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 №1

на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ, ПОТОКАМИ, НИТЯМИ.

Выполнил: студент гр.253505 Павлович В.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177309059)

[2 Описание функций программы 4](#_Toc177309060)

[2.1 Выбор процесса для запуска 4](#_Toc177309061)

[2.2 Отображение статуса открытого процесса. 5](#_Toc177309062)

[2.3 Закрытие открытого программой процесса 6](#_Toc177309063)

[Список использованных источников 7](#_Toc177309064)

[Приложение А Исходный код программы 8](#_Toc177309065)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы возобновление, закрепление и развитие навыков программирования приложений Windows. В ходе выполнения работы необходимо изучить концепции вычислительных процессов, потоков, нитей и их реализацию в Windows; основные этапы жизненного цикла процессов (потоков) и элементарное управление ими: порождение, завершение, получение и изменение состояния; Типичное (простое) использование многозадачности и многопоточности.

В качестве задачи необходимо разработать процесс-диспетчер, а именно процесс, выполняющий:

– выбор исполняемого файла, запуск процесса из него;

– хранение списка порожденных процессов;

– отображение состояния контролируемых процессов (достаточно

различать состояния «выполняется», «завершился»);

– возможность послать сообщение WM\_CLOSE выбранному процессу;

– отображение возникающих ошибок.

В качестве контролируемых процессов можно использовать произвольные подходящие программы либо специально написанный процесс: оконное приложение, способное наглядно показывать свое выполнение.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Выбор процесса для запуска (с обработкой ошибок при выборе некорректного файла).

– Отображение статуса открытого процесса.

– Закрытие открытого программой процесса.

## **2.1 Выбор процесса для запуска**

Для открытия меню выбора исполняемого файла необходимо нажать левой кнопкой мыши по экрану приложения.

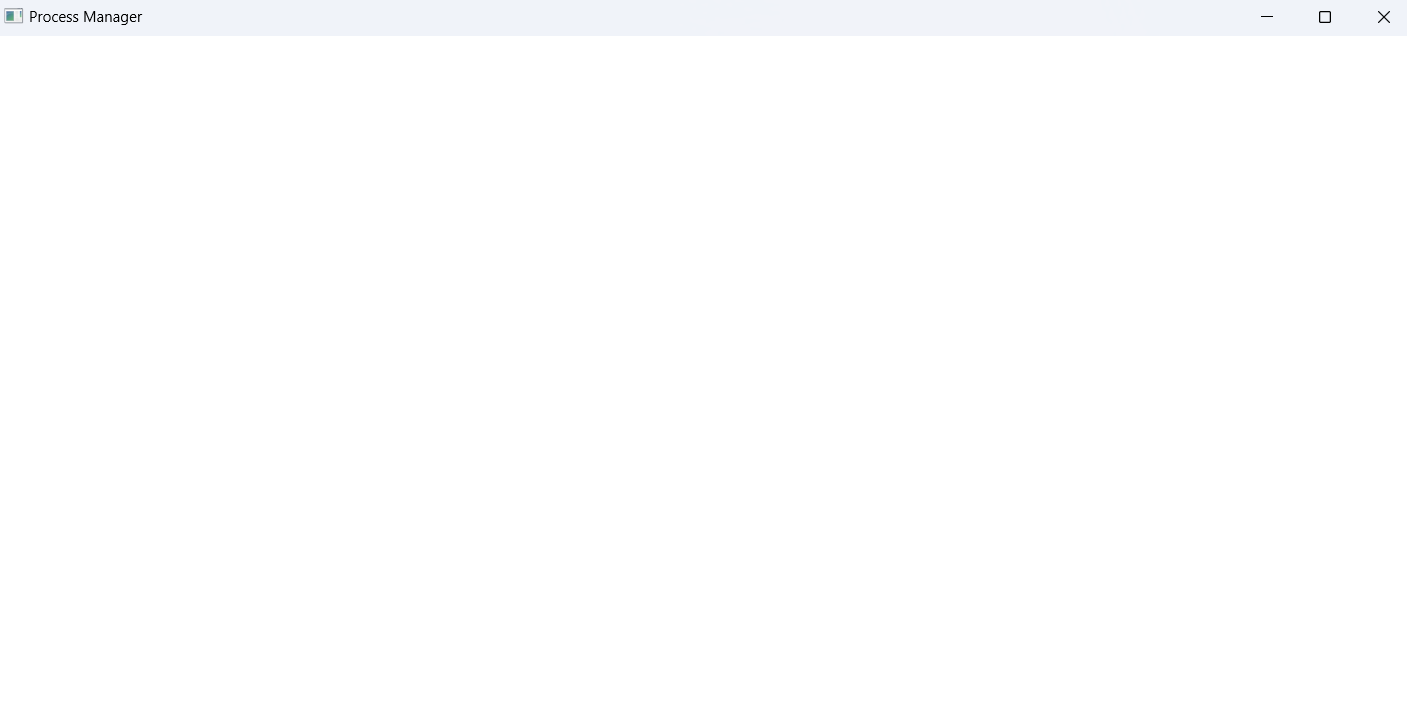


Рисунок 1 – Экран приложения

Далее следует выбрать исполняемый файл в открывшемся проводнике (рисунок 2) и нажать кнопку *Открыть*.

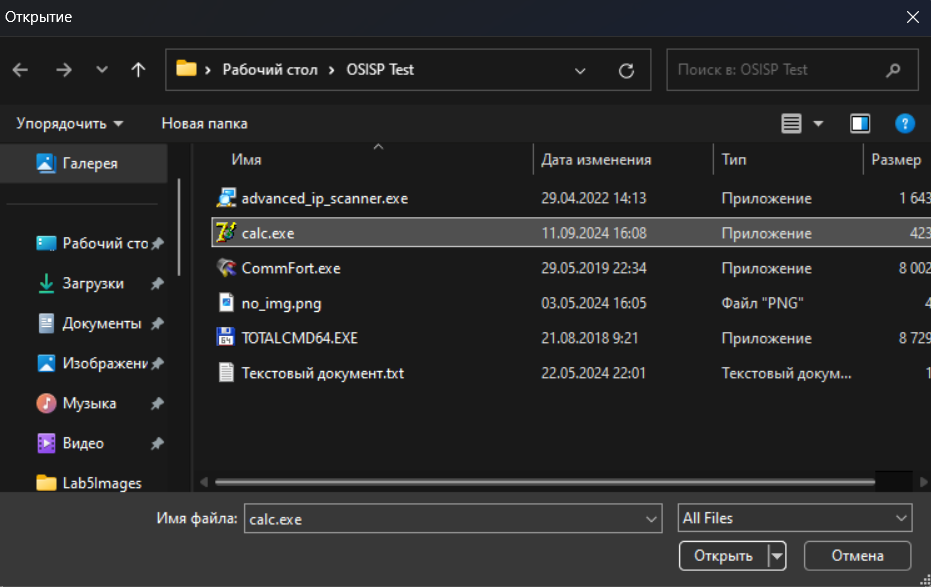


Рисунок 2 – Меню выбора файла

При попытке открыть некорректный исполняемый файл (например текстовый документ) появится всплывающее окно с сообщением об ошибке.

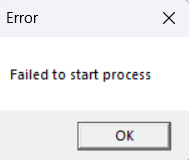


Рисунок 3 – Сообщение об ошибке

## **2.2 Отображение статуса открытого процесса.**

При выборе исполняемого файла, будет запущен процесс, а информация о нем появится на экране приложения.

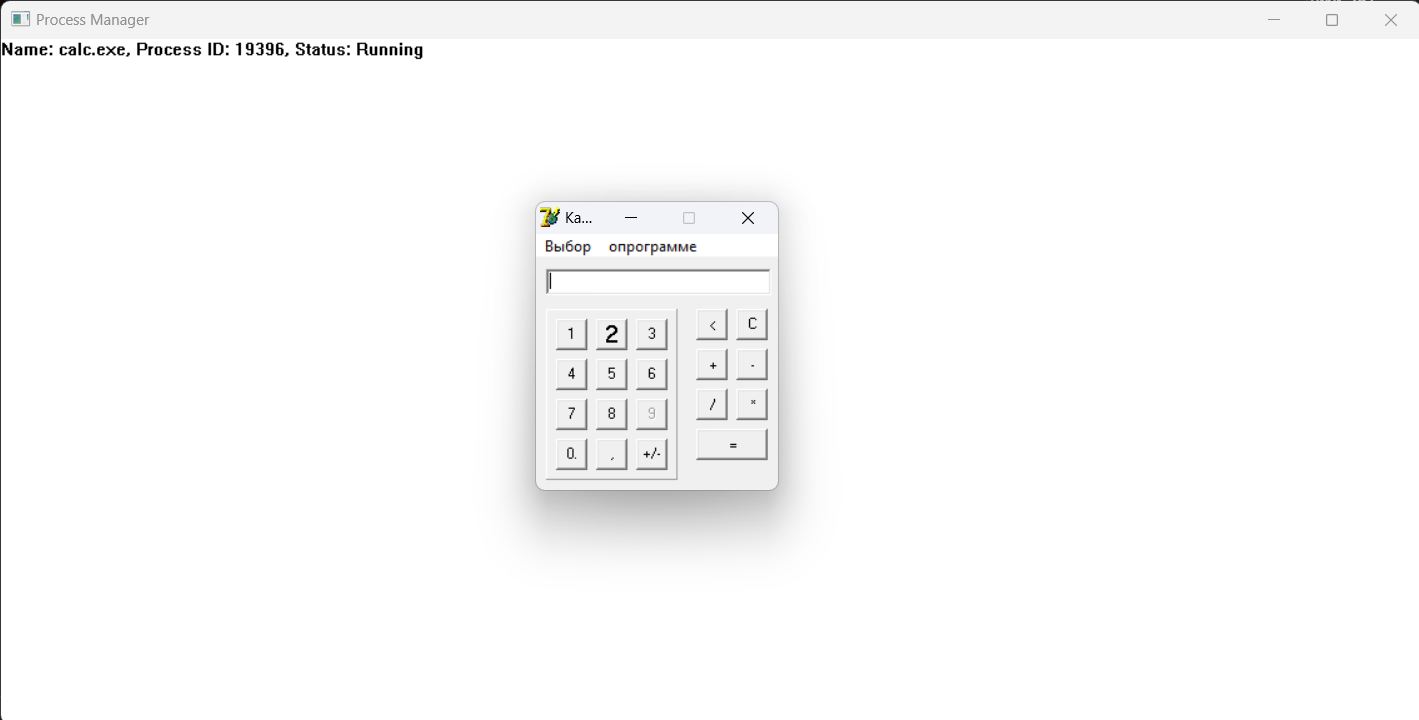


Рисунок 4 – Отображение информации о порожденном процессе

При закрытии процесса (или через само приложение, или из вне) его статус будет обновлен на завершенный.

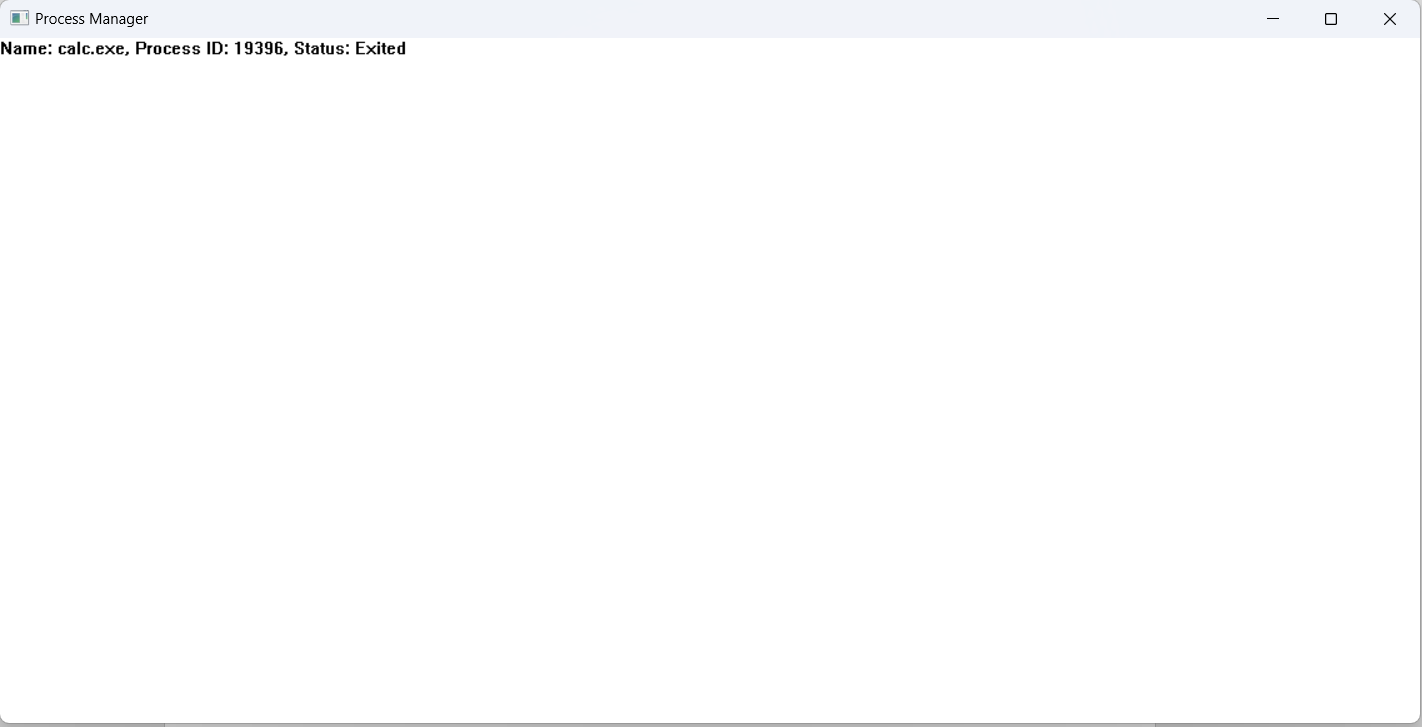


Рисунок 5 – Отображение информации о завершенном процессе

## **2.3 Закрытие открытого программой процесса**

Для закрытия открытого программой процесса необходимо нажать на строку информации о нем правой кнопкой мыши.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка приложений с помощью WinAPI. – Электронные данные. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

**Исходный код программы**

#include <windows.h>

#include <vector>

#include <string>

#include <sstream>

#include <commdlg.h>

#include <shlwapi.h>

#pragma comment(lib, "shlwapi.lib")

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

void StartProcess(const wchar\_t\* executablePath);

void CheckProcessStatus();

void DisplayProcessStatus(HDC hdc, RECT& rect);

std::wstring OpenFileDialog(HWND hwnd);

void TerminateProcessById(DWORD processId);

struct ProcessInfo {

HANDLE hProcess;

DWORD processId;

std::wstring name;

std::string status;

};

std::vector<ProcessInfo> processList;

void StartProcess(const wchar\_t\* executablePath) {

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

if (CreateProcess(

executablePath,

NULL, NULL, NULL, FALSE,

0, NULL, NULL,

&si, &pi)) {

ProcessInfo pInfo;

pInfo.hProcess = pi.hProcess;

pInfo.processId = pi.dwProcessId;

pInfo.name = PathFindFileName(executablePath);

pInfo.status = "Running";

processList.push\_back(pInfo);

CloseHandle(pi.hThread);

}

else {

MessageBox(NULL, L"Failed to start process", L"Error", MB\_OK);

}

}

void CheckProcessStatus() {

for (auto& process : processList) {

DWORD exitCode;

if (process.status != "Exited") {

if (GetExitCodeProcess(process.hProcess, &exitCode)) {

if (exitCode != STILL\_ACTIVE) {

process.status = "Exited";

CloseHandle(process.hProcess);

}

}

}

}

}

void DisplayProcessStatus(HDC hdc, RECT& rect) {

std::wstringstream ss;

for (const auto& process : processList) {

ss << L"Name: " << process.name

<< L", Process ID: " << process.processId

<< L", Status: " << process.status.c\_str() << L" \n";

}

DrawText(hdc, ss.str().c\_str(), -1, &rect, DT\_LEFT | DT\_TOP | DT\_NOCLIP);

}

std::wstring OpenFileDialog(HWND hwnd) {

OPENFILENAME ofn;

wchar\_t szFile[260] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"All Files\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName(&ofn)) {

return szFile;

}

return L"";

}

void TerminateProcessById(DWORD processId) {

for (auto process = processList.begin(); process != processList.end(); ++process) {

if (process->processId == processId) {

if (process->status != "Exited") {

TerminateProcess(process->hProcess, 0);

CloseHandle(process->hProcess);

process->status = "Exited";

break;

}

}

}

}

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

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0, CLASS\_NAME, L"Process Manager", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, NULL, hInstance, NULL);

if (hwnd == NULL) {

return 0;

}

ShowWindow(hwnd, nCmdShow);

SetTimer(hwnd, 1, 1000, NULL);

MSG msg = { };

while (GetMessage(&msg, NULL, 0, 0) > 0) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

switch (uMsg) {

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

CheckProcessStatus();

DisplayProcessStatus(hdc, ps.rcPaint);

EndPaint(hwnd, &ps);

return 0;

}

case WM\_TIMER:

InvalidateRect(hwnd, NULL, TRUE);

return 0;

case WM\_LBUTTONDOWN: {

std::wstring filePath = OpenFileDialog(hwnd);

if (!filePath.empty()) {

StartProcess(filePath.c\_str());

InvalidateRect(hwnd, NULL, TRUE);

}

return 0;

}

case WM\_RBUTTONDOWN: {

POINT pt = { LOWORD(lParam), HIWORD(lParam) };

int lineHeight = 16;

int index = pt.y / lineHeight;

if (index < processList.size()) {

TerminateProcessById(processList[index].processId);

InvalidateRect(hwnd, NULL, TRUE);

}

return 0;

}

}

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

}

int main()

{

HINSTANCE hInstance = GetModuleHandle(NULL);

LPWSTR pCmdLine = GetCommandLine();

int nCmdShow = SW\_SHOWDEFAULT;

return wWinMain(hInstance, NULL, pCmdLine, nCmdShow);

}