МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Российский государственный университет им. А.Н. Косыгина   
(Технологии. Дизайн. Искусство)»**

Институт ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ЦИФРОВЫЕ ТРАНСФОРМАЦИИ

**Отчет по лабораторной работе № 11**

**по дисциплине «Операционные системы»**

**Тема: «Управление процессами»**

Выполнил: Сидоров Д. С., группа ИТС-123

Проверил: к.т.н., доц. Семёнов А. А.

Москва, 2025г

**Управление процессами**

Часть 1.

Разработать GUI-приложение для управления процессами. Приложение должно иметь графический интерфейс пользователя согласно представленному ниже прототипу.

На форме необходимо разместить следующие компоненты:

· Button1 (для реализации функции сворачивания приложения в системный трей);

· Button2 (для завершения приложения);

· ListBox1 (для вывода списка активных процессов);

· Timer1 (таймер для подсчета времени работы Блокнота)

Приложение должно реализовывать следующий функционал.

1. Находиться в системном трее. Сворачиваться и разворачиваться в трей.

2. По таймеру обновлять список активных процессов и проверять присутствует ли среди них процесс notepad.exe.

3. Принудительно завершать работу Блокнота, если он работает более 2-ух минут.

Часть 2.

Запустите представленную ниже программу и убедитесь в её работоспособности:

- выводится информация о внешнем процессе;

- внешний процесс уничтожается.

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

int main(int argc, char\* argv[]) {

//HWND hwnd = FindWindow(NULL, "Калькулятор");

HWND hwnd = FindWindow(NULL, "Безымянный - Paint");

if (!hwnd)

{

std::cout << "Error find window\n";

return 1;

}

std::cout << "HWND=" << std::hex << hwnd << "\n";

DWORD id;

GetWindowThreadProcessId(hwnd, &id);

HANDLE hProcess;

hProcess=OpenProcess(PROCESS\_ALL\_ACCESS, false, id);

if (!hProcess)

60

{

}

std::cout << "Error get handle process\n";

return 1;

std::cout << "hProcess=" << std::hex << hProcess << "\n";

FILETIME ft[4];

GetProcessTimes(hProcess, &ft[0], &ft[1], &ft[2], &ft[3]);

SYSTEMTIME tm[4];

FileTimeToLocalFileTime(&(ft[0]), &(ft[0]));

for (int i = 0; i < 4; i++)

tm[0].wSecond);

FileTimeToSystemTime(&ft[i], &tm[i]);

printf("Created at %02d:%02d:%02d\n", tm[0].wHour, tm[0].wMinute,

printf("Kernel time %02d:%02d:%02d:%02d\n", tm[2].wHour, tm[2].wMinute,

tm[2].wSecond, tm[2].wMilliseconds);

printf("User time %02d:%02d:%02d:%02d\n", tm[3].wHour, tm[3].wMinute,

tm[3].wSecond, tm[3].wMilliseconds);

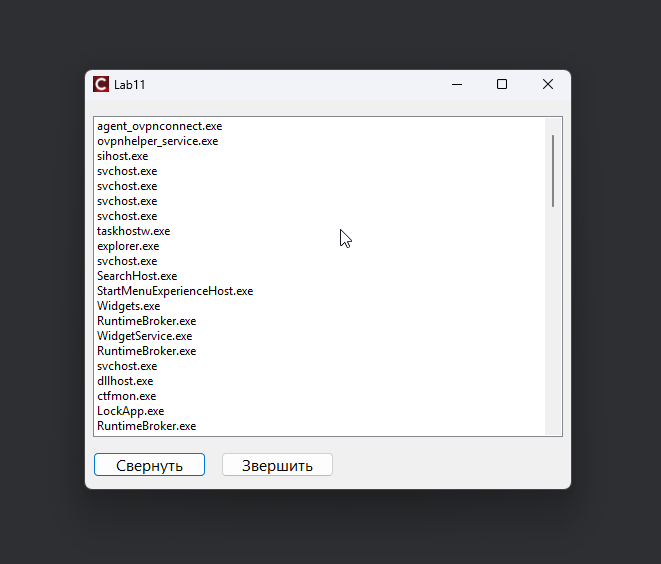
TerminateProcess(hProcess, NO\_ERROR);

CloseHandle(hProcess);

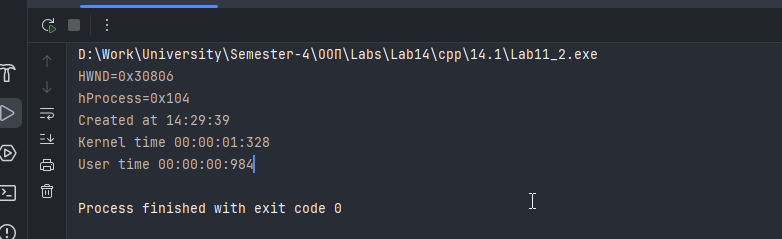
return 0;

}

Часть 3. Разработайте аналогичную представленной в части 2 программу на языке Python.

  
*Рисунок 1. Работающее приложение из п. 1.*

  
*Рисунок 2. Сворачивание приложения в трей*

  
*Рисунок 3. Завершение приложения по названию из п.2.*

**Листинг кода**

**Unit1.cpp**#include <vcl.h>

#include <tlhelp32.h>

#include <Psapi.h>

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

TNotifyIconData NID;

PInf \*Inf;

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

void \_\_fastcall TForm1::MTIcon(TMessage &a)

{

switch (a.LParam)

{

case WM\_LBUTTONDBLCLK:

case WM\_RBUTTONDOWN:

Form1->Show();

SetForegroundWindow(Handle);

break;

}

}

void \_\_fastcall TForm1::spisokprocessov()

{

for (int i = 0; i < ListBox1->Count; i++) {

PInf\* item = (PInf\*)(ListBox1->Items->Objects[i]);

if (item && item->H) CloseHandle(item->H);

delete item;

}

ListBox1->Clear();

HANDLE HS = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

if (HS == INVALID\_HANDLE\_VALUE) return;

PROCESSENTRY32 P;

P.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(HS, &P)) {

do {

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_LIMITED\_INFORMATION, FALSE, P.th32ProcessID);

if (hProcess) {

PInf \*inf = new PInf;

inf->Usage = P.cntUsage;

inf->Threads = P.cntThreads;

inf->PriClassBase = P.pcPriClassBase;

inf->ID = P.th32ProcessID;

inf->Suspended = false;

inf->H = hProcess;

FILETIME ftCreate, ftExit, ftKernel, ftUser;

if (GetProcessTimes(hProcess, &ftCreate, &ftExit, &ftKernel, &ftUser)) {

inf->StartTime = ftCreate;

} else {

ZeroMemory(&inf->StartTime, sizeof(FILETIME));

}

ListBox1->AddItem(P.szExeFile, (TObject\*)inf);

}

} while (Process32Next(HS, &P));

}

CloseHandle(HS);

}

\_\_int64 FileTimeToInt64(FILETIME ft)

{

ULARGE\_INTEGER uli;

uli.LowPart = ft.dwLowDateTime;

uli.HighPart = ft.dwHighDateTime;

return uli.QuadPart;

}

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

Application->ShowMainForm = true;

NID.cbSize = sizeof(TNotifyIconData);

NID.hWnd = Handle;

NID.uID = 1;

NID.uFlags = NIF\_ICON | NIF\_MESSAGE | NIF\_TIP;

NID.uCallbackMessage = WM\_USER + 1;

NID.hIcon = Application->Icon->Handle;

wcscpy(NID.szTip, L"Terminate Process");

Shell\_NotifyIcon(NIM\_ADD, &NID);

spisokprocessov();

}

void \_\_fastcall TForm1::Timer1Timer(TObject \*Sender)

{

spisokprocessov();

FILETIME ftNow;

GetSystemTimeAsFileTime(&ftNow);

for (int i = 0; i < ListBox1->Count; i++) {

AnsiString procName = ListBox1->Items->Strings[i];

if (procName == "notepad.exe") {

PInf\* inf = (PInf\*)ListBox1->Items->Objects[i];

if (inf && inf->H) {

\_\_int64 now = FileTimeToInt64(ftNow);

\_\_int64 start = FileTimeToInt64(inf->StartTime);

if (now - start > 1200000000) {

TerminateProcess(inf->H, 1);

WaitForSingleObject(inf->H, 3000);

}

}

}

}

spisokprocessov();

}

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

Form1->Visible = false;

}

void \_\_fastcall TForm1::Button2Click(TObject \*Sender)

{

Form1->Close();

}

void \_\_fastcall TForm1::FormDestroy(TObject \*Sender)

{

Shell\_NotifyIcon(NIM\_DELETE, &NID);

}

//---------------------------------------------------------------------------

**Unit1.h**

// Unit1.h

#ifndef Unit1H

#define Unit1H

//---------------------------------------------------------------------------

#include <Classes.hpp>

#include <Controls.hpp>

#include <StdCtrls.hpp>

#include <Forms.hpp>

#include <ExtCtrls.hpp>

#include <Windows.h>

#include <ctime>

//---------------------------------------------------------------------------

struct PInf {

int Usage;

int Threads;

int PriClassBase;

DWORD ID;

DWORD V1, V2;

HANDLE H;

bool Suspended;

FILETIME StartTime;

};

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TListBox \*ListBox1;

TButton \*Button1;

TButton \*Button2;

TTimer \*Timer1;

void \_\_fastcall FormCreate(TObject \*Sender);

void \_\_fastcall Timer1Timer(TObject \*Sender);

void \_\_fastcall FormDestroy(TObject \*Sender);

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall Button2Click(TObject \*Sender);

private: // User declarations

void \_\_fastcall spisokprocessov();

void \_\_fastcall MTIcon(TMessage &a);

BEGIN\_MESSAGE\_MAP

MESSAGE\_HANDLER(WM\_USER + 1, TMessage, MTIcon)

END\_MESSAGE\_MAP(TForm)

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

};

//---------------------------------------------------------------------------

extern PACKAGE TForm1 \*Form1;

//---------------------------------------------------------------------------

#endif

**Код приложения из п. 2.**

#include <Windows.h>

#include <iostream>

int main(int argc, char\* argv[]) {

HWND hwnd = FindWindowW(NULL, L"GitHub Desktop");

if (!hwnd) {

std::cout << "Error find window\n";

return 1;

}

std::cout << "HWND=" << std::hex << hwnd << "\n";

// Получаем ID процесса

DWORD id;

GetWindowThreadProcessId(hwnd, &id);

// Получаем хэндл процесса

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, id);

if (!hProcess) {

std::cout << "Error get handle process\n";

return 1;

}

std::cout << "hProcess=" << std::hex << hProcess << "\n";

// Получаем информацию о времени работы процесса

FILETIME ft[4];

if (!GetProcessTimes(hProcess, &ft[0], &ft[1], &ft[2], &ft[3])) {

std::cout << "Error getting process times\n";

CloseHandle(hProcess);

return 1;

}

SYSTEMTIME tm[4];

FileTimeToLocalFileTime(&(ft[0]), &(ft[0]));

for (int i = 0; i < 4; i++) {

FileTimeToSystemTime(&(ft[i]), &(tm[i]));

}

// Выводим время создания процесса

printf("Created at %02d:%02d:%02d\n", tm[0].wHour, tm[0].wMinute, tm[0].wSecond);

// Выводим время процессора (ядро и пользователь)

printf("Kernel time %02d:%02d:%02d:%02d\n", tm[2].wHour, tm[2].wMinute, tm[2].wSecond, tm[2].wMilliseconds);

printf("User time %02d:%02d:%02d:%02d\n", tm[3].wHour, tm[3].wMinute, tm[3].wSecond, tm[3].wMilliseconds);

TerminateProcess(hProcess, NO\_ERROR);

CloseHandle(hProcess);

return 0;

}

**Код приложения на питоне из п. 3.**

import win32gui

import win32process

import win32api

import psutil

import time

def get\_process\_info\_and\_terminate(window\_title):

hwnd = win32gui.FindWindow(None, window\_title)

if not hwnd:

print("Error find window")

return

print(f"HWND = {hwnd}")

\_, pid = win32process.GetWindowThreadProcessId(hwnd)

print(f"PID = {pid}")

try:

process = psutil.Process(pid)

except psutil.NoSuchProcess:

print("Error get handle process!")

return

print(f"Created at: {time.ctime(process.create\_time())}")

print(f"User time: {process.cpu\_times().user:.2f} сек")

print(f"Kernel time: {process.cpu\_times().system:.2f} сек")

try:

process.terminate()

except Exception as e:

print(f"Error")

if \_\_name\_\_ == "\_\_main\_\_":

target\_window = "GitHub Desktop"

get\_process\_info\_and\_terminate(target\_window)

**Вывод:** В результате выполнения лабораторной работы было создано GUI-приложение для мониторинга и управления процессами в системе Windows с функцией автоматического завершения процесса Блокнота, работающего более 2 минут. Изучен механизм свертывания приложения в системный трей и разработан интерфейс для отображения активных процессов. Во второй части работы была протестирована программа на C++ для получения информации о внешнем процессе и его принудительного завершения с использованием Windows API. Также разработан аналог этой программы на языке Python, что позволило сравнить подходы к управлению процессами в разных языках программирования. Освоенные методы позволяют эффективно контролировать выполнение процессов и получать системную информацию о них.