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

ОТЧЁТ

к лабораторной работе №4

на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (WINDOWS). ПОРОЖДЕНИЕ, ЗАВЕРШЕНИЕ, ИЗМЕНЕНИЕ ПРИОРИТЕТОВ ПРОЦЕССОВ И ПОТОКОВ, ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ.

Студент: гр.153502 Матвеев Н.С.

Руководитель: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Формулировка задачи 3

2 Теоретические сведения 4

3 Описание функций программы 5

Список использованных источников 6

Приложение А (обязательное) Исходный код программы 7

**1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка приложения для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы.

В качестве задачи необходимо реализовать возможность выбора процесса из списка всех запущенных и его дальнейшее завершение, приостановка, возобновление.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Win32 API* предоставляет множество функций и возможностей для создания, управления и мониторинга процессов и потоков в операционной системе *Windows*.

Для создания нового процесса используется функция *CreateProcess*. Она позволяет указать исполняемый файл, командную строку, права доступа и множество других параметров.

Для взаимодействия с уже существующим процессом используется функция *OpenProcess*. Она позволяет получить дескриптор процесса и работать с ним.

Для мониторинга запущенных процессов можно использовать функцию *NtQuerySystemInformation*. C ее помощью можно получить список всех процессов и потоков системы, а также информацию об их активности [1].

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

При запуске приложения в элемент *TreeView* загружаются все системные процессы и потоки. Потоки отображаются как дочерние элементы процессов.

Для управления добавлены кнопки завершения (*Terminate*), остановки (*Suspend*) и возобновления (*Resume*) выделенного на данный момент процесса или потока. В левом верхнем углу окна добавлен статический элемент, отображающий статус выделенного элемента.

Графический интерфейс программы представлен на рисунке 1.

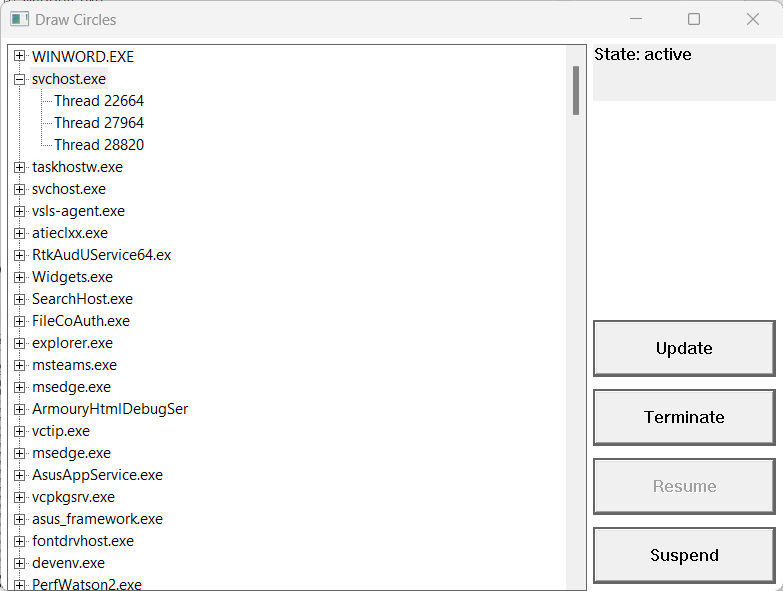


Рисунок 1 – Графический интерфейс программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] NtQuerySystemInformation function (winternl.h) - Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseWindow.h**

*#pragma once*

*template <class DERIVED\_TYPE>*

*class BaseWindow*

*{*

*public:*

*const PCWSTR CLASS\_NAME;*

*static LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*DERIVED\_TYPE \*pThis = NULL;*

*if (uMsg == WM\_NCCREATE)*

*{*

*CREATESTRUCT\* pCreate = (CREATESTRUCT\*)lParam;*

*pThis = (DERIVED\_TYPE\*)pCreate->lpCreateParams;*

*SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)pThis);*

*pThis->m\_hwnd = hwnd;*

*}*

*else*

*{*

*pThis = (DERIVED\_TYPE\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);*

*}*

*if (pThis)*

*{*

*return pThis->HandleMessage(uMsg, wParam, lParam);*

*}*

*else*

*{*

*return DefWindowProc(hwnd, uMsg, wParam, lParam);*

*}*

*}*

*BaseWindow(PCWSTR CLASS\_NAME) : CLASS\_NAME(CLASS\_NAME), m\_hwnd(NULL) { }*

*BOOL Create(*

*PCWSTR lpWindowName,*

*DWORD dwStyle,*

*HWND hWndParent = 0,*

*DWORD dwExStyle = 0,*

*int x = CW\_USEDEFAULT,*

*int y = CW\_USEDEFAULT,*

*int nWidth = CW\_USEDEFAULT,*

*int nHeight = CW\_USEDEFAULT,*

*HMENU hMenu = 0*

*)*

*{*

*WNDCLASS wc = {};*

*wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;*

*wc.hInstance = GetModuleHandle(NULL);*

*wc.lpszClassName = CLASS\_NAME;*

*RegisterClass(&wc);*

*m\_hwnd = CreateWindowEx(*

*dwExStyle, CLASS\_NAME, lpWindowName, dwStyle, x, y,*

*nWidth, nHeight, hWndParent, hMenu, GetModuleHandle(NULL), this*

*);*

*return (m\_hwnd ? TRUE : FALSE);*

*}*

*HWND Window() const { return m\_hwnd; }*

*protected:*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;*

*HWND m\_hwnd;*

*};*

**Файл DPIScale.h**

*#pragma once*

*#include <d2d1.h>*

*class DPIScale*

*{*

*static float scaleX;*

*static float scaleY;*

*public:*

*static void Initialize()*

*{*

*FLOAT dpi = GetDpiForSystem();*

*scaleX = dpi / 96.0f;*

*scaleY = dpi / 96.0f;*

*}*

*template <typename T>*

*static float PixelsToDipsX(T x)*

*{*

*return static\_cast<float>(x) / scaleX;*

*}*

*template <typename T>*

*static float PixelsToDipsY(T y)*

*{*

*return static\_cast<float>(y) / scaleY;*

*}*

*template <typename T>*

*static T DipXToPixels(float x)*

*{*

*return static\_cast<T>(x) \* scaleX;*

*}*

*template <typename T>*

*static T DipYToPixels(float y)*

*{*

*return static\_cast<T>(y) \* scaleY;*

*}*

*};*

**Файл DPIScale.cpp**

*#include "DPIScale.h"*

*float DPIScale::scaleX = 1;*

*float DPIScale::scaleY = 1;*

**Файл main.cpp**

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <Winuser.h>*

*#include <list>*

*#include <memory>*

*using namespace std;*

*#pragma comment(lib, "d2d1")*

*#pragma comment(lib, "Ntdll")*

*#pragma comment(lib, "Comctl32")*

*#include "BaseWindow.h"*

*#include "MainWindow.h"*

*#include <objidl.h>*

*#include <gdiplus.h>*

*#include <winternl.h>*

*#include <CommCtrl.h>*

*int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR, int nCmdShow)*

*{*

*//Gdiplus::GdiplusStartupInput gdiplusStartupInput;*

*//ULONG\_PTR gdiplusToken;*

*//Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);*

*MainWindow win = MainWindow();*

*//Mode mode = Mode::SelectMode;*

*//Figure figure = Figure::Ellipse;*

*//D2D1\_COLOR\_F color = D2D1::ColorF(D2D1::ColorF::Black);*

*//SceneControl win = SceneControl(&mode, &figure, &color);*

*//GraphicsScene win = GraphicsScene(&mode, &figure, &color);*

*if (!win.Create(L"Draw Circles",*

*WS\_OVERLAPPEDWINDOW,*

*NULL,*

*NULL,*

*CW\_USEDEFAULT,*

*CW\_USEDEFAULT,*

*800,*

*600))*

*{*

*return 0;*

*}*

*//SetWindowsHookEx(WH\_GETMESSAGE, GetMsgProcCustom, NULL, GetCurrentThreadId());*

*//HACCEL hAccel1 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL1));*

*//HACCEL hAccel2 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL2));*

*ShowWindow(win.Window(), nCmdShow);*

*MSG msg;*

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

*{*

*//if (!TranslateAccelerator(GetFocus(), hAccel1, &msg) && !TranslateAccelerator(GetFocus(), hAccel2, &msg))*

*//{*

*TranslateMessage(&msg);*

*DispatchMessage(&msg);*

*//}*

*}*

*return 0;*

*}*

**Файл MainWindow.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <compressapi.h>*

*#include <shlwapi.h>*

*#include <string>*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "Initializer.h"*

*class MainWindow : public BaseWindow<MainWindow>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const float MARGIN\_X;*

*static const float MARGIN\_Y;*

*public:*

*MainWindow(PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME);*

*~MainWindow();*

*void update();*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*Initializer initializer;*

*HWND suspendButton;*

*HWND resumeButton;*

*HWND terminateButton;*

*HWND updateButton;*

*HWND sleepLabel;*

*HWND tree;*

*BaseItem\* selected;*

*};*

**Файл MainWindow.cpp**

*#include "MainWindow.h"*

*const PCWSTR MainWindow::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const float MainWindow::MARGIN\_X = 6.0F;*

*const float MainWindow::MARGIN\_Y = 6.0F;*

*MainWindow::MainWindow(PCWSTR CLASS\_NAME) :*

*BaseWindow<MainWindow>(CLASS\_NAME), initializer(), selected(NULL)*

*{*

*}*

*MainWindow::~MainWindow()*

*{*

*}*

*void MainWindow::update()*

*{*

*HTREEITEM hSelectedItem = TreeView\_GetSelection(tree);*

*TVITEM item = TVITEM();*

*item.hItem = hSelectedItem;*

*item.mask = TVIF\_PARAM;*

*TreeView\_GetItem(tree, &item);*

*selected = (BaseItem\*)item.lParam;*

*if (!selected)*

*{*

*SetWindowText(sleepLabel, NULL);*

*EnableWindow(suspendButton, FALSE);*

*EnableWindow(resumeButton, FALSE);*

*EnableWindow(terminateButton, FALSE);*

*}*

*else if (selected->isSuspended())*

*{*

*SetWindowText(sleepLabel, TEXT("State: suspended"));*

*EnableWindow(suspendButton, FALSE);*

*EnableWindow(resumeButton, TRUE);*

*EnableWindow(terminateButton, TRUE);*

*}*

*else*

*{*

*SetWindowText(sleepLabel, TEXT("State: active"));*

*EnableWindow(suspendButton, TRUE);*

*EnableWindow(resumeButton, FALSE);*

*EnableWindow(terminateButton, TRUE);*

*}*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*//static HBRUSH hBrush = CreateSolidBrush(RGB(230, 2, 2));*

*LRESULT MainWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*CreateLayout();*

*return 0;*

*case WM\_PAINT:*

*{*

*PAINTSTRUCT ps;*

*HDC hdc = BeginPaint(m\_hwnd, &ps);*

*FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));*

*EndPaint(m\_hwnd, &ps);*

*return 0;*

*}*

*case WM\_INITDIALOG:*

*InitCommonControls();*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*case WM\_SIZING:*

*{*

*RECT\* dragRc = (RECT\*)lParam;*

*RECT rcWindow;*

*GetWindowRect(m\_hwnd, &rcWindow);*

*int WINDOW\_HEIGHT\_PIX = rcWindow.bottom - rcWindow.top;*

*int WINDOW\_WIDTH\_PIX = rcWindow.right - rcWindow.left;*

*switch (wParam)*

*{*

*case WMSZ\_BOTTOM:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_BOTTOMLEFT:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_LEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_TOPLEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOP:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOPRIGHT:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_RIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_BOTTOMRIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*}*

*return TRUE;*

*}*

*case WM\_NOTIFY:*

*{*

*NMHDR\* notification = (NMHDR\*)lParam;*

*if (notification->code == TVN\_SELCHANGED)*

*{*

*update();*

*return 0;*

*}*

*break;*

*}*

*case WM\_COMMAND:*

*if (HIWORD(wParam) == BN\_CLICKED)*

*{*

*if ((HWND)lParam == suspendButton)*

*{*

*selected->suspend();*

*initializer.initialize(tree);*

*update();*

*}*

*else if ((HWND)lParam == resumeButton)*

*{*

*selected->resume();*

*initializer.initialize(tree);*

*update();*

*}*

*else if ((HWND)lParam == terminateButton)*

*{*

*selected->terminate();*

*/\*MessageBox(*

*NULL,*

*NULL,*

*NULL,*

*NULL*

*);\*/*

*initializer.initialize(tree);*

*update();*

*}*

*else*

*{*

*initializer.initialize(tree);*

*update();*

*}*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*return 0;*

*}*

*break;*

*/\*case WM\_CTLCOLORSTATIC:*

*{*

*HDC hdcStatic = (HDC)wParam;*

*SetTextColor(hdcStatic, RGB(0, 0, 0));*

*SetBkColor(hdcStatic, RGB(230, 2, 2));*

*return (INT\_PTR)hBrush;*

*}\*/*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

*void MainWindow::CreateLayout()*

*{*

*tree = CreateWindow(WC\_TREEVIEW,*

*L"Tree view",*

*WS\_VISIBLE | WS\_CHILD | WS\_BORDER | TVS\_HASLINES | TVS\_HASBUTTONS | TVS\_LINESATROOT | TVS\_SHOWSELALWAYS,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*suspendButton = CreateWindow(L"BUTTON",*

*L"Suspend",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*resumeButton = CreateWindow(L"BUTTON",*

*L"Resume",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*terminateButton = CreateWindow(L"BUTTON",*

*L"Terminate",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*updateButton = CreateWindow(L"BUTTON",*

*L"Update",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*sleepLabel = CreateWindow(L"STATIC",*

*NULL,*

*WS\_VISIBLE | WS\_CHILD | SS\_LEFT,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*initializer.initialize(tree);*

*update();*

*}*

*void MainWindow::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*int MARGIN\_XPix = DPIScale::DipXToPixels<int>(MARGIN\_X);*

*int MARGIN\_YPix = DPIScale::DipYToPixels<int>(MARGIN\_Y);*

*int WINDOW\_HEIGHT\_PIX = rcClient.bottom - rcClient.top;*

*int WINDOW\_WIDTH\_PIX = rcClient.right - rcClient.left;*

*MoveWindow(tree,*

*MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 - MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX - MARGIN\_YPix,*

*FALSE);*

*MoveWindow(sleepLabel,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(suspendButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 1) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(resumeButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 2) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(terminateButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 3) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(updateButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 4) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

**Файл BaseItem.h**

*#pragma once*

*#include <windows.h>*

*class BaseItem*

*{*

*public:*

*BaseItem() : valid(true) {}*

*bool valid;*

*virtual bool suspend() = 0;*

*virtual bool terminate() = 0;*

*virtual bool resume() = 0;*

*virtual bool isSuspended() const = 0;*

*virtual LPCWSTR getName() const = 0;*

*virtual DWORD getId() const = 0;*

*virtual ~BaseItem() = default;*

*bool isValid() { return valid; }*

*void invalidate() { valid = false; }*

*};*

**Файл BaseNode.h**

*#pragma once*

*#include <commctrl.h>*

*class BaseNode*

*{*

*public:*

*TVINSERTSTRUCTW getNode() { return node; }*

*const TVINSERTSTRUCTW getNode() const { return node; }*

*protected:*

*TVINSERTSTRUCTW node;*

*};*

**Файл ProcessItem.h**

*#pragma once*

*#include "ThreadItem.h"*

*#include <vector>*

*#include <deque>*

*class ProcessItem : public BaseItem*

*{*

*private:*

*static LPCWSTR NAME\_PREFIX;*

*public:*

*bool suspended;*

*size\_t process\_id;*

*std::wstring name;*

*std::deque<ThreadItem\*> threads;*

*ProcessItem(\_SYSTEM\_PROCESS\_INFORMATION\* processInfo);*

*virtual bool suspend() override;*

*virtual bool terminate() override;*

*virtual bool resume() override;*

*virtual bool isSuspended() const override { return suspended; }*

*virtual LPCWSTR getName() const override { return name.c\_str(); }*

*virtual DWORD getId() const override { return process\_id; }*

*virtual ~ProcessItem();*

*std::deque<ThreadItem\*>\* getThreads() { return &threads; }*

*bool update(\_SYSTEM\_PROCESS\_INFORMATION\* info);*

*};*

**Файл ProcessItem.cpp**

*#include "ProcessItem.h"*

*LPCWSTR ProcessItem::NAME\_PREFIX = L"Process ";*

*ProcessItem::ProcessItem(\_SYSTEM\_PROCESS\_INFORMATION\* info) : threads(), suspended(true)*

*{*

*process\_id = (DWORD)(info->UniqueProcessId);*

*if (info->ImageName.Buffer)*

*name = info->ImageName.Buffer;*

*else*

*name = NAME\_PREFIX + std::to\_wstring((DWORD)(info->UniqueProcessId));*

*size\_t si = sizeof(\_SYSTEM\_PROCESS\_INFORMATION);*

*\_SYSTEM\_THREAD\_INFORMATION\* threadsData = (\_SYSTEM\_THREAD\_INFORMATION\*)(info + 1);*

*std::qsort(*

*threadsData,*

*info->NumberOfThreads,*

*sizeof(\_SYSTEM\_THREAD\_INFORMATION),*

*[](const void\* x, const void\* y) {*

*const \_SYSTEM\_THREAD\_INFORMATION\* arg1 = static\_cast<const \_SYSTEM\_THREAD\_INFORMATION\*>(x);*

*const \_SYSTEM\_THREAD\_INFORMATION\* arg2 = static\_cast<const \_SYSTEM\_THREAD\_INFORMATION\*>(y);*

*return ((int)((DWORD)(arg1->ClientId.UniqueThread) > (DWORD)(arg2->ClientId.UniqueThread)) - (int)((DWORD)(arg1->ClientId.UniqueThread) < (DWORD)(arg2->ClientId.UniqueThread)));*

*}*

*);*

*for (size\_t i = 0; i < info->NumberOfThreads; ++i)*

*{*

*if (threadsData->WaitReason != 5)*

*suspended = false;*

*threads.emplace\_back(new ThreadItem(threadsData++));*

*}*

*}*

*ProcessItem::~ProcessItem()*

*{*

*for (auto& thread : threads)*

*delete thread;*

*//CloseHandle(process);*

*}*

*bool ProcessItem::suspend()*

*{*

*valid = false;*

*for (auto& thread : threads)*

*{*

*if (thread->suspend() < 0)*

*return true;*

*}*

*}*

*bool ProcessItem::resume()*

*{*

*valid = false;*

*for (auto & thread : threads)*

*{*

*if (thread->resume() < 0)*

*return true;*

*}*

*}*

*bool ProcessItem::terminate()*

*{*

*valid = false;*

*if (HANDLE process = OpenProcess(THREAD\_ALL\_ACCESS, false, process\_id))*

*{*

*DWORD ret = TerminateProcess(process, NULL);*

*CloseHandle(process);*

*return ret == 0;*

*}*

*}*

**Файл ThreadItem.h**

*#pragma once*

*#include "BaseItem.h"*

*#include <winternl.h>*

*#include <string>*

*class ThreadItem : public BaseItem*

*{*

*private:*

*static LPCWSTR NAME\_PREFIX;*

*public:*

*ThreadItem(\_SYSTEM\_THREAD\_INFORMATION\* info);*

*virtual bool suspend() override;*

*virtual bool terminate() override;*

*virtual bool resume() override;*

*virtual bool isSuspended() const override { return suspended; }*

*virtual LPCWSTR getName() const override { return name.c\_str(); }*

*virtual DWORD getId() const override { return thread\_id; }*

*virtual ~ThreadItem();*

*bool update(\_SYSTEM\_THREAD\_INFORMATION\* info);*

*bool operator<(const ThreadItem& obj) const { return thread\_id < obj.getId(); }*

*bool operator==(const ThreadItem& obj) const { return thread\_id == obj.getId(); }*

*protected:*

*size\_t thread\_id;*

*std::wstring name;*

*bool suspended;*

*};*

**Файл ThreadItem.cpp**

*#include "ThreadItem.h"*

*LPCWSTR ThreadItem::NAME\_PREFIX = L"Thread ";*

*ThreadItem::ThreadItem(\_SYSTEM\_THREAD\_INFORMATION\* info)*

*{*

*thread\_id = (DWORD)(info->ClientId.UniqueThread);*

*name = NAME\_PREFIX + std::to\_wstring(thread\_id);*

*suspended = info->WaitReason == 5;*

*}*

*ThreadItem::~ThreadItem()*

*{*

*//CloseHandle(thread);*

*}*

*bool ThreadItem::suspend()*

*{*

*valid = false;*

*if (HANDLE thread = OpenThread(THREAD\_ALL\_ACCESS, false, thread\_id))*

*{*

*DWORD ret = SuspendThread(thread);*

*CloseHandle(thread);*

*return ret < 0;*

*}*

*}*

*bool ThreadItem::resume()*

*{*

*valid = false;*

*if (HANDLE thread = OpenThread(THREAD\_ALL\_ACCESS, false, thread\_id))*

*{*

*DWORD ret;*

*while ((ret = ResumeThread(thread)) > 0);*

*CloseHandle(thread);*

*return ret < 0;*

*}*

*}*

*bool ThreadItem::terminate()*

*{*

*valid = false;*

*if (HANDLE thread = OpenThread(THREAD\_ALL\_ACCESS, false, thread\_id))*

*{*

*DWORD ret = TerminateThread(thread, NULL);*

*CloseHandle(thread);*

*return ret == 0;*

*}*

*}*

*bool ThreadItem::update(\_SYSTEM\_THREAD\_INFORMATION\* info)*

*{*

*bool updated = false;*

*if ((info->WaitReason == 5) != suspended)*

*{*

*suspended = info->WaitReason == 5;*

*updated = true;*

*}*

*valid = true;*

*return updated;*

*}*

**Файл ThreadNode.h**

*#pragma once*

*#include "ProcessItem.h"*

*#include "BaseNode.h"*

*#include <commctrl.h>*

*class ThreadNode : public BaseNode*

*{*

*static const size\_t COLUMNT\_COUNT;*

*static const size\_t STR\_SIZE;*

*static const size\_t PU\_COLUMNS[];*

*static LPCWSTR SUSPEND\_VAL;*

*static LPCWSTR NOT\_SUSPEND\_VAL;*

*public:*

*ThreadNode(ThreadItem\* data, HWND tree, const BaseNode\* parent, const BaseNode\* after);*

*DWORD getId() { return data->getId(); }*

*~ThreadNode();*

*protected:*

*ThreadItem\* data;*

*HWND tree;*

*};*

**Файл ThreadNode.cpp**

*#include "ThreadNode.h"*

*#include <strsafe.h>*

*const size\_t ThreadNode::COLUMNT\_COUNT = 1;*

*const size\_t ThreadNode::PU\_COLUMNS[] = { 1 };*

*const size\_t ThreadNode::STR\_SIZE = 20;*

*LPCWSTR ThreadNode::SUSPEND\_VAL = L"Suspended";*

*LPCWSTR ThreadNode::NOT\_SUSPEND\_VAL = L"Active";*

*ThreadNode::ThreadNode(ThreadItem\* data, HWND tree, const BaseNode\* parent, const BaseNode\* after) : data(data), tree(tree)*

*{*

*node = TVINSERTSTRUCTW();*

*if (parent != NULL)*

*node.hParent = parent->getNode().item.hItem;*

*if (after != NULL)*

*node.hInsertAfter = after->getNode().item.hItem;*

*else*

*node.hInsertAfter = TVI\_FIRST;*

*node.item.mask = TVIF\_PARAM | TVIF\_TEXT;*

*node.item.lParam = (LPARAM)data;*

*node.item.pszText = new WCHAR[STR\_SIZE];*

*node.item.cchTextMax = STR\_SIZE;*

*StringCchCopy(node.item.pszText, node.item.cchTextMax, data->getName());*

*node.item.hItem = (HTREEITEM)SendMessage(tree, TVM\_INSERTITEM, 0, (LPARAM)&node);*

*}*

*ThreadNode::~ThreadNode()*

*{*

*TreeView\_DeleteItem(tree, node.item.hItem);*

*delete[] (WCHAR\*)node.item.pszText;*

*}*

**Файл ProcessNode.h**

*#pragma once*

*#include "ThreadNode.h"*

*#include <commctrl.h>*

*class ProcessNode : public BaseNode*

*{*

*static const size\_t COLUMNT\_COUNT;*

*static const size\_t STR\_SIZE;*

*static const size\_t PU\_COLUMNS[];*

*static LPCWSTR SUSPEND\_VAL;*

*static LPCWSTR NOT\_SUSPEND\_VAL;*

*public:*

*ProcessNode(\_SYSTEM\_PROCESS\_INFORMATION\* info, HWND tree, const BaseNode\* parent, const BaseNode\* after);*

*void update(\_SYSTEM\_PROCESS\_INFORMATION\* info);*

*DWORD getId() { return data.getId(); }*

*~ProcessNode();*

*protected:*

*ProcessItem data;*

*std::deque<ThreadNode\*> threads;*

*HWND tree;*

*};*

**Файл ProcessNode.cpp**

*#include "ProcessNode.h"*

*#include <strsafe.h>*

*const size\_t ProcessNode::COLUMNT\_COUNT = 1;*

*const size\_t ProcessNode::PU\_COLUMNS[] = { 1 };*

*const size\_t ProcessNode::STR\_SIZE = 20;*

*LPCWSTR ProcessNode::SUSPEND\_VAL = L"Suspended";*

*LPCWSTR ProcessNode::NOT\_SUSPEND\_VAL = L"Active";*

*ProcessNode::ProcessNode(\_SYSTEM\_PROCESS\_INFORMATION\* info, HWND tree, const BaseNode\* parent, const BaseNode\* prev) : tree(tree), data(info), threads()*

*{*

*node = TVINSERTSTRUCTW();*

*if (parent != NULL)*

*node.hParent = parent->getNode().item.hItem;*

*if (prev != NULL)*

*node.hInsertAfter = prev->getNode().item.hItem;*

*else*

*node.hInsertAfter = TVI\_FIRST;*

*node.item.mask = TVIF\_PARAM | TVIF\_TEXT;*

*node.item.lParam = (LPARAM)&data;*

*node.item.pszText = new WCHAR[STR\_SIZE];*

*node.item.cchTextMax = STR\_SIZE;*

*StringCchCopy(node.item.pszText, node.item.cchTextMax, data.getName());*

*node.item.hItem = (HTREEITEM)SendMessage(tree, TVM\_INSERTITEM, 0, (LPARAM)&node);*

*ThreadNode\* after = NULL;*

*for (size\_t i = 0; i < data.threads.size(); i++)*

*{*

*threads.emplace(threads.begin() + i, new ThreadNode(data.threads[i], tree, this, after));*

*after = threads.back();*

*}*

*}*

*void ProcessNode::update(\_SYSTEM\_PROCESS\_INFORMATION\* info)*

*{*

*\_SYSTEM\_THREAD\_INFORMATION\* threadsData = (\_SYSTEM\_THREAD\_INFORMATION\*)(info + 1);*

*std::qsort(*

*threadsData,*

*info->NumberOfThreads,*

*sizeof(\_SYSTEM\_THREAD\_INFORMATION),*

*[](const void\* x, const void\* y) {*

*const \_SYSTEM\_THREAD\_INFORMATION\* arg1 = static\_cast<const \_SYSTEM\_THREAD\_INFORMATION\*>(x);*

*const \_SYSTEM\_THREAD\_INFORMATION\* arg2 = static\_cast<const \_SYSTEM\_THREAD\_INFORMATION\*>(y);*

*return ((int)((DWORD)(arg1->ClientId.UniqueThread) > (DWORD)(arg2->ClientId.UniqueThread)) - (int)((DWORD)(arg1->ClientId.UniqueThread) < (DWORD)(arg2->ClientId.UniqueThread)));*

*}*

*);*

*data.suspended = true;*

*ThreadNode\* after = NULL;*

*size\_t i = 0, j = 0;*

*while (i < threads.size() && j < info->NumberOfThreads)*

*{*

*while (i < threads.size() && j < info->NumberOfThreads && (DWORD)((threadsData + j)->ClientId.UniqueThread) < threads[i]->getId())*

*{*

*if ((threadsData + j)->WaitReason != 5)*

*data.suspended = false;*

*data.threads.emplace(data.threads.begin() + i, new ThreadItem(threadsData + j));*

*threads.emplace(threads.begin() + i, new ThreadNode(data.threads[i], tree, this, after));*

*after = threads[i];*

*j++;*

*i++;*

*}*

*while (i < threads.size() && j < info->NumberOfThreads && (DWORD)((threadsData + j)->ClientId.UniqueThread) == threads[i]->getId())*

*{*

*if ((threadsData + j)->WaitReason != 5)*

*data.suspended = false;*

*data.threads[i]->update(threadsData + j);*

*after = threads[i];*

*j++;*

*i++;*

*}*

*while (i < threads.size() && j < info->NumberOfThreads && (DWORD)((threadsData + j)->ClientId.UniqueThread) > threads[i]->getId())*

*{*

*delete threads[i];*

*threads.erase(threads.begin() + i);*

*delete data.threads[i];*

*data.threads.erase(data.threads.begin() + i);*

*}*

*}*

*while (i < threads.size())*

*{*

*delete threads[i];*

*threads.erase(threads.begin() + i);*

*delete data.threads[i];*

*data.threads.erase(data.threads.begin() + i);*

*}*

*while (j < info->NumberOfThreads)*

*{*

*if ((threadsData + j)->WaitReason != 5)*

*data.suspended = false;*

*data.threads.emplace(data.threads.begin() + i, new ThreadItem(threadsData + j));*

*threads.emplace(threads.begin() + i, new ThreadNode(data.threads[i], tree, this, after));*

*after = threads.back();*

*j++;*

*i++;*

*}*

*data.valid = true;*

*}*

*ProcessNode::~ProcessNode()*

*{*

*for (auto& thread : threads)*

*delete thread;*

*TreeView\_DeleteItem(tree, node.item.hItem);*

*delete[] (WCHAR\*)node.item.pszText;*

*}*