RAD Studio Delphi/ Lazarus):

```
// Объявление переменных для замера времени
```

```
FbeginCount, FendCount, Ffrequence: TlargeInteger;
Ftime: extended;
// Замер времени начала (до выполнения действий)
QueryPerformanceFrequency(Ffrequence);
QueryPerformanceCounter(FbeginCount);
/// Выполнение некоторых вычислений (вызов функций из dll)
// Замер времени окончания (после выполнения действий)
QueryPerformanceCounter(FendCount);
Ftime := ((FendCount – FbeginCount) / Ffrequence) * 1000;
```

– на языке Python:

```
// импорт библиотеки времени
import time;
// Замер времени начала (до выполнения действий)
start_time = time.perf_counter()
/// Выполнение некоторых вычислений (вызов функций из dll)
// Замер времени окончания (после выполнения действий)
end_time = time.perf_counter()
Ftime = (end_time - start_time) * 1000;
```

Данные конструкции позволяют получить время в миллисекундах.

Отчет

Отчет в форме документа Microsoft Word сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы и постановку задачи;

– описание реализованных алгоритмов на *русском* языке с точки зрения исполнения или в виде блок схемы/пошагового описания выполнения алгоритма для задания 1;

– снимки экрана, демонстрирующие работоспособность приложений и выполнения реализованных алгоритмов (задания 2 и 3).

– информация о компьютере, на котором выполнялось экспериментальное исследование;

– заполненную статистическую таблицу для задания 4;

– выводы по результатам экспериментального исследования;

– выводы по лабораторной работе в целом.

Дополнительно к файлу отчета необходимо предоставить архив содержащий:

– исходные коды динамических библиотек (для каждой реализации) с комментариями к основным моментам;

– исходные коды программ работающих с динамическими библиотеками (для каждой реализации);

Контрольные вопросы и задания

1. Опишите последовательность действий для создания DLL-библиотеки в среде разработки Visual Studio на языке C++.
2. Приведите типовую структуру исходного кода DLL-библиотеки на языке C++.
3. Опишите особенности создания DLL-библиотеки в среде разработки RAD Studio на языке C++.
4. Приведите последовательность действий для создания DLL-библиотеки в среде разработки RAD Studio на языке Delphi.
5. Какие отличия существуют между статическим и динамическим подключением DLL-библиотеки?
6. Каким образом осуществляется статическое подключение DLL-библиотеки в программе на языке C++?
7. Опишите структуру DLL-библиотеки на языке Delphi.
8. Каким образом можно организовать статическое использование функций DLL-библиотеки в языке C#?
9. Представьте схему динамического подключения и использования функции DLL-библиотеки на языке Delphi.
10. Приведите пример объявления функции DLL-библиотеки на языке Delphi при использовании статического способа подключения.
11. Какие API-функции используются при динамическом подключении DLL-библиотеки?
12. Приведите схему использования функции DLL-библиотеки на языке C++.
13. Каким образом на Python можно подключить DLL-библиотеку?
14. Какую информацию дает соглашение о вызове?
15. Охарактеризуйте директиву cdecl.
16. Приведите сравнительное описание директив для указания соглашения о вызове.
17. Для чего в Python используется модуль ctypes при работе с DLL?
18. Охарактеризуйте особенности использования директивы stdcall.
19. Каким образом лучше организовывать обработку массивов данных?
20. При помощи каких API-функций можно получить доступ к подпрограммам DLL-библиотеки?
21. Опишите процесс работы с функциями DLL-библиотеки на языке Python.
22. Поясните разработанный вами программный код.

Лабораторная работа 2

РАЗРАБОТКА ПЛАГИНОВ К ПОЛЬЗОВАТЕЛЬСКОМУ ПРИЛОЖЕНИЮ НА ОСНОВЕ DLL-БИБЛИОТЕК

(*6 часов*)

Цель работы: разработать программный продукт с системой плагинов на основе DLL-библиотек.

Задачи работы:

– ознакомиться с принципами организации модулей расширения функционала программных продуктов (плагинов) на основе динамических библиотек;

– изучить способы динамического подключения DLL-библиотеки во время работы приложений и использовать их для расширения функциональных возможностей программного продукта.

Порядок выполнения работы

1. Ознакомиться с п. 3 основных теоретических сведений.
2. Изучить примеры с сервера дистанционного образования.
3. Выполнить задания 1, 2.
4. Подготовить отчет по лабораторной работе.
5. Защитить лабораторную работу перед преподавателем.

Задания

Задание 1. Для выбранной предметной области (целесообразно согласовать с ее темой курсовой работы) разработать программный продукт (систему), в котором реализовать следующие функциональные возможности:

– подключение модулей расширения, т. е. плагинов (DLL-библиотек с необходимой функциональностью);

– автоматический поиск плагинов в определенном каталоге и подключение их к системе при ее запуске;

– получение информации о подключенном плагине: названии алгоритма/метода, версии, сведений о разработчике;

– подключение нового плагина во время работы приложения посредством диалогового окна;

– настройка параметров обработки данных для плагинов с возможностью дальнейшей модификации настроек при помощи пользовательского интерфейса.

– выполнение обработки данных на основе алгоритмов реализованных в виде плагинов.

Задание 2. Разработать не менее трех алгоритмов по предметной области и реализовать их в виде модулей расширения, т. е. плагинов (DLL-библиотек с необходимой функциональностью), с учетом:

– возможности конфигурирования работы алгоритма пользователем (например, использования диалогового окна или формирования панели с настройками);

– возможности отображения окна/панели с информации о времени работы алгоритмов обработки данных (если такая опция поставлена в настройках).

Отчет

Отчет в форме документа Microsoft Word сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы и постановку задачи;

– краткие теоретические сведения о предметной области;

– схему организации плагина (описание интерфейса программного взаимодействия);

– схемы функционирования механизма работы с модулями расширения (плагинами), включая действия, связанные с подключением плагинов, управлением списком плагинов, формированием графического интерфейса для настройки параметров исполнения алгоритма;

– снимки экрана, демонстрирующие работу приложения и выполнения алгоритмов/методов обработки данных с отображением их параметров настройки (конфигурационного окна);

– выводы по лабораторной работе.

Дополнительно к файлу отчета необходимо предоставить архив содержащий:

– исходные коды модулей расширения, т.е. плагинов (DLL с необходимой функциональностью) с комментариями;

– исходные коды программы работающей с модулями расширения.

Контрольные вопросы и задания

1. Чем отличается плагин (модуль расширения) от обычной динамической библиотеки?
2. Приведите типовую структуру реализации интерфейса взаимодействия с плагином.
3. Почему для работы с модулем расширения нельзя использовать статическое подключение?

4. Охарактеризуйте особенности реализации плагинов с возможностями настройки параметров работы в пользовательском интерфейсе.

5. Укажите известные способы реализации плагинов на основе классов.

6. Опишите процесс передачи информации о конфигурации графического интерфейса.

7. Какой способ хранения информации о подключенных плагинах в основной программе является наиболее рациональным?

8. Каким образом можно организовать передачу строковых констант из плагинов в основную программу?

9. Поясните разработанный вами программный код.

Лабораторная работа 3

РАЗРАБОТКА ПРИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ПОТОКИ ДЛЯ ОБРАБОТКИ ДАННЫХ, ПОЛУЧЕННЫХ ИЗ СЕТИ ИНТЕРНЕТ

(*4 часа*)

Цель работы: разработать приложение, использующее потоки для обработки данных, полученных из сети Интернет.

Задачи работы:

– ознакомиться с принципами разработки приложений, использующих выделенные потоки для обработки данных, на примере результатов HTTP-запросов;

– получить навыки разработки прикладных программ с учетом обработки в потоках.

Порядок выполнения работы

1. Ознакомиться с п. 4 основных теоретических сведений.
2. Изучить примеры с сервера дистанционного образования.
3. Выполнить задания 1, 2.
4. Подготовить отчет по лабораторной работе.
5. Защитить лабораторную работу перед преподавателем.

Задания

Задание 1. Разработать приложение, использующее данные, получаемые на основе HTTP/FTP-запросов одновременно с нескольких источников. Например, можно реализовать вывод изображений и/или

видеопотоков одновременно с двух IP-камер AXIS на основе HTTP-запросов[1]. При организации работы программы необходимо:

– реализовать механизм работы потоков, получающих данные из сети Интернет, например кадры видеопоследовательности MJPEG или серию отдельных картинок;

– для обработки данных, полученных в результате HTTP-запросов, например декодирования кадров/картинок, использовать динамическую библиотеку, например TurboJPEG[2];

– организовать управление потоками, т. е. запуск/паузу/ остановку с сообщением об этом в основном интерфейсе текущего состояния.

Задание 2. Разработать и реализовать два алгоритма обработки получаемых данных в виде плагинов (например, для изображения кадра: инвертирование, изменение яркости, цветовой сдвиг, гамма коррекция). Модифицировать программу из задания 1, предоставив возможность применять один из реализованных алгоритмов обработки по выбору пользователя для каждого потока. Например, при обработке в потоке изображений/MJPEG алгоритм из плагина будет применяться перед показом изображения (выводом на экран).

Внимание! По предварительному согласованию с преподавателем формулировка задания может быть несколько изменена, но она должна включать в себя обработку результатов HTTP-запросов с использованием DLL-библиотеки в организуемых потоках.

Требования к отчету

Отчет в форме документа Microsoft Word сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы;

– постановку задачи;

– краткие теоретические сведения о предметной области и способах работы с потоками и выполнением HTTP-запросов в выбранной среде разработки;

– схему обработки данных потоков для задания 1 (возможно в виде укрупнённых шагов);

– описание разработанных алгоритмов на русском языке с точки зрения реализации или в виде блок-схемы для задания 2;

---

[1] Для получения кадров формата JPEG используется http-запрос формата http://<ip>/axis-cgi/jpg/image.cgi в случае MJPEG потока: http://<ip>//axis-cgi/mjpg/video.cgi через опцию resolution можно указать желаемое поддерживаемое разрешение, например http://<ip:port>//axis-cgi/mjpg/video.cgi&resolution=1920x1080

[2] Описание библиотеки TurboJPEG см. по адресу: http://www.libjpeg-turbo.org/About/TurboJPEG

– снимки экрана, демонстрирующие работу приложения и выполнения алгоритмов обработки данных;

– выводы по лабораторной работе.

Дополнительно к файлу отчета необходимо предоставить архив, содержащий:

– исходные коды модулей расширения, т. е. плагинов (DLL-библиотек с необходимой функциональностью), с комментариями;

– исходные коды итоговой программы с учетом задания 2.

Контрольные вопросы и задания

1. В каких случаях следует использовать потоки?
2. Опишите процесс создания потока на языке Delphi.
3. Опишите процесс создания потока на языке C++.
4. Опишите процесс создания потока на языке C#.
5. В чем заключаются особенности работы с потоками на языке
6. Охарактеризуйте объект класса Tthread в языке Delphi.
7. Каким образом можно получить статистику о текущем потоке в языке C#?
8. Перечислите и охарактеризуйте виды приоритетов, используемые в потоках.
9. Охарактеризуйте особенности взаимодействия потоков с объектами VCL.
10. Каким образом можно сделать GET-запрос в среде разработки Visual Studio C#?
11. Какие действия необходимо выполнить для того, чтобы можно было влиять на интерфейс приложения из потока?
12. Опишите процесс получения результата GET-запроса на языке Delphi.
13. Приведите пример получения результата запроса к веб-странице на языке C++.
14. Поясните разработанный вами программный код.

Лабораторная работа 4

РАЗРАБОТКА МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ, ИСПОЛЬЗУЮЩИХ ОЧЕРЕДИ ОБРАБОТКИ И ОБМЕН ДАННЫМИ МЕЖДУ ПОТОКАМИ

(*6 часов*)

Цель работы: разработать многопоточные приложения, использующие очереди обработки и обмен данными между потоками.

Задачи работы:

– ознакомиться с принципами разработки приложений, использующих выделенные потоки, с организацией очереди заявок на обработку потоков;

– изучить особенности реализации очереди заявок в средах разработки RAD Studio и Visual Studio на примере организации ведения протокола обработки.

Порядок выполнения работы

1. Ознакомиться с п. 4 основных теоретических сведений.
2. Изучить примеры с сервера дистанционного образования.
3. Выполнить задание.
4. Подготовить отчет по лабораторной работе.
5. Защитить лабораторную работу перед преподавателем.

Задание

Разработать приложение, обрабатывающее данные в независимых потоках, например изображения, текстовые данные, числовые массивы и т. п.), с использованием функций, хранящихся в DLL-библиотеке. В приложении необходимо:

– организовать очередь задач на обработку данных, например список файлов, список URL, параметры массивов и т. п.;

– реализовать обработку данных предметной области при помощи потоков, при этом количество потоков пользователь должен указать во время выполнения программы (не менее трех). Непосредственную обработку данных организовать на основе системы плагинов или динамического подключения функций из DLL-библиотеки;

– реализовать возможность формирования очереди заявок на занесение сведений в протокол из рабочих потоков (например, формирование конструкции INSERT);

– организовать ведение протокола в базе данных на основе отдельного потока с низким приоритетом и режимом простоя (в случае отсутствия в очереди заявок на обработку поток должен перейти в режим неактивности на установленный пользователем интервал времени). Для занесения сведений в БД использовать SQL-запросы.

В интерфейсе приложения показывать статистику о выполненной каждым из рабочих потоков обработке данных: количество обработанных задач, минимальное время, а также среднее и максимальное время, затраченное на обработку. Осуществляется при помощи запроса SELECT к таблице с данными.

Требования к отчету

Отчет в форме документа Microsoft Word сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы и постановку задачи;

– краткие теоретические сведения о способах организации обмена данных между потоками в выбранной среде разработки;

– описание реализованных алгоритмов обработки данных на русском языке с точки зрения реализации или в виде блок-схемы или диаграммы активности;

– схему реализации обработки в рабочих потоках и их взаимодействия с системой протоколирования;

– снимки экрана, демонстрирующие работу приложения, выполнения алгоритмов обработки данных и ведения протокола в базе данных;

– выводы по лабораторной работе.

Дополнительно к файлу отчета необходимо предоставить архив, содержащий:

– исходные коды DLL-библиотек и/или модулей расширения (плагинов) с необходимой функциональностью;

– исходные коды приложения, обрабатывающего данные в независимых потоках.

Контрольные вопросы и задания

1. Охарактеризуйте существующие в Windows способы управления потоками.
2. Каким образом можно передать данные из одного потока в другой на языке Delphi?
3. Перечислите и охарактеризуйте способы синхронизации данных интерфейса приложения и потоков на языке C#.
4. Приведите схему реализации обработки списка заявок с использованием нескольких потоков.
5. В чем заключаются особенности организации доступа к общим переменным на языке C++?
6. Охарактеризуйте класс Mutex языков C++/C#.
7. Каким образом можно приостановить/возобновить работу потока?
8. Приведите типичную схему реализации обработки в рабочих потоках и их взаимодействия с системой протоколирования.
9. Охарактеризуйте API-процедуру WaitForSingleObject.
10. Охарактеризуйте API-процедуру WaitForMultipleObjects.

11. В чем состоит сущность проблемы взаимных блокировок (deadlock)?

12. Охарактеризуйте класс std::lock_guard.

13. Поясните разработанный вами программный код.

Лабораторная работа 5

ОРГАНИЗАЦИЯ ОБМЕНА ДАННЫМИ МЕЖДУ ПРИЛОЖЕНИЯМИ

(*6 часов*)

Цель работы: разработать программный комплекс, реализующий механизмы обмена данными между приложениями.

Задачи работы:

– ознакомиться с принципами организации взаимодействия приложений между собой;

– изучить различные виды реализации взаимодействия приложений с использованием сокетов, буфера обмена Windows, сообщений **WM_COPYDATA** (**SendMessage**, **PostMessage**);

– получить навыки разработки программных комплексов с возможностями передачи команд и данных между отдельными модулями.

Порядок выполнения работы

1. Ознакомиться с п. 5 основных теоретических сведений.
2. Изучить примеры с сервера дистанционного образования.
3. Выполнить задание.
4. Подготовить отчет по лабораторной работе.
5. Защитить лабораторную работу перед преподавателем.

Задание

Разработать программный комплекс, содержащий четыре приложения:

– первое «Клиент» организует управление обработкой данных путем пересылки команд и данных (текстовые, графические, произвольные структуры);

– второе приложение «Сервер» получает данные и обрабатывает текстовых и графические данные на основе команд (набора из нескольких простых функций) и возвращает «Клиенту», в случае получения данных произвольной структуры и получения команды вычисления запускается третье приложение;

– третье приложение консольного типа «Калькулятор» выполняет вычисления согласно параметрам командной строки и заносит информацию в протокол (таблицу базы данных): дата и время запуска; параметры запуска (параметры задачи); результат вычисления; дата и время завершения работы приложения.

При реализации работы программного комплекса необходимо организовать для приложений «Клиент» и «Сервер»:

– взаимодействие приложений через сокеты, т. е. передачу команд и данных нескольких видов (текстовых, графических, произвольных структур);

– пересылку данных нескольких видов (текстовых, графических, бинарных массивов) с использованием буфера обмена Windows;

– передачу команд и данных нескольких видов (текстовых, графических, произвольных структур) при помощи сообщений **WM_COPYDATA** (**SendMessage**, **PostMessage**).

Требования к отчету

Отчет в форме документа Microsoft Word сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы и постановку задачи;

– краткие теоретические сведения о реализации способов межпрограммного взаимодействия в выбранной среде разработки;

– описание формата пересылаемых команд и данных;

– описание команд обработки данных;

– снимки экрана, демонстрирующие работу приложений, входящих в программный комплекс;

– выводы по лабораторной работе.

Дополнительно к файлу отчета необходимо предоставить архив, содержащий исходные коды программного комплекса.

Контрольные вопросы и задания

1. Перечислите известные вам способы передачи команд и данных между приложениями.
2. Приведите типовую схему организации взаимодействия приложений через сокеты.
3. Охарактеризуйте компоненты языков Delhpi/C# для работы с сокетами.
4. Каким образом можно определить тип данных, находящихся в буфера обмена Windows?

5. Представьте фрагмент кода на языке C# для копирования информации в буфер обмена Windows.

6. При помощи какой функции можно выяснить дескриптор окна приложения по его имени?

7. Опишите процесс передачи команд и данных с использованием сообщения **WM_COPYDATA**.

8. Сравните способы передачи данных и команд с использованием функций **SendMessage** и **PostMessage**.

9. Приведите пример обработчика команд с использованием механизма сообщений.

10. Поясните разработанный вами программный код.

Лабораторная работа 6

ОРГАНИЗАЦИЯ СИСТЕМЫ ЗАЩИТЫ ПРИЛОЖЕНИЯ НА ОСНОВЕ USB FLASH DRIVE

(*6 часов*)

Цель работы: реализовать систему защиты приложения на основе USB Flash Drive.

Задачи работы:

– ознакомиться с принципами организации защиты программного продукта;

– изучить методы реализации аппаратной защиты на основе USB Flash Drive и кодированного файла лицензионного ключа;

– получить навыки разработки модулей для защиты программного продукта с возможностями ограничения функционала.

Порядок выполнения работы

1. Ознакомиться с п. 6 основных теоретических сведений.
2. Изучить примеры с сервера дистанционного образования.
3. Выполнить задание.
4. Подготовить отчет по лабораторной работе.
5. Защитить лабораторную работу перед преподавателем.

Задание

Разработать защищаемый программный комплекс, в котором защита реализована на уровне проверки полномочий в соответствии с файлом ключа и серийным номером USB Flash Drive. В программном комплексе

необходимо реализовать три модуля, обеспечивающих защиту основной программы:

1) модуль (программу) генерации файла ключа, который хранит информацию в недоступном для обычного просмотра виде (возможна реализация простых алгоритмов шифрования и/или с дополнительным использования хеш-кодов). Ключ должен хранить следующую информацию:

– сведения о том, кому выдана лицензия (имя, организация);

– ограничения работы основной программы по времени (возможно указание дней или даты, ограничивающей действия ключа);

– ограничения функциональных возможностей (от одной до трех функций), например скрытие части интерфейса, параметров работы алгоритмов, количества рабочих потоков и т. п.;

2) модуль защиты основной программы на основе сверки данных ключа с параметрами системы и USB Flash Drive;

3) отладочный модуль (отдельное приложение) для получения данных о USB Flash Drive, в котором необходимо реализовать возможность ведения протокола включающего учет:

– изменения состояния USB накопителей: дата и время, тип действия (подключение/отключение), данные накопителя в том числе серийный номер и буква диска.

– проверок лицензии, которые получаются из защищаемого приложения (если оно запущено) при помощи любой технологии обмена данных. При обнаружении использовании неверного (измененного) файла ключа также заносится запись в протокол.

Основная программа для демонстрации возможностей защиты должна показывать сведения о лицензии (берутся из файла ключа) и реализовывать функции, на которые будут накладываться ограничения[1]. Для разработки программы можно использовать результаты предыдущих лабораторных работ.

Требования к отчету

Отчет сдается преподавателю в электронном виде и должен включать в себя:

– титульный лист;

– цель работы;

– постановку задачи;

– описание структуры лицензионного ключа;

[1] Допускается использование простых функций, например включение отключения таймера изменяющего цвет панели или функцию подсчета времени работы, а также простое изменение состояние активности элементов интерфейса (блокировка части интерфейса для действий).

– схему распределения функциональных возможностей согласно политике ограничения работы основной программы;

– сведения о методах и алгоритмах, использованных для организации механизма защиты;

– снимки экрана, демонстрирующие работу модулей защиты и основного приложения;

– выводы по лабораторной работе.

Дополнительно к файлу отчета необходимо предоставить архив, содержащий:

– исходные коды модулей защиты и основной программы.

Контрольные вопросы и задания

1. Охарактеризуйте способы защиты программного обеспечения.
2. Каким образом лучше хранить данные в лицензионном ключе?
3. Перечислите известные вам механизмы шифрования данных, используемые в Windows.
4. Каким образом можно защитить файл ключа от его изменения?
5. Охарактеризуйте хеш-коды CRC32, MD5, SHA-1/5.
6. Какая системная библиотека в средах Windows 7/10/11 может предоставить информацию об подключенных устройствах?
7. Какие системные функции могут понадобится для реализации защиты на основе USB Flash Drive?
8. Перечислите основные этапы, которые нужно выполнить для получения идентификатора и серийного номера USB Flash Drive.
9. Какую информацию содержит строка идентификатора устройства?
10. Поясните разработанный вами программный код.

Лабораторная работа 7
СОЗДАНИЕ ИНСТАЛЛЯЦИОННЫХ КОМПЛЕКТОВ
(*6 часов*)

Цель работы: создать инсталляционные комплекты.

Задачи работы:

– Ознакомиться с системой создания инсталляторов для Windows-программ Inno Setup;

– изучить способы создания ярлыков и элементов реестра, а также задания инструкций для выполнения после успешной установки программы.

Порядок выполнения работы

1. Ознакомиться с п. 7 основных теоретических сведений.
2. Выполнить задания 1, 2.
3. Подготовить отчет по лабораторной работе.
4. Защитить лабораторную работу перед преподавателем.

Задания

Задание 1. Создать инсталляционный комплект набора программ, созданного в ходе лабораторной работы 1, в котором должны быть реализованы следующие возможности:

– выбор типа установки (**Полная**, **Visual Studio**, **RAD Studio**, **Выборочная**), регламентирующей перечень используемых компонентов;

– выбор создаваемых ярлыков для программ;

– развертывание .NET-фреймворка требуемой версии.

Задание 2. Создать инсталляционный комплект для программного комплекса, разработанного в ходе лабораторной работы 6, в котором должны быть реализованы следующие возможности:

– выбор типа установки (**Полная**, **Компактная**, **Выборочная**), регламентирующей перечень используемых компонентов с обязательным компонентом;

– выбор создаваемых ярлыков;

– автоматическая установка службы Windows по работе с USB Flash Drive;

– создание ветки реестра с ключом, содержащим путь к программному комплексу.

Требования к отчету

Отчет сдается преподавателю в электронном виде и должен содержать:

– титульный лист;

– цель работы;

– постановку задачи;

– скрипт-код инсталляционного комплекта для задания 1;

– скрипт-код инсталляционного комплекта для задания 2;

– снимки экрана, демонстрирующие работу инсталляторов;

– выводы по лабораторной работе.

Контрольные вопросы и задания

1. Охарактеризуйте структуру скрипта Inno Setup.
2. Для каких целей используется уникальный идентификатор приложения (GUID)?
3. Каким образом можно указать инсталлятору, чтобы он мог создавать ярлыки для приложений?
4. Опишите порядок настройки для установки соответствующей версии фреймворка.
5. Перечислите константы, определяющие директории в скрипте Inno Setup.
6. Какие действия нужно выполнить для того, чтобы выводить предупреждения при установке в существующую папку?
7. Каким образом можно выполнить настройки для замены существующих файлов?
8. Назовите константы папок окружения.
9. Каким образом происходит отладка дистрибутива в скрипте
10. Inno Setup?
11. За что отвечает секция Types скрипта Inno Setup?
12. Для чего предназначена секция Tasks скрипта Inno Setup?
13. Какая секция скрипта Inno Setup определяет перечень дополнительные папок, кроме папки приложения, которые нужно создавать?
14. Каким образом можно настроить выбор устанавливаемых компонентов?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Основной

15. Албахари, Д. C# 5.0. Справочник. Полное описание языка : пер. с англ. / Д. Албахари, Б. Албахари. – М. : Вильямс, 2014. – 1008 с.

16. Архангельский, А. Я. Программирование в C++ Builder / А. Я. Архангельский. – 7-е изд. – М. : Бином-Пресс, 2010. – 1230 с.

17. Арчер, Т. Основы C#. Новейшие технологии : пер. с англ. / Т. Арчер. – М. : Рус. ред., 2001. – 418 с.

18. Литвиненко, Н. А. Технология программирования на C++. Win32 API-приложения : учеб. пособие / Н. А. Литвиненко. – СПб. : БХВ-Петербург, 2010. – 288 с.

19. Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов : пер. с англ. / К. Нейгел, Б. Ивьен, Д. Глин. – М. : Вильямс, 2014. – 1440 с.

20. Осипов, Д. Delphi. Профессиональное программирование / Д. Осипов. – СПб. : Символ-Плюс, 2006. – 1056 с.

21. Рихтер, Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows : пер. с англ. / Дж. Рихтер. – 4-е изд. – СПб. : Питер ; М. : Рус. ред., 2008. – 720 с.

22. Скляров, Д. В. Искусство защиты и взлома информации / Д. В. Скляров. – СПб. : БХВ-Петербург, 2004. – 288 с.

23. Щупак, Ю. А. Win32 API. Эффективная разработка приложений / Ю. А. Щупак. – СПб. : Питер, 2007. – 572 с.

24. Эхтер, Ш. Многоядерное программирование / Ш. Эхтер, Дж. Робертс. – СПб. : Питер, 2010. – 316 с.

Электронные ресурсы

1. Документация по языку C# [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/ . – Загл. с экрана.

2. Документация по Microsoft C++/ C и ассемблеру [Электронный ресурс]. – Электрон. дан. – Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170 . – Загл. с экрана.

3. C# для Windows [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://wladm.narod.ru/C_Sharp/index.html Загл. с экрана.

4. CodeProject – For those who code [Electronic resource]. – Electronic data. – Access mode: https://www.codeproject.com. – Title from screen.

5. Delphi-исходники, компоненты и примеры программ на Borland Делфи 7/XE [Электронный ресурс]. – Электрон. дан. – Режим доступа: http://www.delphisources.ru. – Загл. с экрана.