МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»
(РУТ (МИИТ)

Одобрено кафедрой
«ЖЕЛЕЗНОДОРОЖНАЯ АВТОМАТИКА ТЕЛЕМЕХАНИКА И СВЯЗЬ»

Протокол №___ от_____ 201___г.

Автор: _____

ЗАДАНИЕ НА КОНТРОЛЬНУЮ РАБОТУ С МЕТОДИЧЕСКИМИ УКАЗАНИЯМИ

ПО ДИСЦИПЛИНЕ

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Уровень ВО: *Бакалавриат*

Форма обучения: Заочная

Курс: *3*

Специальность/Направление: *27.03.04 Управление в технических системах (УТб)*

Специализация/Профиль/Магистерская программа: *(УТ) Системы и технические средства автоматизации и управления*

Москва

ОБЩИЕ УКАЗАНИЯ

Контрольная работа выполняется в системе программирования Delphi. Её целью является закрепление знаний системного программирования.

В контрольной работе предлагается разработать приложение, запускающее внешний процесс.

Вариант задачи выбирается по последней цифре учебного шифра.

Выполненная контрольная работа предъявляется на компакт-диске, содержащем тексты программ задач, результаты их выполнения, а также условия задач. Кроме того, прилагается титульный лист формата А4, на котором указаны наименование дисциплины, данные студента и его учебный шифр.

ЗАДАНИЕ НА КОНТРОЛЬНУЮ РАБОТУ

Разработать программу, позволяющую осуществить запуск на исполнение и закрытие внешнего дочернего процесса. После создания процесса вывести значения идентификаторов процесса в соответствии с вариантом задания, который представлен в таблице и определяется по последней цифре учебного шифра студента.

ТАБЛИЦА ВАРИАНТОВ

№	Внешнее приложение	Действие, выполняемое при запуске приложения	Идентификаторы потока	Приоритет процесса	Способ закрытия приложения
0	MS Word	Открытие заданного документа Word	дескриптор созданного процесса, глобальный идентификатор процесса	Выполняется во время простоя системы	Сообщение WM_CLOSE
1	MathCad (или любой другой математический пакет)	Открытие заданного документа MathCad	дескриптор первого потока, глобальный идентификатор потока	Высокий	Сообщение WM_CLOSE
2	Просмотрщик презентаций MS Power Point	Запуск на просмотр заданной презентации	дескриптор первого потока, глобальный идентификатор процесса	Нормальный	Функция TerminateProcess
3	Delphi (или другая используемая среда программирования	Открытие заданного проекта Delphi	дескриптор созданного процесса, глобальный	Нормальный	Сообщение WM_CLOSE

			идентификатор		
5	Windows Media Player (или любой другой media player	Просмотр заданного видео-файла	дескриптор созданного процесса, глобальный идентификатор процесса	Высокий	Функция TerminateProcess
6	MS Excel	Открытие заданного документа Excel	дескриптор первого потока, глобальный идентификатор потока	Выполняется во время простоя системы	Сообщение WM_CLOSE
7	MS Internet Explorer	Выход на заданный ресурс Internet (или локальный Web-ресурс)	дескриптор первого потока, глобальный идентификатор процесса	Нормальный	Функция TerminateProcess
8	Paint (или любой другой графический редактор	Открытие для редактирования графического файла	дескриптор созданного процесса, глобальный идентификатор потока	Высокий	Сообщение WM_CLOSE
9	Kaspersky Anti - Virus Scaner (или любой другой антивирусный сканер)	Проверка дискеты на наличие вируса	дескриптор созданного процесса, дескриптор первого потока	Критическая задача	Функция TerminateProcess

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧИ

Задача выполняется в среде программирования Delphi .
Для создания нового процесса используется функция
CreateProcessA.

function CreateProcessA(lpApplicationName: PAnsiChar; lpCommandLine: PAnsiChar; lpProcessAttributes, lpThreadAttributes: PSecurityAttributes; bInheritHandles: BOOL; dwCreationFlags: DWORD; lpEnvironment: Pointer; lpCurrentDirectory: PAnsiChar; const lpStartupInfo: TStartupInfo; var lpProcessInformation: TProcessInformation): BOOL

Функция порождает новый дочерний процесс и его первый поток (нить). В рамках этого процесса выполняется указанный файл **IpApplicationName** скомандной строкой **IpCommandLine**. Параметр **IpApplicationName** может быть равен **nil**, а имя выполняемого модуля в этом случае должно быть первым элементом командной строки, задаваемой параметром **IpCommandLine**. Сам выполняемый модуль может быть любого вида: 32-разрядным приложением Windows, приложением MS-DOS, OS/2 и т.п.

Однако, если из приложения Windows создается процесс MS-DOS, то параметр **IpApplicationName** должен быть равен **nil**, а имя файла и его командная строка включаются в **IpCommandLine**.

Поэтому, чтобы не ошибиться, проще всегда задавать **IpApplicationName = nil**, и помещать всю информацию в **IpCommandLine**.

Рассмотрим некоторые параметры функции (более подробную информацию можно получить в справке Delphi).

Параметры **IpProcessAttributes**, **IpThreadAttributes**, **IpEnvironment**, **bInhe-ritHandles** определяют наследование дочерним процессом свойств родительскогопроцесса. По умолчанию, можно первые три из этих параметров задавать равными **nil**, а последний — **false**.

Параметр **IpCurrent-Directory** указывает на строку, определяющую текущий каталог и диск дочернего процесса. Это используется в приложениях-оболочках, выполняющих различные приложения с различными рабочими каталогами. Если параметр равен **nil**, текущий каталог совпадает с родительским.

Параметр **dwCreationFlags** определяет флаги, задающие характеристики создаваемого процесса. Эти флаги определяют тип процесса (например, **CREATE_NEW_CONSOLE** — создание нового консольного приложения), характер взаимоотношения с родительским процессом и класс приоритета нового процесса:

HIGH_PRIORITY_CLASS	Указывает на процесс как на критическую задачу, которая должна выполняться немедленно
IDLE_PRIORITY_CLASS	Все потоки процесса выполняются только во время простоя системы. Пример — хранители экрана. Все наследники такого процесса будут иметь тот же класс приоритета
NORMAL_PRIORITY_CLASS	Нормальный приоритет процесса
REALTIME_PRIORITY_CLASS	Высокий приоритет, превышающий приоритеты других процессов, включая приоритеты процессов операционной системы

Параметр **lpStartupInfo** указывает на структуру типа **TStartupInfo**, определяющую основное окно дочернего процесса. Из всех полей этой структуры обязательным для заполнения является только **cb** — размер в байтах данной структуры. Остальные можно не заполнять, что обеспечит вид окна по умолчанию.

Параметр **lpProcessInformation** указывает на структуру **TProcessInformation**, из которой родительское приложение может получать информацию о выполнении нового процесса. Ее поля обозначают следующее:

hProcess	Возвращает дескриптор созданного процесса. Используется во всех функциях, осуществляющих операции с объектом процесса
hThread	Возвращает дескриптор первого потока (нити) созданного процесса. Используется во всех функциях, осуществляющих операции с объектом потока
dwProcessId	Возвращает глобальный идентификатор процесса. Значение доступно с момента создания процесса и до момента его завершения
dwThreadId	Возвращает глобальный идентификатор потока. Значение доступно с момента создания потока и до момента его завершения

Если функция **CreateProcess** успешно выполнена, она возвращает ненулевое значение (**true**). Если произошла ошибка — возвращается 0 (**false**). Тогда информацию об ошибке можно получить, вызвав функцию **GetLastError**.

Закрывает процесс и все его потоки функция **ExitProcess.** Однако эта функция может закрыть процесс, если она вызывается из потока, созданного этим процессом. В более общем случае для немедленного завершения работы внешнего процесса используется функция **TerminateProcess**.

function TerminateProcess(hProcess: THandle; uExitCode: UINT): BOOL;

Параметр **hProcess** - дескриптор завершамого процесса, его значение можно взять из одноименного поля структуры **TProcessInformation** , которая храниться в процессе-родителе.

Параметр **uExitCode** содержит код завершения процесса. Для его определения используется функция **GetExitCodeProcess.**

function GetExitCodeProcess(hProcess: THandle; var lpExitCode: DWORD): BOOL;

Параметр **hProcess** - дескриптор завершамого процесса.

Параметр **lpExitCode** собственно и содержит код завершения процесса.

Типы UNIT и DWORD являются целыми типами Windows, пришедшими из C++, они переопределены в Delphi как тип LongWord, поэтому при написании программы можно в этих функциях использовать переменную знакомого вам типа LongWord.

Замечание. Отметим, что функцией **TerminateProcess** работа приложения завершается немедленно, при этом возможны потери данных. Например, если в окне "Блокнота" вы производили редактирование текста, то после выполнения функции **TerminateProcess**

внесенные изменения будут потерены. Корректно завершить работу приложения, как вы уже знаете, можно послав окну приложения сообщение **WM_CLOSE**.

Ниже приводится листинг программы, содержащей необходимые комментарии.

```
{Модуль Project1.dpr}
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.

{Модуль Unit1.pas}
{Эта программа даёт возможность запустить
программу Блокнот с загрузкой в неё заданного
файла Zapiska.txt, а затем закрыть эту пр-мму
Блокнот. Приведённая программа после своего
запуска даёт экран формы Form1, а на нём - шесть
компонентов, - Edit1, Edit2, Label1, Label2, Button1,
Button2.
Здесь Label1 и Label2 служат для подсказок,
а в поле Edit1 надо ввести путь к файлу
запускаемой программы Блокнот (Notepad.exe),
(например, C:\Windows\), а в поле Edit2 надо ввести
путь к загружаемому в этот редактор текст.файлу
Zapiska.txt (например, C:\My Documents\).
После этого щёлчком на кнопке Button1
("Запустить Блокнот") запускаем Блокнот с
загрузкой в него указанного текст.файла.
Обеспечиваем приоритет процесса - "Критическая задача".
(Если такого файла по указанному пути не
окажется, то программа даст возможность его
создать заново) Для закрытия Блокнота
следует щёлкнуть кнопку Button2 ("Завершить Блокнот").
После создания процесса помещаем в поля Edit1 и Edit2
значения дескриптора созданного процесса и глобального
идентификатора процесса. Т.е. используем прежние
компоненты Edit1, Edit2, а заодно и Label1, Label2
```

```
(для подсказки к этим новым значениям полей
Label1, Label2). }
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants,  Classes,  Graphics, Controls,
 Forms,  Dialogs, StdCtrls; //Не опускать элементы списка

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Label1: TLabel;



    Button2: TButton;
    Edit2: TEdit;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;


var
  Form1: TForm1;
  pInfo: TProcessInformation; //Структура,
  //содержащая информацию о созданном процессе
implementation

{$R *.DFM}

{Обработка события активации Form1 (On Activate)}
procedure TForm1.FormActivate(Sender: TObject);
begin
  Form1.Caption := 'Запуск программы Блокнот' +
  ' с загрузкой Zapiska.txt и завершение её работы';
  Label1.Caption := 'Введите путь к Notepad.exe';
  Label2.Caption := 'Введите путь к Zapiska.txt';
  Button1.Caption := 'Запустить Блокнот';
```

```
  Button2.Caption := 'Завершить Блокнот';
end;

{Обработка щелчка на кнопке Button1.
Открываем программу Блокнот и загружаем
в неё файл Zapiska.txt. Выводим дескриптор
созданного процесса и глобальный идентификатор
процесса }
procedure TForm1.Button1Click(Sender: TObject);
  var
  SInfo : TStartupInfo; //Структура, заполняемая
                  // при создании процесса
begin


  FillChar(SInfo, SizeOf(SInfo), #0);
  //Очистили поля структуры SInfo
  SInfo.cb := SizeOf(SInfo);
  //Заполнили в структуре SInfo поле cb
  //размером этой структуры в байтах
  If Not CreateProcess(nil,
       PChar(Trim(Edit1.Text) + 'Notepad.exe' +
       ' ' + Trim(Edit2.Text) + 'Zapiska.txt'),
       //Выше стоит командная строка запуска Блокнот с
       //загрузкой заданного текст.файла Zapiska.txt
       nil,
       nil,
       False,
       HIGH_PRIORITY_CLASS,
       // Приоритет процесса - "Критическая задача"
       nil,
       nil,
       SInfo,
       pInfo)
  Then
     //Выдать сообщение об ошибке
     ShowMessage(IntToStr(GetLastError));
     //Вывести дескриптор созд. процесса
     //и глоб.идентиф-р процесса
     Label1.Caption := 'Дескриптор созданн. процесса';
     Label2.Caption := 'Глобальн. идентиф-тор процесса';
     Edit1.Text := IntToStr(pInfo.hProcess);
     Edit2.Text := IntToStr(pInfo.dwProcessId);
end;

{Обработка щелчка на кнопке Button2. Закрываем
```

```
программу Блокнот с помощью функции TerminateProcess.
Если даже текст в ред-ре был изменён, то закроет его,
не предложив сохранить внесённые изменения и их
не сохранит}
procedure TForm1.Button2Click(Sender: TObject);
var
   uExitCode: LongWord;
begin
  if not GetExitCodeProcess(pInfo.hProcess, uExitCode)
  //Определили код выхода процесса uExitCode


  then ShowMessage('Ошибка завершения процесса N' +
  IntToStr(GetLastError));
  TerminateProcess(pInfo.hProcess,uExitCode);
  //Процедура завершила процесс
  end;
{ Альтернативный вариант закрытия программы Блокнот
с помощью функции PostMessage. Эта функция даёт
возможность сохранить изменения текста в редакторе.
DES - дескриптор окна вызываемой пр-ммы, функция
FindWindow получает в качестве вх.параметров имя
класса окна и имя окна (отображаемое в строке заго-
ловка окна). Для отыскания имени класса окна надо
заранее в Windows запустить программу
Пуск\Программы\Delphi6(или 7)\WinSight32
и затем выполнить команду Spy\Find Window
и найти в появившемся списке строку Overlapped с
именем нужного окна, в ней также будет стоять в
фигурных скобках и имя класса окна.}
procedure TForm1.Button2Click(Sender: TObject);
var DES: hWnd;
  begin
    DES:=FindWindow('Notepad','Zapiska.txt - Блокнот');
    PostMessage(DES,WM_Close,0,0);
  end;
}
end.
```

ЛИТЕРАТУРА

Олифер В.Г.,Олифер Н.А. Сетевые операционные системы. 羊 М.: Изд-во "Питер", 2007 г.