Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные среды и системное программирование

ОТЧЕТ
к лабораторной работе №4
на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (WINDOWS)

Студент Н. С. Шмидт
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью данной лабораторной работы является изучение методов управления процессами и потоками Windows, такими как, порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности. Данные методы будут продемонстрированы на оконном приложении, позволяющим управлять приоритетом потоков с помощью нажатия на кнопки.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Управление потоками и процессами в Windows API предоставляет средства для эффективной организации и контроля многозадачных приложений. Краткие сведения о данной теме:

1 Процесс - это изолированное выполнение приложения, имеющее своё собственное адресное пространство, ресурсы и потоки.

2 CreateProcess: Эта функция используется для создания нового процесса. Она позволяет указать исполняемый файл, аргументы командной строки, настройки безопасности и другие параметры. Возвращает информацию о новом процессе и его главном потоке.

3 OpenProcess: Позволяет открыть существующий процесс для управления им. Используется, например, для отправки сигналов, прерывания или изменения приоритета процесса.

4 TerminateProcess: Позволяет завершить указанный процесс. Предоставляет возможность принудительно завершить процесс, но следует использовать осторожно.

5 GetExitCodeProcess: Позволяет получить код завершения процесса после его завершения.

6 CreateJobObject: Создает объект задания, который может использоваться для группировки процессов и управления ими.

7 Поток - это наименьшая единица выполнения внутри процесса. Потоки в одном процессе разделяют адресное пространство процесса и ресурсы.

8 CreateThread - функция для создания нового потока в процессе.

9 Критические секции - это механизм для ограничения доступа к общим ресурсам из нескольких потоков.

10 InitializeCriticalSection, EnterCriticalSection, и LeaveCriticalSection - функции для работы с критическими секциями.

11 Освобождение ресурсов и завершение работы процессов и потоков обеспечивают функции ExitProcess и ExitThread.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

Основой является приложение, в котором при нажатии на кнопку, можно изменять приоритет одного из трёх потоков(рисунок 1).

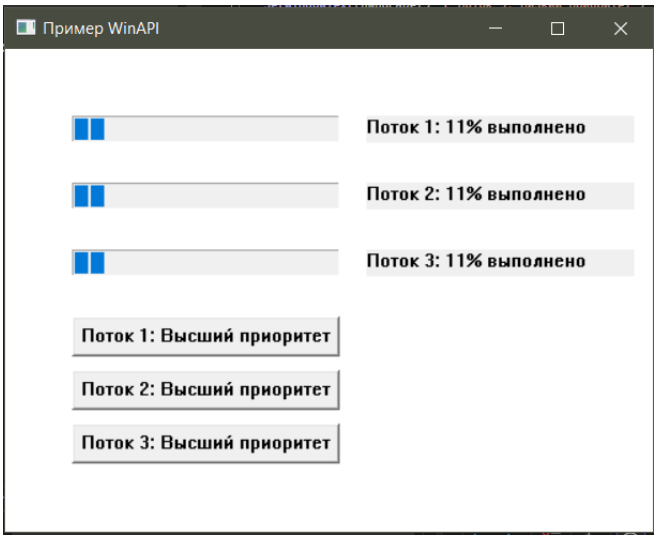

Рисунок 1 – Окно приложения

В ходе выполнения задания было создано приложение с тремя *ProgressBar*, каждый из этих элементов запускается в своём потоке. По нажатию на кнопку приоритет соответствующего *ProgressBar* потока увеличивается, что приводит к более быстрому заполнению (рисунок 2).

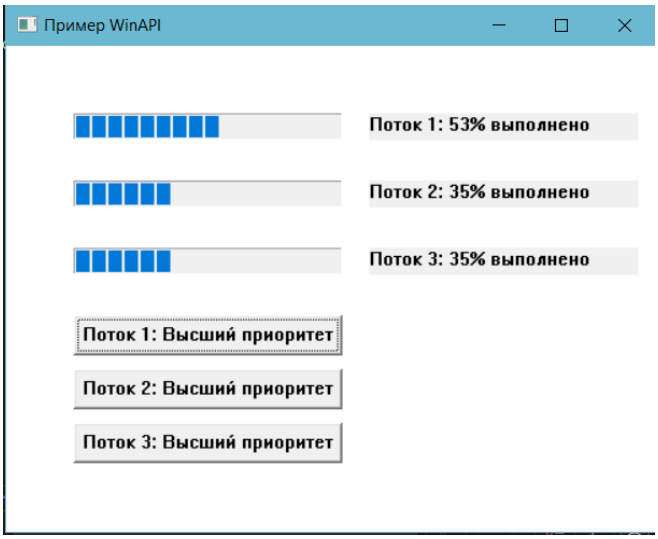

Рисунок 2 – Окно приложения после нажатия на первую кнопку

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные принципы работы с методами управления процессами и потоками Windows, такими как, порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности. Было создано оконное приложении, позволяющим управлять приоритетом потоков с помощью нажатия на кнопки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: https://dims.karelia.ru/win32/.

[2] Сопоставление файлов [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/memory/file-mapping.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл *main.cpp*

```
#include <windows.h>
#include <commctrl.h>
#include <process.h>
#include <cmath>
#include <sstream>
#include <string>

// Глобальные переменные
HANDLE hThread1, hThread2, hThread3;
HWND hwndProgressBar1, hwndProgressBar2, hwndProgressBar3;
HWND hwndLabel1, hwndLabel2, hwndLabel3;



// Функция потока для ProgressBar 1
bool isThread1HighPriority = false;
bool isThread2HighPriority = false;
bool isThread3HighPriority = false;

DWORD WINAPI ThreadFunc1(LPVOID lpParam) {
    SetWindowText(hwndLabel1, L"Поток 1: Низкий приоритет"); // Изначально
установим низкий приоритет
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_LOWEST);

    while (true) {
        if (isThread1HighPriority) {
            SetWindowText(hwndLabel1, L"Поток 1: Высший приоритет");
            SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_HIGHEST);
            isThread1HighPriority = false;
        }
        SendMessage(hwndProgressBar1, PBM_SETSTATE, PBST_NORMAL, 0);
        for (int i = 0; i <= 100; i++) {
            SendMessage(hwndProgressBar1, PBM_SETPOS, i, 0);
            Sleep(100);

            std::wstring percentText = L"Поток 1: " + std::to_wstring(i) + L"%
выполнено";
            SetWindowText(hwndLabel1, percentText.c_str());

            if            (GetThreadPriority(GetCurrentThread())           ==
THREAD_PRIORITY_HIGHEST) {
                Sleep(10);
            }
            else {
                Sleep(100);
            }
        }
```

```
    }
    return 0;
}

DWORD WINAPI ThreadFunc2(LPVOID lpParam) {
    SetWindowText(hwndLabel2, L"Поток 2: Низкий приоритет");
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_LOWEST);

    while (true) {

        if (isThread2HighPriority) {
            SetWindowText(hwndLabel2, L"Поток 2: Высший приоритет");
            SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_HIGHEST);
            isThread2HighPriority = false;
        }

        SendMessage(hwndProgressBar2, PBM_SETSTATE, PBST_NORMAL, 0);
        for (int i = 0; i <= 100; i++) {
            SendMessage(hwndProgressBar2, PBM_SETPOS, i, 0);
            Sleep(100);

            std::wstring percentText = L"Поток 2: " + std::to_wstring(i) + L"%
выполнено";
            SetWindowText(hwndLabel2, percentText.c_str());

            if            (GetThreadPriority(GetCurrentThread())            ==
THREAD_PRIORITY_HIGHEST) {
                Sleep(10);
            }
            else {
                Sleep(100);
            }
        }
    }
    return 0;
}

DWORD WINAPI ThreadFunc3(LPVOID lpParam) {
    SetWindowText(hwndLabel3, L"Поток 3: Низкий приоритет");
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_LOWEST);

    while (true) {
        if (isThread3HighPriority) {
            SetWindowText(hwndLabel3, L"Поток 3: Высший приоритет");
            SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_HIGHEST);
            isThread3HighPriority = false;
        }

        SendMessage(hwndProgressBar3, PBM_SETSTATE, PBST_NORMAL, 0);
        for (int i = 0; i <= 100; i++) {
            SendMessage(hwndProgressBar3, PBM_SETPOS, i, 0);
            Sleep(100);
```

```
            std::wstring percentText = L"Поток 3: " + std::to_wstring(i) + L"%
выполнено";
            SetWindowText(hwndLabel3, percentText.c_str());

            if            (GetThreadPriority(GetCurrentThread())            ==
THREAD_PRIORITY_HIGHEST) {
                Sleep(10);
            }
            else {
                Sleep(100);
            }
        }
    }
    return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg) {
    case WM_CREATE: {

        hwndProgressBar1 = CreateWindowEx(0, PROGRESS_CLASS, NULL,
            WS_CHILD | WS_VISIBLE | PBS_MARQUEE, 50, 50, 200, 20,
            hwnd, NULL, NULL, NULL);
        SendMessage(hwndProgressBar1, PBM_SETMARQUEE, 1, 100);

        hwndProgressBar2 = CreateWindowEx(0, PROGRESS_CLASS, NULL,
            WS_CHILD | WS_VISIBLE | PBS_MARQUEE, 50, 100, 200, 20,
            hwnd, NULL, NULL, NULL);
        SendMessage(hwndProgressBar2, PBM_SETMARQUEE, 1, 100);

        hwndProgressBar3 = CreateWindowEx(0, PROGRESS_CLASS, NULL,
            WS_CHILD | WS_VISIBLE | PBS_MARQUEE, 50, 150, 200, 20,
            hwnd, NULL, NULL, NULL);
        SendMessage(hwndProgressBar3, PBM_SETMARQUEE, 1, 100);

        hwndLabel1 = CreateWindow(L"STATIC", L"Поток 1: Высший приоритет",
            WS_CHILD | WS_VISIBLE, 270, 50, 200, 20,
            hwnd, NULL, NULL, NULL);

        hwndLabel2 = CreateWindow(L"STATIC", L"Поток 2: Обычный приоритет",
            WS_CHILD | WS_VISIBLE, 270, 100, 200, 20,
            hwnd, NULL, NULL, NULL);

        hwndLabel3 = CreateWindow(L"STATIC", L"Поток 3: Низкий приоритет",
            WS_CHILD | WS_VISIBLE, 270, 150, 200, 20,
            hwnd, NULL, NULL, NULL);

        CreateWindowEx(0, L"BUTTON", L"Поток 1: Высший приоритет", WS_CHILD |
WS_VISIBLE,
            50, 200, 200, 30, hwnd, (HMENU)1, NULL, NULL);
```

```
        CreateWindowEx(0, L"BUTTON", L"Поток 2: Высший приоритет", WS_CHILD |
WS_VISIBLE,
            50, 240, 200, 30, hwnd, (HMENU)2, NULL, NULL);

        CreateWindowEx(0, L"BUTTON", L"Поток 3: Высший приоритет", WS_CHILD |
WS_VISIBLE,
            50, 280, 200, 30, hwnd, (HMENU)3, NULL, NULL);

        break;
    }
    case WM_COMMAND: {
        int wmId = LOWORD(wParam);
        switch (wmId) {
        case 1:
            SetThreadPriority(hThread1, THREAD_PRIORITY_HIGHEST);
            SetWindowText(hwndLabel1, L"Поток 1: Высший приоритет");
            SetThreadPriority(hThread2, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel2, L"Поток 2: Низший приоритет");
            SetThreadPriority(hThread3, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel3, L"Поток 3: Низший приоритет");
            break;
        case 2:
            SetThreadPriority(hThread2, THREAD_PRIORITY_HIGHEST);
            SetWindowText(hwndLabel2, L"Поток 2: Высший приоритет");
            SetThreadPriority(hThread1, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel1, L"Поток 1: Низший приоритет");
            SetThreadPriority(hThread3, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel3, L"Поток 3: Низший приоритет");
            break;
        case 3:
            SetThreadPriority(hThread3, THREAD_PRIORITY_HIGHEST);
            SetWindowText(hwndLabel3, L"Поток 3: Высший приоритет");
            SetThreadPriority(hThread2, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel2, L"Поток 2: Низший приоритет");
            SetThreadPriority(hThread1, THREAD_PRIORITY_LOWEST);
            SetWindowText(hwndLabel1, L"Поток 1: Низший приоритет");
            break;
        }
        break;
    }
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
```

```
    WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_HREDRAW | CS_VREDRAW, WindowProc,
0, 0, GetModuleHandle(NULL), NULL, NULL, NULL, NULL, L"WinAPIExample", NULL };
    RegisterClassEx(&wc);

    HWND hwnd = CreateWindow(L"WinAPIExample", L"Пример WinAPI",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 500, 400, NULL, NULL,
hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    hThread1 = CreateThread(NULL, 0, ThreadFunc1, NULL, 0, NULL);
    hThread2 = CreateThread(NULL, 0, ThreadFunc2, NULL, 0, NULL);
    hThread3 = CreateThread(NULL, 0, ThreadFunc3, NULL, 0, NULL);

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

    TerminateThread(hThread1, 0);
    TerminateThread(hThread2, 0);
    TerminateThread(hThread3, 0);

    return msg.wParam;
}
```