МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №5

Тема: Вторичные потоки. Синхронизация потоков

Курс: Системное программирование

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Филипишин Д.А.

Донецк – 2021

**Задание**

1. Разработать многопоточное приложение Windows. Первый вторичный поток создаётся в приостановленном состоянии при обработке события WM\_CREATE главного окна. С помощью пунктов меню пользователь может возобновить или приостановить работу этого потока. Второй поток создаётся при выборе соответствующего пункта меню. Ему необходимо передать параметры. Результаты своей работы этот поток выводит в клиентскую область окна. Потоковые функции вторичных потоков должны выполнять действия, описанные в варианте заданий.

2. Разработать новое приложение, в котором создать вторичный поток, выполняющий те же действия, что и первый вторичный поток из задания 1, только с другим объектом (см. таблицу). Внести следующие изменения в приложение из задания 1:

– добавить пункт в меню, по которому вызывается новое приложение;

– добавить работу с объектом синхронизации, который будет синхронизировать работу «Первого потока» из родительского процесса и вторичного потока из дочернего процесса таким образом, чтобы операции вывода в окно выполнялись сначала одной потоковой функцией определённое количество раз, потом другой и т.д.

**ЗАДАНИЕ 1**

5. Первый поток: перемещает строку случайным образом в пределах клиентской области окна с интервалом 0,3 сек.

Второй поток: рисует эллипс; координаты центра, длины полуосей и дескриптор окна передать входной функции вторичного потока в качестве параметра.

**ЗАДАНИЕ 2**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Перемещаемый объект для вторичного потока дочернего процесса | Коли-чество повторов | Объект синхронизации |
| 5 | Прямоугольник с закруглёнными углами (RoundRect) | 5 | Ожидаемый таймер (наследуемый) |

**Листинг главного приложения**

// SP\_lab\_5.cpp : Определяет точку входа для приложения.

//

#include "framework.h"

#include "SP\_lab\_5.h"

#include "PaintGuard.hpp"

#include "utility.hpp"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

HANDLE htimer;

LARGE\_INTEGER li;

struct WindowSize {

int width;

int height;

};

WindowSize GetWindowSize(HWND window\_handle) {

RECT window\_rect;

GetClientRect(window\_handle, &window\_rect);

WindowSize result;

result.width = window\_rect.right - window\_rect.left;

result.height = window\_rect.bottom - window\_rect.top;

return result;

}

WindowSize window\_size;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

DWORD WINAPI ThreadFunc1(PVOID pvParam);

DWORD WINAPI ThreadFunc2(PVOID pvParam);

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

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

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

LoadStringW(hInstance, IDC\_SPLAB5, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_SPLAB5));

MSG msg;

// Цикл основного сообщения:

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_SPLAB5));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_SPLAB5);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

(GetSystemMetrics(SM\_CXSCREEN) - 500) / 2,

(GetSystemMetrics(SM\_CYSCREEN) - 500) / 2,

500, 500, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

typedef struct {

HWND hwnd;

int centerx;

int centery;

int radiusa;

int radiusb;

} thread\_data;

thread\_data td\_ellipse;

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

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

{

srand(1);

PaintGuard pg(hWnd);

HDC hdc = pg.GetHDC();

static HANDLE thread1, thread2;

HWND label;

static wchar\_t str[100], cmdline[100];

static PROCESS\_INFORMATION pi;

STARTUPINFO si;

/////////////////////////////////////Нужно сделать SetWaitableTimer + Сделать запуск приложения (дочернего)

switch (message)

{

case WM\_CREATE: {

label = CreateWindow(L"static", L"Cтрока", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | SS\_CENTER, 10, 10, 60, 20, hWnd, NULL, NULL, NULL);

thread1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadFunc1, label, CREATE\_SUSPENDED, NULL);

li.QuadPart = -10000000LL;

SECURITY\_ATTRIBUTES sa;

sa.bInheritHandle = true;

sa.lpSecurityDescriptor = {};

LPSECURITY\_ATTRIBUTES psa = &sa;

htimer = CreateWaitableTimer(psa, FALSE, L"Global\\MyTimer");

SetWaitableTimer(htimer, &li, 1.5 \* 1000, NULL, NULL, FALSE);

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_START\_FIRST:

if (thread1)

window\_size = GetWindowSize(hWnd);

ResumeThread(thread1);

break;

case IDM\_STOP\_FIRST:

if (thread1)

SuspendThread(thread1);

break;

case IDM\_START\_SECOND:

if (DialogBox(NULL, (LPCTSTR)IDD\_ELLIPSE\_DIALOG, hWnd,

(DLGPROC)EllipseDialogFunc) == IDOK) {

InvalidateRect(hWnd, NULL, TRUE);

}

td\_ellipse.hwnd = hWnd;

thread2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadFunc2, &td\_ellipse, NULL, NULL);

break;

case IDM\_START\_PROCESS: {

wsprintf(str, L"%x", htimer);

std::wstring appName(L"C:/Users/Arsenii/source/repos/SP\_lab\_5\_pt\_2/Debug/SP\_lab\_5\_pt\_2.exe");

LPWSTR lpwAppName = &appName[0];

wcscpy\_s(cmdline, L"C:/Users/Arsenii/source/repos/SP\_lab\_5\_pt\_2/Debug/SP\_lab\_5\_pt\_2.exe");

wcscat\_s(cmdline, str);

ZeroMemory(&pi, sizeof(pi));

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

if (!CreateProcess(lpwAppName, cmdline, NULL, NULL, true, 0, NULL, NULL, &si, &pi)) {

MessageBox(hWnd, L"Cannot create process", L"Error",

MB\_OK | MB\_ICONSTOP | MB\_SETFOREGROUND | MB\_TOPMOST);

}

break;

}

default:

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

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

//Arc(hdc, 100, 100, 500, 200, 0, 0, 0, 0);

EndPaint(hWnd, &ps);

}

break;

case WM\_SIZE:

UpdateWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

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

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

DWORD WINAPI ThreadFunc1(PVOID pvParam) {

HWND label = (HWND)pvParam;

while (true) {

WaitForSingleObject(htimer, INFINITE);

for (int a = 0; a < 5; ++a) {

int randPosX = rand() % window\_size.width;

int randPosX2 = min(randPosX, window\_size.width - 60);

int randPosY = rand() % window\_size.height;

int randPosY2 = min(randPosY, window\_size.height - 20);

MoveWindow(label, randPosX2, randPosY2, 60, 20, TRUE);

Sleep(300);

}

}

DWORD dwResult = 0;

return dwResult;

}

DWORD WINAPI ThreadFunc2(PVOID pvParam) {

thread\_data \* td\_ellipse = (thread\_data\*)pvParam;

HWND hwnd = td\_ellipse->hwnd;

int centerx = td\_ellipse->centerx;

int centery = td\_ellipse->centery;

int radiusa = td\_ellipse->radiusa;

int radiusb = td\_ellipse->radiusb;

int lhx = centerx - radiusa;

int lhy = centery - radiusb;

int rbx = centerx + radiusa;

int rby = centery + radiusb;

HDC hdc = GetDC(hwnd);

HPEN hpen = CreatePen(PS\_DASHDOTDOT, 2, NULL);

SelectObject(hdc, hpen);

//Rectangle(hdc, 1, 1, 100, 100);

Ellipse(hdc, lhx, lhy, rbx, rby);

//Arc(hdc, lhx, lhy, rbx, rby, 0, 0, 0, 0);

//Sleep(300);

DeleteObject(hpen);

DWORD dwResult = 0;

return dwResult;

}

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

auto GetInt = [=](const int idc) {

String buffer(100, 0);

GetDlgItemText(hDlg, idc, &buffer[0], buffer.size());

return std::stoi(buffer);

};

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK) {

td\_ellipse.centerx = GetInt(IDC\_CENTER\_X);

td\_ellipse.centery = GetInt(IDC\_CENTER\_Y);

td\_ellipse.radiusa = GetInt(IDC\_RADIUS\_A);

td\_ellipse.radiusb = GetInt(IDC\_RADIUS\_B);

EndDialog(hDlg, IDOK);

return TRUE;

}

if (LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, IDCANCEL);

return TRUE;

}

}

return 0;

}

**Листинг дочернего приложения**

#include "framework.h"

#include "SP\_lab\_5\_pt\_2.h"

#include "utility.hpp"

#include "PaintGuard.hpp"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

struct WindowSize {

int width;

int height;

};

WindowSize GetWindowSize(HWND window\_handle) {

RECT window\_rect;

GetClientRect(window\_handle, &window\_rect);

WindowSize result;

result.width = window\_rect.right - window\_rect.left;

result.height = window\_rect.bottom - window\_rect.top;

return result;

}

WindowSize window\_size;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

DWORD WINAPI ThreadFunc1(PVOID pvParam);

DWORD WINAPI ThreadFunc2(PVOID pvParam);

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

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

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

LoadStringW(hInstance, IDC\_SPLAB5PT2, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_SPLAB5PT2));

MSG msg;

// Цикл основного сообщения:

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDC\_SPLAB5PT2));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_SPLAB5PT2);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

(GetSystemMetrics(SM\_CXSCREEN) - 500) / 2,

(GetSystemMetrics(SM\_CYSCREEN) - 500) / 2,

500, 500, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

HANDLE htimer;

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

{

srand(1);

PaintGuard pg(hWnd);

HDC hdc = pg.GetHDC();

static HANDLE thread1, thread2;

//HWND label;

switch (message)

{

case WM\_CREATE:

//label = CreateWindow(L"static", L"Cтрока", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | SS\_CENTER, 10, 10, 60, 20, hWnd, NULL, NULL, NULL);

thread1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadFunc1, hWnd, CREATE\_SUSPENDED, NULL);

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_START\_FIRST:

if (thread1)

window\_size = GetWindowSize(hWnd);

htimer = OpenWaitableTimer(TIMER\_ALL\_ACCESS, true, L"Global\\MyTimer");

if (!htimer) {

MessageBox(hWnd, L"Cannot open timer", L"Error",

MB\_OK | MB\_ICONSTOP | MB\_SETFOREGROUND | MB\_TOPMOST);

}

ResumeThread(thread1);

break;

case IDM\_STOP\_FIRST:

if (thread1)

SuspendThread(thread1);

break;

default:

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

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

//Arc(hdc, 100, 100, 500, 200, 0, 0, 0, 0);

EndPaint(hWnd, &ps);

}

break;

case WM\_SIZE:

UpdateWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

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

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

DWORD WINAPI ThreadFunc1(PVOID pvParam) {

HWND hwnd = (HWND)pvParam;

HDC hdc = GetDC(hwnd);

int i, j = 0;

int rectSize = 40;

while (true) {

WaitForSingleObject(htimer, INFINITE);

for (int a = 0; a < 5; ++a) {

BitBlt(hdc, 0, 0, 1000, 1000, 0, 0, 0, WHITENESS);

int randPosX = rand() % window\_size.width;

i = min(randPosX, window\_size.width - rectSize);

int randPosY = rand() % window\_size.height;

j = min(randPosY, window\_size.height - rectSize);

RoundRect(hdc, i, j, i + rectSize, j + rectSize, 20, 40);

Sleep(300);

}

}

DWORD dwResult = 0;

return dwResult;

}

**Демонстрация работы приложения приведена в видео SP\_lab\_5.mp4**