

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

А. И. Рябченко, А. С. Вегера

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по одноименной дисциплине
для слушателей специальности 1-40 01 73
«Программное обеспечение информационных систем»
заочной формы обучения**

Гомель 2014

УДК 004.42-004.43(075.8)
ББК 32.973.26-018.1я73
Р98

*Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем ГГТУ им. П. О. Сухого
(протокол № 12 от 24.06.2013 г.)*

Рецензент: д-р техн. наук, проф. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого *А. И. Мурашко*

Рябченко, А. И.

Р98 Системное программирование : лаб. практикум по одной дисциплине для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / А. И. Рябченко, А. С. Вегера. – Гомель : ГГТУ им. П. О. Сухого, 2014. – 85 с. Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: <http://library.gstu.by>. – Загл. с титул. экрана.

Представлены задания к лабораторным работам по дисциплине «Системное программирование». Рассмотрены вопросы, связанные с применением процедур и функций, созданием динамических библиотек (DLL), применением WinAPI для создания собственных приложений и управления чужими программами, способами построения многопоточных приложений, созданием служб и драйверов режима ядра. Все задания к лабораторным работам снабжены предварительными теоретическими сведениями, которые будут полезны для выполнения заданий. Для каждой лабораторной работы приведен пошаговый ход выполнения заданий, что позволяет выполнить лабораторные работы самостоятельно.

Для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» ИПК и ПК.

**УДК 004.42-004.43(075.8)
ББК 32.973.26-018.1я73**

Лабораторная работа № 1 Процедуры и функции в Delphi

Цель работы: научиться работать с процедурами и функциями.

Теоретические сведения

Часто, работая над программой, программист замечает, что некоторая последовательность инструкций встречается в разных частях программы несколько раз. Можно избежать дублирования кода в программе. Для этого надо оформить инструкции, которые встречаются в программе несколько раз, как подпрограмму, и заменить инструкции, оформленные в виде подпрограммы, инструкцией вызова подпрограммы.

Во-первых, в программе нет дублирования кода, что сокращает трудоемкость создания программы, делает более удобным процесс отладки и внесения изменений. Представьте, что нужно изменить пояснительный текст, выводимый программой пересчета веса из фунтов в килограммы. В программе, не использующей подпрограмму, нужно просмотреть весь текст и сделать необходимые изменения. Если программа использует подпрограмму, то изменения надо внести только в текст подпрограммы.

Во-вторых, значительно повышается надежность программы. Следует обратить внимание, что подпрограммы используют не только тогда, когда нужно избежать дублирования кода.

Удобно большую задачу разделить на несколько подзадач и оформить каждую задачу как подпрограмму. В этом случае значительно улучшается "читаемость" программы и, как следствие, существенно облегчается процесс отладки.

Подпрограмма — это небольшая программа, которая решает часть общей задачи. В языке Delphi есть два вида подпрограмм — **процедура и функция**.

У каждой подпрограммы есть имя, которое используется в программе для вызова подпрограммы (процедуры).

Отличие функции от процедуры состоит в том, что с именем функции связано значение, поэтому функцию можно использовать в качестве операнда выражения, например, инструкции присваивания.

Delphi позволяет программисту поместить свои функции и процедуры в отдельный модуль, а затем использовать процедуры и

функции модуля в своих программах, указав имя модуля в списке модулей, необходимых программе (инструкция uses).

Для того чтобы в программе могли применяться функции и процедуры модуля, программист должен добавить этот модуль к проекту и указать имя модуля в списке используемых модулей (обычно имя модуля программиста помещают в конец сформированного Delphi списка используемых модулей).

Практическое задание

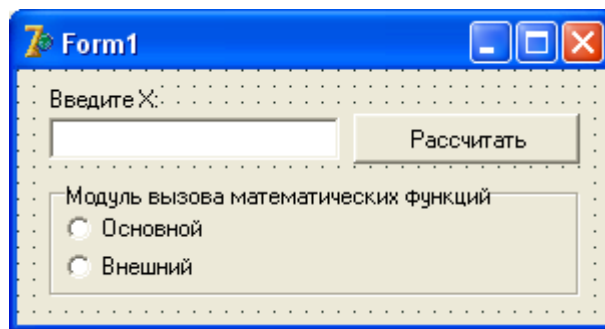
Разработать приложение позволяющее производить некоторые математические вычисления. Предусмотреть реализацию обработки исключений и защищенного ввода. Вычисляемую функцию реализовать, как в основном коде программы, так и в подключаемом модуле.

Варианты заданий

Вариант	Формула для вычислений	Вариант	Формула для вычислений
1	$y(x)=\sin(x)+\ln(x)$	11	$y(x)=\text{tg}^2(x)/40+\sin(x)$
2	$y(x)=\cos(x)/40+\sin(x)$	12	$y(x)=\sin^2(x^2)-\ln(x)$
3	$y(x)=12*\text{ctg}(x)+14*\sin(x)*\cos(x)$	13	$y(x)=\text{ctg}(x)+\cos(x)$
4	$y(x)=\text{tg}(x)+\cos(x)$	14	$y(x)=\ln(x)-\log(x)$
5	$y(x)=\ln(x)+\cos^2(x)$	15	$y(x)=\log^2(x)+\cos(x)$
6	$y(x)=\sin^2(x^2)-45*\text{tg}(x)$	16	$y(x)=x^2+\cos(x)$
7	$y(x)=\sin^2(x^2)+14*\sin(x)*\cos(x)$	17	$y(x)=\log(x)-45*\text{tg}(x)$
8	$y(x)=\text{tg}(x)-45*\text{tg}(x)$	18	$y(x)=\text{ctg}(x)*\log(x)$
9	$y(x)=\cos(x)/40+\sin(x)+\cos(x)$	19	$y(x)=-45*\text{tg}(x)/\cos(x)$
10	$y(x)=\ln(x)+14*\sin(x)$	20	$y(x)=\sin^2(x^2)*2$

Ход выполнения работы

1. Добавить на форму компоненты согласно рисунку:



2. Написать исходный код основного модуля:

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs, StdCtrls, ExtCtrls, Buttons, Unit_External, Math;  
  
type  
  TForm1 = class (TForm)  
    BitBtn1: TBitBtn;  
    LabeledEdit1: TLabeledEdit;  
    RadioGroup1: TRadioGroup;  
    procedure BitBtn1Click(Sender: TObject);  
  private  
    procedure calculateFromThisUnit;  
    procedure calculateFromExternalUnit;  
    { Private declarations }  
  public  
    { Public declarations }  
  end;  
  
var  
  Form1: TForm1;  
  
implementation  
  
{ $R *.dfm }  
  
function getY(x : real) : real;  
begin  
  result := cos(x) + sin(x);  
end;
```

```

function getX(LE : TLabelledEdit) : real;
begin
    try
        Result:=StrToFloat (LE.Text);
    except
        ShowMessage('Parsing error. ');
        Halt(10);
        Result := null;
    end;
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    case RadioGroup1.ItemIndex of
        0 : calculateFromThisUnit;
        1 : calculateFromExternalUnit;
    else ShowMessage('Не выбран модуль для вызова функции. ');
    end;

end;

procedure TForm1.calculateFromThisUnit;
var x, y: Real;
begin
    x := getX(LabeledEdit1);
    y := getY(x);
    ShowMessage(FloatToStr(y));
end;

procedure TForm1.calculateFromExternalUnit;
var x, y: Real;
begin
    x := getX(LabeledEdit1);
    y := Unit_External.getY(x);
    ShowMessage(FloatToStr(y));
end;
end.

```

3. Написать исходный код подключаемого модуля:



```
unit Unit_External;  
  
interface  
uses  
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms;  
  
  function getY(x : real) : real;  
  
implementation  
  
function getY(x : real) : real;  
begin  
  result := cos(x) + sin(x);  
end;  
end.
```

4. Реализовать вызов математических функций согласно варианту. Изменить формулу расчета функции $y(x)$ в исходном коде программы.

Вопросы для контроля по лабораторной работе № 1

1. Для чего применяются подпрограммы? В чем их преимущество?
2. Какие виды подпрограмм существуют?
3. В чем отличие функции от процедуры?
4. Опишите основные этапы работы с модулями (unit).

Компонент	Тип	Описание
 кнопка с графикой	BitBtn	Используется для создания кнопок, на которых располагается битовая графика (например, кнопка ОК с галочкой). Компонент визуальный.
 кнопка с фиксацией	SpeedButton	Используется для создания инструментальных панелей и в других случаях, когда требуется кнопка с фиксацией нажатого состояния. Компонент визуальный.
 маскированный ввод	MaskEdit	Используется для форматирования данных или для ввода символов в соответствии с шаблоном. Компонент визуальный.
 таблица строк	StringGrid	Используется для отображения текстовой информации в таблице из строк и столбцов. Компонент визуальный.
 таблица рисунков	DrawGrid	Используется для отображения в строках и столбцах нетекстовых данных. Компонент визуальный.
 изображение	Image	Используется для отображения графики: пиктограмм, битовых матриц и метафайлов. Компонент визуальный.
 форма	Shape	Используется для рисования фигур: квадратов, кругов и т.д. Компонент визуальный.
 рамка	Bevel	Используется для рисования выступающих или утопленных линий или прямоугольных рамок. Компонент визуальный.
 окно с прокруткой	ScrollBar	Используется для создания зон отображения с прокруткой. Компонент визуальный.
 список с флажками	CheckBox	Компонент является комбинацией свойств списка ListBox и индикаторов CheckBox в одном компоненте. Компонент визуальный.
 разделитель панелей	Splitter	Используется для создания в приложении панелей с изменяемыми пользователем размерами. Компонент визуальный.
 метка с бордюром	StaticText	Компонент подобен компоненту Label, но обеспечивает дополнительные возможности по заданию стиля бордюра. Компонент визуальный.
 инструментальная панель	ControlBar	Используется для размещения компонентов инструментальной панели. Компонент визуальный.

 события приложения	ApplicationEvents	Перехватывает события на уровне приложения. Компонент не визуальный.
 диаграммы и графики	Chart	Компонент принадлежит к семейству компонентов TChart, которые используются для создания диаграмм и графиков. Компонент визуальный.

Страница Win32 содержит компоненты общего назначения, позволяющие разрабатывать приложения в стиле Windows 95/98 и NT 4.x.




Компонент	Тип	Описание
 страница с закладкой	TabControl	Позволяет организовывать страницы с закладками в стиле Windows 95, которые может выбирать пользователь. Компонент визуальный.
 многостраничное окно	PageControl	Позволяет создавать страницы в стиле Windows 95/98, управляемые закладками или иными органами управления, для локальной работы на рабочем столе. Компонент визуальный.
 список изображений	ImageList	Предназначен для работы со списками изображений одинакового размера в меню, инструментальных панелях и т.п. Компонент визуальный.
 окно редактирования в формате RTF	RichEdit	Представляет собой окно редактирования в стиле Windows 95/98, позволяющее применять выбор цвета и шрифта, поиск текста и многое другое. Компонент визуальный.
 ползунок	TrackBar	Управляющий элемент в виде ползунка в стиле Windows 95/98. Компонент визуальный.
 отображение хода процесса	ProgressBar	Используется для отображения в стиле Windows 95/98 хода процессов, занимающих заметное время. Компонент визуальный.
 кнопка-счетчик	UpDown	Кнопки-счетчик в стиле Windows 95/98 для ввода целых чисел. Компонент визуальный.
 горячие клавиши	HotKey	Дает возможность реализовать в приложениях поддержку горячих клавиш. Компонент визуальный.

кнопки		визуальный.
 воспроизведение немых кино	Animate	Используется для воспроизведения немых кино AVI, подобных используемым в Windows 95/98 графическим копированием файлов и т.п. Компонент визуальный.
 ввод дат и времени	Date Time Picker	Ввод дат и времени с выпадающим календарем. Компонент визуальный.
 ввод дат	Month Calendar	Ввод дат с выбором из календаря. Компонент визуальный.
 дерево	TreeView	Предоставляет возможность просмотра структуры иерархических данных в стиле Windows 95/98. Компонент визуальный.
	List View	Отображает списки в стиле Windows 95/98. Компонент визуальный.
списки		
 заголовок	Header Control	Позволяет создавать вкладки переключаемые заголовки в стиле Windows 95/98. Компонент визуальный.
 полоса состояния	Status Bar	Полоса состояния программы, при необходимости — на нескольких панелях. Компонент визуальный.
 инструментальная панель	ToolBar	Инструментальная панель для быстрого доступа к часто используемым функциям приложения. Компонент визуальный.
 инструментальная перестраиваемая панель	CoolBar	Контейнер инструментальной панели, размеры которой могут изменяться пользователем. Компонент визуальный.
 прокрутка страниц	Page Scroller	Обеспечивает прокрутку больших окон, например, инструментальных панелей. Компонент визуальный.

Страница System содержит компоненты, позволяющие использовать системные средства Windows.



Компонент Timer позволяет задавать в приложении интервалы времени.

Компонент	Тип	Описание
 таймер	Timer	Используется для запуска процедур, функций и событий в указанные интервалы времени. Компонент невизуальный.

Таймер находит многочисленные применения: синхронизация мультимедиа, закрытие каких-то окон, с которыми пользователь долгое время не работает, включение хранителя экрана или закрытие связей с удаленным сервером при отсутствии действий пользователя, регулярный опрос каких-то источников информации, задание времени на ответ в обучающих программах — все это множество задач, в которых требуется задавать интервалы времени, решается с помощью таймера.

Таймер — невизуальный компонент, который может размещаться в любом месте формы.

Он имеет два свойства, позволяющие им управлять:

Interval — интервал времени в миллисекундах







Enabled — доступность.

Свойство *Interval* задает период срабатывания таймера. Через заданный интервал времени после предыдущего срабатывания, или после программной установки свойства *Interval*, или после запуска приложения, если значение *Interval* установлено во время проектирования, таймер срабатывает, вызывая событие *OnTimer*. В обработчике этого события записываются необходимые операции.

Если задать *Interval* = 0 или *Enabled* = false, то таймер перестает работать. Чтобы запустить отсчет времени надо или задать *Enabled* = true, если установлено положительное значение *Interval*, или задать положительное значение *Interval*, если *Enabled* = true.

Например, если требуется, чтобы через 5 секунд после запуска приложения закрылась форма — заставка, отображающая логотип приложения, на ней надо разместить таймер, задать в нем интервал *Interval* = 5000, а в обработчик события *OnTimer* вставить оператор *Close*, закрывающий окно формы.

Таймер точно выдерживает заданные интервалы *Interval*, если они достаточно велики — сотни и тысячи миллисекунд. Если же задавать интервалы длительностью десятки или единицы миллисекунд, то реальные интервалы времени оказываются заметно больше вследствие различных накладных расходов, связанных с вызовами функций и иными вычислительными аспектами.

 панель для рисования	PaintBox	Используется для создания на экране рабочей области, в которой можно рисовать. Компонент визуальный.
 аудио и видео плеер	MediaPlayer	Используется для создания панели управления непосредственно звуками и видео файлами, а также устройства мультимедиа. Компонент визуальный.
 контейнер OLE	OLEContainer	Используется при создании области клиента для объекта OLE. Компонент визуальный.
 диалог с сервером DDE	DDEClientConv	Используется клиентом DDE для организации диалога с сервером DDE. Компонент визуальный.
 данные, передаваемые серверу DDE	DDEClientItem	Используется для определения данных клиента, передаваемых в диалоге серверу DDE. Компонент не визуальный.
 диалог с клиентом DDE	DDEServerConv	Компонент используется сервером DDE при проведении диалога с клиентом DDE. Компонент визуальный.
 данные, передаваемые клиенту DDE	DDEServerItem	Компонент используется для определения данных сервера, передаваемых клиенту DDE в течение диалога. Компонент визуальный.

Практическое задание 1

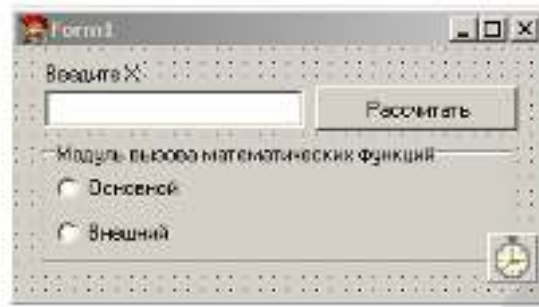
В соответствии со своим вариантом на основе предыдущей лабораторной работы разработать программу, которая позволяет производить математические вычисления со сдвигом во времени (компонент Timer). Отложенный запуск осуществить для функции находящейся во вспомогательном модуле.

Практическое задание 2

В соответствии со своим вариантом на основе предыдущей лабораторной работы разработать 2 программы, которые позволяют производить математические вычисления, и используют технологию DDE. В первой программе должен быть реализован ввод значения аргумента и передача этого значения во вторую программу. Вторая программа должна при получении аргумента вычислять значение функции, и передавать его в первую программу.

Ход выполнения задания 1

1. Добавить на разработанную ранее форму (см. лаб. работу 1) компонент «таймер»:



2. Написать исходный код основного модуля:

```
unit UDDESERV;  
  
interface  
  
uses  
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
    Dialogs, DdeMan, StdCtrls;  
  
type  
    TForm1 = class(TForm)  
        Label1: TLabel;  
        DdeServerItem1: TDdeServerItem;  
        DdeServerConv1: TDdeServerConv;  
        Memo1: TMemo;  
        Label2: TLabel;  
        procedure DdeServerItem1PokeData(Sender: TObject);  
    private  
        { Private declarations }  
    public  
        { Public declarations }  
    end;  
  
var  
    Form1: TForm1;  
  
implementation  
  
uses Math;  
  
{$R *.dfm}  
  
function getY(x : real) : real;  
begin  
    result := cos(x) + sin(x);  
end;
```

```

function getX(LE : TLabelEdit) : real;
begin
    try
        Result:=StrToFloat(LE.Text);
    except
        ShowMessage('Parsing error. ');
        Halt(10);
        Result := null;
    end;
end;

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    case RadioGroup1.ItemIndex of
        0 : calculateFromThisUnit;
        1 : Timer1.Enabled := true;
    else ShowMessage('Не выбран модуль для вызова функции. ');
    end;

end;

procedure TForm1.calculateFromThisUnit;
var x, y: Real;
begin
    x := getX(LabeledEdit1);
    y := getY(x);
    ShowMessage(FloatToStr(y));
end;

procedure TForm1.calculateFromExternalUnit;
var x, y: Real;
begin
    x := getX(LabeledEdit1);
    y := Unit_External.getY(x);
    ShowMessage(FloatToStr(y));
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    calculateFromExternalUnit;
    Timer1.Enabled := false;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    Timer1.Enabled := false;
    Timer1.Interval := 3000;
end;
end.

```

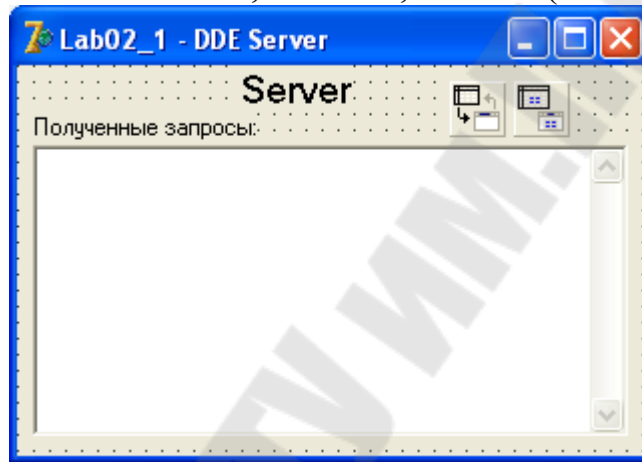
Суть вносимых изменений в добавлении методов FormCreate() и Timer1Timer(). А также по установке значения Timer1.Enabled = true. Это значение активирует таймер.

Ход выполнения задания 2

I часть – сервер

1. Создаем проект с названием DDESERV (это имя будет использоваться в дальнейшем).

2. Добавить на форму следующие компоненты: TDdeServerItem, TDdeServerConv, TMemo, TLabel (согласно рисунку).



3. Написать исходный код основного модуля:

```
unit UDDESERV;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DdeMan, StdCtrls;

type
  TForm1 = class (TForm)
    Label1: TLabel;
    DdeServerItem1: TDdeServerItem;
    DdeServerConv1: TDdeServerConv;
    Memo1: TMemo;
    Label2: TLabel;
    procedure DdeServerItem1PokeData(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Math;

({$R *.dfm})

function getY(x : real) : real;
begin
  result := cos(x) + sin(x);
end;
```

```

function getX(): real;
begin
  try
    Form1.Memo1.Lines.AddStrings(Form1.DdeServerItem1.Lines);
    Result := StrToFloat(Form1.DdeServerItem1.Lines[0]);
  except
    Form1.DdeServerItem1.Text := 'Неверный формат данных. Введите число!';
    Result := Infinity;
  end;
end;

procedure TForm1.DdeServerItem1PokeData(Sender: TObject);
var x, y: real;
begin
  x := getX();
  if (x <> Infinity) then
  begin
    try
      y := getY(x);
      DdeServerItem1.Text := FloatToStr(y);
    except
      DdeServerItem1.Text := 'На сервере произошла ошибка!';
    end;
  end;
end;
end.

```

4. Установить свойство `ServerConv = DdeServerConv1` для компоненты `DdeServerItem1`. Аналогично установить событие `ServerConv = DdeServerConv1`. А также событие `DdeServerItem1PokeData()` (см. исходный код выше).

5. Скомпилировать приложение.

II часть – клиент

6. В этой же папке (что и проект для сервера) создаем проект с названием `DDECLI`.

7. Добавить на форму для программы выполняющей функции клиента `DdeClientConv`, `DdeClientItem`, `TEdit`, `TLabel` (согласно рисунку):



8. Написать исходный код основного модуля:

```
unit UDDECLI;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DdeMan, StdCtrls;

type
  TForm1 = class (TForm)
    Edit1: TEdit;
    Label1: TLabel;
    DdeClientItem1: TDdeClientItem;
    DdeClientConv1: TDdeClientConv;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
    Button1: TButton;
    procedure DdeClientItem1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  DdeClientItem1.DdeItem := 'DdeServerItem1';
end;

procedure TForm1.DdeClientItem1Change(Sender: TObject);
begin
  Edit2.Text := DdeClientItem1.Text;
end;

procedure TForm1.Button1Click(Sender: TObject);
var strings: TStringList;
begin
  strings := TStringList.Create;
  strings.Add(Edit1.Text);
  DdeClientConv1.PokeDataLines(DdeClientItem1.DdeItem, strings);
end;

end.
```

9. Запустить приложение «DDESERV.exe».
10. Настроить свойства у компонента DdeClientConv1:
ServiceApplication = DDESERV
DdeService = DDESERV
DdeTopic = DdeServerConv1
11. Настроить свойства у компонента DdeClientItem1:
DdeConv = DdeClientConv1
DdeItem = DdeServerItem1

А также события:

DdeConv = DdeClientConv1
OnChange = DdeClientItem1Change

Вопросы для контроля по лабораторной работе № 2

1. Перечислите основные компоненты палитры Additional и их свойства.
2. Перечислите основные компоненты палитры Win32 и их свойства.
3. Перечислите основные компоненты палитры System и их свойства.

Лабораторная работа № 3 Создание динамических библиотек (DLL) в Delphi

Цель работы: научиться работать с динамическими библиотеками.

Теоретические сведения

DLL являются неотъемлемой частью функционирования операционных систем семейства Microsoft Windows. DLL - это один или несколько логически законченных фрагментов кода, сохраненных в файле с расширением.dll. Этот код может быть запущен на выполнение в процессе функционирования какой-либо другой программы (такие приложения называются вызываемыми по отношению к библиотеке), но сама DLL не является запускаемым файлом.

Существует два типа динамических библиотек - исполняемые и библиотеки ресурсов. Однако это не означает, что в одном файле не может находиться и код некоторой функции и какие-либо ресурсы.

Разработка динамических библиотек не представляет собой некий сверхсложный процесс, доступный лишь избранным. Если вы достаточно хорошо знакомы с разработкой приложений на Object Pascal, то вам не составит особого труда научиться работать с механизмом DLL. Итак, рассмотрим те особенности создания DLL, которые вам необходимо знать, а в завершении статьи разработаем свою собственную библиотеку.

Как и любой другой модуль, модуль динамической библиотеки имеет фиксированный формат.

```
library MyFirstDLL;
uses SysUtils, Classes, Forms, Windows;

procedure HelloWorld(AForm : TForm);
begin
    MessageBox(AForm.Handle, 'Hello world!',
        'DLL Message Box', MB_OK or
            MB_ICONEXCLAMATION);
end;

exports
    HelloWorld;
```

```
begin  
end.
```

Для экспорта процедур и функций из DLL, необходимо использовать ключевое слово `export`.

Существуют следующие способы экспорта процедур и функций: экспорт по имени и экспорт по порядковому номеру.

Практическое задание

Разработать динамическую библиотеку, позволяющую производить математические вычисления из задания для лабораторной работы №1. Модифицировать программу из лабораторной работы №2 для работы с разработанной библиотекой. Вызов подпрограмм из библиотеки осуществить различными способами (статически, динамически и по индексу).

Ход выполнения работы

Часть 1. Создание DLL

1. Создать новый проект, выбрав в меню DLL File => New => DLL Wizard.
2. Сохранить проект с новым именем (например, TestLibrary).
3. Написать исходный код основного модуля:

```

library TestLibrary;

uses
  SysUtils,
  Classes;

({R *.res})

function getY(x : real): real; stdcall;
begin
  result := sin(x) + cos(x);
end;

function getPartY(x : real): real; stdcall;
begin
  result := sin(x);
end;

exports
  getY index 1,
  getPartY index 2;

begin
end.

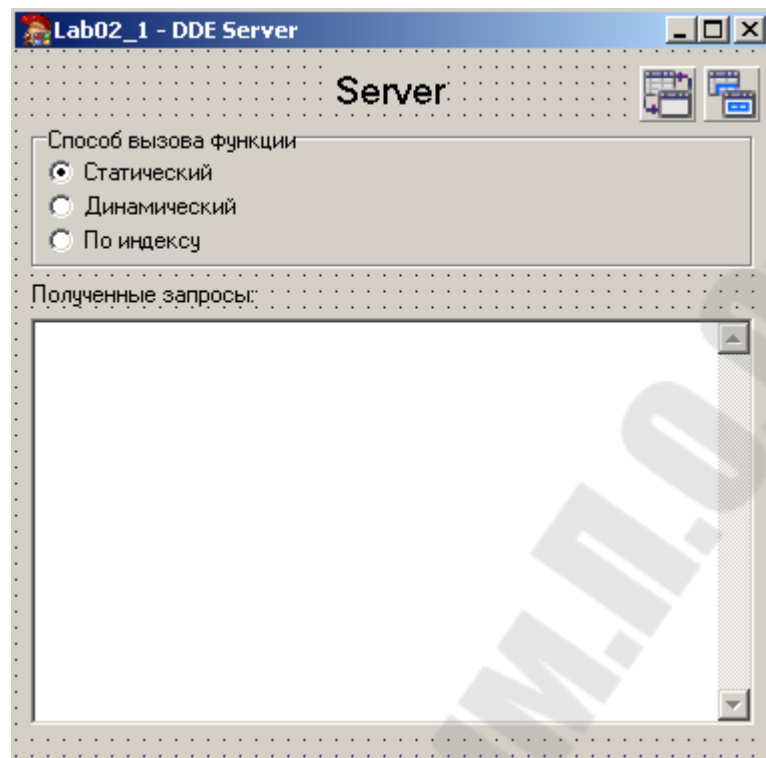
```

4. Скомпилировать библиотеку, нажав комбинацию клавиш «Ctrl + F9» или выбрав в меню: Project => Compile TestLibrary.

Часть 2. Вызов функций из DLL

5. Скопировать в другую папку проект, разработанный для лабораторной работы №2.

6. Открыть проект, выполняющий функции сервера и добавить на форму компонент RadioGroup согласно рисунку:



7. Изменяем исходный код основного модуля на следующий:

```
unit ПППКСКВ;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DdeWin, SdcOcs, ErcOcs, Ksch;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    DdeServerItem1: TDdeServerItem;
    DdeServerConn1: TDdeServerConn;
    Memo1: TMemo;
    Label2: TLabel;
    DdeGroup1: TDdeGroup;
    procedure DdeServerItem1PokeData(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

function getDynamicConnY(x: TConn): TConn;
```



```

var
  Form1: TForm1;

function getY(x : real): real; stdcall; external 'TestLibrary.dll';

function getByIndexY(x : real): real; stdcall; external 'TestLibrary.dll' index 1;

implementation

{$R *.dfm}

function getX(): real;
begin
  try
    Form1.Memo1.Lines.AddStrings(Form1.DdeServerItem1.Lines);
    Result := StrToFloat(Form1.DdeServerItem1.Lines[0]);
  except
    Form1.DdeServerItem1.Text := 'Неверный формат данных. Введите число!';
    Result := Infinity;
  end;
end;

procedure TForm1.DdeServerItem1PokeData(Sender: TObject);
var x, y: real;
begin
  x := getX();
  if (x <> Infinity) then
    begin
      try
        case RadioGroup1.ItemIndex of
          0: y := 10 + getY(x);
          1: y := 20 + getDynamicLoadY(x);
          2: y := 30 + getByIndexY(x);
        end;
        DdeServerItem1.Text := FloatToStr(y);
      except
        DdeServerItem1.Text := 'На сервере произошла ошибка!';
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  //Устанавливаем статический способ вызова по умолчанию
  RadioGroup1.ItemIndex := 0;
end;

```

```

function getDynamicLoadY(x: real): real;
type
  TgetY = function(x : real): real;
var
  DLLInstance : THandle;
  getY : TgetY;
begin
  //Загрузка DLL
  DLLInstance := LoadLibrary('TestLibrary.dll');
  if (DLLInstance = 0) then begin
    MessageDlg('Не удалось загрузить DLL.', mtError, [mbOK], 0);
    Exit;
  end;

  //Устанавливаем связь с процедурой
  @getY := GetProcAddress(DLLInstance, 'getY');

  if (@getY <> nil) then
    Result := getY(x)
  else
    begin
      Result := Infinity;
      MessageDlg('Не удалось найти процедуру.', mtError, [mbOK], 0);
    end;

  //Выгружаем DLL
  FreeLibrary(DLLInstance);
end;

end.

```

Суть изменений: добавлены вызовы функций (статический, по индексу и динамический), событие FormCreate() устанавливающее статический вызов функции по умолчанию и добавлен выбор способа вызова с учетом выбранного пункта в RadioGroup.

Вопросы для контроля по лабораторной работе № 3

1. Основы разработки динамических библиотек.
2. Правила экспорта функций из DLL.
3. Правила импорта функция в программу из сторонних DLL.

Лабораторная работа № 4

Хранение форм в динамических библиотеках

Цель работы: научиться создавать DLL, содержащие внутри ресурсы (формы).

Теоретические сведения

В DLL можно хранить не только код, но и формы. Причем создание и помещение форм в динамическую библиотеку не слишком сильно отличается от работы с формами в обычном проекте. В ходе выполнения данной лабораторной работы рассмотрен пример, каким образом можно написать библиотеку, содержащую обычные формы и дочерние формы, созданные по технологии MDI.

Практическое задание

Разработать пользовательскую форму и поместить ее в динамическую библиотеку. Выполнить вызов формы из другой программы.

Ход выполнения работы

Часть 1. Создание DLL

1. Создать новый проект, выбрав в меню DLL File => New => DLL Wizard.
2. Сохранить проект с новым именем (например, FormLibrary).
3. Написать исходный код основного модуля:

```

library FormLibrary;

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Unit1 in 'Unit1.pas' {frmCld};

{$R *.res}
var DllApp:TApplication;

procedure ShowMDIChild(MainApp : TApplication);
var
  Child : TfrmCld;
begin
  if not Assigned(DllApp) then
  begin
    DllApp := Application;
    Application := MainApp;
  end;
  Child := TfrmCld.Create(Application.MainForm);
  Child.Show;
end;

procedure MyDLLProc(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
  { DLL is выгружается. Восстанавливаем значение указателя Application}
  if Assigned(DllApp) then Application := DllApp;
end;

exports ShowMDIChild;

begin
  DLLProc := @MyDLLProc;
end.

```

4. Создать пустую форму со следующим исходным кодом модуля:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TfrmCld = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmCld: TfrmCld;

implementation

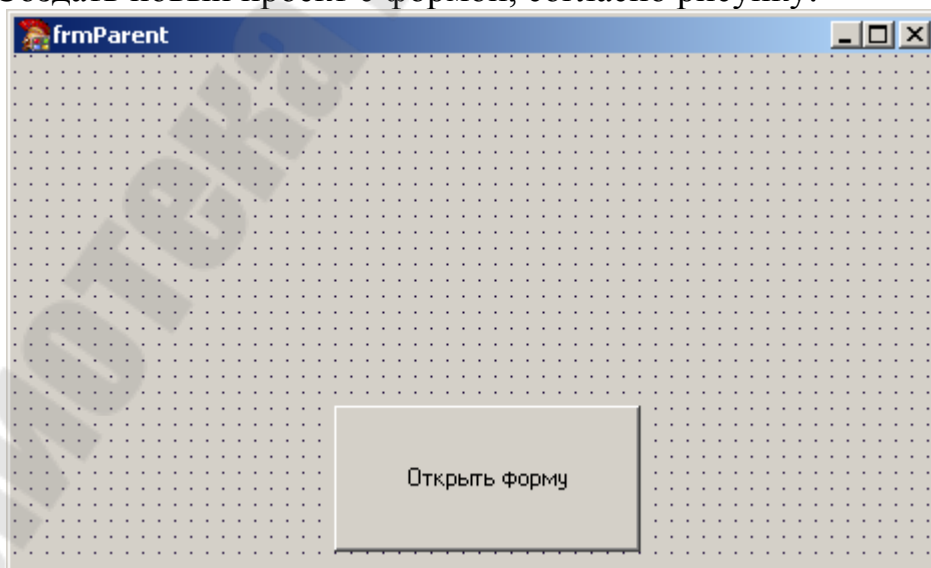
{$R *.dfm}

end.
```

5. Скомпилировать библиотеку, нажав комбинацию клавиш «Ctrl + F9» или выбрав в меню: Project => Compile FormLibrary.

Часть 2. Вызов формы из DLL

6. Создать новый проект с формой, согласно рисунку:



7. Скопировать разработанную в части 1, библиотеку «FormLibrary.dll» в папку с проектом.

8. Написать исходный код основного модуля:

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TfrmParent = class (TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmParent: TfrmParent;

procedure ShowMDIChild(MainApp : TApplication); external 'FormLibrary.dll';

implementation

({$R *.dfm})

procedure TfrmParent.Button1Click(Sender: TObject);
begin
  ShowMDIChild(Application)
end;

end.
```

Вопросы для контроля по лабораторной работе № 4

1. Основные этапы создания форм в DLL.
2. Основные особенности сохранения дочерних форм по технологии MDI в DLL.
3. Правила экспорта функций из DLL.
4. Правила импорта функция в программу из сторонних DLL.

Лабораторная работа № 5

Использование сторонних динамических библиотек в Delphi

Цель работы: научиться использовать сторонние библиотеки.

Теоретические сведения

Для экспорта процедур и функций из DLL, необходимо использовать ключевое слово `export`. Существуют следующие способы экспорта процедур и функций: экспорт по имени и экспорт по порядковому номеру. Наиболее распространенный способ экспорта - по имени.

Следует обратить внимание на то, что Delphi автоматически назначает порядковый номер каждой экспортируемой функции (процедуре) независимо от того, определяете вы его явно или нет. Явное определение индекса позволяет вам лично управлять порядковым номером экспортируемой функции или процедуры.

Для того, чтобы определить выполняется ли ваш код в DLL или в вызывающем приложении, можно воспользоваться глобальной переменной `IsLibrary`. Она принимает значение `true` в том случае, если код вызывается из библиотеки и `false` в случае выполнения процедуры или функции из вызывающего приложения.

Кроме этого, в поставку Delphi входит весьма полезная утилита `tdump`, которая предоставляет данные о том, какая информация экспортируется из указанной DLL.

Практическое задание

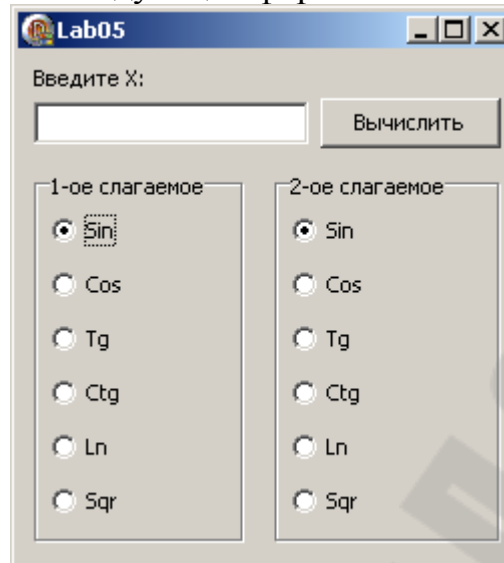
Разработать программу, использующую процедуры и функции сторонней библиотеки. Для получения списка экспортируемых процедур и функций использовать утилиту `Tdump`.

Ход выполнения работы

1. Получить при помощи утилиты `TDUMP` список доступных к использованию функций библиотеки `MathLibrary.dll`. Для этого необходимо либо запустить файл «`Run_tdump.bat`», либо выполнить команду в командной строке Windows:

```
tdump MathLibrary.dll > dump.txt -ee
```

2. Создать проект со следующей формой:



3. Написать исходный код основного модуля:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Math;

type
  TForm1 = class(TForm)
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
    Button1: TButton;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  function cosExt(x : real): real; stdcall; external 'MathLibrary.dll';
  function sinExt(x : real): real; stdcall; external 'MathLibrary.dll';
  function tgExt(x : real): real; stdcall; external 'MathLibrary.dll';
  function ctgExt(x : real): real; stdcall; external 'MathLibrary.dll';
  function lnExt(x : real): real; stdcall; external 'MathLibrary.dll';
  function sqrExt(x : real): real; stdcall; external 'MathLibrary.dll';
```



```

implementation

{$R *.dfm}
function getX(Edit : TEdit) : real;
begin
  try
    Result:=StrToFloat(Edit.Text);
  except
    Result := Infinity;
  end;
end;

function getFunctionValue(RadioGroup : TRadioGroup; x: real) : real;
begin
  case RadioGroup.ItemIndex of
    0: result := sinExt(x);
    1: result := cosExt(x);
    2: result := tgExt(x);
    3: result := ctgExt(x);
    4: result := lnExt(x);
    5: result := sqrExt(x);
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var x, y: real;
begin
  x:= getX(Edit1);
  if (x <> Infinity) then
  begin
    y := getFunctionValue(RadioGroup1, x) + getFunctionValue(RadioGroup2, x);
    ShowMessage(FloatToStr(x));
  end
  else
    ShowMessage('Неверный формат числа.');
```

```

end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.ItemIndex := 0;
  RadioGroup2.ItemIndex := 0;
  Edit1.Text := '2';
end;

end.
```

Вопросы для контроля по лабораторной работе № 5

1. Какие ключевые слова применяются в Delphi при импорте функций из сторонних DLL.
2. Какие существуют способы экспорта процедур и функций из сторонних DLL?
3. Как узнать список экспортируемых из сторонней DLL процедур и функций?

Лабораторная работа № 6

Создание простейшего приложения с помощью WinAPI

Цель работы: научиться разрабатывать простейшее приложение с помощью WinAPI. Получить первоначальный опыт работы с WinAPI.

Теоретические сведения

Windows API (англ. application programming interfaces) — общее наименование целого набора базовых функций интерфейсов программирования приложений операционных систем семейств Windows и Windows NT корпорации «Майкрософт». Является самым прямым способом взаимодействия приложений с Windows.

Windows API был изначально спроектирован для использования в программах, написанных на языке C (или C++).

Работа через Windows API — это наиболее близкий к системе способ взаимодействия с ней из прикладных программ. Более низкий уровень доступа, необходимый только для драйверов устройств, в текущих версиях Windows предоставляется через Windows Driver Model.

WinAPI - это те функции которыми управляется работа приложений в Windows. Они являются частью системы, и подгружаются вместе с windows в библиотеке kernel32.dll.

В Delphi эти функции преимущественно описаны в библиотеке Windows, которая автоматически включается в ваш новый проект. Вы можете открыть эту библиотеку и посмотреть сами. Большая часть VCL — это надстройка над WinAPI.

Для каждого запущенного приложения создается процесс и в этом процессе основной поток (приложение может создавать свои дополнительные потоки — все они будут принадлежать его процессу), а уж потоки создают окна.

Каждый поток имеет уникальный числовой идентификатор называемый ThreadID. Это просто целое число которое дается (ассоциируется) этому потоку. Точно так же имеет свой уникальный идентификатор каждое окно в системе, называемый Handle. Он обозначается обычно типом HWND, но это просто целое. 4-х байтное.

В windows взаимодействие построено на сообщениях.

Сообщение - это небольшой набор данных (record, условно говоря), который содержит:

- Handle — Handle окна, которому сообщение предназначается.
- Message — целое число, которое указывает, что же это за сообщение.

- wParam - целое, значение зависит от сообщения
- lParam - целое, значение зависит от сообщения.

Для системных сообщений определены константы типа WM_KEYPRESSED, WM_MOUSEMOVE и т. д.

Их значение (числовое) совершенно никого не интересует, однако его легко узнать:

```
ShowMessage('WM_MOUSEMOVE: ' + IntToStr(WM_MOUSEMOVE));
```

Для каждого потока отводится специальное место в памяти, куда складываются сообщения по мере их поступления – называется это очередью сообщений.

Сами окна сообщения не получают – все они складываются в очередь потока. Чтобы достать следующее сообщение, используется функция GetMessage(PeekMessage).

Если вы хотите доставить сообщение окну, то проще всего это сделать, вызвав DispatchMessage, передав в качестве параметра полученное сообщение.

Эта функция находит нужное окно в вашем потоке, и вызывает WindowsProc – процедуру окна, которая должна обработать это сообщение.

Адрес этой процедуры (для каждого окна свой) известен системе – он передается ей во время регистрации окна.

Практически каждое приложение осуществляет цикл обработки сообщений. То есть цикл, который вызывает GetMessage и обрабатывает сообщение (рассылает окнам), пока не попадет сообщение WM_QUIT, после чего приложение должно завершить работу.

В Delphi этот цикл представлен в методе Application.Run.

Поведение окна и его внешний вид определяются классом окна. Класс – это внутренняя структура Windows, описывающая шаблон, на основании которого операционная система создает окна. Перед созданием окна Вы должны зарегистрировать его класс при помощи функции:

```
function RegisterClassEx(
  const WndClass: TWndClassEx
): ATOM; stdcall;
```

После того, как класс зарегистрирован, приложение может создавать окна этого класса функцией:

```
function CreateWindowEx(
  dwExStyle: DWORD; // расширенный стиль окна
  lpClassName: PChar; // имя класса
  lpWindowName: PChar; // заголовок окна
  dwStyle: DWORD; // стиль окна
```

X, Y, nWidth,
nHeight: Integer; // размеры и позиция на экране
hWndParent: HWND; // идентификатор окна-владельца
hMenu: HMENU; // идентификатор меню окна
hInstance: HINST; // идент-ор модуля, ассоц-ного с окном
lpParam: Pointer // дополнительный параметр, передаваемый в
// оконную процедуру с сообщением WM_CREATE
) : HWND; stdcall;

Функция CreateWindowEx позволяет задать конкретный вид окна и уточнить информацию, полученную от класса окна.

Практическое задание

Разработать приложение с помощью WinAPI, которое представляет собой пустую форму с заголовком. В заголовке формы отобразить фразу «Hello, World! (N)», где N – номер варианта.

Ход выполнения работы

1. Создать консольное приложение и написать исходный код в файле с расширением *.dpr.

```

program Pr11;

uses
  Windows, Messages, Graphics;

{$R *.res}

var
  wnd: HWND;
  Msg: TMsg;
  wndclass: TWndClass;
  Canvas: TCanvas;

function WndProc(wnd: HWND; iMsg: UINT; wParam: WPARAM; lParam: LPARAM): LResult stdcall;
var
  dc: HDC;
  ps: PAINTSTRUCT;
begin
  case iMsg of
    WM_CREATE:
      begin
        // здесь указывается стиль окна
        SetWindowLong(wnd, GWL_EXSTYLE, Integer(GetWindowLong(wnd, GWL_EXSTYLE) or WS_EX_LAYERED));
        // здесь оно делается прозрачным на 200/255
        // если ты хочешь сделать прозрачным
        // определенный цвет, со стилем LWA_COLORKEY
        // а user вторым параметром
        SetLayeredWindowAttributes(wnd, 0, 200, LWA_ALPHA);
      end;
    WM_PAINT:
      begin
        dc:=BeginPaint(wnd, ps);
        TextOut(dc, 10, 10, '21a', 5); // это WinAPI
        Canvas:=TCanvas.Create;
        Canvas.Handle:=dc;
        Canvas.TextOut(10, 30, '21a'); // это VCL
        Canvas.Free;
        EndPaint(wnd, ps);
      end;
  end;
end;

```

```

WM_DESTROY:
begin
PostQuitMessage(0);
Result:=0;
end;
end;
Result:=DefWindowProc(wnd, Msg, wParam, lParam);
end;

begin
// заполняем структуру класса
ZeroMemory(&wndclass, sizeof(wndclass));
wndclass.style=CS_HREDRAW or CS_VREDRAW;
wndclass.lpfnWndProc=@WndProc; // адрес оконной функции
wndclass.hInstance=hInstance;
wndclass.hIcon=LoadIcon(0, IDI_APPLICATION);
wndclass.hCursor=LoadCursor(0, IDC_ARROW);
wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
wndclass.lpszClassName='ZisApp';
// регистрируем его
RegisterClass(&wndclass);
// создаем окно зарегистрированного класса
wnd=CreateWindow('ZisApp', 'Hello, WORD!', WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
0, 0, hInstance, nil);
// показываем его
ShowWindow(wnd, SW_SHOWNORMAL);
UpdateWindow(wnd);
// входим в цикл обработки сообщений
while GetMessage(Msg, 0, 0, 0) do
begin
TranslateMessage(Msg);
DispatchMessage(Msg);
end;
end;
end.

```

Вопросы для контроля по лабораторной работе № 6

1. Что такое WinAPI?
2. Понятие процесса в понимании ОС Windows.
3. Понятие потока в понимании ОС Windows.
4. Понятие сообщения в понимании ОС Windows.
5. Как зарегистрировать класс окна?
6. Как создать окно класса?

Лабораторная работа № 7

Добавление визуальных компонент на форму с помощью WinAPI

Цель работы: научиться создавать визуальные компоненты средствами WinAPI.

Теоретические сведения

Каждый наследник TWinControl может служить контейнером для других компонентов. Для связи между родительским и дочерними компонентами служат свойства Controls и Parent. Каждый визуальный компонент имеет свойство Parent, ссылающееся на оконный компонент, являющийся его владельцем.

До тех пор, пока не установлено это свойство, компонент не может обрабатывать сообщения Windows и отображаться на экране. Окно-предок ведет список вставленных в него компонентов, доступ к которому можно получить при помощи его свойства Controls.

Delphi вводит новую для Windows концепцию – форма. Такого понятия нет в Windows, однако оно довольно часто встречается в высокоуровневых средствах разработки. Форма инкапсулирует окно верхнего уровня (top-level window), которое служит контейнером для остальных визуальных элементов программы.

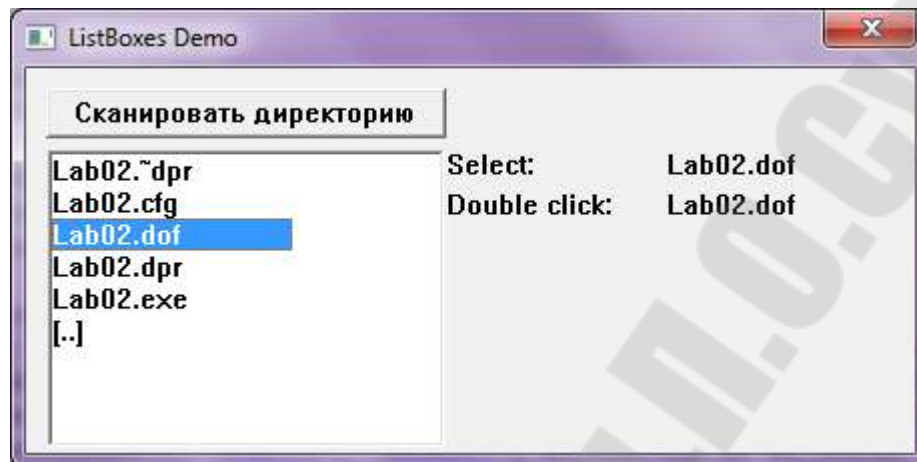
С точки зрения Windows – это такое же окно, как и все остальные, только с соответствующим образом подобранным набором стилей. В частности, на этом уровне реализована поддержка главного меню, управление дочерними окнами (MDI). Как и любое окно TForm имеет свойство Handle, которое может быть использовано в вызовах Windows API.

Концепция форм помогает разработчику в управлении окнами верхнего уровня, однако многие её понятия (например, главное окно приложения, список окон (Screen.Forms)) не имеет прямых аналогов в Windows API.

Практическое задание

Разработать приложение, содержащее на форме следующие компоненты: listbox, кнопка и 4 текстовые метки. По нажатию на кнопку должно происходить сканирование директории, результатом которого является список файлов находящихся в данной папке. Затем

поместить этот список в listbox. При смене выбранного значения в listbox изменять значение первой метки. По двойному клику на элемент из listbox изменять значение метки на текст элемента.



Ход выполнения работы

1. Создать консольное приложение и написать исходный код в файле с расширением *.dpr.


```

program Lab02;
uses windows, messages;

var
  wc: TWndClassEx;
  Wnd: HWND;
  Msg: TMsg;

function WindowProc(wnd : HWND; Msg : Integer;
  WParam : WParam; LParam : Longint): Longint; stdcall;
const BUFFER = MAX_PATH;
var
  I : Integer;
  CurDir : array[0..BUFFER-1] of Char;
begin
  Result := 0;
  case msg of
    wm_destroy: begin
      postquitmessage(0);
      exit;
      Result:=0;
    end;
    wm_command: begin
      case loword(wParam) of
        200:
          case hiWord(wParam) of
            lbn_selchange:
              begin
                I := SendMessage(lParam, LB_GETCURSEL, 0, 0);
                SendMessage(lParam, LB_GETTEXT, I, Integer(@CurDir));
                SetWindowText(GetDlgItem(Wnd, 222), CurDir);
              end;
            lbn_dblick:
              begin
                I := SendMessage(lParam, LB_GETCURSEL, 0, 0);
                SendMessage(lParam, LB_GETTEXT, I, Integer(@CurDir));
                SetWindowText(GetDlgItem(Wnd, 221), CurDir);
              end;
          end;
        100:
          begin
            GetCurrentDirectory(SizeOf(CurDir), CurDir);
            DlgDirList(Wnd, CurDir, 200, 0, DDL_DIRECTORY);
          end;
        end;
      end;
    else
      Result := DefWindowProc(wnd, msg, wParam, lParam);
    end;
  end;
end;

```

```

]Begin
  wc.cbSize := sizeof(wc);
  wc.style := ws_hredraw or ws_vredraw;
  wc.lpfnWndProc := @WindowProc;
  wc.cbClsExtra := 0;
  wc.cbWndExtra := 0;
  wc.hInstance := HInstance;
  wc.hIcon := LoadIcon(0, IDI_APPLICATION);
  wc.hCursor := LoadCursor(0, IDC_ARROW);
  wc.hbrBackground := COLOR_BTNFACE + 1;
  wc.lpszMenuName := nil;
  wc.lpszClassName := 'listbox example';

  RegisterClassEx(wc);
  Wnd := CreateWindowEx(0, 'listbox example', 'ListBoxes Demo',
    ws_overlapped or ws_sysmenu, 100, 150, 450, 220, 0, 0, HInstance, nil);
  CreateWindowEx(0, 'BUTTON', 'Сканировать директорию', BS_PUSHBUTTON or
    WS_CHILD or WS_VISIBLE, 10, 10, 200, 25, Wnd, 100, HInstance, NIL);

  CreateWindowEx(WS_EX_CLIENTEDGE, 'LISTBOX', NIL, WS_BORDER or LBS_NOTIFY or
    WS_CHILD or WS_VISIBLE or LBS_MULTICOLUMN or LBS_SORT or
    LBS_EXTENDEDSEL, 10, 40, 200, 150, Wnd, 200, HInstance, NIL);
  CreateWindowEx(0, 'STATIC', 'Select:', WS_VISIBLE or WS_CHILD,
    210, 40, 60, 20, Wnd, 0, HInstance, NIL);
  CreateWindowEx(0, 'STATIC', NIL, WS_VISIBLE or WS_CHILD,
    320, 40, 300, 20, Wnd, 222, HInstance, NIL);
  CreateWindowEx(0, 'STATIC', 'Double click:', WS_VISIBLE or WS_CHILD,
    210, 60, 100, 20, Wnd, 0, HInstance, NIL);
  CreateWindowEx(0, 'STATIC', NIL, WS_VISIBLE or WS_CHILD,
    320, 60, 300, 20, Wnd, 221, HInstance, NIL);

  ShowWindow(Wnd, CmdShow);

  While GetMessage(Msg, 0, 0, 0) do
  begin
    TranslateMessage(Msg);
    DispatchMessage(Msg);
  end;
]End.

```

Вопросы для контроля по лабораторной работе № 7

1. Класс TWinControl.
2. Свойство Controls.
3. Свойство Parent.

Лабораторная работа № 8

Управление другими приложениями при помощи WinAPI

Цель работы: изучить основные возможности управления и взаимодействия с внешними приложениями.

Теоретические сведения

Сообщения – это базовый механизм информирования программ о событиях, на которые они должны реагировать.

Ядром программы является функция обработки сообщений, зарегистрированная в классе окна, которая вызывается ядром Windows при появлении событий, на которые программа должна отреагировать.

Получение сообщения окном означает вызов его оконной функции с параметрами, описывающими передаваемое сообщение.

Например, сразу после создания окна оно получает сообщение WM_CREATE, при нажатии клавиш на клавиатуре – WM_KEYDOWN, WM_KEYUP, при перемещении мыши WM_MOUSEMOVE и т.п.

Без обработки сообщений окно не сможет даже отрисовать себя – рисование выполняется по получению сообщений WM_PAINT, WM_NCPAINT. В программе, написанной с использованием только WinAPI, функция обработки сообщений обычно представляет собой оператор case, альтернативами которого являются различные сообщения, которые эта функция должна обработать.

Программирование создания окон и цикла обработки сообщений вручную является непростой и довольно низкоуровневой задачей. VCL реализует классы, позволяющие избежать возникающих при этом сложностей.

Базовым классом, инкапсулирующим окно Windows, является TWinControl. При создании экземпляра наследника этого класса, VCL автоматически регистрирует соответствующий класс окна Windows и создает окно. Благодаря этому, наследники TWinControl могут содержать в себе другие окна и обрабатывать сообщения Windows. Визуальные компоненты, не являющиеся наследниками TWinControl (такие, как TLabel, TSpeedButton) не являются окнами в понимании Windows. Все их события эмулируются компонентом, в который они помещены.

Центральным свойством компонента TWinControl является свойство Handle. Это свойство представляет идентификатор окна Windows, полученного при создании этого компонента. Этот идентификатор можно использовать с любыми функциями Windows API, работающими с окнами. Например, следующий код прокручивает текст в TMemo на одну строку вниз:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  PostMessage(Memo1.Handle, WM_VSCROLL, SB_LINEDOWN, 0);  
end;
```

Каждое окно Windows должно обрабатывать сообщения. VCL берет на себя работу по организации цикла сообщений и их базовой обработке. Для большинства сообщений Windows, которые должно обрабатывать окно, уже предусмотрена обработка «по умолчанию».

Сообщения, требующие специфической обработки, приводят к вызовам функций-обработчиков событий, например:

<i>WM_MOUSEMOVE</i>	<i>OnMouseMove</i>
<i>WM_LBUTTONDOWN,</i>	<i>OnMouseDown</i>
<i>WM_RBUTTONDOWN</i>	
<i>WM_LBUTTONUP,</i>	<i>OnMouseUp</i>
<i>WM_RBUTTONUP</i>	
<i>WM_LBUTTONDBLCLK</i>	<i>OnDblClick</i>
<i>WM_KEYDOWN</i>	<i>OnKeyDown</i>
<i>WM_KEYUP</i>	<i>OnKeyUp</i>
<i>WM_PAINT</i>	<i>OnPaint</i>

Показателен в этом смысле метод WndProc класса TWinControl или его наследников.

При этом VCL перед вызовом обработчиков производит «упаковку» параметров сообщений в удобный для обработки и анализа вид. Понимание, какое сообщение Windows вызывает срабатывание того или иного события VCL очень помогает при программировании обработчиков и совершенно необходимо при написании собственных компонентов.

Разумеется, предусматривать отдельные обработчики для всех из сотен сообщений, которые могут поступить в окно – значит неоправданно усложнить код VCL. Поэтому для обработки остальных

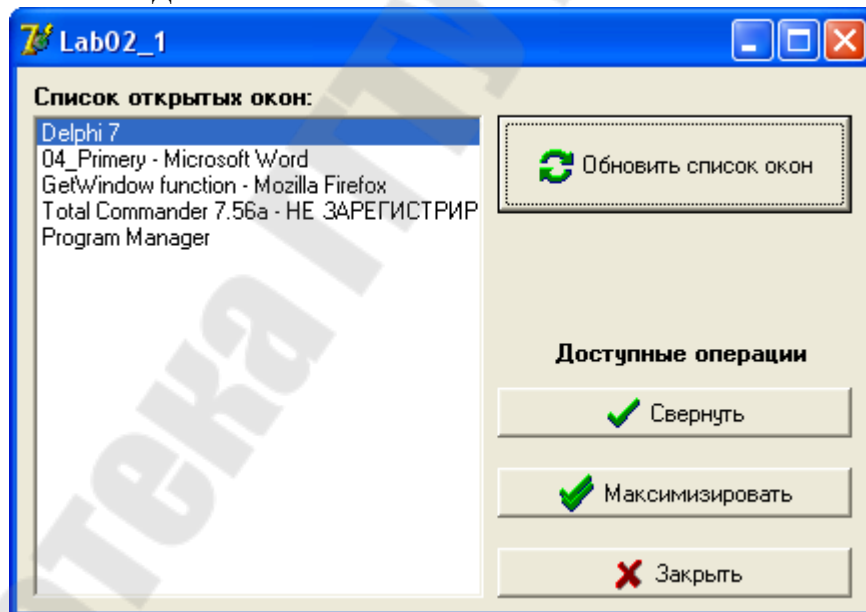
сообщений синтаксис Object Pascal предусматривает создание процедур-обработчиков сообщений.

Такие процедуры объявляются как
procedure WMSize(var Message: TWMSize); message WM_SIZE;

В качестве параметра такая функция получает указатель на структуру TMessage, содержащую информацию о сообщении, переданном окну. Для многих часто используемых сообщений в модуле Messages.pas определены структуры, позволяющие более удобно работать с конкретными сообщениями.

Практическое задание

Разработать приложение (используя VCL), содержащее на форме следующие компоненты: listbox и четыре кнопки. По нажатию на первую кнопку в Listbox должен загрузиться список заголовков открытых окон системы. Три оставшиеся кнопки должны соответственно отвечать за операцию «сворачивания/разворачивания», «минимизации/максимизации» и «закрытия» окна выделенного в списке.



Ход выполнения работы

1. Написать исходный код основного модуля:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Label1: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    Label2: TLabel;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```

procedure TForm1.BitBtn1Click(Sender: TObject);
VAR
  Wnd : hWnd;
  buff: ARRAY [0..127] OF Char;
begin
  ListBox1.Clear;
  Wnd := GetWindow(Handle, gw_HWndFirst);
  WHILE Wnd <> 0 DO
    BEGIN {Не показываем}
      IF (Wnd <> Application.Handle) AND {-Собственное окно}
        IsWindowVisible(Wnd) AND {-Невидимые окна}
          (GetWindow(Wnd, gw_Owner) = 0) AND {-Дочерние окна}
            (GetWindowText(Wnd, buff, sizeof(buff)) <> 0) {-Окна без заголовков}
        THEN
          BEGIN
            GetWindowText(Wnd, buff, sizeof(buff));
            ListBox1.Items.Add(StrPas(buff));
          END;
          Wnd := GetWindow(Wnd, gw_hWndNext);
        END;
  ListBox1.ItemIndex := 0;
  BitBtn1.Caption := 'Обновить список окон';
end;

Function Find(s: string): hWnd;
var
  Wnd: hWnd;
  buff: array[0..127] of Char;
begin
  Find := 0;
  Wnd := GetWindow(Application.Handle, gw_HWndFirst);
  while Wnd <> 0 do
    begin
      if (Wnd <> Application.Handle) and
        IsWindowVisible(Wnd) and
          (GetWindow(Wnd, gw_Owner) = 0) and
            (GetWindowText(Wnd, buff, sizeof(buff)) <> 0) then
        begin
          GetWindowText(Wnd, buff, sizeof(buff));
          if pos(s, StrPas(buff)) > 0 then
            begin
              Find := Wnd;
              Break;
            end;
          end;
        end;
        Wnd := GetWindow(Wnd, gw_hWndNext);
    end;
end;

```

```

procedure TForm1.BitBtn2Click(Sender: TObject);
var
    Wnd: hWnd;
begin
    Wnd := Find(ListBox1.Items[ListBox1.ItemIndex]);
    if (IsZoomed(Wnd) = true) then
        begin
            ShowWindow(Wnd, SW_MINIMIZE);
            BitBtn2.Caption := 'Развернуть';
        end
    else
        begin
            ShowWindow(Wnd, SW_MAXIMIZE);
            BitBtn2.Caption := 'Свернуть';
        end;
    end;

procedure TForm1.BitBtn3Click(Sender: TObject);
var
    Wnd: hWnd;
begin
    Wnd := Find(ListBox1.Items[ListBox1.ItemIndex]);
    ShowWindow(Wnd, SW_SHOWMAXIMIZED);
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
var
    Wnd: hWnd;
begin
    Wnd := Find(ListBox1.Items[ListBox1.ItemIndex]);
    PostThreadMessage(Wnd, WM_CLOSE, 0, 0);
    PostMessage(Wnd, WM_CLOSE, 0, 0);
end;

end.

```

Вопросы для контроля по лабораторной работе № 8

1. Что является ядром программы?
2. Какой класс является Базовым классом, инкапсулирующим окно Windows?
3. Что является центральным свойством компонента TWinControl?

Лабораторная работа № 9

Разработка нестандартных визуальных приложений

Цель работы: изучить возможности WinAPI по переопределению стандартных свойств визуальных приложений.

Теоретические сведения

Перед созданием окна TWinControl вызывает виртуальный метод CreateParams, позволяя программисту задать низкоуровневые параметры создаваемого окна. В процедуру передается структура данных

```
TCreateParams = record
  Caption: PChar;           // Заголовок окна, соответствует параметру
                           // lpWindowName
  Style: Longint;          // Стиль окна, соответствует параметру
dwStyle
  ExStyle: Longint;        // Расширенный стиль окна (dwExStyle)
  X, Y: Integer;
  Width, Height: Integer; // Координаты окна
  WndParent: HWND;         // Идентификатор окна-владельца (hWndParent)
  Param: Pointer           // Дополнительный параметр (lpParam)
  WindowClass: TWndClass; // Структура TWndClass, позволяющая
                          // задать
                           // параметры класса окна
  WinClassName: array[0..63] of Char; // Имя класса окна
                                       // (lpClassName)
end;
```

Наследники TWinControl могут перекрыть CreateParams, создавая окна с требуемыми внешним видом и поведением.

Например, требуется создать форму, не имеющую заголовка, однако позволяющую изменять свои размеры. Delphi не предоставляет возможности задать такое поведение визуальными средствами, однако, перекрыв TForm.CreateParams мы легко добиваемся нужного эффекта:

```
procedure TForm1.CreateParams(var Params: TCreateParams);
begin
  inherited; // Вызываем унаследованный обработчик, позволяя
            // VCL подготовить «типовую» конфигурацию окна
```

```
with Params do  
    // И изменяем требуемые параметры  
    Style := Style and (not WS_CAPTION) or WS_THICKFRAME or  
    WS_POPUP;  
end;
```

Если посмотреть на некоторые фрагменты реализации класса TWinControl, расположенного в модуле Controls.pas, то можно увидеть, что метод CreateWnd сначала вызывает метод CreateParams, заполняющий структуру WndClass с параметрами класса окна и параметры для CreateWindow, а затем регистрирует класс и создает окно. Также очень показателен метод TCustomForm.CreateParams, расположенный в модуле Forms.pas. Хорошо видно, как по свойствам Position, BorderIcons и FormStyle формируется набор флагов стиля окна для функции CreateWindow.

Практическое задание

Разработать приложение, имеющее окно нестандартной формы.

Ход выполнения задания

1. Создать в графическом редакторе (например, MS Paint) файл bitmap.bmp. Нарисовать в нем произвольную фигуру. При рисовании обязательно использовать белый и черный цвета.
2. Создать визуальное приложение в Delphi.
3. Разместить компоненту Timage на форме. Загрузить созданный графический файл bitmap.bmp в Image.
4. Свойство формы BorderStyle установить в bsNone, свойство Color установить в clWhite.
5. Изменить размер формы под размер компоненты Timage.
6. Прописать следующий код.

```

function BitmapToRegion(Bitmap: TBitmap; TransColor: TColor): HRGN;
var X, Y: Integer;
    XStart: Integer;
begin
    Result := 0;
    with Bitmap do
    for Y := 0 to Height - 1 do
    begin
        X := 0;
        while X < Width do
        begin
            // Пропускаем прозрачные точки
            while (X < Width) and (Canvas.Pixels[X, Y] = TransColor) do
                Inc(X);
            if X >= Width then
                Break;
            XStart := X;
            // Пропускаем непрозрачные точки
            while (X < Width) and (Canvas.Pixels[X, Y] <> TransColor) do
                Inc(X);
            // Создаем новый прямоугольный регион и добавляем его к
            // региону всей картинки
            if Result = 0 then
                Result := CreateRectRgn(XStart, Y, X, Y + 1)
            else
                CombineRgn(Result, Result,
                    CreateRectRgn(XStart, Y, X, Y + 1), RGN_OR);
            end;
        end;
    end;
end;

```

7. Прописать следующий код в обработчик события формы OnCreate

```

procedure TForm1.FormCreate(Sender: TObject);
Var RGN : HRGN;
begin
    // создаем регион для картинки
    RGN:= BitmapToRegion(Image1.Picture.Bitmap, clWhite);
    // устанавливаем новый регион для картинки
    SetWindowRgn(Form1.Handle, RGN, True); // второй параметр всегда равен True
end;

```

Вопросы для контроля по лабораторной работе № 9

1. Какой виртуальный метод вызывается перед созданием окна TWinControl?
2. Могут ли наследники TWinControl перекрыть CreateParams?

Лабораторная работа № 10

Разработка многопоточного приложения

Цель работы: научиться создавать многопоточные приложения.

Теоретические сведения

Концепция многопоточного исполнения впервые появилась на системах с разделением времени (time sharing), где несколько человек могли одновременно работать на центральном компьютере.

Важно отметить, что процессорное время просто делилось между пользователями, а уже сама операционная система использовала концепции "процессов" и "потоков".

Приложение – самодостаточный набор машинных инструкций, обеспечивающий решение конкретной задачи.

Процесс обычно определяют как экземпляр (или копию) выполняемой программы (приложения).

В Win32 процессу отводится 4Гбайта адресного пространства. (В Win64 – 16 экзбайт(2^{64})).

Поток можно определить как такой фрагмент программного кода, который выполняется последовательно.

Именно потоки отвечают за исполнение программного кода, помещенного в адресное пространство процесса. При создании процесса в Win32 первый (первичный – primary) поток создается системой автоматически.

Далее этот поток может порождать другие потоки, которые в свою очередь могут порождать третьи и т.д. Таким образом, процесс может включать произвольное число потоков, которые можно создавать и уничтожать. Процесс завершается, когда завершается выполнение первичного потока.

Несмотря на то, что Win32 API включает все средства, необходимые для работы с потоками, класс TThread удобнее использовать в частности потому, что он предоставляет средства, гарантирующие совместимость с библиотекой визуальных компонентов Delphi.

Для создания потока необходимо выполнить следующие действия:

1. создать новый проект, разместить на форме все необходимые компоненты и добавить требуемый код;

2. выполнить тему меню *File/New/Thread Object*, задать имя класса поток. После выполнения этих действий Delphi создаст новый модуль-шаблон для потока;

3. сохранить этот модуль под требуемым именем в папке проекта;

4. новый модуль добавить в предложение *uses* там, где это необходимо;

5. добавить в модуль (потока) требуемый код.

В сгенерированном Delphi модуле-заготовке присутствует описание класса TThread с одним единственным методом Execute.

С этого метода, который нет необходимости вызывать "напрямую", начинается выполнения потока. Но еще раньше поток необходимо создать с помощью конструктора Create.

Реализация (абстрактного) метода Execute может выглядеть примерно так:

```
Procedure TMyThread.Execute;  
Begin  
    repeat  
        {выполнение требуемого кода}  
    until CancelCondition or Terminated;  
End;
```

Изменение значения свойства **Suspended** аналогично вызову методов **Resume** и **Suspend**, т.е. выполнение оператора **Suspended:=true** приводит к приостановке выполнения потока, а **Suspended:=false** – к возобновлению его выполнения с точки останова.

Метод **Synchronize** гарантирует, что к любому объекту VCL одновременно получит доступ только один поток.

Одновременный доступ к объекту VCL двух и более потоков может вызвать непредсказуемые последствия. Метод потока, который содержит обращение к какому-либо объекту VCL, не может иметь параметров и должен быть передан процедуре Synchronize в качестве параметра

Практическое задание 1

Разработать приложение, содержащее три потока: главный и два дополнительных.

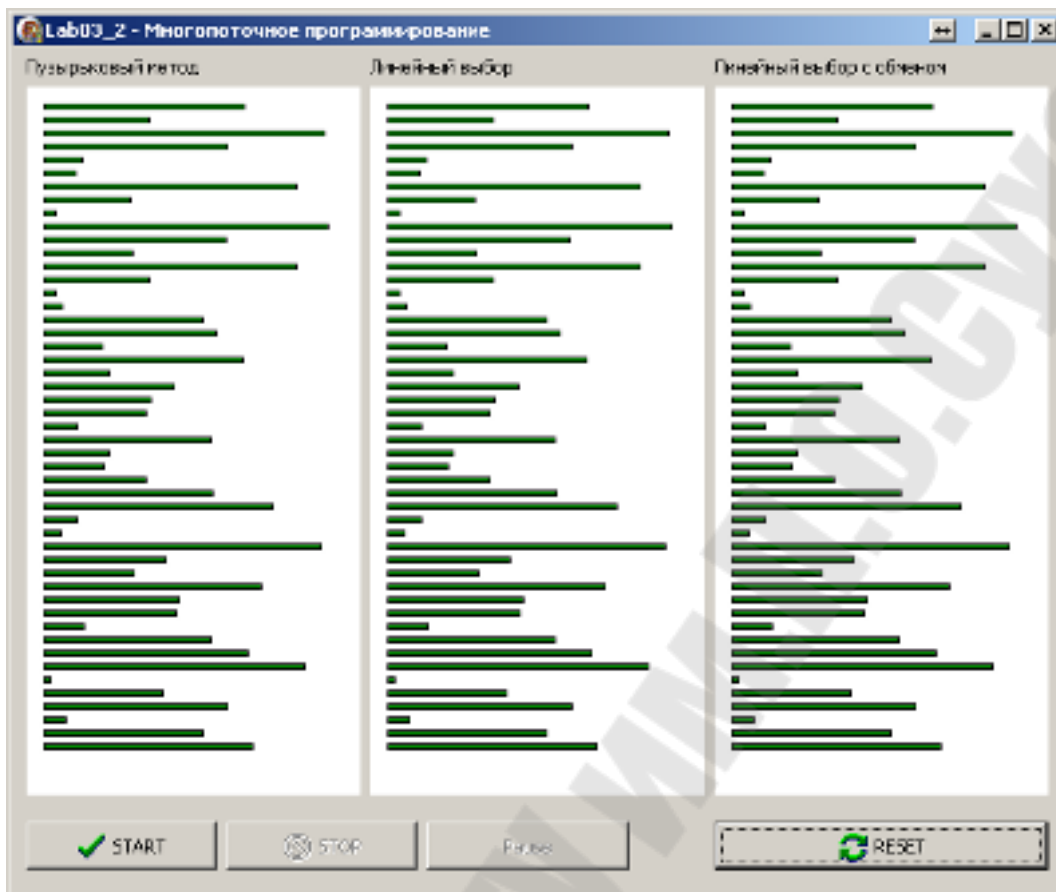
Среди компонентов приложения использовать два регулятора для установки приоритетов потоков и две строки редактирования — для наблюдения за их изменением. Потоки должны выполнять некоторые простые вычисления, а главная программа — отображать, сколько вычислений в секунду выполнено в каждом потоке.

Практическое задание 2

Разработать приложение (используя VCL), содержащее на форме следующие компоненты:

- 1) Label – название метода сортировки (3 экземпляра соответственно);
- 2) Image – визуализация текущей итерации метода сортировки (3 экземпляра для каждого метода соответственно);
- 3) Reset button – кнопка повторно вызывающая генерацию значений массива;
- 4) Start button – кнопка запуска трех различных потоков;
- 5) Stop button – кнопка принудительной остановки всех исполняющихся потоков;
- 6) Pause button – кнопка временной остановки всех исполняющихся потоков.

Разработать три класса потоков (TThread), каждый из которых реализует свой собственный (отличный от двух других) метод сортировки.



При открытии приложения программа должна сгенерировать массив из 50 случайных чисел в некотором диапазоне. Затем данный массив необходимо визуализировать (представить в виде диаграммы) в каждом из трех изображений на форме.

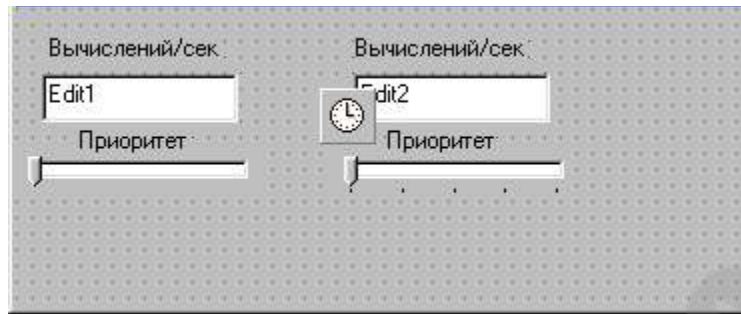
Нажатие на кнопку «Start button» должно приводить к запуску трех экземпляров различных потоков реализованных в пункте 2. При этом каждый поток «в реальном времени» (а именно после каждой выполненной итерации в вызванном методе сортировки) визуально отображает текущие состояние значений переменных массива.

По нажатию на кнопку «Stop button» должна производиться принудительная остановка всех трех исполняющихся потоков.

Кнопка «Reset button» повторно вызывает генерацию значений исходного массива.

Ход выполнения задания 1

1. Расположите на форме два компонента Edit, два регулятора TrackBar и один компонент типа TTimer, как показано ниже. Поместите одну строку редактирования и один регулятор слева, а другую пару — справа.



2. Откройте меню File и выберите пункт Save Project As. Сохраните модуль как Thrdunit, а проект — как Thrdproj.

3. Откройте меню File и выберите пункт New. Затем выполните двойной щелчок на объекте типа поток (значок Thread Object). Откроется диалоговое окно New Items



4. Когда появится диалоговое окно для именованния объекта поток, введите TSiriipieThread и нажмите клавишу <Enter>.



5. Delphi создаст шаблон для нового потока, который показан

В ЛИСТИНГЕ:

```
unit Unit-1;
interface
uses
  Classes ;
type
  TSimpleThread = class(TThread)
  private { Private declarations }
  protected procedure Execute; override;
```



```
end;  
implementation
```

{Важно: методы и свойства объектов из состава VCL могут быть использованы посредством метода под названием Synchronize, например, Synchronize(UpdateCaption);

где UpdateCaption может выглядеть так:

```
procedure ThrdProj Thread.UpdateCaption;  
begin
```

```
    Form1.Caption := 'Updated in a thread';
```

```
end;
```

```
}
```

```
( TSimpleThread ) procedure TSimpleThread.Execute;
```

```
begin
```

```
{ Код потока помещается здесь }
```

```
end;
```

```
end.
```

6. Измените объявление класса TSimpleThread, чтобы включить в секцию public поле count. Поле count будет использовано, чтобы подсчитать, сколько вычислений в секунду выполняется в потоке:

```
TSimpleThread = class(TThread) private  
    { Private declarations }  
    protected procedure Execute; override;  
    public  
        Count : Integer;  
end;
```

7. Изменения, вносимые в модуль Execute, заключаются в том, чтобы подсчитать среднее значение десяти случайных чисел и затем увеличить на единицу значение count. Эти изменения показаны ниже:

```
procedure TSimpleThread.Execute;  
Var  
    I, Total, Avg : Integer;  
Begin  
    While True Do  
        Begin  
            Total := 0;  
            For I := 1 To 10 Do Inc ( Total, Random( Maxint ) );  
            Avg := Avg Div 10;  
            Inc( Count ) ;  
        End;  
    end;
```

8. Откройте меню File и выберите пункт Save As. Сохраните модуль с потоком как Thrd.pas.

9. Отредактируйте главный файл модуля ThrdUnit.pas, и добавьте модуль Thrd к списку используемых модулей. Он должен выглядеть так:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
  Forms, Dialogs, Thrd, ExtCtrls, StdCtrls, ComCtrls;
```

10. В секции public формы TForm1 добавьте следующую строку:

```
Thread1, Thread2: TSimpleThread;
```

11. Выполните двойной щелчок на свободном месте рабочей области формы, чтобы объявить два потока, которые будут использоваться программой; при этом создастся шаблон метода Formcreate. В этом методе произойдет создание потоков, присвоение им приоритетов и запуск. Поместите в шаблон Formcreate следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Thread1 := TSimpleThread.Create( False );
  Thread1.Priority := tpLowest;
  Thread2 := TSimpleThread.Create( False );
  Thread2.Priority := tpLowest;
end;
```

12. Выполните двойной щелчок на компоненте TTimer для создания пустого шаблона метода Timer. Этот метод будет автоматически вызываться каждую секунду, чтобы приложение могло отслеживать состояние потоков. Метод Timer должен выглядеть следующим образом:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Edit1.Text := IntToStr( Thread1.Count );
  Edit2.Text := IntToStr( Thread2.Count );
  Thread1.Count := 0;
  Thread2.Count := 0;
end;
```

13. Щелкните на левом регуляторе (TrackBar1) и выберите страницу Events в окне Object Inspector. Выполните двойной щелчок напротив имени метода onchange для создания шаблона метода,

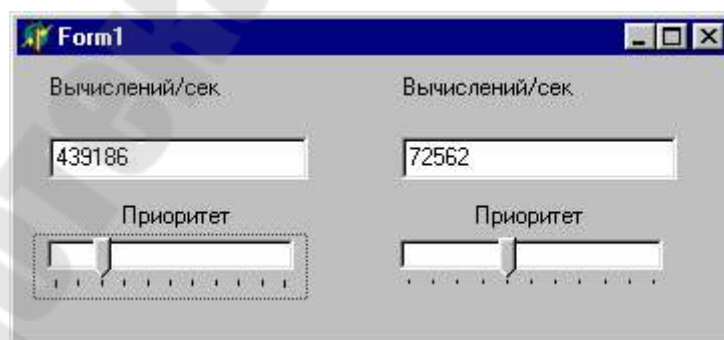
который будет вызываться каждый раз при изменении положения регулятора. Метод будет устанавливать регулятор в соответствии с приоритетом потока. Он должен содержать следующий код:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
Var
  I : Integer;
  Priority : TThreadPriority;
begin
  Priority := tpLowest;
  For I := 0 To ( Sender as tTrackBar ).Position- 1
    Do inc( Priority
           );
  If Sender = TrackBar1 Then Thread1.Priority :=
Priority
  Else Thread2.Priority := Priority;
end;
```

14. Чтобы связать метод, созданный на шаге 14, со вторым регулятором, выберите TrackBar2 в окне Object Inspector, откройте комбинированный список события OnChange И выберите метод TrackBar1Change.

15. Чтобы учесть прозвучавшее выше предупреждение о недопустимости приоритета, высшего чем tpHigher, максимальное положение регуляторов должно быть ограничено четырьмя. Выберите TrackBar1, затем, удерживая клавишу <Shift>, TrackBar2. Когда оба компонента будут выбраны, выберите в окне Object Inspector страницу properties и придайте свойству max значение 4.

Таким образом, многопоточное приложение готово к запуску.



Ход выполнения задания 2

1. Создать и описывать новый модуль (в примере Unit2.pas). Записывать исходный код объявления класса потока в модуле:

```
unit Unit2;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Buttons, ExtCtrls;
```

```
type
```

```
SortThread1 = class(TThread)
```

```
private
```

```
{ Private declarations }
```

```
protected
```

```
procedure Execute; override;
```

```
procedure SortArray;
```

```
procedure VisualizeArray;
```

```
public
```

```
a: array[1..50] of byte;
```

```
end;
```

```
implementation
```

```
uses Unit1;
```

2. Описать процедуру вызываемую при старте потока:

```
procedure SortThread1.Execute;
```

```
begin
```

```
SortArray;
```

```
end;
```

3. Реализовать процедуру выполняющую сортировку массива
пузырьковым методом.

```
procedure SortThread1.SortArray;
```

```
var
```

```
i, j, r: byte;
```

```
begin
```

```
For i := 49 downto 1 do
```

```
begin
```

```
For j:=1 to i do
```

```
begin
```

```
If (a[j] > a[j + 1]) then
```

```
begin
```

```
r := a[j];
```

```
a[j] := a[j + 1];
```

```
a[j + 1] := r;
```

```
end;
```

```
end;
```

```
Synchronize(VisualizeArray);
```

```
sleep(40);
```

```
end;
```

```
end;
```

4. Записать процедуру вызова визуализации массива.

```

procedure SortThread1.VisualizeArray;
begin
    Form1.ShowArray(a, Form1.Image1);
end;

```

5. Создать аналогичные классы SortedThread2 и SortedThread3, но реализующие иные методы сортировки.

6. Написать исходный код основного приложения:

```

unit Unit1;

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, '
    Dialogs, StdCtrls, Buttons, ExtCtrls, Unit2, Unit3, Unit4;

type
    TForm1 = class(TForm)
        Image1: TImage;
        ResetButton: TBitBtn;
        Image2: TImage;
        Image3: TImage;
        StopButton: TBitBtn;
        StartButton: TBitBtn;
        PauseButton: TSpeedButton;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        procedure ResetButtonClick(Sender: TObject);
        procedure StartButtonClick(Sender: TObject);
        procedure StopButtonClick(Sender: TObject);
        procedure PauseButtonClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Thread1: SortThread1;
    Thread2: SortThread2;
    Thread3: SortThread3;
    a: array of byte;
    Image: TImage;

implementation
    {$R *.res}

```

```
procedure TForm1.ResetButtonClick(Sender: TObject);  
var i: byte;  
begin  
    //Создаем массив  
    Randomize;  
    for i := 1 to 50 do  
    begin  
        a[i] := Random(180) + 1;  
    end;  
  
    //Отображаем массивы  
    ShowArray(a, Image1);  
    ShowArray(a, Image2);  
    ShowArray(a, Image3);  
  
    StartButton.Enabled := true;  
    PauseButton.Enabled := false;  
    StopButton.Enabled := false;  
end;
```

```
procedure TForm1.PauseButtonClick(Sender: TObject);  
begin  
    if (PauseButton.Down = false) then  
    begin  
        PauseButton.AllowAllUp := True;  
        PauseButton.GroupIndex := 1;  
        Thread1.Suspend;  
        Thread2.Suspend;  
        Thread3.Suspend;  
    end  
    else  
    begin  
        PauseButton.AllowAllUp := False;  
        PauseButton.GroupIndex := 1;  
        Thread1.Resume;  
        Thread2.Resume;  
        Thread3.Resume;  
    end;  
end;
```

```

procedure TForm1.StopButtonClick(Sender: TObject);
begin
    StartButton.Enabled := true;
    PauseButton.Enabled := false;
    StopButton.Enabled := false;

    //Останавливаем потоки
    Thread1.Suspend;
    Thread2.Suspend;
    Thread3.Suspend;
    Thread1.Terminate;
    Thread2.Terminate;
    Thread3.Terminate;
end;

procedure TForm1.StartButtonClick(Sender: TObject);
var i: byte;
begin
    StartButton.Enabled := false;
    PauseButton.Enabled := true;
    StopButton.Enabled := true;

    //Создаем потоки
    Thread1 := SortThread1.Create(True);
    Thread2 := SortThread2.Create(True);
    Thread3 := SortThread3.Create(True);

    Thread1.Priority := tpLowest;
    Thread2.Priority := tpLowest;
    Thread3.Priority := tpLowest;

    for i := 1 to 50 do
    begin
        Thread1.a[i] := a[i];
        Thread2.a[i] := a[i];
        Thread3.a[i] := a[i];
    end;

    //Запускаем потоки
    Thread1.Resume;
    Thread2.Resume;
    Thread3.Resume;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    StartButton.Enabled := false;
    StopButton.Enabled := false;
    PauseButton.Enabled := false;
end;

```

```

procedure TForm1.ShowArray(a: array of byte; image: TImage);
const
    indentX = 10;
    indentY = 10;
    height = 3;
    interval = 5;
var
    i: byte;
    currentHeight: Integer;
begin
    image.Canvas.Brush.Color := clWhite;
    image.Canvas.FillRect(image.Canvas.ClipRect);

    image.Canvas.Brush.Color := clGreen;
    currentHeight := indentY;
    for i := 1 to 50 do
        begin
            image.Canvas.Rectangle(indentX, currentHeight,
                indentX + a[i], height + currentHeight);
            inc(currentHeight, interval + height);
        end;
    image.Repaint;
end;

```

Вопросы для контроля по лабораторной работе № 10

1. Понятие приложения.
2. Понятие процесса.
3. Понятие потока.
4. Какие основные действия необходимо выполнить для создания потока?
5. Для чего нужны методы **Resume** и **Suspend**?
6. Для чего нужен метод **Synchronize**?

Лабораторная работа № 11

Создание службы с использованием WinAPI

Цель работы: научиться разрабатывать службы с использованием WinAPI.

Теоретические сведения

Чтобы создать службу на Delphi, достаточно нажать F1, набрать TService и прочитать раздел using tservice. Но можно написать службу и на более низком уровне WinAPI.

Обычный Win32-сервис это обычная программа.

Программу рекомендуется сделать консольной (DELPHI MENU | Project | Options.. | Linker [X]Generate Console Application) и крайне рекомендуется сделать ее **без форм** и удалить модуль Forms из Uses.

Рекомендуется потому, что, во-первых, это окошко показывать не стоит потому, что оно позволит любому юзеру, прибавив ваше окошко прибить и сервис и, во-вторых, конечно же, размер файла.

В главном модуле проекта добавляется uses Windows и WinSvc.

Как уже отмечалось - сервис это обычная программа. Программа в Delphi находится между begin и end.

После запуска нашего сервиса (здесь и далее под запуском сервиса понимается именно запуск его из Менеджера сервисов, а не просто запуск exe-файла сервиса) менеджер сервисов ждет пока наш сервис вызовет функцию StartServiceCtrlDispatcher. Ждать он будет недолго - если в нашем exe'шнике несколько сервисов то секунд 30, если один - около секунды, поэтому помещаем вызов StartServiceCtrlDispatcher поближе к begin.

StartServiceCtrlDispatcher качестве аргумента требует `_SERVICE_TABLE_ENTRY`, поэтому добавляем в `var DispatchTable : array [0..кол-во сервисов] of _SERVICE_TABLE_ENTRY`;

и заполняем этот массив (естественно перед вызовом StartServiceCtrlDispatcher).

Т.к. в нашем exe-файле будет 1 сервис, то заполняем его так :

```
DispatchTable[0].lpServiceName:=ServiceName;  
DispatchTable[0].lpServiceProc:=@ServiceProc;  
DispatchTable[1].lpServiceName:=nil;  
DispatchTable[1].lpServiceProc:=nil;
```

Следует завести константы ServiceName - имя сервиса и ServiceDisplayName - отображаемое имя.

ServiceProc - основная функция сервиса, а в функцию мы передаем ее адрес.

В DispatchTable[кол-во сервисов] все равно nil - это показывает функции, что предыдущее поле было последним.

begin

```
DispatchTable[0].lpServiceName:=ServiceName;  
DispatchTable[0].lpServiceProc:=@ServiceProc;
```

```
DispatchTable[1].lpServiceName:=nil;  
DispatchTable[1].lpServiceProc:=nil;
```

```
if not StartServiceCtrlDispatcher(DispatchTable[0])  
then LogError('StartServiceCtrlDispatcher Error');
```

end.

StartServiceCtrlDispatcher выполнится только после того, как все сервисы будут остановлены.

Функция LogError протоколирует ошибки - напишите ее сами.

ServiceMain - основная функция сервиса. Если в ехешнике несколько сервисов, но для каждого сервиса пишется своя ServiceMain функция.

Имя функции может быть любым и передается в DispatchTable.lpServiceProc:=@ServiceMain (см.предыдущий абзац). У меня она называется ServiceProc и описывается так:

```
procedure ServiceProc(argc : DWORD;var argv : array of  
PChar);stdcall;
```

argc кол-во аргументов и их массив argv передаются менеджером сервисов из настроек сервиса.

ServiceMain требуется выполнить подготовку к запуску сервиса и зарегистрировать обработчик сообщений от менеджера сервисов (Handler).

Опять после запуска ServiceMain и до запуска RegisterServiceCtrlHandler должно пройти минимум времени. Если сервису надо делать что-нибудь очень долго и обязательно до вызова RegisterServiceCtrlHandler, то надо посылать сообщение SERVICE_START_PENDING функцией SetServiceStatus.

Итак, в RegisterServiceCtrlHandler передаем название нашего сервиса и адрес функции Handler'a.

Далее выполняем подготовку к запуску и настройку сервиса. Остановимся на настройке поподробнее.

```
var ServiceStatus : SERVICE_STATUS;
```

(ServiceStatusHandle : SERVICE_STATUS_HANDLE и ServiceStatus надо сделать глобальными переменными и поместить их выше всех функций).

dwServiceType - тип сервиса	
SERVICE_WIN32_OWN_PROCESS	Одиночный сервис
SERVICE_WIN32_SHARE_PROCESS	Несколько сервисов в одном процессе
SERVICE_INTERACTIVE_PROCESS	интерактивный сервис (может взаимодействовать с пользователем).

dwWin32ExitCode и dwServiceSpecificExitCode - коды ошибок сервиса. Если все идет нормально, то они должны быть равны нулю, иначе коду ошибки.

dwCheckPoint - если сервис выполняет какое-нибудь долгое действие при остановке, запуске и т.д. то dwCheckPoint является индикатором прогресса (увеличивайте его, чтобы дать понять, что сервис не завис), иначе он должен быть равен нулю.

dwWaitHint - время, через которое сервис должен послать свой новый статус менеджеру сервисов при выполнении действия (запуска, остановки и т.д.). Если dwCurrentState и dwCheckPoint через это кол-во миллисекунд не изменится, то менеджер сервисов решит, что произошла ошибка.

dwCurrentState - Ставим его в SERVICE_RUNNING, если сервис запущен.

После заполнения этой структуры посылаем наш новый статус функцией SetServiceStatus и мы работаем.

Функция Handler будет вызываться менеджером сервисов при передаче сообщений сервису. Опять же название функции - любое.

Адрес функции передается с помощью функции RegisterServiceCtrlHandler (см. выше). Функция имеет один параметр

типа DWORD (Cardinal) - сообщение сервису. Если в одном процессе несколько сервисов - для каждого из них должна быть своя функция.

```
procedure ServiceCtrlHandler(Opcode : Cardinal);stdcall;
```

В функции ServiceMain пишем код сервиса. Так как сервис обычно постоянно находится в памяти компьютера, то скорее всего код будет находиться в цикле. Например, в таком:

```
repeat
```

```
  Что-нибудь делаем пока сервис не завершится.
```

```
until ServiceStatus.dwCurrentState = SERVICE_STOPPED;
```

Но это сработает, если сервис не обрабатывает сообщения приостановки/перезапуска, иначе сервис никак не прореагирует.

Другой вариант:

```
repeat
```

```
  if ServiceStatus.dwCurrentState <> SERVICE_PAUSED
```

```
  then чего-то делаем
```

```
until ServiceStatus.dwCurrentState = SERVICE_STOPPED;
```

И третий, самый правильный вариант = использование потока:

```
function MainServiceThread(p:Pointer):DWORD;stdcall;
```

```
begin
```

```
  что-то делаем
```

```
end;
```

и в ServiceMain создаем поток

```
var
```

```
  ThID : Cardinal;
```

```
  hThread:=CreateThread(nil,0,@MainServiceThread,nil,0,ThID);
```

и ждем его завершения

```
WaitForSingleObject(hThread,INFINITE);
```

закрывая после этого его дескриптор

```
CloseHandle(hThread);
```

При этом hThread делаем глобальной переменной.

Теперь при приостановке сервиса (в Handler) делаем так

```
SERVICE_CONTROL_PAUSE :
```

```
begin
```

```
  ServiceStatus.dwCurrentState := SERVICE_PAUSED;
```

```
  SuspendThread(hThread); // приостанавливаем поток
```

```
end;
```

и при возобновлении работы сервиса

```
SERVICE_CONTROL_CONTINUE :  
begin  
ServiceStatus.dwCurrentState := SERVICE_RUNNING;  
ResumeThread(hThread); // возобновляем поток  
end;
```

Практическое задание

Разработать службу, которая выводит на экран сообщение с текстом «Проверка службы», воспроизводит звук через встроенный динамик ПК и включает/выключает три светодиода на клавиатуре (Num Lock, Caps Lock, Scroll Lock).

Ход выполнения работы

1. Написать исходный код службы.

```

program Project1;

uses
  Windows, WinSvc, SysUtils;

const
  c_ServiceName = 'Service_001';

type
  KeyType = (k_CopLock, k_BusLock, k_ScrollLock);

var
  DispatchTable: array [0..1] of _SERVICE_TABLE_ENTRY;
  sst: SERVICE_STATUS;
  sstHandler: SERVICE_STATUS_HANDLER;

procedure SetServiceStatus;
begin
  if not SetServiceStatus(sstHandler, sst) then
    RaiseLastOSError;
end;

procedure ServiceCtrlHandler(Opcode : Cardinal);stdcall;
begin
  case Opcode of
    SERVICE_CONTROL_STOP:
      begin
        sst.dwWin32ExitCode := 0;
        sst.dwCurrentState := SERVICE_STOPPED;
        sst.dwCheckPoint := 0;
        sst.dwWaitHint := 0;
        SetServiceStatus;
        exit;
      end;
    SERVICE_CONTROL_INTERROGATE : ;
  end;

  SetServiceStatus;
end;

```



```

var
  schService, schSCManager: SC_HANDLE;
  binExec pchar;
begin
  if (ParamStr[1] = '' or ParamStr[1] = '?') then
  begin
    binExec := pchar(ParamStr[1]);
    schSCManager := OpenSCManager(
      nil,                               local machine
      nil,                               LocalMachine Database
      SC_MANAGER_ALL_ACCESS);           full access rights

    if (schSCManager = 0) then RaiseLastOSError;

    schService := CreateService(
      schSCManager,                       SCManager Database
      c_ServiceName,                       name of service
      c_ServiceName,                       service name to display
      SERVICE_ALL_ACCESS,                 desired access
      SERVICE_WOW64_64BIT_PROCESS,       service type
      SERVICE_DEMAND_START,              start type
      SERVICE_ERROR_NORMAL,              error control type
      binExec,                             service's binary
      nil,                                  no local recovery path
      nil,                                  no password
      nil,                                  no dependencies
      nil,                                  no dependencies
      nil,                                  no dependencies
      nil);

    if schService = 0 then RaiseLastOSError;
    if not CloseServiceHandle(schService) then RaiseLastOSError;
  end;
end

```

```

uses k (Winutils) (.....) Mass;
begin
  schSCManager := OpenSCManager(
    nil,                               local machine
    nil,                               LocalMachine Database
    SC_MANAGER_ALL_ACCESS);           full access rights

  if schSCManager = 0 then RaiseLastOSError;

  schService := CreateService(
    schSCManager,                       SCManager Database
    c_ServiceName,                       name of service
    c_ServiceName,                       name of service
    SERVICE_ALL_ACCESS);               full access rights

  if schService = 0 then RaiseLastOSError;
  if not CloseServiceHandle(schService) then RaiseLastOSError;
end;

begin
  schService := CreateService(
    schSCManager,                       SCManager Database
    c_ServiceName,                       name of service
    c_ServiceName,                       name of service
    SERVICE_ALL_ACCESS);               full access rights

  if schService = 0 then RaiseLastOSError;
  if not CloseServiceHandle(schService) then RaiseLastOSError;
end;

```


2. Далее напишем скрипты удаления и добавления службы в систему. Для этого необходимо создать текстовый файл с именем «install.bat» и поместим в него следующий текст:

```
Project1.exe /Install
```

Аналогично создадим файл с именем «delete.bat» с содержимым:

```
Project1.exe /Delete
```

3. Затем открываем программу для управления службами, следующим способом:

Пуск => Панель управления => Администрирование => Службы

4. По названию службы находим ее в списке и щелкаем по ней правой кнопкой мыши. В открывшемся окне на закладке «Вход в систему» установить флаг на пункте // "Разрешить взаимодействие с рабочим столом" (необходимо чтобы открывались окна сообщений).

5. Для запуск службы щелкаем по ней правой кнопкой мыши и нажимаем кнопку «Пуск» (для остановки соответственно нажимаем «Стоп»).

Вопросы для контроля по лабораторной работе № 11

1. Как запускается сервис в ОС Windows?
2. С помощью какой функции запускается сервис?
3. Как работать с DispatchTable?
4. Назначение функции RegisterServiceCtrlHandler .
5. Назначение функции SetServiceStatus.

Лабораторная работа № 12

Создание драйверов режима ядра

Цель работы: научиться создавать внешние и внутренние ссылки, якоря. Освоить принципы создания картинок-ссылок. Научиться создавать и использовать карты изображений.

Теоретические сведения

Программирование в системах Windows линейки NT можно условно разделить на две принципиально различных части:

- создание кода пользовательского режима
- кода режима ядра.

Такое разделение вызвано особенностями внутреннего строения Windows. Поскольку основным семейством процессоров для всего семейства Windows являются процессоры Intel семейства x86. Известно, что эти процессоры этого семейства имеют четыре уровня защиты (от нулевого до третьего), называемые кольцами. Кольца различаются множеством разрешённых к выполнению операций, например в 3-м кольце существуют ограничения на операции с портами ввода-вывода и на доступ к памяти по физическим адресам.

В архитектуре ОС Windows используются всего два кольца: 0-е и 3-е. В нулевом кольце выполняется код уровня абстрагирования от аппаратуры (HAL), ядро системы и различные драйверы, в том числе и драйверы устройств. В 3-м кольце выполняются системные службы, программы, взаимодействующие с пользователем, а также вспомогательный код для вызова функций ядра из пользовательского режима.

Для разработки драйверов корпорация Microsoft предоставляет Driver Development Kit (DDK), представляющий собой набор заголовочных файлов, утилит и документации. Из соображений соблюдения внутрикорпоративного стандарта вся документация, примеры кода и инструменты сборки в DDK ориентированы на языки C/C++.

Так сложилось, что ОС семейства Windows пишутся на языках C/C++. Поэтому неудивительно, что DDK ориентирован на C/C++-компиляторы.

Для этих языков процесс преобразования исходного кода программы в машинный код традиционно происходит в два этапа — компиляции и сборки.

В процессе компиляции исходный код программы превращается в так называемые объектные модули, которые обычно содержат машинный код и информацию об экспорте переменных и функций. Слово "обычно" употреблено здесь по той причине, что некоторые компиляторы предоставляют возможность поместить в объектные модули не машинный код, а так называемый промежуточный код (отдаленно напоминающий MSIL или байт-код Java), что позволяет впоследствии оптимизировать код на уровне целого приложения, а не отдельных объектных модулей.

Вторым этапом является сборка. Сборщик после компиляции формирует из одного или нескольких объектных модулей и статически подключаемых библиотек так называемый исполняемый образ (executable image). Исполняемый образ строится в соответствии с требованиями целевой ОС и содержит непосредственно выполняемый процессором машинный код, а также различную вспомогательную информацию.

Создание файла NT-драйвера режима ядра также подчиняется этой схеме. Сборщик генерирует выполняемый образ, указывая в его заголовке, что это именно NT-драйвер режима ядра, в соответствии с указанными при сборке опциями.

Исполняемый образ для native-подсистемы Windows, проще говоря — NT-драйвер режима ядра, исходный код которого написан на языке Object Pascal (сейчас и сам язык называется Delphi), создать можно. Но перед тем, как заняться этим, необходимо прояснить некоторые принципиальные моменты.

Во-первых, сборщик от Microsoft на сегодня единственный, который способен сгенерировать такой исполняемый образ, поэтому без него не обойтись в любом случае.

Во-вторых, встроенный в Delphi сборщик по умолчанию внедряет в любой исполняемый файл код своего Run-Time Library (RTL), а затем компилятор вызывает некоторые функции из этого RTL для реализации некоторых возможностей языка.

Поскольку Delphi RTL рассчитан на выполнение в режиме пользователя, то, очевидно, и возможностями языка Delphi, которые ориентированы на функциональность RTL, придется пожертвовать. К таким возможностям, относятся, например, поддержка динамических

массивов, в том числе строковых типов и операций со строками, поддержка классов, Run-Time Type Information (RTTI) и т.д.

В-третьих, единственными Win32-версиями Delphi, которые создавали объектные файлы, полностью соответствующие стандарту OMF, являются Delphi 2 и Delphi 3. Так что тем, кто привык пользоваться возможностями языка позднейших версий, такими как перегрузка функций и процедур директивой `overload`, придётся отказаться от них.

В-четвёртых, исходный текст драйвера на Delphi также будет иметь свою специфику. Например, сборщику необходимо будет указать так называемую точку входа — функцию в теле драйвера, вызываемую системой при инициализации драйвера. Объектные файлы, которые генерирует Delphi для проектов типа `program` или `library`, не содержат символьного имени точки входа, поэтому придётся воспользоваться проектом типа `unit`.

Практическое задание

Написать драйвер режима ядра для операционной системы Windows.

Ход выполнения работы

1. Набрать следующий код, реализующий минимальный функционал драйвера:

```

unit tiny;

interface

TYPE
  UShort = Word; // unsigned 16-bit
  Short = Smallint; // signed 16-bit
  ULONG = Cardinal;
  Size_T = Cardinal;
  PVoid = Pointer;

  NTStatus = ULONG;
  CShort = Short;

TYPE
  PUNICODE_STRING = ^UNICODE_STRING;
  UNICODE_STRING = packed record
    Length : UShort;
    MaximumLength : UShort;
    Buffer : PWideChar;
  end;

CONST
  NTOSKrn1 = 'ntoskrnl.exe';

CONST
  IRP_MJ_MAXIMUM_FUNCTION = $1B;

TYPE
  PDRIVER_OBJECT = ^DRIVER_OBJECT;
  DRIVER_OBJECT = packed record

```

```

DRIVER_OBJECT = packed record
csType : CShort;
csSize : CShort;
DeviceObject : Pointer; // SHOULD BE PDEVICE_OBJECT
Flags : ULONG;
DriverStart : Pointer;
DriverSize : ULONG;
DriverSection : Pointer;
DriverExtension : Pointer; // SHOULD BE PDRIVER_EXTENSION
DriverName : UNICODE_STRING;
HardwareDatabase : PUNICODE_STRING;
FastIoDispatch : Pointer; // SHOULD BE PFAST_IO_DISPATCH
DriverInit : Pointer; // PDRIVER_INITIALIZE
DriverStartIo : Pointer; // PDRIVER_STARTIO
DriverUnload : Pointer; // PDRIVER_UNLOAD
MajorFunction : array [0..IRP_MJ_MAXIMUM_FUNCTION] of Pointer;
end;

CONST STATUS_SUCCESS = NTSTATUS( $00000000 );

function DriverEntry(
  const DriverObject : PDRIVER_OBJECT;
  const RegistryPath : PUNICODE_STRING
) : NTSTATUS; stdcall;

implementation

function DbgPrint(
  const Format : PAnsiChar
) : NTSTATUS; cdecl; external NTDOSKrn1 name '_DbgPrint';

procedure ADriverUnload(
  const DriverObject : PDRIVER_OBJECT
); stdcall;

begin
  DbgPrint('Tiny: DriverUnload()');
end;

function DriverEntry;
begin
  DriverObject^.DriverUnload := @ADriverUnload;
  DbgPrint('Tiny: DriverEntry()');
  Result := STATUS_SUCCESS;
end;

end.

```

2. Ниже приведён перечень файлов, которые понадобятся для создания драйвера.

- dcc32.exe — компилятор, вызываемый из командной строки;
- rlink32.dll — библиотека, реализующая сборку исполняемого образа (хотя нам именно эта сборка и не нужна, без такой библиотеки компилятор может не запуститься);

– sysinit.dcu и system.dcu — скомпилированные модули Delphi RTL.

– link.exe — сборщик;

– mspdb50.dll — вспомогательная библиотека сборщика;

– ntoskrnl.lib — библиотека, описывающая функции, импортируемые из ntoskrnl.exe — модуля, содержащего большинство API режима ядра.

3. Компиляция кода запускается так:

```
dcc32.exe -jP -$A-,B-,C-,D-,G-,H-,I-,J-,L-,M-,O+,P-,Q-,R-,T-,U-,V-,W+,X+,Y- tiny.pas
```

Принципиальным моментом здесь является наличие ключа `-jP`, вызывающего генерацию объектного файла.

4. Сборка объектного файла в исполняемый образ вызывается такой командой:

```
link.exe /NOLOGO /ALIGN:32 /BASE:0x10000 /SUBSYSTEM:NATIVE /DRIVER /FORCE:UNRESOLVED /ENTRY:DriverEntry$qqsxp13DRIVER_OBJECTxp14UNICODE_STRING tiny.obj /out:tiny.sys ntoskrnl.lib
```

Здесь принципиальными являются опции `/FORCE:UNRESOLVED` и `/ENTRY`.

Компилятор Delphi внёс в объектный файл несколько символов, относящихся к Delphi RTL, таких как `@@HandleFinally$qqrv`.

Чтобы не писать пустые процедуры-заглушки с требуемыми именами, указываем опцию `/FORCE`, и сборщик пропускает сборку таких символов, так как они всё равно не нужны.

Опция `/ENTRY`: указывает на символ, которым компилятор Delphi обозначил точку входа — функцию `DriverEntry`. Поскольку компилятор Delphi в название этого символа вносит также типы передаваемых параметров, то в случае несовпадения объявления функции следует в объектном файле найти правильное имя символа, начинающееся с `DriverEntry`.

В результате компиляции и сборки будет создан файл **tiny.sys**.

Для проверки работоспособности драйвера скопируем его в каталог `%SYSTEMROOT%\system32\drivers` и воспользуемся примитивным инсталлятором драйвера:

```
{ $APPTYPE CONSOLE }
```

```
program drvinst;
```

```
uses Windows, WinSVC;
```

```
var hSCM, hSRV : THandle;
```

```
R : LongBool;
```

```
Param : AnsiString;
```

```
begin
```

```
if ParamCount = 1 then
```

```
begin
```

```
hSCM := OpenSCManager(nil, nil, SC_MANAGER_ALL_ACCESS);
```

```
Writeln('OpenSCManager ', (hSCM <> INVALID_HANDLE_VALUE));
```

```
Param := AnsiString(ParamStr(1));
```

```
// создание системной записи о драйвере
```

```
hSRV := CreateService( hSCM, @Param[1], @Param[1],
```

```
SERVICE_ALL_ACCESS,
```

```
SERVICE_KERNEL_DRIVER,
```

```
SERVICE_DEMAND_START,
```

```
SERVICE_ERROR_NORMAL,
```

```
PAnsiChar('System32\DRIVERS\' + Param + '.sys'),
```

```
nil, nil, nil, nil, nil);
```

```
Writeln('CreateService ', hSRV <> INVALID_HANDLE_VALUE);
```

```
// очистка ресурсов
```

```
R := CloseServiceHandle(hSRV);
```

```
Writeln('CloseServiceHandle ', R);
```

```
R := CloseServiceHandle(hSCM);
```

```
Writeln('CloseServiceHandle ', R);
```

```
end;
```

```
end.
```

Запустив инсталлятор с параметром tiny, мы установим созданный драйвер в систему. Теперь запустим программу DebugView, а после этого запустим драйвер:

```
>net start tiny
```

```
The tiny service was started successfully.
```

Если в драйвере нет ошибок, и операционная система не вызвала BSOD, мы увидим, что драйвер успешно загружен, с помощью утилиты drivers.exe из DDK:

ModuleName	Code	Data	Bss	Paged	Init	LinkDate
ntoskrnl.exe	643072	114688	0	1400832	184320	Fri Aug 08 17:40:11 2003
hal.dll	36864	49152	0	40960	16384	Fri Aug 08 15:58:53 2003
tiny.sys		192	64	0	0	96 Fri Aug 06

12:54:19 2004

...
ntdll.dll 503808 24576 0 0 0 Fri Aug 08 18:37:28 2003

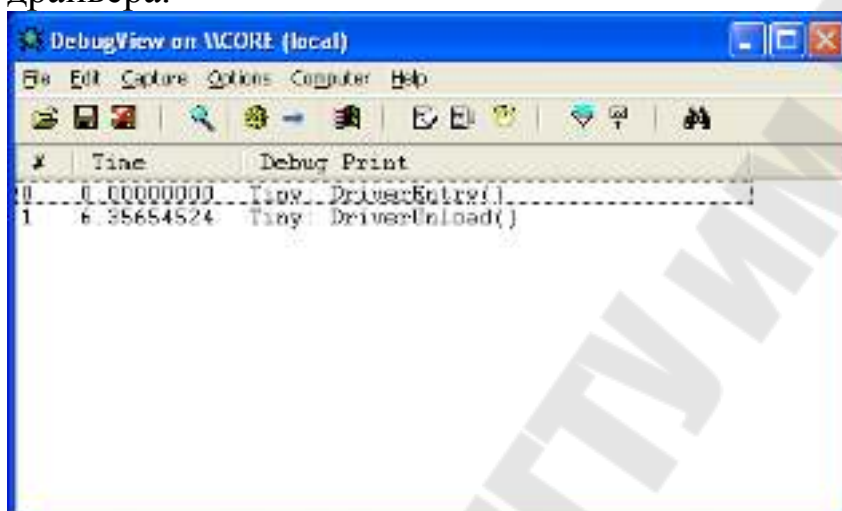
Total 9296112 1675376 0 5221088 850272

После этого остановим драйвер:

```
>net stop tiny
```

The tiny service was stopped successfully.

В окне программы DebugView увидим результаты деятельности драйвера.



С помощью drivers.exe можно проверить, выгрузился драйвер из памяти или нет.

Удалить запись о драйвере можно с помощью приведённого ниже примитивного деинсталлятора:

```
{ $APPTYPE CONSOLE }  
program drvremove;  
  
uses Windows, WinSVC;  
  
var hSCM, hSRV : THandle;  
    R : LongBool;  
    Param : AnsiString;  
  
begin  
  if ParamCount=1 then  
    begin  
      hSCM := OpenSCManager(nil, nil, SC_MANAGER_ALL_ACCESS);  
      Writeln('OpenSCManager ', hSCM <> INVALID_HANDLE_VALUE);
```

```
Param := AnsiString(ParamStr(1));  
// удаление системной записи о драйвере  
hSRV := OpenService(hSCM, @Param[1], SERVICE_ALL_ACCESS);  
Writeln('OpenService ', hSRV <> INVALID_HANDLE_VALUE);  
R := DeleteService(hSrv);  
Writeln('DeleteService ', R);  
// очистка ресурсов  
R := CloseServiceHandle(hSRV);  
Writeln('CloseServiceHandle ', R);  
R := CloseServiceHandle(hSCM);  
Writeln('CloseServiceHandle ', R);  
end;  
end.
```

Вопросы для контроля по лабораторной работе № 12

1. Как условно можно разделить программирование в для ОС Windows?
2. Сколько уровней защиты имеют процессоры Intel семейства x86?
3. Перечислите основные этапы создания драйверов для ОС Windows.

Литература

1. А. П. Побегайло "Системное программирование в Windows" .– СПб.: БХВ-Петербург, 2006. – 1056 с.: ил.
2. Д. Кузана и В. Шапорова "Программирование Win32 API в Delphi".– СПб.: БХВ-Петербург, 2005. – 368 с.: ил.
3. Delphi. Программирование на языке высокого уровня: Учебник для вузов / В.В. Фаронов . – СПб.: Питер, 2004. – 640 с.
4. Delphi7: Учебный курс / С.И. Бобровский. - СПб.: Питер, 2004.-735с.
5. Основы программирования в Turbo Delphi / Н.Культин. – СПб.:ВНУ, 2007 – 384с.

Содержание

Лабораторная работа № 1 Процедуры и функции в Delphi	3
Теоретические сведения	3
Практическое задание	4
Ход выполнения работы	4
Вопросы для контроля по лабораторной работе № 1	7
Лабораторная работа № 2 Обзор палитры компонент Delphi	8
Теоретические сведения	8
Практическое задание 1	13
Практическое задание 2	13
Ход выполнения задания 1	13
Ход выполнения задания 2	16
Вопросы для контроля по лабораторной работе № 2	20
Лабораторная работа № 3 Создание динамических библиотек (DLL) в Delphi	21
Теоретические сведения	21
Практическое задание	22
Ход выполнения работы	22
Вопросы для контроля по лабораторной работе № 3	26
Лабораторная работа № 4	
Хранение форм в динамических библиотеках	27
Теоретические сведения	27
Практическое задание	27
Ход выполнения работы	27
Вопросы для контроля по лабораторной работе № 4	30
Лабораторная работа № 5 Использование сторонних динамических библиотек в Delphi	31
Теоретические сведения	31
Практическое задание	31
Ход выполнения работы	31
Вопросы для контроля по лабораторной работе № 5	33
Лабораторная работа № 6 Создание простейшего приложения с помощью WinAPI	34
Теоретические сведения	34
Практическое задание	36
Ход выполнения работы	36
Вопросы для контроля по лабораторной работе № 6	38

Лабораторная работа № 7 Добавление визуальных компонент на форму с помощью WinAPI	39
Теоретические сведения.....	39
Практическое задание.....	39
Ход выполнения работы.....	40
Вопросы для контроля по лабораторной работе № 7	42
Лабораторная работа № 8 _Управление другими приложениями при помощи WinAPI.....	43
Теоретические сведения.....	43
Практическое задание.....	45
Ход выполнения работы.....	45
Вопросы для контроля по лабораторной работе № 8	48
Лабораторная работа № 9 Разработка нестандартных визуальных приложений.....	49
Теоретические сведения.....	49
Практическое задание.....	50
Ход выполнения задания.....	50
Вопросы для контроля по лабораторной работе № 9	51
Лабораторная работа № 10 Разработка многопоточного приложения ...	52
Теоретические сведения.....	52
Практическое задание 1.....	53
Практическое задание 2.....	54
Ход выполнения задания 1.....	55
Ход выполнения задания 2.....	59
Вопросы для контроля по лабораторной работе № 10	64
Лабораторная работа № 11	
Создание службы с использованием WinAPI	65
Теоретические сведения.....	65
Практическое задание.....	69
Ход выполнения работы.....	69
Вопросы для контроля по лабораторной работе № 11	73
Лабораторная работа № 12	
Создание драйверов режима ядра.....	74
Теоретические сведения.....	74
Практическое задание.....	76
Ход выполнения работы.....	76
Вопросы для контроля по лабораторной работе № 12	82
Литература	83

Рябченко Алексей Иванович
Вегера Артем Сергеевич

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум
по одноименной дисциплине
для слушателей специальности 1-40 01 73
«Программное обеспечение информационных систем»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 28.01.14.

Рег. № 53Е.

E-mail: ic@gstu.by

<http://www.gstu.by>