Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №3

по дисциплине «Операционные системы»

«Архитектура памяти Windows»

Выполнил: студент гр..

Проверил:.

Тамбов, 20

***Цели и задачи****.*

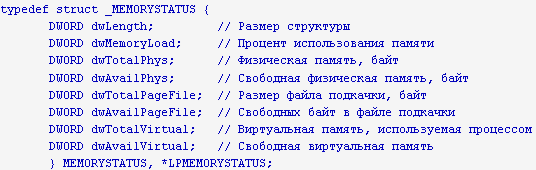
Получение практических навыков по программированию в Win32 API и изучение архитектуры памяти операционной системы Windows.

***Решение задачи****.*

**Функция GlobalMemoryStatus**

Функция GlobalMemoryStatus из библиотеки Kernel32.dll предоставляет информацию об использовании физической и виртуальной памяти компьютера. Вот её прототип:

Тип данных LPMEMORYSTATUS - это указатель на структуру *MemoryStatus*:

Функция не возвращает никаких значений.

**Снимки системы**

В основе **Tool Help Library** лежит понятие «snapshot» - мгновенного снимка состояния запущенных приложений. Перед вызовом всех функций библиотеки необходимо создать snapshot. Это осуществляется вызовом функции

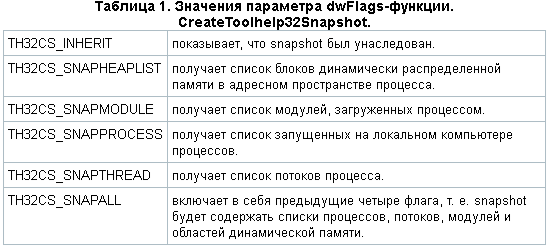
Параметр **dwFlags** указывает, какая именно информация интересует пользователя. Его значения приведены в Таблице 1.

Параметр **th32ProcessID** идентифицирует процесс, состояние которого исследуется. Он необходим при использовании **dwFlags** со значениями **TH32CS\_SNAPHEAPLIST**, **TH32CS\_SNAPMODULE** и **TH32CS\_SNAP-THREAD**. В остальных случаях этот параметр игнорируется.

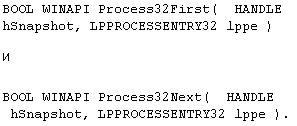
Так, например, для получения списка модулей процесса **dwProcessID** необходим следующий вызов:

Уничтожается объект snapshot стандартным вызовом **CloseHandle**( hSnapshot );

Все ошибки, возникающие при выполнении функций THL, возвращаются стандартным вызовом GetLastError и FormatMesssage.



**Список процессов**

Чтобы получить список действующих в настоящий момент процессов, необходимо задействовать две функции:

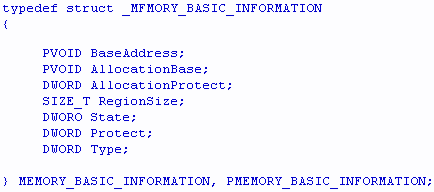
В обеих функциях два параметра. Первый - дескриптор, возвращаемый предыдущим вызовом **CreateToolhelp32Snapshot**, а второй указывает на структуру **PROCESSENTRY32**, в которой возвращается результат выполнения функции. Перед вызовом **Process32First** необходимо установить в поле dwSize структуры **PROCESSENTRY32** значение, равное размеру самой структуры - sizeof(PROCESSENTRY32), иначе функция вернет ошибку.

**Определение состояния адресного пространства**

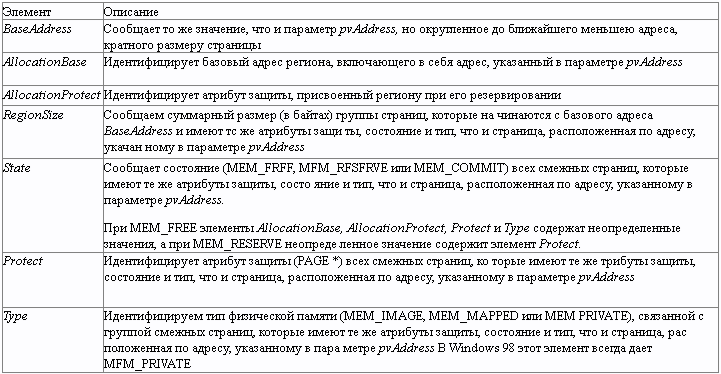
В Windows имеется функция, позволяющая запрашивать определенную информацию об участке памяти по заданному адресу (в пределах адресного пространства вызыва ющего процесса): размер, тип памяти и атрибуты защиты. Это функция:



Парная ей функция, ***VirtualQueryEx****,* сообщает ту же информацию о памяти, но в другом процессе:

Эти функции идентичны с тем исключением, что ***VirtualQueryEx***принимает описатель процесса, об адресном пространстве которого Вы хотите получить информацию. При вызове ***VirtualQitery(Ex)***параметр ***pvAddress***должен содержать адрес виртуальной памяти, о которой Вы хо тите получить информацию. Параметр ***ртbi***— это адрес структуры **MEMORY\_BA SIC\_INFORMATION**, которую надо создать перед вызовом функции. Данная структура определена в файле WinNT.h так

Параметр ***dwLength***задает размер структуры **MEMORY\_BASIC\_INFORMATION**. Функция ***VirtualQuery(Ex)***возвращает число байтов, скопированных в буфер.

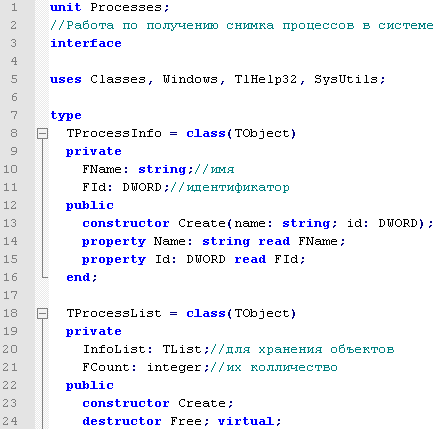
Используя адрес, указанный Вами в параметре ***pvAddress****,* функция ***VirtualQuery(Ex)***заполняет структуру информацией о диапазоне смежных страниц, имеющих одинаковые состояние, атрибуты защиты и тип. Описание элементов структуры приведено в таблице ниже

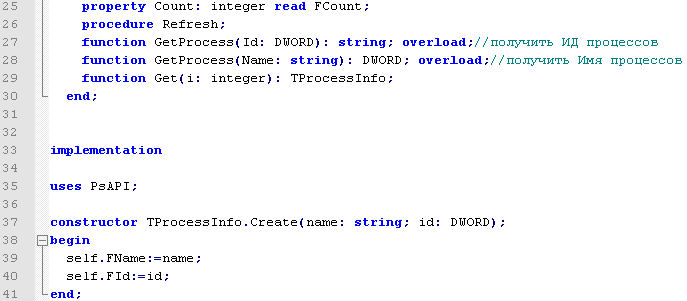
Итого:

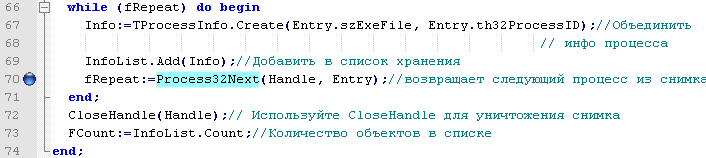
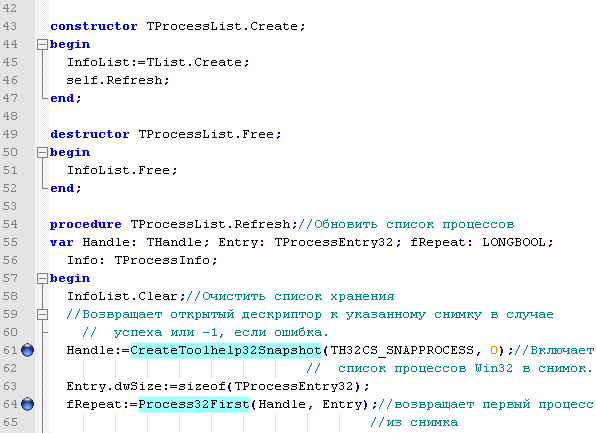
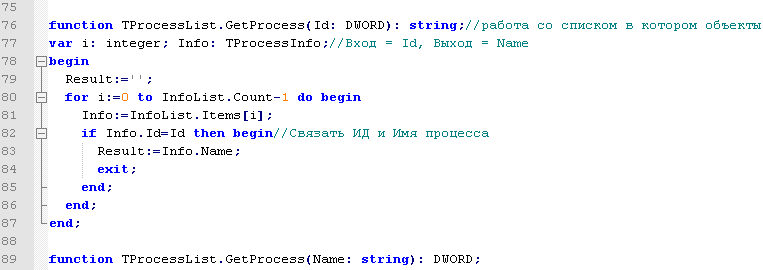
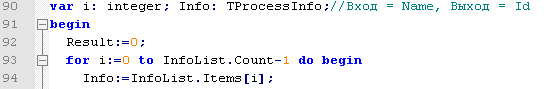
* получаем инфо о памяти Windows.**GlobalMemoryStatus**(lpBuffer)
* делаем снимок процессов **CreateToolhelp32Snapshot**(TH32CS\_SNAPPROCESS, 0)
* листаем снимок при помощи функций

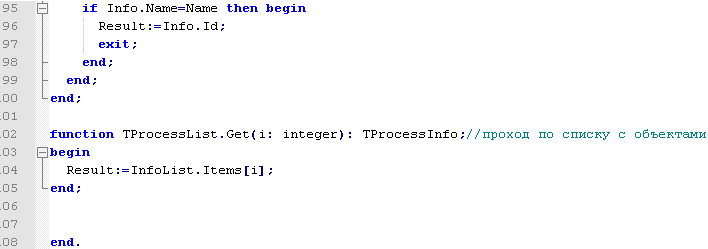
**Process32First**(Handle, Entry) и **Process32Next**(Handle, Entry)

* копаем в глубь снимка функцией **VirtualQueryEx**, строим карты

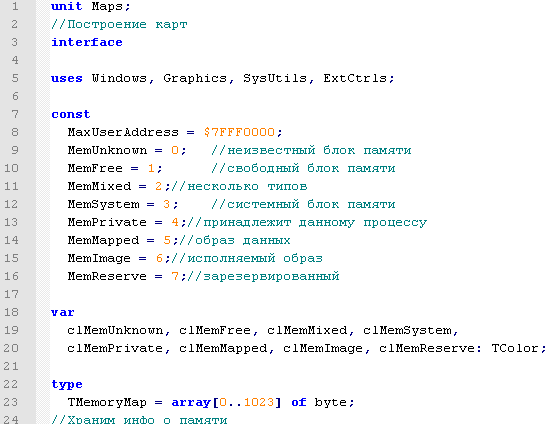
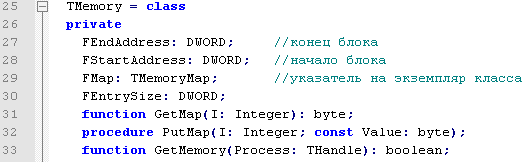
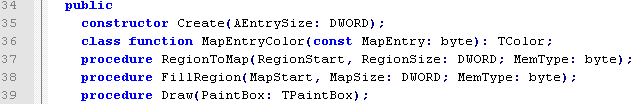
**Код для процессов:**

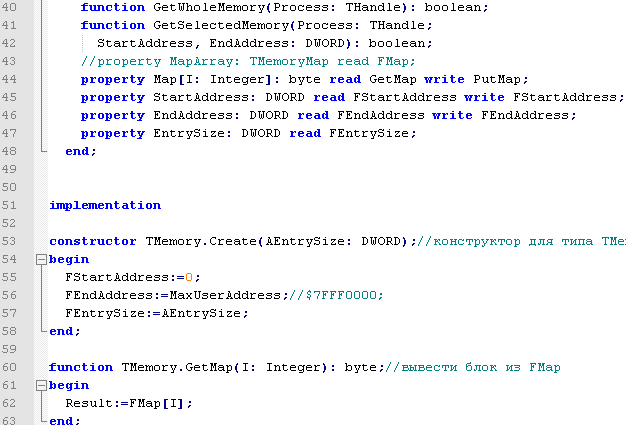
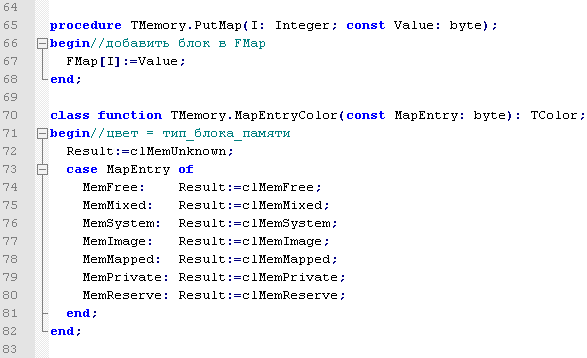
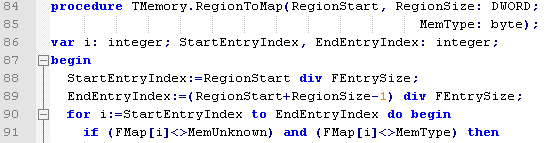


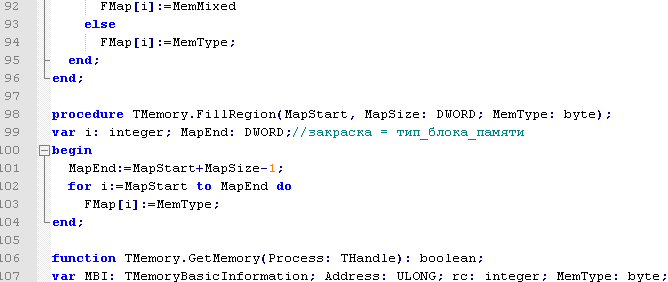
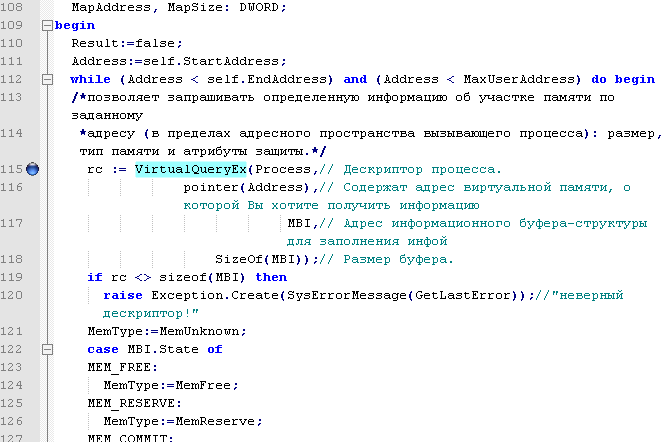
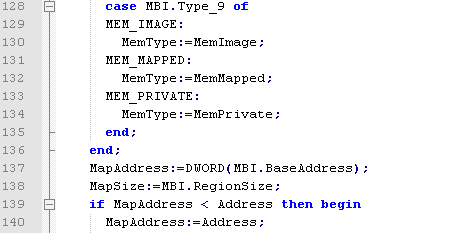


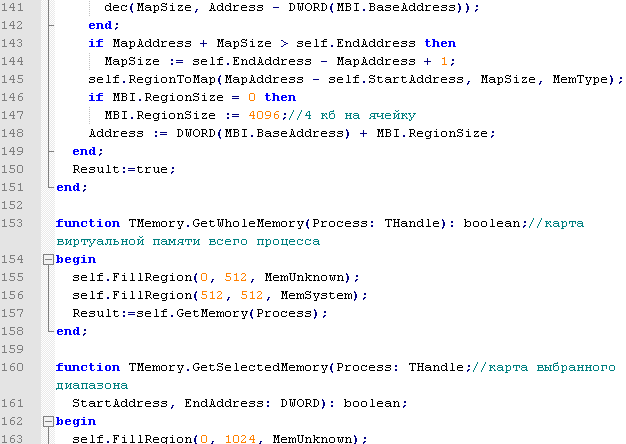
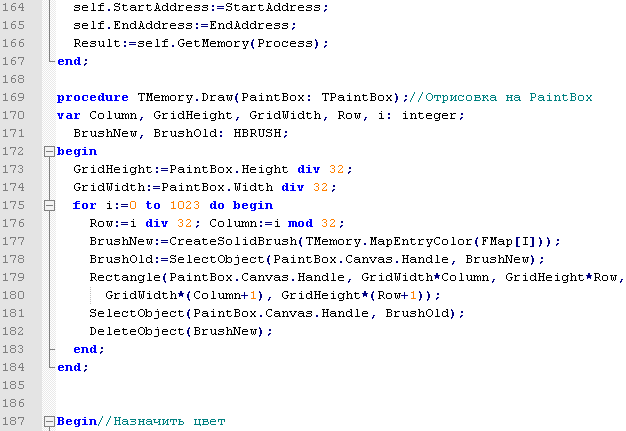
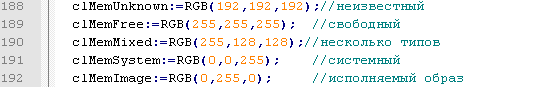


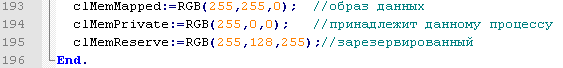
**Код для построения карт процесса**











**Код для формы**

unit MainForm;

//Вывод-связка инфы с компонентами формы

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, TeEngine, Series, ExtCtrls, TeeProcs, Chart,

Processes, Maps;

type

TForm1 = class(TForm)

MemoryChart: TChart; //карта памяти

Series1: TPieSeries;

PhysicalChart: TChart; //карта физической памяти

PieSeries1: TPieSeries; //сектор-диаграмма

PageFileChart: TChart; //карта файла подкачки

PieSeries2: TPieSeries; //сектор-диаграмма

VirtualChart: TChart; //карта виртуальной памяти

PieSeries3: TPieSeries; //сектор-диаграмма

ProcessesComboBox: TComboBox; //выпадСписок с процессами

ProcessHandleLabel: TLabel; //иденификатор выбранного процесса

RefreshProcessesButton: TButton;//кнопка обновления //выпадСписок с процессами

DiagramsTimer: TTimer; //таймер обновления всегоИвся

PaintBoxWhole: TPaintBox; //

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

lbSelectedRange: TLabel;

lbLowestSelected: TLabel;

lbMiddleSelectedLow: TLabel;

lbMiddleSelectedHigh: TLabel;

lbHighestSelected: TLabel;

PaintBoxSelected: TPaintBox;

Label14: TLabel;

Label15: TLabel;

Label16: TLabel;

lbSelectedPage: TLabel;

gbLegend: TGroupBox; //типы блоков памяти

ShapeUnknown: TShape;

Label5: TLabel;

ShapeFree: TShape;

Label6: TLabel;

ShapeMixed: TShape;

Label7: TLabel;

ShapeSystem: TShape;

Label8: TLabel;

ShapePrivate: TShape;

ShapeMapped: TShape;

ShapeImage: TShape;

Label9: TLabel;

ShapeReserved: TShape;

Label10: TLabel;

Label11: TLabel;

Label12: TLabel;

btnRefreshProcessMap: TButton;//Обновить карту процесса

procedure FormCreate(Sender: TObject);

procedure ProcessesComboBoxChange(Sender: TObject);

procedure RefreshProcessesButtonClick(Sender: TObject);

procedure DiagramsTimerTimer(Sender: TObject);

procedure FormDestroy(Sender: TObject);

procedure PaintBoxWholePaint(Sender: TObject);//карта

procedure PaintBoxSelectedPaint(Sender: TObject);//карта

procedure PaintBoxSelectedMouseDown(Sender: TObject;//карта

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

procedure btnRefreshProcessMapClick(Sender: TObject);

procedure PaintBoxWholeMouseDown(Sender: TObject; Button: TMouseButton;//карта

Shift: TShiftState; X, Y: Integer);

private

{ Private declarations }

ProcessList: TProcessList;

fCouldRepaintSelectedMap: boolean;

hCurrentProcess: THandle;

WholeMemory: TMemory;

SelectedMemory: TMemory;

fPrevSuccess: boolean;

procedure RefreshDiagrams;

procedure MakeProcessList;

procedure RefreshHandleLabel;

procedure DrawLegend;

procedure DrawWholeMap;

procedure DrawSelectedMap;

procedure SelectWholeMapEntry(EntryIndex: Integer);

procedure RefreshProcessMap;

function GetPaintBoxIndex(X, Y: integer; PaintBox: TPaintBox): integer;

public

{ Public declarations }

end;

const

DefaultProcessItem = 'OSLab4.exe';

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.RefreshDiagrams;

var lpBuffer: \_MEMORYSTATUS;

begin

Windows.GlobalMemoryStatus(lpBuffer);//позволяет отслеживать текущее состояние памяти

with Series1 do begin

Clear;

AddPie(lpBuffer.dwMemoryLoad, 'Занято', clRed);

AddPie(100-lpBuffer.dwMemoryLoad, 'Свободно', clLime);

end;

with PieSeries1 do begin

Clear;

AddPie(lpBuffer.dwTotalPhys-lpBuffer.dwAvailPhys, 'Занято', clRed);

AddPie(lpBuffer.dwAvailPhys, 'Свободно', clLime);

end;

with PieSeries2 do begin

Clear;

AddPie(lpBuffer.dwTotalPageFile-lpBuffer.dwAvailPageFile, 'Занято', clRed);

AddPie(lpBuffer.dwAvailPageFile, 'Свободно', clLime);

end;

with PieSeries3 do begin

Clear;

AddPie(lpBuffer.dwTotalVirtual-lpBuffer.dwAvailVirtual, 'Занято', clRed);

AddPie(lpBuffer.dwAvailVirtual, 'Свободно', clLime);

end;

end;

procedure TForm1.MakeProcessList;

var i: integer;

begin

ProcessesComboBox.Clear;

for i:=0 to ProcessList.Count-1 do

ProcessesComboBox.Items.Add(ProcessList.Get(i).Name);

if (ProcessesComboBox.Items.Count > 0) then

with ProcessesComboBox do begin

ItemIndex:=0;

for i:=0 to Items.Count-1 do

if (UpperCase(Items[i])=UpperCase(DefaultProcessItem)) then begin

ItemIndex:=i;

break;

end;

end;

RefreshHandleLabel;

end;

procedure TForm1.RefreshHandleLabel;

var index: integer; item: string; id: DWORD;

begin

fPrevSuccess:=true;

index:=ProcessesComboBox.ItemIndex;

item:=ProcessesComboBox.Items[index];

id:=ProcessList.GetProcess(item);

if hCurrentProcess <> GetCurrentProcess then

CloseHandle(hCurrentProcess);

hCurrentProcess:=OpenProcess(PROCESS\_QUERY\_INFORMATION, true, id);

ProcessHandleLabel.Caption:='Идентификатор: ' + IntToStr(id);

end;

procedure TForm1.DrawLegend;

begin

ShapeUnknown.Brush.Color:= TMemory.MapEntryColor(MemUnknown);

ShapeFree.Brush.Color:= TMemory.MapEntryColor(MemFree);

ShapeMixed.Brush.Color:= TMemory.MapEntryColor(MemMixed);

ShapeSystem.Brush.Color:= TMemory.MapEntryColor(MemSystem);

ShapePrivate.Brush.Color:= TMemory.MapEntryColor(MemPrivate);

ShapeMapped.Brush.Color:= TMemory.MapEntryColor(MemMapped);

ShapeImage.Brush.Color:= TMemory.MapEntryColor(MemImage);

ShapeReserved.Brush.Color:= TMemory.MapEntryColor(MemReserve);

end;

procedure TForm1.DrawWholeMap;

begin

PaintBoxWhole.Invalidate;

end;

procedure TForm1.DrawSelectedMap;

begin

fCouldRepaintSelectedMap:=true;

PaintBoxSelected.Invalidate;

end;

procedure TForm1.SelectWholeMapEntry(EntryIndex: Integer);//выбор на карте процесса

var StartAddress, EndAddress: DWORD;

begin

StartAddress := DWORD(EntryIndex)\*WholeMemory.EntrySize;

EndAddress := DWORD((EntryIndex+1))\*WholeMemory.EntrySize-1;

lbSelectedRange.Caption := Format('Выбранный диапазон: 0x%.8x-0x%.8x',

[StartAddress, EndAddress]);

if EntryIndex >= 512 then begin

SelectedMemory.FillRegion(0, 1024, MemSystem);

SelectedMemory.StartAddress := StartAddress;

SelectedMemory.EndAddress := EndAddress;

end else begin

fPrevSuccess:=false;

SelectedMemory.GetSelectedMemory(hCurrentProcess, StartAddress, EndAddress);

fPrevSuccess:=true;

end;

lbLowestSelected.Caption := Format('0x%.8x', [StartAddress]);

lbMiddleSelectedLow.Caption := Format('0x%.8x', [StartAddress +

DWORD(WholeMemory.EntrySize div 2 - 1)]);

lbMiddleSelectedHigh.Caption := Format('0x%.8x', [StartAddress +

DWORD(WholeMemory.EntrySize div 2)]);

lbHighestSelected.Caption := Format('0x%.8x', [EndAddress]);

lbSelectedPage.Caption := 'не выбран';

DrawSelectedMap;

end;

procedure TForm1.RefreshProcessMap;

begin

fPrevSuccess:=false;

try

WholeMemory.GetWholeMemory(hCurrentProcess);

fPrevSuccess:=true;

except

SelectedMemory.FillRegion(0, 1024, MemUnknown);

WholeMemory.FillRegion(0, 1024, MemUnknown);

DrawWholeMap;

DrawSelectedMap;

end;

DrawWholeMap;

SelectWholeMapEntry(0);

end;

function TForm1.GetPaintBoxIndex(X, Y: integer; PaintBox: TPaintBox): integer;

var EntryX, EntryY, GridHeight, GridWidth: integer;

begin

GridHeight:=PaintBox.Height div 32;

GridWidth :=PaintBox.Width div 32;

EntryX:=(X div GridWidth);

EntryY:=(Y div GridHeight);

Result:=EntryY\*32+EntryX;

end;

procedure TForm1.ProcessesComboBoxChange(Sender: TObject);//выбор из списка

begin

RefreshHandleLabel;

RefreshProcessMap;

end;

procedure TForm1.RefreshProcessesButtonClick(Sender: TObject);

begin

RefreshDiagrams;

ProcessList.Refresh;

MakeProcessList;

RefreshProcessMap;

end;

procedure TForm1.DiagramsTimerTimer(Sender: TObject);

begin

RefreshDiagrams;

end;

procedure TForm1.FormCreate(Sender: TObject);

begin

fPrevSuccess:=true;

hCurrentProcess:=GetCurrentProcess;

DrawLegend;

ProcessList:=TProcessList.Create;

MakeProcessList;

WholeMemory:=TMemory.Create(4096\*1024);

SelectedMemory:=TMemory.Create(4096);

RefreshProcessMap;

end;

procedure TForm1.FormDestroy(Sender: TObject);

begin

ProcessList.Free;

WholeMemory.Free;

SelectedMemory.Free;

if hCurrentProcess <> GetCurrentProcess then

CloseHandle(hCurrentProcess);

end;

procedure TForm1.PaintBoxWholePaint(Sender: TObject);

begin

WholeMemory.Draw(Sender as TPaintBox);

end;

procedure TForm1.PaintBoxSelectedPaint(Sender: TObject);

begin

if not fCouldRepaintSelectedMap then exit;

SelectedMemory.Draw(Sender as TPaintBox);

end;

procedure TForm1.PaintBoxSelectedMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

var EntryIndex: integer; Address: DWORD;

PaintBox : TPaintBox;

begin

PaintBox:=Sender as TPaintBox;

if Button=mbLeft then begin

EntryIndex:=GetPaintBoxIndex(X, Y, PaintBox);

Address:=DWORD(DWORD(EntryIndex)\*SelectedMemory.EntrySize)+

SelectedMemory.StartAddress;

lbSelectedPage.Caption := Format('%.8x', [Address]);

end;

end;

procedure TForm1.btnRefreshProcessMapClick(Sender: TObject);

begin

RefreshProcessMap;

end;

procedure TForm1.PaintBoxWholeMouseDown(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

var EntryIndex: integer; PaintBox: TPaintBox;

begin

PaintBox:=Sender as TPaintBox;

if Button = mbLeft then begin

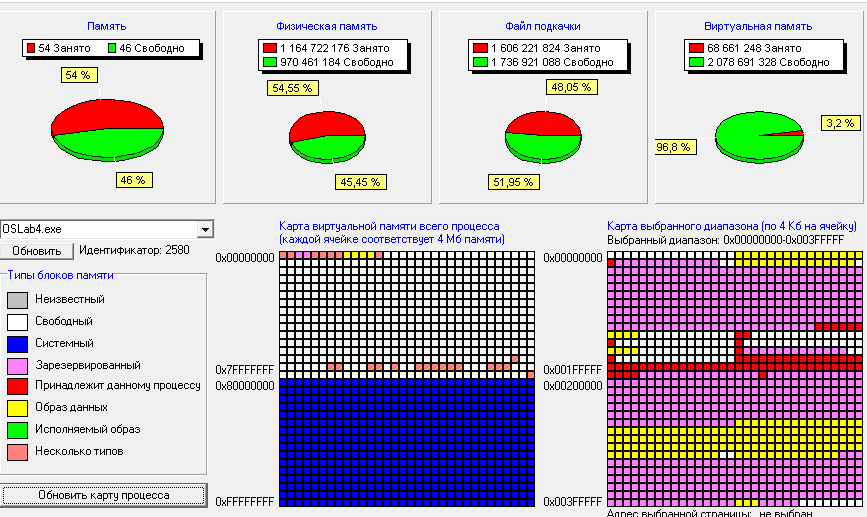
EntryIndex:=GetPaintBoxIndex(X, Y, PaintBox);

SelectWholeMapEntry(EntryIndex);

end;

end;

end.



Форма: