// Thr3.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "resource.h"

//#include "windowsx.h"

#define MAX\_LOADSTRING 100

//Look into Windowsx.h for System definitions without the prefix R

#define RHANDLE\_MSG(hwnd, message, fn) \

case (message) : return RHANDLE\_##message((hwnd), (wParam), (lParam), (fn))

#define RHANDLE\_WM\_COMMAND(hwnd, wParam, lParam, fn) \

((fn)((hwnd), (int)(LOWORD(wParam)), (HWND)(lParam), (UINT)HIWORD(wParam)), -1L)

#define RHANDLE\_WM\_INITDIALOG(hwnd, wParam, lParam, fn) \

(LRESULT)(DWORD)(UINT)(BOOL)(fn)((hwnd), (HWND)(wParam), lParam)

#define RFORWARD\_WM\_COMMAND(hwnd, id, hwndCtl, codeNotify, fn) \

(LONG)(fn)((hwnd), WM\_COMMAND, MAKEWPARAM((UINT)(id), (UINT)(codeNotify)), (LPARAM)(HWND)(hwndCtl))

//Globals to control threads

TCHAR szMutexName[] = "$MutexFor3Threads$";

HANDLE hMutex;

HANDLE hThreadE[2];

BOOL fTerminateE;

BOOL fTerminateR;

bool bSuspendE,

bSuspendR;

unsigned thridE; //thread identifier

DWORD nWM\_PAINT = 0; //WM\_PAINT count

// Global Variables:

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // The title bar text

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

//\_stdcall for CALLBACK (see Windef.h)

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK ChangeThreadPriority\_Box(HWND, UINT, WPARAM, LPARAM);

BOOL ChThPr\_OnInitDialog(HWND hDlg, HWND hwndFocus, LPARAM lParam);

LONG ChThPr\_OnCommand(HWND hDlg, int id, HWND hwndCtl, UINT codeNotify);

LONG WndProc\_OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct);

LONG WndProc\_OnDestroy(HWND hWnd);

void WndProc\_OnPaint(HWND hWnd);

LONG WndProc\_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify);

void WndProc\_OnClose(HWND hWnd);

unsigned int \_\_stdcall PaintEllipse(void \*hWnd);

unsigned int \_\_stdcall PaintRectangle(void \*hWnd);

void SuspendEllipse(HMENU hMenu, bool \*bSuspend);

void SuspendRectangle(HMENU hMenu, bool \*bSuspend);

void TerminateEllipse(HWND hWnd, HMENU hMenu, BOOL \*fTerminateE);

//\_stdcall for APIENTRY (see Windef.h), used for new and last versions

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{//Create an object Mutex (named)

//As it is named it is possible to verify if there is only one application instance too!

if (!(hMutex = CreateMutex(NULL, FALSE, szMutexName))){//FALSE sets it In SIGNALED STATE!

//NO OWNERSHIP

MessageBox(NULL, "CreateMutex Error", "Four Threads",

MB\_OK | MB\_ICONEXCLAMATION);

return 01;

}

if (GetLastError() == ERROR\_ALREADY\_EXISTS){

MessageBox(NULL, "Mutex already started", "Four Threads",

MB\_OK | MB\_ICONEXCLAMATION);

return 01;

}

TCHAR msgHINST[250];

wsprintf(msgHINST, TEXT("HINSTANCE is %ld"), hInstance);

MessageBox(NULL, msgHINST, "Four Threads",

MB\_OK);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_THR3, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC\_THR3);

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

// COMMENTS:

//

// This function and its usage is only necessary if you want this code

// to be compatible with Win32 systems prior to the 'RegisterClassEx'

// function that was added to Windows 95. It is important to call this function

// so that the application will get 'well formed' small icons associated

// with it.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = (WNDPROC)WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI\_THR3);

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = (LPCSTR)IDC\_THR3;

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI\_SMALL);

return RegisterClassEx(&wcex);

}

//

// FUNCTION: InitInstance(HANDLE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance; // Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, unsigned, WORD, LONG)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

// int wmId, wmEvent;

switch (message)

{

HANDLE\_MSG(hWnd, WM\_COMMAND, WndProc\_OnCommand);//!

HANDLE\_MSG(hWnd, WM\_PAINT, WndProc\_OnPaint);//!

HANDLE\_MSG(hWnd, WM\_CREATE, WndProc\_OnCreate);//!

HANDLE\_MSG(hWnd, WM\_DESTROY, WndProc\_OnDestroy);//!

HANDLE\_MSG(hWnd, WM\_CLOSE, WndProc\_OnClose);//!

default:return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;//BY CHANCE TO BE USED

}

LONG WndProc\_OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)

{

// Parse the menu selections:

switch (id)

{

case IDM\_SUSE:

SuspendEllipse(GetMenu(hWnd), &bSuspendE);

return 0L;

case IDM\_SUSR:

SuspendRectangle(GetMenu(hWnd), &bSuspendR);

return 0L;

case IDM\_TERME:

TerminateEllipse(hWnd, GetMenu(hWnd), &fTerminateE);

return 0L;

case IDM\_CH\_THR\_PR\_ELL:

DialogBox(hInst, (LPCTSTR)IDD\_DIALOG\_PRIORITY, hWnd,

(DLGPROC)ChangeThreadPriority\_Box);

return 0L;

case IDM\_ABOUT:

DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

return 0L;// break;

case IDM\_EXIT:

DestroyWindow(hWnd); //PostQuitMessage(0);

return 0L;// break;

default:break;

// return DefWindowProc(hWnd, message, wParam, lParam);

}

return RFORWARD\_WM\_COMMAND(hWnd, id, hwndCtl, codeNotify, DefWindowProc);

}

void WndProc\_OnPaint(HWND hWnd)

{

PAINTSTRUCT ps;

HDC hdc;

TCHAR szHello[MAX\_LOADSTRING];

TCHAR sznWM\_PAINT[50];

DWORD dwRetCode;

LoadString(hInst, IDS\_HELLO, szHello, MAX\_LOADSTRING);

dwRetCode = WaitForSingleObject(hMutex, INFINITE);

if (dwRetCode == WAIT\_OBJECT\_0)

{

hdc = BeginPaint(hWnd, &ps);

// TODO: Add any drawing code here...

RECT rt;

GetClientRect(hWnd, &rt);

DrawText(hdc, szHello, strlen(szHello), &rt, DT\_CENTER);

wsprintf(sznWM\_PAINT, TEXT("nWM\_PAINT=%d"), nWM\_PAINT);

rt.top = rt.top + 50;

DrawText(hdc, sznWM\_PAINT, strlen(sznWM\_PAINT), &rt, DT\_CENTER);

EndPaint(hWnd, &ps);

nWM\_PAINT++;

ReleaseMutex(hMutex);

}

// return 0;

}

// This program requires the multithreaded library. For example,

// compile with the following command line:

// CL /MT /D "\_X86\_" BEGTHRD.C

LONG WndProc\_OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)

{

fTerminateE = FALSE;

fTerminateR = FALSE;

//Thread PaintEllipse starting not suspended!

bSuspendE = false;

//Thread PaintRectangle starting not suspended!

bSuspendR = false;

unsigned ususpend = 0;

hThreadE[0] = (HANDLE)\_beginthreadex(NULL,//must be FOR W95 ,SA

0,//stack size committed

PaintEllipse,

(void \*)hWnd, //\*arglist

ususpend,//0

&thridE

);

if (!hThreadE[0]){

MessageBox(NULL, "Thread start Error",

"PaintEllipse Thread",

MB\_OK | MB\_ICONEXCLAMATION);

return FALSE;

}

hThreadE[1] = (HANDLE)\_beginthreadex(NULL,//must be FOR W95

0,//stack size

PaintRectangle,

(void \*)hWnd,

ususpend,//0

&thridE

);

if (!hThreadE[1]){

MessageBox(NULL, "Thread start Error",

"PaintRectangle Thread",

MB\_OK | MB\_ICONEXCLAMATION);

return FALSE;

}

return TRUE;//BOOL

}

LONG WndProc\_OnDestroy(HWND hWnd)

{

if (bSuspendE)

ResumeThread(hThreadE[0]);

if (bSuspendR)

ResumeThread(hThreadE[1]);

fTerminateE = TRUE;

fTerminateR = TRUE;

TCHAR szRetRes[100];

DWORD dwRet;

switch (dwRet = WaitForMultipleObjects(2, hThreadE, TRUE, INFINITE))

{

case WAIT\_ABANDONED\_0:

wsprintf(szRetRes, TEXT("WAIT\_ABANDONED\_0=%d dwRet=%d"), WAIT\_ABANDONED\_0, dwRet);

MessageBox(NULL, szRetRes, "WaitForMultipleObjects", MB\_OK | MB\_ICONEXCLAMATION);

break;

case WAIT\_FAILED:

break;

case WAIT\_OBJECT\_0:

wsprintf(szRetRes, TEXT("WAIT\_OBJECT\_0=%d dwRet=%d"), WAIT\_OBJECT\_0, dwRet);

MessageBox(NULL, szRetRes, "WaitForMultipleObjects", MB\_OK | MB\_ICONEXCLAMATION);

break;

}

if (!CloseHandle(hThreadE[0]))

{

MessageBox(NULL, "CloseHandle failed",//NULL , no hWnd(the window is destroied)

"PaintEllipse Thread", MB\_OK | MB\_ICONEXCLAMATION);

};

if (!CloseHandle(hThreadE[1]))

{

MessageBox(NULL, "CloseHandle failed",

"PaintEllipse Thread", MB\_OK | MB\_ICONEXCLAMATION);

};

CloseHandle(hMutex);

PostQuitMessage(0);

return (0);

}

void WndProc\_OnClose(HWND hWnd)

{

/\*

LONG InterlockedExchangeAdd (

PLONG Addend, // pointer to the addend

LONG Increment // increment value

);

\*/

DWORD dwn = InterlockedExchangeAdd((PLONG)&nWM\_PAINT, 0);

TCHAR sznWM[50];

sprintf(sznWM, "nWM\_PAINT=%11d\n", dwn);// no nWM\_PAINT

int r = MessageBox(hWnd, sznWM, "Primary Thread WM\_PAINT count",

MB\_OKCANCEL | MB\_ICONEXCLAMATION);

//return CLFORWARD\_WM\_CLOSE(hWnd, DefWindowProc);

DestroyWindow(hWnd); // or

//FORWARD\_WM\_CLOSE(hWnd, DefWindowProc);

}

// Message handler for about box.

LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE; //SetDlgMsgResult-???ABSENT

}

unsigned int \_\_stdcall PaintEllipse(void \*hWnd){

HDC hDC;

RECT rect;

LONG xLeft, xRight, yTop, yBottom;

short nRed, nGreen, nBlue;

HBRUSH hBrush, hOldBrush;

DWORD dwRetCode;

srand((unsigned int)hWnd + 100);

while (!fTerminateE){ // Is not it to be continued?

// To be continued!

switch (dwRetCode = WaitForSingleObject(hMutex, INFINITE)){

case WAIT\_ABANDONED:break;

case WAIT\_FAILED:break;

case WAIT\_OBJECT\_0:

{

hDC = GetDC((HWND)hWnd);

nRed = rand() % 255; nGreen = rand() % 255; nBlue = rand() % 255;

GetWindowRect((HWND)hWnd, &rect);

xLeft = rand() % (rect.left + 1);

xRight = rand() % (rect.right + 1);

yTop = rand() % (rect.top + 1);

yBottom = rand() % (rect.bottom + 1);

hBrush = CreateSolidBrush(RGB(nRed, nGreen, nBlue));

hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);

Ellipse(hDC, min(xLeft, xRight), min(yTop, yBottom),

max(xLeft, xRight), max(yTop, yBottom));

SelectObject(hDC, hOldBrush);

DeleteObject(hBrush);

ReleaseDC((HWND)hWnd, hDC);

ReleaseMutex(hMutex);

break;

};

}

Sleep(100);

// InvalidateRect((HWND)hWnd,NULL,TRUE); //NULL- the whole client region

//TRUE - the background is erased when the BeginPaint function is called.

Sleep(100);

}//End of while

//InvalidateRect((HWND)hWnd,NULL,TRUE);

return 0;

}//End of PaintEllipse

unsigned int \_\_stdcall PaintRectangle(void \*hWnd){

////////void PaintRectangle(void \*hWnd){

HDC hDC;

RECT rect;

LONG xLeft, xRight, yTop, yBottom;

short nRed, nGreen, nBlue;

HBRUSH hBrush, hOldBrush;

DWORD dwRetCode;

srand((unsigned int)hWnd);

while (!fTerminateR){

// Is not it to be continued?

// To be continued!

switch (dwRetCode = WaitForSingleObject(hMutex, INFINITE)){

case WAIT\_ABANDONED:break;

/\*

The specified object is a mutex object that was not released by the thread

that owned the mutex object before the owning thread terminated.

Ownership of the mutex object is granted to the calling thread,

and the mutex is set to nonsignaled.

\*/

case WAIT\_FAILED:

// DWORD GetLastError(VOID)

break;

case WAIT\_TIMEOUT:break; //by INFINITE never occurs

case WAIT\_OBJECT\_0:

{

hDC = GetDC((HWND)hWnd);

nRed = rand() % 255; nGreen = rand() % 255; nBlue = rand() % 255;

GetWindowRect((HWND)hWnd, &rect);

xLeft = rand() % (rect.left + 1);

xRight = rand() % (rect.right + 1);

yTop = rand() % (rect.top + 1);

yBottom = rand() % (rect.bottom + 1);

hBrush = CreateSolidBrush(RGB(nRed, nGreen, nBlue));

hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);

Rectangle(hDC, min(xLeft, xRight), min(yTop, yBottom),

max(xLeft, xRight), max(yTop, yBottom));

SelectObject(hDC, hOldBrush);

DeleteObject(hBrush);

ReleaseDC((HWND)hWnd, hDC);

//Sleep(10000);

ReleaseMutex(hMutex);

break;

};

}

Sleep(100);

InvalidateRect((HWND)hWnd, NULL, TRUE);

Sleep(100);

}//End of while

return 0;

}//End of PaintRectangle

//Wrong solution whithout using hMutex???

void SuspendEllipse(HMENU hMenu, bool \*bSuspend)

{

TCHAR message[260];

if (!\*bSuspend){

if (0xFFFFFFFF == SuspendThread(hThreadE[0]))//or -1 (==0xFFFFFFFF)

{

wsprintf(message, TEXT("SuspendThread Error %ld"), GetLastError());

MessageBox(NULL, message, "PaintEllipse Thread", MB\_OK | MB\_ICONEXCLAMATION);

return;

}

\*bSuspend = true;

//Check

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu, IDM\_SUSE, MF\_CHECKED);

}

else

{

if (0xFFFFFFFF == ResumeThread(hThreadE[0]))

{

wsprintf(message, TEXT("ResumeThread Error %ld"), GetLastError());

MessageBox(NULL, message, "PaintEllipse Thread", MB\_OK | MB\_ICONEXCLAMATION);

return;

}

\*bSuspend = false;

//Uncheck

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu, IDM\_SUSE, MF\_UNCHECKED);

}

return;

}

//Right solution

void SuspendRectangle(HMENU hMenu, bool \*bSuspend)

{

TCHAR message[260];

DWORD dwRetCode;

switch (dwRetCode = WaitForSingleObject(hMutex, INFINITE))

{

case WAIT\_ABANDONED:break;

case WAIT\_FAILED:break;

case WAIT\_OBJECT\_0:

if (!\*bSuspend){

if (0xFFFFFFFF == SuspendThread(hThreadE[1]))

{

wsprintf(message, TEXT("SuspendThread Error %ld"), GetLastError());

MessageBox(NULL, message, "PaintRectangle Thread", MB\_OK | MB\_ICONEXCLAMATION);

ReleaseMutex(hMutex);

return;

}

\*bSuspend = true;

//Check

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu, IDM\_SUSR, MF\_CHECKED);

}

else

{

if (0xFFFFFFFF == ResumeThread(hThreadE[1]))

{

wsprintf(message, TEXT("ResumeThread Error %ld"), GetLastError());

MessageBox(NULL, message, "PaintRectangle Thread", MB\_OK | MB\_ICONEXCLAMATION);

ReleaseMutex(hMutex);

return;

}

\*bSuspend = false;

//Uncheck

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu, IDM\_SUSR, MF\_UNCHECKED);

}

ReleaseMutex(hMutex);

return;

break;

default: //is never reached!!

break;

}//switch

}//SuspendRectangle

void TerminateEllipse(HWND hWnd, HMENU hMenu, BOOL \*fTerminateE)

{

if (!\*fTerminateE)

{

//Check

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu, IDM\_TERME, MF\_CHECKED);

\*fTerminateE = true;

if (bSuspendE)

ResumeThread(hThreadE[0]);

WaitForSingleObject(hThreadE[0], INFINITE);//Block the primary thread if PaintEllipse

// is in the suspended state

if (!CloseHandle(hThreadE[0]))

{

MessageBox(NULL, "CloseHandle failed",

"PaintEllipse Thread",

MB\_OK | MB\_ICONEXCLAMATION);

};

}

else

{

//Uncheck

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu, IDM\_TERME, MF\_UNCHECKED);

\*fTerminateE = false;

unsigned ususpend = (bSuspendE) ? CREATE\_SUSPENDED : 0;

hThreadE[0] = (HANDLE)\_beginthreadex(NULL,//must be FOR W95

0,//stack size

PaintEllipse,

(void \*)hWnd,

ususpend,//0

&thridE

);

if (!hThreadE[0])MessageBox(NULL, "Thread start Error",

"PaintEllipse Thread",

MB\_OK | MB\_ICONEXCLAMATION);

}

return;

}

//Dialog Box Function

//\_stdcall for CALLBACK (see Windef.h)

LRESULT CALLBACK ChangeThreadPriority\_Box(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

RHANDLE\_MSG(hDlg, WM\_INITDIALOG, ChThPr\_OnInitDialog);

RHANDLE\_MSG(hDlg, WM\_COMMAND, ChThPr\_OnCommand);

//return true; for all handled messages

default:

return FALSE;//not handled messages

}

}//ChangeThreadPriority\_Box

LONG ChThPr\_OnCommand(HWND hDlg, int id, HWND hwndCtl, UINT codeNotify)

{

// Parse the Change Thread Priority Box commands:

switch (id)

{

case IDOK:

{

int nPriorityEll = THREAD\_PRIORITY\_NORMAL;

int nIDCheckButton = IDC\_RADIO\_NORMAL, i;

for (i = 0; i<7; i++)

if (BST\_CHECKED == IsDlgButtonChecked(hDlg, IDC\_RADIO\_TMCRITICAL + i))

{

nIDCheckButton = IDC\_RADIO\_TMCRITICAL + i;

break;

}

switch (nIDCheckButton)

{

case IDC\_RADIO\_TMCRITICAL:

nPriorityEll = THREAD\_PRIORITY\_TIME\_CRITICAL;

break;

case IDC\_RADIO\_HGHST:

nPriorityEll = THREAD\_PRIORITY\_HIGHEST;

break;

case IDC\_RADIO\_ABOVE:

nPriorityEll = THREAD\_PRIORITY\_ABOVE\_NORMAL;

break;

case IDC\_RADIO\_NORMAL:

nPriorityEll = THREAD\_PRIORITY\_NORMAL;

break;

case IDC\_RADIO\_BELOW:

nPriorityEll = THREAD\_PRIORITY\_BELOW\_NORMAL;

break;

case IDC\_RADIO\_LOWEST:

nPriorityEll = THREAD\_PRIORITY\_LOWEST;

break;

case IDC\_RADIO\_IDLE:

nPriorityEll = THREAD\_PRIORITY\_NORMAL;

break;

default:

break;

}

SetThreadPriority(hThreadE[0], nPriorityEll);

}

case IDCANCEL:

EndDialog(hDlg, id);

return TRUE;//handled

default:

return TRUE;

}//return TRUE for all handled commands,else ones are not here

}

BOOL ChThPr\_OnInitDialog(HWND hDlg, HWND hwndFocus, LPARAM lParam)

{

//Set Current Priority Class and Thread Priority

int nPriorityEll = GetThreadPriority(hThreadE[0]);

int nIDCheckButton;

switch (nPriorityEll)

{

case THREAD\_PRIORITY\_TIME\_CRITICAL:

nIDCheckButton = IDC\_RADIO\_TMCRITICAL;

break;

case THREAD\_PRIORITY\_HIGHEST:

nIDCheckButton = IDC\_RADIO\_HGHST;

break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL:

nIDCheckButton = IDC\_RADIO\_ABOVE;

break;

case THREAD\_PRIORITY\_NORMAL:

nIDCheckButton = IDC\_RADIO\_NORMAL;

break;

case THREAD\_PRIORITY\_BELOW\_NORMAL:

nIDCheckButton = IDC\_RADIO\_BELOW;

break;

case THREAD\_PRIORITY\_LOWEST:

nIDCheckButton = IDC\_RADIO\_LOWEST;

break;

case THREAD\_PRIORITY\_IDLE:

nIDCheckButton = IDC\_RADIO\_IDLE;

break;

default:

break;

}

CheckRadioButton(hDlg, // handle to dialog box

IDC\_RADIO\_TMCRITICAL, // identifier of first radio button in group

IDC\_RADIO\_IDLE, // identifier of last radio button in group

nIDCheckButton // identifier of radio button to select

);

/////////////CheckDlgButton(hwnd,IDC\_RADIO\_DIRECT,bDirection);

DWORD fdwPrClass = GetPriorityClass(GetCurrentProcess());

TCHAR szPrClass[100];

switch (fdwPrClass)

{

case IDLE\_PRIORITY\_CLASS:

wsprintf(szPrClass, TEXT("%s"), TEXT("IDLE\_PRIORITY\_CLASS"));

break;

// case BELOW\_NORMAL\_PRIORITY\_CLASS:

// wsprintf(szPrClass,TEXT("%s"),TEXT("BELOW\_NORMAL\_PRIORITY\_CLASS"));

// break;

case NORMAL\_PRIORITY\_CLASS:

wsprintf(szPrClass, TEXT("%s"), TEXT("NORMAL\_PRIORITY\_CLASS"));

break;

// case ABOVE\_NORMAL\_PRIORITY\_CLASS:

// wsprintf(szPrClass,TEXT("%s"),TEXT("ABOVE\_NORMAL\_PRIORITY\_CLASS"));

// break;

case HIGH\_PRIORITY\_CLASS:

wsprintf(szPrClass, TEXT("%s"), TEXT("HIGH\_PRIORITY\_CLASS"));

break;

case REALTIME\_PRIORITY\_CLASS:

wsprintf(szPrClass, TEXT("%s"), TEXT("REALTIME\_PRIORITY\_CLASS"));

break;

default:

break;

}

SendDlgItemMessage(hDlg, IDC\_STATIC\_PR\_CL, WM\_SETTEXT, 0, (LPARAM)szPrClass);//EM\_REPLACESEL

return TRUE;//handled

}

skype: kondratjeva314

фамилия\_номерГруппы\_(ле/лаб)Номер\_дата

Кондратьва\_1\_ле3\_22сент

// Thr3.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "resource.h"

//#include "windowsx.h"

#define MAX\_LOADSTRING 100

//Look into Windowsx.h for System definitions without the prefix R

#define RHANDLE\_MSG(hwnd, message, fn) \

case (message): return RHANDLE\_##message((hwnd), (wParam), (lParam), (fn))

#define RHANDLE\_WM\_COMMAND(hwnd, wParam, lParam, fn) \

((fn)((hwnd), (int)(LOWORD(wParam)), (HWND)(lParam), (UINT)HIWORD(wParam)), -1L)

#define RHANDLE\_WM\_INITDIALOG(hwnd, wParam, lParam, fn) \

(LRESULT)(DWORD)(UINT)(BOOL)(fn)((hwnd), (HWND)(wParam), lParam)

#define RFORWARD\_WM\_COMMAND(hwnd, id, hwndCtl, codeNotify, fn) \

(LONG)(fn)((hwnd), WM\_COMMAND, MAKEWPARAM((UINT)(id),(UINT)(codeNotify)), (LPARAM)(HWND)(hwndCtl))

//Globals to control threads

TCHAR szMutexName[]="$MutexFor3Threads$";

HANDLE hMutex;

HANDLE hThreadE[2];

BOOL fTerminateE;

BOOL fTerminateR;

bool bSuspendE,

bSuspendR;

unsigned thridE; //thread identifier

DWORD nWM\_PAINT=0; //WM\_PAINT count

// Global Variables:

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // The title bar text

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

//\_stdcall for CALLBACK (see Windef.h)

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK ChangeThreadPriority\_Box(HWND, UINT, WPARAM, LPARAM);

BOOL ChThPr\_OnInitDialog(HWND hDlg, HWND hwndFocus, LPARAM lParam);

LONG ChThPr\_OnCommand (HWND hDlg,int id,HWND hwndCtl,UINT codeNotify);

LONG WndProc\_OnCreate(HWND hWnd,LPCREATESTRUCT lpCreateStruct);

LONG WndProc\_OnDestroy(HWND hWnd);

void WndProc\_OnPaint(HWND hWnd);

LONG WndProc\_OnCommand (HWND hWnd,int id,HWND hwndCtl,UINT codeNotify);

void WndProc\_OnClose(HWND hWnd);

unsigned int \_\_stdcall PaintEllipse(void \*hWnd);

unsigned int \_\_stdcall PaintRectangle(void \*hWnd);

void SuspendEllipse(HMENU hMenu,bool \*bSuspend);

void SuspendRectangle(HMENU hMenu,bool \*bSuspend);

void TerminateEllipse(HWND hWnd,HMENU hMenu,BOOL \*fTerminateE);

//\_stdcall for APIENTRY (see Windef.h), used for new and last versions

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{//Create an object Mutex (named)

//As it is named it is possible to verify if there is only one application instance too!

if(!(hMutex=CreateMutex(NULL,FALSE,szMutexName))){//FALSE sets it In SIGNALED STATE!

//NO OWNERSHIP

MessageBox(NULL,"CreateMutex Error","Four Threads",

MB\_OK|MB\_ICONEXCLAMATION);

return 01;

}

if(GetLastError()==ERROR\_ALREADY\_EXISTS){

MessageBox(NULL,"Mutex already started","Four Threads",

MB\_OK|MB\_ICONEXCLAMATION);

return 01;

}

TCHAR msgHINST[250];

wsprintf(msgHINST,TEXT("HINSTANCE is %ld"),hInstance);

MessageBox(NULL,msgHINST,"Four Threads",

MB\_OK);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_THR3, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC\_THR3);

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

// COMMENTS:

//

// This function and its usage is only necessary if you want this code

// to be compatible with Win32 systems prior to the 'RegisterClassEx'

// function that was added to Windows 95. It is important to call this function

// so that the application will get 'well formed' small icons associated

// with it.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = (WNDPROC)WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI\_THR3);

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = (LPCSTR)IDC\_THR3;

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI\_SMALL);

return RegisterClassEx(&wcex);

}

//

// FUNCTION: InitInstance(HANDLE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance; // Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, unsigned, WORD, LONG)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

// int wmId, wmEvent;

switch (message)

{

HANDLE\_MSG(hWnd,WM\_COMMAND,WndProc\_OnCommand);//!

HANDLE\_MSG(hWnd,WM\_PAINT, WndProc\_OnPaint);//!

HANDLE\_MSG(hWnd,WM\_CREATE,WndProc\_OnCreate);//!

HANDLE\_MSG(hWnd,WM\_DESTROY,WndProc\_OnDestroy);//!

HANDLE\_MSG(hWnd,WM\_CLOSE,WndProc\_OnClose);//!

default:return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;//BY CHANCE TO BE USED

}

LONG WndProc\_OnCommand (HWND hWnd,int id,HWND hwndCtl,UINT codeNotify)

{

// Parse the menu selections:

switch (id)

{

case IDM\_SUSE:

SuspendEllipse(GetMenu(hWnd),&bSuspendE);

return 0L;

case IDM\_SUSR:

SuspendRectangle(GetMenu(hWnd),&bSuspendR);

return 0L;

case IDM\_TERME:

TerminateEllipse(hWnd,GetMenu(hWnd), &fTerminateE);

return 0L;

case IDM\_CH\_THR\_PR\_ELL:

DialogBox(hInst, (LPCTSTR)IDD\_DIALOG\_PRIORITY, hWnd,

(DLGPROC)ChangeThreadPriority\_Box);

return 0L;

case IDM\_ABOUT:

DialogBox(hInst, (LPCTSTR)IDD\_ABOUTBOX, hWnd, (DLGPROC)About);

return 0L;// break;

case IDM\_EXIT:

DestroyWindow(hWnd); //PostQuitMessage(0);

return 0L;// break;

default:break;

// return DefWindowProc(hWnd, message, wParam, lParam);

}

return RFORWARD\_WM\_COMMAND(hWnd, id, hwndCtl, codeNotify,DefWindowProc);

}

void WndProc\_OnPaint(HWND hWnd)

{

PAINTSTRUCT ps;

HDC hdc;

TCHAR szHello[MAX\_LOADSTRING];

TCHAR sznWM\_PAINT[50];

DWORD dwRetCode;

LoadString(hInst, IDS\_HELLO, szHello, MAX\_LOADSTRING);

dwRetCode=WaitForSingleObject(hMutex,INFINITE);

if(dwRetCode==WAIT\_OBJECT\_0)

{

hdc = BeginPaint(hWnd, &ps);

// TODO: Add any drawing code here...

RECT rt;

GetClientRect(hWnd, &rt);

DrawText(hdc, szHello, strlen(szHello), &rt, DT\_CENTER);

wsprintf(sznWM\_PAINT,TEXT("nWM\_PAINT=%d"),nWM\_PAINT);

rt.top = rt.top + 50;

DrawText(hdc, sznWM\_PAINT, strlen(sznWM\_PAINT), &rt, DT\_CENTER);

EndPaint(hWnd, &ps);

nWM\_PAINT++;

ReleaseMutex(hMutex);

}

// return 0;

}

// This program requires the multithreaded library. For example,

// compile with the following command line:

// CL /MT /D "\_X86\_" BEGTHRD.C

LONG WndProc\_OnCreate(HWND hWnd,LPCREATESTRUCT lpCreateStruct)

{

fTerminateE=FALSE;

fTerminateR=FALSE;

//Thread PaintEllipse starting not suspended!

bSuspendE = false;

//Thread PaintRectangle starting not suspended!

bSuspendR = false;

unsigned ususpend=0;

hThreadE[0]=(HANDLE)\_beginthreadex(NULL,//must be FOR W95 ,SA

0,//stack size committed

PaintEllipse,

(void \*)hWnd, //\*arglist

ususpend ,//0

&thridE

);

if(!hThreadE[0]){MessageBox(NULL,"Thread start Error",

"PaintEllipse Thread",

MB\_OK|MB\_ICONEXCLAMATION);

return FALSE;

}

hThreadE[1]=(HANDLE)\_beginthreadex(NULL,//must be FOR W95

0,//stack size

PaintRectangle,

(void \*)hWnd,

ususpend ,//0

&thridE

);

if(!hThreadE[1]){MessageBox(NULL,"Thread start Error",

"PaintRectangle Thread",

MB\_OK|MB\_ICONEXCLAMATION);

return FALSE;

}

return TRUE;//BOOL

}

LONG WndProc\_OnDestroy(HWND hWnd)

{

if(bSuspendE)

ResumeThread(hThreadE[0]);

if(bSuspendR)

ResumeThread(hThreadE[1]);

fTerminateE=TRUE;

fTerminateR=TRUE;

TCHAR szRetRes [100];

DWORD dwRet;

switch(dwRet=WaitForMultipleObjects(3???,hThreadE,TRUE,INFINITE))

провести эксперимент: завершить из меню 1-2 дочерних потока и завершить приложение

{

case WAIT\_ABANDONED\_0 :

wsprintf(szRetRes,TEXT("WAIT\_ABANDONED\_0=%d dwRet=%d"),WAIT\_ABANDONED\_0,dwRet);

MessageBox(NULL,szRetRes,"WaitForMultipleObjects",MB\_OK|MB\_ICONEXCLAMATION);

break;

case WAIT\_FAILED:

break;

case WAIT\_OBJECT\_0:

wsprintf(szRetRes,TEXT("WAIT\_OBJECT\_0=%d dwRet=%d"),WAIT\_OBJECT\_0,dwRet);

MessageBox(NULL,szRetRes,"WaitForMultipleObjects",MB\_OK|MB\_ICONEXCLAMATION);

break;

}

if(!CloseHandle(hThreadE[0]))

{MessageBox(NULL,"CloseHandle failed",//NULL , no hWnd(the window is destroied)

"PaintEllipse Thread",MB\_OK|MB\_ICONEXCLAMATION);

};

if(!CloseHandle(hThreadE[1]))

{MessageBox(NULL,"CloseHandle failed",

"PaintEllipse Thread", MB\_OK|MB\_ICONEXCLAMATION);

};

CloseHandle(hMutex);

PostQuitMessage(0);

return (0);

}

void WndProc\_OnClose(HWND hWnd)

{

/\*

LONG InterlockedExchangeAdd (

PLONG Addend, // pointer to the addend

LONG Increment // increment value

);

\*/

DWORD dwn=InterlockedExchangeAdd((PLONG)&nWM\_PAINT,0);

TCHAR sznWM[50];

sprintf(sznWM,"nWM\_PAINT=%11d\n", dwn);// no nWM\_PAINT

int r=MessageBox(hWnd,sznWM,"Primary Thread WM\_PAINT count",

MB\_OKCANCEL|MB\_ICONEXCLAMATION);

//return CLFORWARD\_WM\_CLOSE(hWnd, DefWindowProc);

DestroyWindow(hWnd); // or

//FORWARD\_WM\_CLOSE(hWnd, DefWindowProc);

}

// Message handler for about box.

LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return TRUE;

}

break;

}

return FALSE; //SetDlgMsgResult-???ABSENT

}

unsigned int \_\_stdcall PaintEllipse(void \*hWnd){

HDC hDC;

RECT rect;

LONG xLeft,xRight,yTop,yBottom;

short nRed,nGreen,nBlue;

HBRUSH hBrush,hOldBrush;

DWORD dwRetCode;

srand((unsigned int)hWnd +100);

while(!fTerminateE){ // Is not it to be continued?

// To be continued!

switch(dwRetCode=WaitForSingleObject(hMutex,INFINITE)){

case WAIT\_ABANDONED:break;

case WAIT\_FAILED:break;

case WAIT\_OBJECT\_0:

{

hDC=GetDC((HWND)hWnd);

nRed=rand()%255; nGreen=rand()%255; nBlue=rand()%255;

GetWindowRect((HWND)hWnd,&rect);

xLeft=rand()%(rect.left +1);

xRight=rand()%(rect.right +1);

yTop=rand()%(rect.top +1);

yBottom=rand()%(rect.bottom +1);

hBrush=CreateSolidBrush(RGB(nRed,nGreen,nBlue));

hOldBrush=(HBRUSH)SelectObject(hDC,hBrush);

Ellipse(hDC,min(xLeft,xRight),min(yTop,yBottom),

max(xLeft,xRight),max(yTop,yBottom));

SelectObject(hDC,hOldBrush);

DeleteObject(hBrush);

ReleaseDC((HWND)hWnd,hDC);

ReleaseMutex(hMutex);

break;

};

}

Sleep(100);

// InvalidateRect((HWND)hWnd,NULL,TRUE); //NULL- the whole client region

//TRUE - the background is erased when the BeginPaint function is called.

Sleep(100);

}//End of while

//InvalidateRect((HWND)hWnd,NULL,TRUE);

return 0;

}//End of PaintEllipse

unsigned int \_\_stdcall PaintRectangle(void \*hWnd){

////////void PaintRectangle(void \*hWnd){

HDC hDC;

RECT rect;

LONG xLeft,xRight,yTop,yBottom;

short nRed,nGreen,nBlue;

HBRUSH hBrush,hOldBrush;

DWORD dwRetCode;

srand((unsigned int)hWnd);

while(!fTerminateR){

// Is not it to be continued?

// To be continued!

switch(dwRetCode=WaitForSingleObject(hMutex,INFINITE)){

case WAIT\_ABANDONED:break;

/\*

The specified object is a mutex object that was not released by the thread

that owned the mutex object before the owning thread terminated.

Ownership of the mutex object is granted to the calling thread,

and the mutex is set to nonsignaled.

\*/

case WAIT\_FAILED:

// DWORD GetLastError(VOID)

break;

case WAIT\_TIMEOUT:break; //by INFINITE never occurs

case WAIT\_OBJECT\_0:

{

hDC=GetDC((HWND)hWnd);

nRed=rand()%255; nGreen=rand()%255; nBlue=rand()%255;

GetWindowRect((HWND)hWnd,&rect);

xLeft=rand()%(rect.left +1);

xRight=rand()%(rect.right +1);

yTop=rand()%(rect.top +1);

yBottom=rand()%(rect.bottom +1);

hBrush=CreateSolidBrush(RGB(nRed,nGreen,nBlue));

hOldBrush=(HBRUSH)SelectObject(hDC,hBrush);

Rectangle(hDC,min(xLeft,xRight),min(yTop,yBottom),

max(xLeft,xRight),max(yTop,yBottom));

SelectObject(hDC,hOldBrush);

DeleteObject(hBrush);

ReleaseDC((HWND)hWnd,hDC);

//Sleep(10000);

ReleaseMutex(hMutex);

break;

};

}

Sleep(100);

InvalidateRect((HWND)hWnd,NULL,TRUE);

Sleep(100);

}//End of while

return 0;

}//End of PaintRectangle

//Wrong solution whithout using hMutex???

void SuspendEllipse(HMENU hMenu,bool \*bSuspend)

{

TCHAR message[260];

if(!\*bSuspend){

if(0xFFFFFFFF==SuspendThread(hThreadE[0]))//or -1 (==0xFFFFFFFF)

{

wsprintf(message,TEXT("SuspendThread Error %ld"),GetLastError());

MessageBox(NULL,message,"PaintEllipse Thread",MB\_OK|MB\_ICONEXCLAMATION);

return;

}

\*bSuspend = true;

//Check

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu,IDM\_SUSE,MF\_CHECKED);

}

else

{

if(0xFFFFFFFF==ResumeThread(hThreadE[0]))

{

wsprintf(message,TEXT("ResumeThread Error %ld"),GetLastError());

MessageBox(NULL,message,"PaintEllipse Thread",MB\_OK|MB\_ICONEXCLAMATION);

return;

}

\*bSuspend = false;

//Uncheck

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu,IDM\_SUSE,MF\_UNCHECKED);

}

return;

}

//Right solution

void SuspendRectangle(HMENU hMenu,bool \*bSuspend)

{

TCHAR message[260];

DWORD dwRetCode;

switch(dwRetCode=WaitForSingleObject(hMutex,INFINITE))

{

case WAIT\_ABANDONED:break;

case WAIT\_FAILED :break;

case WAIT\_OBJECT\_0:

if(!\*bSuspend){

if(0xFFFFFFFF==SuspendThread(hThreadE[1]))

{

wsprintf(message,TEXT("SuspendThread Error %ld"),GetLastError());

MessageBox(NULL,message,"PaintRectangle Thread",MB\_OK|MB\_ICONEXCLAMATION);

ReleaseMutex(hMutex);

return;

}

\*bSuspend = true;

//Check

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu,IDM\_SUSR,MF\_CHECKED);

}

else

{

if(0xFFFFFFFF==ResumeThread(hThreadE[1]))

{

wsprintf(message,TEXT("ResumeThread Error %ld"),GetLastError());

MessageBox(NULL,message,"PaintRectangle Thread",MB\_OK|MB\_ICONEXCLAMATION);

ReleaseMutex(hMutex);

return;

}

\*bSuspend = false;

//Uncheck

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu,IDM\_SUSR,MF\_UNCHECKED);

}

ReleaseMutex(hMutex);

return;

break;

default: //is never reached!!

break;

}//switch

}//SuspendRectangle

void TerminateEllipse(HWND hWnd,HMENU hMenu,BOOL \*fTerminateE)

{

if(!\*fTerminateE)

{

//Check

//Sets the check-mark attribute to the unchecked state.

CheckMenuItem(hMenu,IDM\_TERME,MF\_CHECKED);

\*fTerminateE=true;

if(bSuspendE)

ResumeThread(hThreadE[0]);

WaitForSingleObject(hThreadE[0],INFINITE);//Block the primary thread if PaintEllipse

// is in the suspended state

if(!CloseHandle(hThreadE[0]))

{MessageBox(NULL,"CloseHandle failed",

"PaintEllipse Thread",

MB\_OK|MB\_ICONEXCLAMATION);

};

}

else

{

//Uncheck

//Sets the check-mark attribute to the checked state.

CheckMenuItem(hMenu,IDM\_TERME,MF\_UNCHECKED);

\*fTerminateE=false;

unsigned ususpend= (bSuspendE)?CREATE\_SUSPENDED:0;

hThreadE[0]=(HANDLE)\_beginthreadex(NULL,//must be FOR W95

0,//stack size

PaintEllipse,

(void \*) hWnd,

ususpend ,//0

&thridE

);

if(!hThreadE[0])MessageBox(NULL,"Thread start Error",

"PaintEllipse Thread",

MB\_OK|MB\_ICONEXCLAMATION);

}

return;

}

//Dialog Box Function

//\_stdcall for CALLBACK (see Windef.h)

LRESULT CALLBACK ChangeThreadPriority\_Box(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

RHANDLE\_MSG(hDlg, WM\_INITDIALOG, ChThPr\_OnInitDialog);

RHANDLE\_MSG(hDlg, WM\_COMMAND, ChThPr\_OnCommand);

//return true; for all handled messages

default:

return FALSE;//not handled messages

}

}//ChangeThreadPriority\_Box

LONG ChThPr\_OnCommand (HWND hDlg,int id,HWND hwndCtl,UINT codeNotify)

{

// Parse the Change Thread Priority Box commands:

switch (id)

{

case IDOK:

{

int nPriorityEll=THREAD\_PRIORITY\_NORMAL;

int nIDCheckButton=IDC\_RADIO\_NORMAL,i;

for( i=0; i<7; i++)

if(BST\_CHECKED==IsDlgButtonChecked(hDlg,IDC\_RADIO\_TMCRITICAL+i))

{

nIDCheckButton=IDC\_RADIO\_TMCRITICAL+i;

break;

}

switch(nIDCheckButton)

{

case IDC\_RADIO\_TMCRITICAL:

nPriorityEll=THREAD\_PRIORITY\_TIME\_CRITICAL;

break;

case IDC\_RADIO\_HGHST:

nPriorityEll=THREAD\_PRIORITY\_HIGHEST;

break;

case IDC\_RADIO\_ABOVE:

nPriorityEll=THREAD\_PRIORITY\_ABOVE\_NORMAL;

break;

case IDC\_RADIO\_NORMAL:

nPriorityEll=THREAD\_PRIORITY\_NORMAL;

break;

case IDC\_RADIO\_BELOW:

nPriorityEll=THREAD\_PRIORITY\_BELOW\_NORMAL;

break;

case IDC\_RADIO\_LOWEST:

nPriorityEll=THREAD\_PRIORITY\_LOWEST;

break;

case IDC\_RADIO\_IDLE:

nPriorityEll=THREAD\_PRIORITY\_NORMAL;

break;

default:

break;

}

SetThreadPriority(hThreadE[0],nPriorityEll);

}

case IDCANCEL:

EndDialog(hDlg, id);

return TRUE;//handled

default:

return TRUE;

}//return TRUE for all handled commands,else ones are not here

}

BOOL ChThPr\_OnInitDialog(HWND hDlg, HWND hwndFocus, LPARAM lParam)

{

//Set Current Priority Class and Thread Priority

int nPriorityEll=GetThreadPriority(hThreadE[0]);

int nIDCheckButton;

switch(nPriorityEll)

{

case THREAD\_PRIORITY\_TIME\_CRITICAL:

nIDCheckButton=IDC\_RADIO\_TMCRITICAL;

break;

case THREAD\_PRIORITY\_HIGHEST:

nIDCheckButton=IDC\_RADIO\_HGHST;

break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL:

nIDCheckButton=IDC\_RADIO\_ABOVE ;

break;

case THREAD\_PRIORITY\_NORMAL:

nIDCheckButton=IDC\_RADIO\_NORMAL;

break;

case THREAD\_PRIORITY\_BELOW\_NORMAL:

nIDCheckButton=IDC\_RADIO\_BELOW;

break;

case THREAD\_PRIORITY\_LOWEST:

nIDCheckButton=IDC\_RADIO\_LOWEST;

break;

case THREAD\_PRIORITY\_IDLE:

nIDCheckButton=IDC\_RADIO\_IDLE;

break;

default:

break;

}

CheckRadioButton( hDlg, // handle to dialog box

IDC\_RADIO\_TMCRITICAL, // identifier of first radio button in group

IDC\_RADIO\_IDLE, // identifier of last radio button in group

nIDCheckButton // identifier of radio button to select

);

/////////////CheckDlgButton(hwnd,IDC\_RADIO\_DIRECT,bDirection);

DWORD fdwPrClass=GetPriorityClass(GetCurrentProcess());

TCHAR szPrClass[100];

switch(fdwPrClass)

{

case IDLE\_PRIORITY\_CLASS:

wsprintf(szPrClass,TEXT("%s"),TEXT("IDLE\_PRIORITY\_CLASS"));

break;

// case BELOW\_NORMAL\_PRIORITY\_CLASS:

// wsprintf(szPrClass,TEXT("%s"),TEXT("BELOW\_NORMAL\_PRIORITY\_CLASS"));

// break;

case NORMAL\_PRIORITY\_CLASS:

wsprintf(szPrClass,TEXT("%s"),TEXT("NORMAL\_PRIORITY\_CLASS"));

break;

// case ABOVE\_NORMAL\_PRIORITY\_CLASS:

// wsprintf(szPrClass,TEXT("%s"),TEXT("ABOVE\_NORMAL\_PRIORITY\_CLASS"));

// break;

case HIGH\_PRIORITY\_CLASS:

wsprintf(szPrClass,TEXT("%s"),TEXT("HIGH\_PRIORITY\_CLASS"));

break;

case REALTIME\_PRIORITY\_CLASS:

wsprintf(szPrClass,TEXT("%s"),TEXT("REALTIME\_PRIORITY\_CLASS"));

break;

default:

break;

}

SendDlgItemMessage(hDlg,IDC\_STATIC\_PR\_CL,WM\_SETTEXT ,0,(LPARAM)szPrClass);//EM\_REPLACESEL

return TRUE;//handled

}