Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ ПО ПРОГРАММИРОВАНИЮ В СРЕДЕ WINDOWS

Студент **Ныркov Илья Алексеевич**
фамилия, имя, отчество

Группа **ИУ5-42Б**

Студент _____ **Ныркov И.А.**
подпись, дата *фамилия, и.о.*

Руководитель _____ **Аксёнова М.В.**
подпись, дата *фамилия, и.о.*

2020 г.

Лабораторная работа №1 (Графика, таймеры, кнопки, тексты и т.п.)

Задание (17 Вариант): Три вложенных друг в друга квадрата с размерами сторон 50, 100 и 150 пикселов, соприкасающиеся левыми верхними углами. По щелчку левой кнопки мыши цвет i-го квадрата становиться цветом (i+1)-го. Щелчок мыши (правая клавиша) - Закрыть окно через 15 сек. **Доп задание:** изменение цветов по таймеру вместо нажатия на ЛКМ, отключение анимации по нажатию ПКМ.

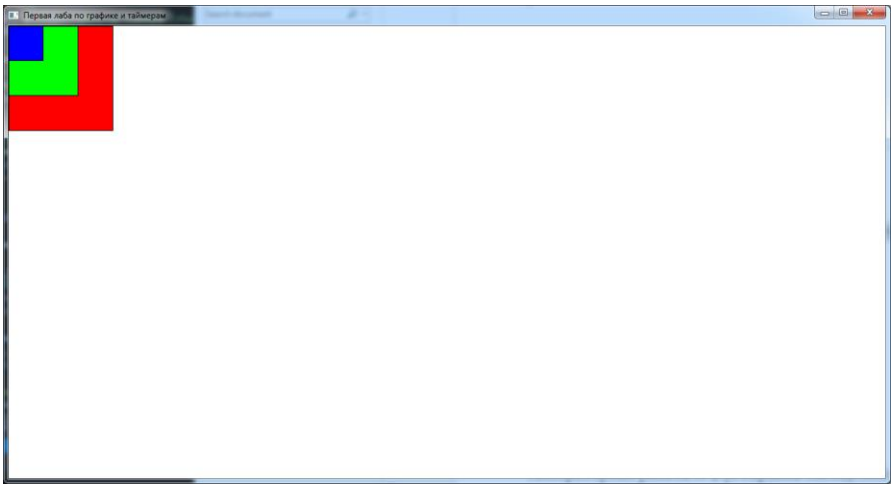
Листинг программы:

```
#include <Windows.h>
#include <tchar.h>
#include <ctime>
#include <vector>
#define TIMER1 1 //идентификатор таймера (обычный инт)
#define TIMER2 2

void PrintSquare(int size, std::vector<int> colors, HDC& hdc, HBRUSH& Hbrush) {
	Hbrush = CreateSolidBrush(RGB(colors[0], colors[1], colors[2]));
	SelectObject(hdc, Hbrush);
	Rectangle(hdc, 0, 0, size, size);
}

//сдвиг массива цветов вправо для последовательной смены цветов
void MoveElems(std::vector<std::vector<int>>& colors) {
	std::vector<std::vector<int>> tmp = { colors[2], colors[0], colors[1] };
	colors = tmp;
}

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

TCHAR WinName[] = _T("MainFrame");

int WINAPI _tWinMain(HINSTANCE This,		// Дескриптор текущего приложения
	HINSTANCE Prev,		// В современных системах всегда 0
	LPTSTR cmd,		// Командная строка
	int mode)		// Режим отображения окна
{
	HWND hWnd;		// Дескриптор главного окна программы
	MSG msg;		// Структура для хранения сообщения
	WNDCLASS wc;  // Класс окна

	// Определение класса окна
	wc.hInstance = This;
	wc.lpszClassName = WinName;				// Имя класса окна
	wc.lpfnWndProc = WndProc;				// Функция окна
	wc.style = CS_HREDRAW | CS_VREDRAW;			// Стиль окна
	wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);		// Стандартная иконка
	wc.hCursor = LoadCursor(NULL, IDC_ARROW);		// Стандартный курсор
```

```
	wc.lpszMenuName = NULL;                                    // Нет меню
	wc.cbClsExtra = 0;                                         // Нет дополнительных
данных класса
	wc.cbWndExtra = 0;                                         // Нет дополнительных
данных окна
	wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);  // Заполнение окна белым цветом

	// Регистрация класса окна
	if (!RegisterClass(&wc)) return 0;

	// Создание окна
	hWnd = CreateWindow(WinName,                       // Имя класса окна
		_T("Первая лаба по графике и таймерам"),        // Заголовок окна
		WS_OVERLAPPEDWINDOW,               // Стиль окна
		CW_USEDEFAULT,                             // x
		CW_USEDEFAULT,                             // y    Размеры окна
		CW_USEDEFAULT,                             // width
		CW_USEDEFAULT,                             // Height
		HWND_DESKTOP,                              // Дескриптор родительского окна
		NULL,                                      // Нет меню
		This,                                      // Дескриптор приложения
		NULL);                                     // Дополнительной информации нет

	ShowWindow(hWnd, mode);                            // Показать окно
	UpdateWindow(hWnd);
	// Цикл обработки сообщений
	while (GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);            // Функция трансляции кодов нажатой
клавиши
		DispatchMessage(&msg);             // Посылает сообщение функции WndProc()
	}
	return 0;
}

// Оконная функция вызывается операционной системой
// и получает сообщения из очереди для данного приложения
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	static std::vector<std::vector<int>> colors = { {0, 0, 255}, {255, 0, 0}, {0, 255,
0} }; // указываем static изменения данных из switch case
	switch (message)              // Обработчик сообщений
	{
		PAINTSTRUCT ps;
		HDC hdc;
		HBRUSH Hbrush;
		int square_size;
	case WM_CREATE:
	{
		return 0;
	}
	case WM_DESTROY:
	{
		PostQuitMessage(0);
		return 0;                    // Завершение программы
	}
	//весь код связанный с рисованием (текст, линии, фигуры, задний фон и т.п)
	//пишем здесь, для сохранения всего нарисованного при изменении окна
```

```
	//(например, при сворачивании окна), при изменении окна каким-либо образом
	//(в том числе и InvalidateRect) программе отправляется сообщение WM_PAINT
	case WM_PAINT:
	{
		UpdateWindow(hWnd);
		hdc = BeginPaint(hWnd, &ps);
		square_size = 50;
		PrintSquare(square_size * 3, colors[0], hdc, Hbrush);
		PrintSquare(square_size * 2, colors[1], hdc, Hbrush);
		PrintSquare(square_size * 1, colors[2], hdc, Hbrush);
		EndPaint(hWnd, &ps);
		return 0;
	}
	case WM_LBUTTONDOWN:
	{
		SetTimer(hWnd, TIMER1, 500, NULL); //время задаётся в миллисекундах,
		//последний параметр - указатель на функцию, вызываемую по истечении
таймера
		return 0;
	}
	case WM_RBUTTONDOWN:
	{
		KillTimer(hWnd, TIMER1);
		return 0;
	}
	case WM_TIMER:
	{
		if (wParam == TIMER1) {
			MoveElems(colors);
			//перерисовка окна после того как поменяли
			//цвета каждого квадрата (обязательный метод для
			//того, чтобы было видно изменения в нарисованных
			//объектах в окне
			InvalidateRect(
				hWnd, // 1 параметр - дискриптор окна для перерисовки
				NULL, //2 параметр указатель на структуру RECT
(прямоугольник), у которой можно задать размеры ,
				//с помощью этой структуры можно ограничить область
перерисовываемого окна
				TRUE //3 параметр - bool, в случае истины всё что нарисовано
на окне стирается перед обновлением (нужно для анимации)
			);
		}
		return 0;
	}
	default:			// Обработка сообщения по умолчанию
		return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Скрины работы:



Лабораторная работа №2 (Отправка сообщений и параметров между окнами разных программ)

Задание (17 Вариант): Сформировать два приложения, которые открывают по одному окну. В окне 1 по щелчку левой клавиши мыши: при помощи FindWindow() найти дескриптор окна 2. Выдать сообщение об этом. Если операция неудачная закрыть приложение 1. При помощи функции SendMessage() и поля WPARAM передать свой дескриптор второму окну. Выдать сообщение об этом. В окне 2: при получении сообщения WM_USER+1 (левая клавиша) выдать список запущенных приложений. При нажатии правой клавиши в первом приложении - сообщение о получении. **Доп. Задание:** вывод вторым окном полученного дискриптора первого окна.

Листинг первой программы:

```
#include <Windows.h>
#include <tchar.h>
#include "Procceess_list.h" //библиотека для вывода списка
//процессов запущенных на эвм

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

TCHAR WinName[] = _T("Part_1");

int WINAPI _tWinMain(HINSTANCE This, HINSTANCE Prev,  LPTSTR cmd, int mode)
{
	HWND hWnd;
	MSG msg;
	WNDCLASS wc;

	wc.hInstance = This;
	wc.lpszClassName = WinName;
	wc.lpfnWndProc = WndProc;
	wc.style = CS_HREDRAW | CS_VREDRAW;
	wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wc.hCursor = LoadCursor(NULL, IDC_ARROW);
	wc.lpszMenuName = NULL;
	wc.cbClsExtra = 0;
	wc.cbWndExtra = 0;
	wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

	if (!RegisterClass(&wc)) return 0;

	hWnd = CreateWindow(WinName, _T("Первое окно"), WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,  CW_USEDEFAULT,
		HWND_DESKTOP, NULL, This, NULL);

	ShowWindow(hWnd, mode);

	while (GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return 0;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message)		// Обработчик сообщений
	{
	case WM_USER: //при получении сообщения от второго окна выводит список процессов
	{
		std::wstring proccesses; //unicode строка c++ для простоты (не указывать
размер и не чистить память вручную)
		show_procceses(proccesses);
		MessageBox(hWnd, proccesses.c_str(), L"Processes_list", NULL);
```

```
                return 0;
        }
        case WM_LBUTTONDOWN:
        {
                HWND hWnd_2 = FindWindow(L"Part_2", NULL); //метод нахождения дискриптора
другого окна по названию
                if (hWnd_2 != NULL) {
                        MessageBox(hWnd, L"Second window handle found, sending the message",
                                L"Handle status", NULL);
                        SendMessage(hWnd_2, WM_USER + 1, (WPARAM)hWnd, NULL); //отправка
дискриптора этого окна второму окну
                        MessageBox(hWnd, L"First window handle sent to second window",
L"Handle status", NULL);
                }
                else {
                        exit(0);
                }
                return 0;
        }
        case WM_RBUTTONDOWN:
        {
                MessageBox(hWnd, L"Processes", L"Running apps list accepted", NULL);
//программа получет сообщение от второго окна
                return 0;
        }
        case WM_DESTROY:
        {
                PostQuitMessage(0);
                return 0;
        }
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

Библиотека вывода списка процессов

(Header-файл)

```
#pragma once
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <psapi.h>
#include <list>
#include <algorithm>
#include <string>
#include <cctype>
#include <sstream>
#include <iostream>
#include <string>
```

```
bool findStringIC(const std::wstring& strHaystack, const std::wstring& strNeedle);

int ProcessInfo(DWORD processID, std::wstring& result);

int show_procceses(std::wstring& result);
```

(CPP-файл)

```
#include "Procceess_list.h"

/// Try to find in the Haystack the Needle - ignore case
bool findStringIC(const std::wstring& strHaystack, const std::wstring& strNeedle)
{
	auto it = std::search(
		strHaystack.begin(), strHaystack.end(),
		strNeedle.begin(), strNeedle.end(),
		[](char ch1, char ch2) { return std::toupper(ch1) == std::toupper(ch2); }
	);
	return (it != strHaystack.end());
}

// To ensure correct resolution of symbols, add Psapi.lib to TARGETLIBS
// and compile with -DPSAPI_VERSION=1

int ProcessInfo(DWORD processID, std::wstring& result)
{
	HMODULE hMods[1024];
	HANDLE hProcess;
	DWORD cbNeeded;
	unsigned int i;
	hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
		PROCESS_VM_READ,
		FALSE, processID);
	if (NULL == hProcess) {
		return 1;
	}
	WCHAR procWChar[MAX_PATH];
	DWORD namelen = GetProcessImageFileName(hProcess, procWChar, sizeof(procWChar) /
sizeof(*procWChar));
	if (0 == namelen)
	{
		result = L"Name was empty, skiping";
		return 1;
	}
	std::wstring procName = std::wstring(procWChar);
	size_t lastPath = procName.find_last_of(L"\\");
	procName = procName.substr(lastPath + 1, procName.length() - lastPath - 1);
	result += L"Process: " + procName + L"\ ";
	CloseHandle(hProcess);
	return 0;
}

int show_procceses(std::wstring& result) {
	DWORD aProcesses[1024];
	DWORD cbNeeded;
	DWORD cProcesses;
```

```
	unsigned int i;
	// Get the list of process identifiers.
	if (!EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded))
		return 1;
	// Calculate how many process identifiers were returned.
	cProcesses = cbNeeded / sizeof(DWORD);
	// Print the names of each process.
	for (i = 0; i < cProcesses; i++)
	{
		ProcessInfo(aProcesses[i], result); //небольшой костыль при выводе списка
процессов
		//для того чтобы помещались все процессы одном окне
		if (i % 3 == 0) {
			result += L"\n";
		}
	}
}
```

Листинг второй программы:

```
#include <Windows.h>
#include <tchar.h>
#include <string>
#include <cstdio>
#pragma warning (disable : 4996) //visual studio не любит _stprintf
//уведомление просто можно отключить

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

TCHAR WinName[] = _T("Part_2");

int WINAPI _tWinMain(HINSTANCE This, HINSTANCE Prev, LPTSTR cmd, int mode)
{
	HWND hWnd;
	MSG msg;
	WNDCLASS wc;

	wc.hInstance = This;
	wc.lpszClassName = WinName;
	wc.lpfnWndProc = WndProc;
	wc.style = CS_HREDRAW | CS_VREDRAW;
	wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wc.hCursor = LoadCursor(NULL, IDC_ARROW);
	wc.lpszMenuName = NULL;
	wc.cbClsExtra = 0;
	wc.cbWndExtra = 0;
	wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

	if (!RegisterClass(&wc)) return 0;

	hWnd = CreateWindow(WinName, _T("Второе окно"), WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
		HWND_DESKTOP, NULL, This, NULL);

	ShowWindow(hWnd, mode);

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

```
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return 0;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message)
	{
		static HWND hwnd_2;
		HDC hdc;
		PAINTSTRUCT Ps;
	case WM_LBUTTONDOWN: {
		if (hwnd_2 != NULL) {
			SendMessage(hwnd_2, WM_USER, wParam, NULL);
		}
		else {
			MessageBox(hWnd, L"Cant'send apps list, no 1st window handle",
L"Error", NULL);
		}
		return 0;
	}
	case WM_PAINT:
	{
		if (hwnd_2 != NULL) {
			hdc = BeginPaint(hWnd, &Ps);
			wchar_t buffer[10];
			_stprintf(buffer, _T("%p"), hwnd_2); //метод для перевода
дискриптора
			//(число в 16ичной системе исчесления) в unicode строку
			TextOut(hdc, 100, 100, buffer, 9);
			EndPaint(hWnd, &Ps);
		}
		return 0;
	}
	case WM_USER + 1:
	{
		hwnd_2 = (HWND)wParam;
		InvalidateRect(hWnd, NULL, NULL);
		return 0;
	}
	case WM_DESTROY:
	{
		PostQuitMessage(0);
		return 0;
	}
	default:
		return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Лабораторная работа №3 (Меню, диалог с пользователем)

Задание (17 Вариант): Составьте программу **"Font".** В программе есть поле ввода и вывод этого текста на экран. Есть возможность выбора размера шрифта и самого шрифта.

Лабораторная работа №4 (Потоки)

Задание (17 Вариант): Первый поток выводит в левую половину окна фигуру плавно перемещающуюся по вертикали от верхнего края окна до нижнего, скачком возвращается назад и повторяет движение вниз. В каждом шаге у-координату фигуры изменяйте на 1 пиксел. Второй поток выводит в правую половину окна вводимый текст.

Листинг программы:

(В данной программе на самом деле 3 потока – 2 через createThread и один основной, но сделано так, для простоты кода и разделения двух заданий на отдельные функции)

```
#include <Windows.h>
#include <tchar.h>
#include <string>
#define TIMER1 1 //square
#define TIMER2 2 //text

#define NORMAL_WINDOW_HEIGHT 720
#define NORMAL_WINDOW_WIDTH 1280
#define BUTTN_1 1
#define EDIT_1 2

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

DWORD WINAPI FirstThread(LPVOID param);
DWORD WINAPI SecondThread(LPVOID param);
HANDLE MutexHwnd = CreateMutex(0, 0, 0); //мьютекс - объект, используемый
//для блокирования других потоков, когда один из потоков доходит
//до критического участка кода
//например при удалении и обращениях к индексам массива в разных потоках, может
//возникнуть обращение к несуществующему элементу, и программа выдаст ошибку

TCHAR WinName[] = _T("MainFrame");

struct Color {
	int r = 0;
```

```
	int g = 0;
	int b = 0;
};

void PrintFigure(int x_pos, int y_pos, int x_size, int y_size,
	HWND hWnd, Color color, HBRUSH& brush, HDC& hdc) {
	SelectObject(hdc, brush);
	Rectangle(hdc, x_pos, y_pos, x_size, y_size);
}

int WINAPI _tWinMain(HINSTANCE This, HINSTANCE Prev, LPTSTR cmd, int mode)
{
	HWND hWnd;
	MSG msg;
	WNDCLASS wc;

	wc.hInstance = This;
	wc.lpszClassName = WinName;
	wc.lpfnWndProc = WndProc;
	wc.style = CS_HREDRAW | CS_VREDRAW;
	wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wc.hCursor = LoadCursor(NULL, IDC_ARROW);
	wc.lpszMenuName = NULL;
	wc.cbClsExtra = 0;
	wc.cbWndExtra = 0;
	wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

	if (!RegisterClass(&wc)) return 0;

	hWnd = CreateWindow(WinName, _T("Каркас Windows-приложения"),
		WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
		NORMAL_WINDOW_WIDTH, NORMAL_WINDOW_HEIGHT,
		HWND_DESKTOP, NULL, This, NULL);

	ShowWindow(hWnd, mode);

	while (GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}
	return 0;
}

static std::wstring result = L"text";
static bool is_text_enter = true;
static bool is_square_moving = true;
static int y_pos = 0;
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch (message)
	{
		HINSTANCE hInst;
		static HWND hEdt1;
		static wchar_t c[21];
		static bool is_working;
		is_working = false;
```

```
	case WM_PAINT:
	{

		if (is_text_enter) {
			is_text_enter = false;
			CreateThread(NULL, 0, SecondThread, hWnd, 0, NULL);
		}
		if (is_square_moving) {
			is_square_moving = false;
			CreateThread(NULL, 0, FirstThread, hWnd, 0, NULL);
		}
		return 0;
	}
	case WM_DESTROY:
	{
		PostQuitMessage(0);
		return 0;
	}
	case WM_TIMER:
	{
		if (LOWORD(wParam) == TIMER1) {
			if (NORMAL_WINDOW_HEIGHT - 130 <= y_pos) {
				y_pos = 0;
			}
			y_pos++; //анимация обеспечена обычной перерисовкой фигуры
			is_square_moving = true;
			RECT rectangle;
			rectangle.top = 0;
			rectangle.right = 100;
			rectangle.bottom = NORMAL_WINDOW_HEIGHT;
			rectangle.left = 0;
			RECT* lpRect = &rectangle; //указатель на объект "прямоугольник"
			//для перерисовки определённого объекта
			InvalidateRect(hWnd, lpRect, TRUE);
		}
		else if (LOWORD(wParam) == TIMER2) {
			is_text_enter = true;
			InvalidateRect(hWnd, NULL, 0);
		}
		return 0;
	}
	case WM_CREATE:
	{
		HWND text_enter;
		text_enter = CreateWindow(L"edit", L"", WS_VISIBLE | WS_CHILD | WS_BORDER |
ES_RIGHT, NORMAL_WINDOW_WIDTH - 300, 200, 200, 20, hWnd, (HMENU)(EDIT_1), NULL, NULL);
		SetTimer(hWnd, TIMER1, 1, NULL);
		SetTimer(hWnd, TIMER2, 60, NULL);
		return 0;
	}
	case WM_COMMAND:
	{

		GetWindowText((HWND)lParam, c, 21);
		result = c;
		return 0;
	}
	case WM_SIZE:
```

```
	{
		is_square_moving = true;
		is_text_enter = true;
		return 0;
	}
	default:
		return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}

DWORD __stdcall FirstThread(LPVOID param) {
	HWND hWnd = static_cast<HWND>(param);
	PAINTSTRUCT ps;
	Color figure_color = { 0, 255 };
	HBRUSH hbrush = CreateSolidBrush(RGB(figure_color.r, figure_color.g,
figure_color.b));
	WaitForSingleObject(MutexHwnd, INFINITY); //метка, на которой мьютекс блокирует
//другие потоки, второй параметр отвечает за время ожидания выполнения остального кода
//(освобождения мьютекса)

	HDC hdc = BeginPaint(hWnd, &ps);
	PrintFigure(0, y_pos, 100, 100 + y_pos, hWnd, figure_color, hbrush, hdc);
	EndPaint(hWnd, &ps);
	ReleaseMutex(MutexHwnd); //освобождение мьютекса
	return 0;
}

DWORD __stdcall SecondThread(LPVOID param) {
	HWND hWnd = static_cast<HWND>(param);
	PAINTSTRUCT ps;
	WaitForSingleObject(MutexHwnd, INFINITY);
	HDC hdc = BeginPaint(hWnd, &ps);
	TextOut(hdc, 200, 200, result.c_str(), result.size());
	EndPaint(hWnd, &ps);
	ReleaseMutex(MutexHwnd);
	return 0;
}
```