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 Гимпель К.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178780699)

[2 Краткие теоритические сведения 4](#_Toc178780700)

[3 Описание функций программы 5](#_Toc178780701)

[3.1 Выбор процесса для запуска 5](#_Toc178780702)

[3.2 Отображение запущенных процессов 7](#_Toc178780703)

[3.3 Закрытие открытого программой процесса 8](#_Toc178780704)

Заключение 9

Список использованных источников 10

Приложение А (обязательное) Исходный код программы 11

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является возобновление, закрепление и развитие навыков программирования приложений для 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.

Задача заключается в создании процесс-диспетчера, который должен выполнять следующие функции:

– выбор исполняемого файла, запуск процесса из него;

– хранение списка активных процессов;

– 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 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Приложение состоит из одного или нескольких процессов. Процесс, в простейших терминах, — это выполняющаяся программа. Один или несколько потоков выполняются в контексте процесса. Поток — это базовая единица, которой операционная система выделяет процессорное время. Поток может выполнять любую часть кода процесса, включая те части, которые в настоящее время выполняются другим потоком.

Создание процессов происходит через механизм порождения, при котором родительский процесс создает дочерний. В Windows создание нового процесса осуществляется с помощью функции CreateProcess(), которая позволяет создавать независимые процессы. [1]

Многозадачность (в ОС) – способность ОС выполнять параллельно (псевдопараллельно) более одной программы («задачи»), более или менее равноправных между собой, с возможностью полноценно и эффективно управлять ими.

Нити — это альтернативная модель выполнения, которая реализуется на уровне пользователя и не требует поддержки от операционной системы. В отличие от потоков, нити переключаются самим приложением, что дает разработчикам больше контроля над выполнением. [2]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В соответствии с поставленной задачей были созданы следующие функции программы:

– выбор процесса для запуска (с обработкой ошибок при выборе некорректного файла);

– отображение запущенных процессов;

– закрытие открытого программой процесса.

## **3.1 Выбор процесса для запуска**

Для запуска исполняемого файла необходимо выбрать в меню «Программа» пункт «Запустить программу» (рисунок 3.1).

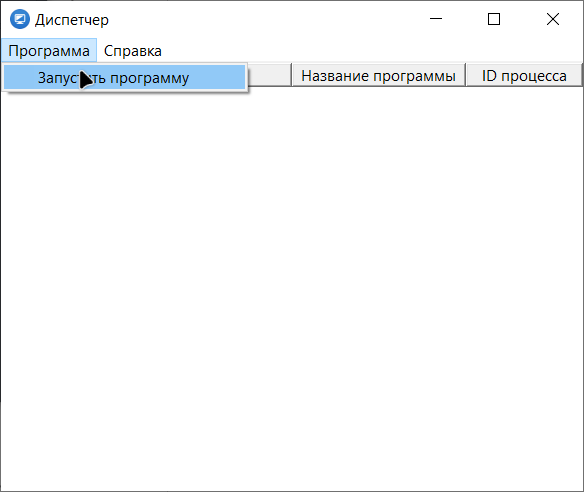


Рисунок 3.1 – Выбор пункта меню «Запустить программу»

Далее следует выбрать исполняемый файл в открывшемся проводнике (рисунок 3.2) и нажать кнопку «Открыть».

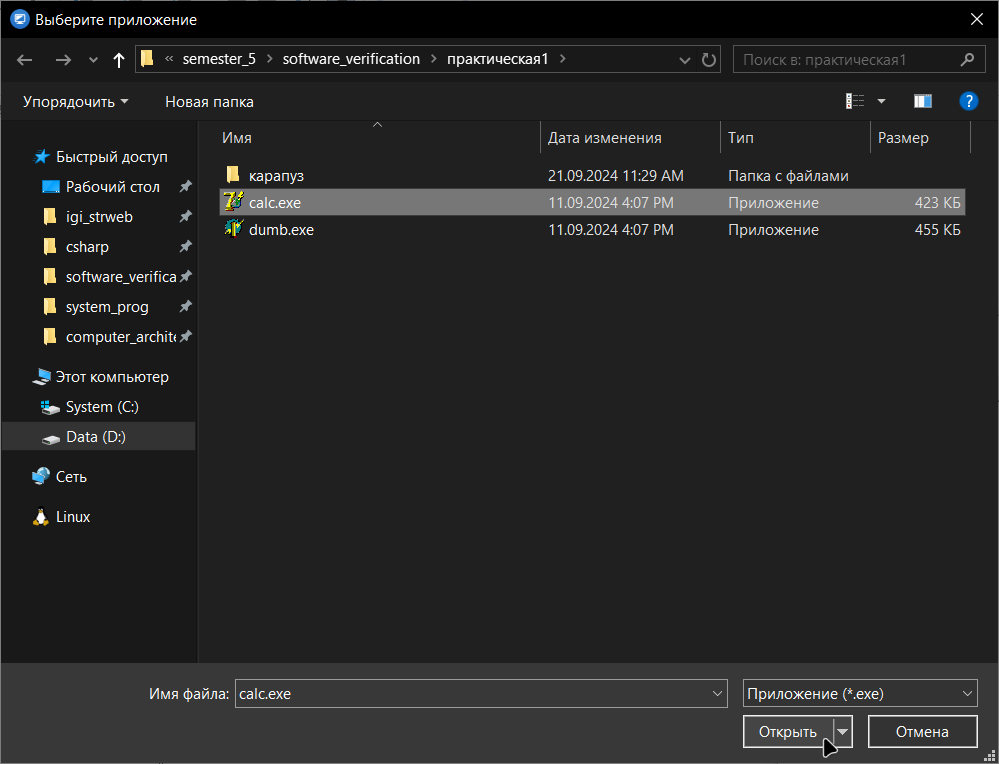


Рисунок 3.2 – Меню выбора файла

При попытке открыть некорректный исполняемый файл (например текстовый документ) появится всплывающее окно с сообщением об ошибке (рисунок 3.3).

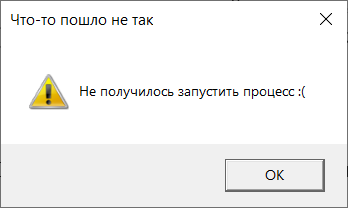


Рисунок 3.3 – Сообщение об ошибке

Сообщение об ошибке появится также, если мы попытаемя открыть файл, которого не существует (рисунок 3.4).

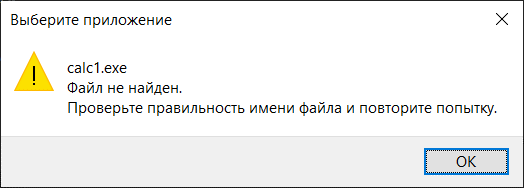


Рисунок 3.4 – Ошибка при открытии несуществующего файла

## **3.2 Отображение запущенных процессов**

При выборе исполняемого файла, будет запущен процесс, а информация о нем появится на экране приложения (рисунок 3.5).

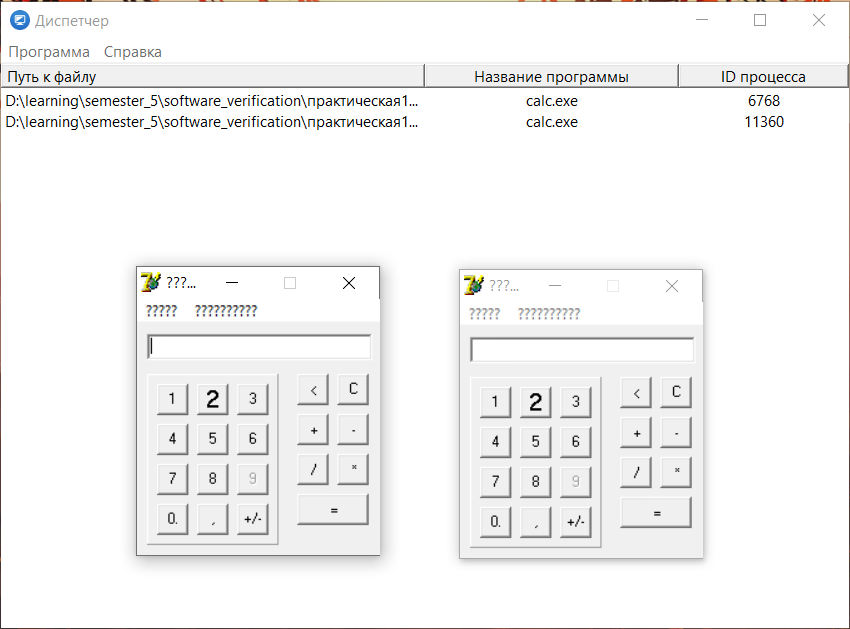


Рисунок 3.5 – Отображение информации о порожденном процессе

При закрытии процесса (через приложение или извне) он будет удален из списка (рисунок 3.6).

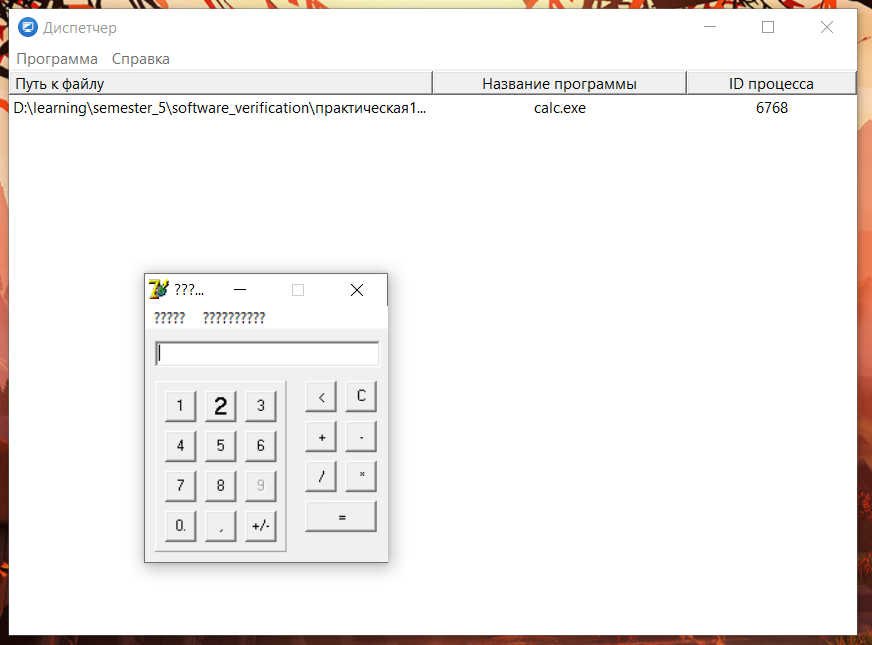


Рисунок 3.6 –Обновление списка активных процессов

## **3.3 Закрытие открытого программой процесса**

Для закрытия открытого программой процесса необходимо дважды щелкнуть правой кнопкой мыши по строке информации о нем.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по дисциплине Операционные системы и системное программирование была разработана программа для управления процессами. Основная задача состояла в том, чтобы создавать и останавливать процессы.

Во время выполнения работы был рассмотрен механизм создания процессов и их завершения, возможность отправки сообщений, таких как WM\_CLOSE, выбранному процессу показала, как взаимодействовать с оконными приложениями и обрабатывать события в Windows.

Успешное выполнение задачи по запуску исполняемых файлов и управлению активными процессами продемонстрировало понимание механизма создания процессов с использованием функции CreateProcess.

Лабораторная работа не только углубила теоретические знания о вычислительных процессах, но и предоставила практические навыки, необходимые для разработки современных приложений на платформе Windows. Эти навыки являются основой для дальнейшего изучения более сложных аспектов программирования в многопоточной среде.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Таненбаум Э. С. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2018. — 1120 с.

[2] Сильбершац А., Галвин П., Гейдж Г. Операционные системы: принципы и практика. — 9-е изд. — М.: Вильямс, 2015. — 1056 с.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#define WIN32\_LEAN\_AND\_MEAN

#include <Windows.h>

#include <windowsx.h>

#include <commdlg.h>

#include <commctrl.h>

#include <cstdio>

#include <vector>

#include <thread>

#include <mutex>

#include <string>

#pragma comment(lib, "comctl32.lib")

#include "resource.h"

#include "constant.h"

#define WINDOW\_CLASS\_NAME L"WindowClass"

#define ID\_MAIN\_LIST 400

#define MIN\_WIDTH 600

#define MIN\_HEIGHT 500

HINSTANCE appInstance;

std::vector<PROCESS\_INFORMATION> processInfos;

std::thread checker;

std::mutex mtx;

bool running = false;

LRESULT CALLBACK WinProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam);

BOOL OpenFileName(HINSTANCE hInstance, HWND hwnd, LPWSTR buffer, int sizeBuffer);

void InitListView(HINSTANCE hInstance, HWND hwnd);

BOOL LaunchApplication(LPWSTR path);

BOOL TerminateApplication(int index);

void CheckProcesses(HWND hwnd);

std::wstring GetFileName(LPWSTR filePath);

int WINAPI WinMain(

\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPSTR lpCmdLine,

\_In\_ int nShowCmd)

{

appInstance = hInstance;

WNDCLASSEX winclass;

HWND hwnd;

MSG msg;

winclass.cbSize = sizeof(winclass);

winclass.hInstance = hInstance;

winclass.lpszClassName = WINDOW\_CLASS\_NAME;

winclass.lpszMenuName = NULL;

winclass.lpfnWndProc = WinProc;

winclass.cbClsExtra = 0;

winclass.cbWndExtra = 0;

winclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

winclass.hCursor = LoadCursor(NULL, IDC\_ARROW);

winclass.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(MY\_APP\_ICON));

winclass.hIconSm = LoadIcon(hInstance, MAKEINTRESOURCE(MY\_APP\_ICON));

winclass.style = CS\_DBLCLKS | CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;

if (!RegisterClassEx(&winclass)) {

return 0;

}

HMENU hmenu = LoadMenu(hInstance, MAKEINTRESOURCE(MainMenu));

hwnd = CreateWindowEx(

NULL,

WINDOW\_CLASS\_NAME,

L"Диспетчер",

WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,

500, 200, MIN\_WIDTH, MIN\_HEIGHT,

NULL,

hmenu,

hInstance,

NULL

);

if (!hwnd) {

return 0;

}

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LRESULT CALLBACK WinProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam) {

switch (msg) {

case WM\_CREATE: {

InitListView(appInstance, hwnd);

running = true;

checker = std::thread(CheckProcesses, hwnd);

return 0;

}

case WM\_COMMAND: {

switch (LOWORD(wparam)) {

case MENU\_PROG\_START: {

WCHAR path[260];

if (OpenFileName(appInstance, hwnd, path, sizeof(path))) {

mtx.lock();

if (LaunchApplication(path)) {

HWND hwndList = GetDlgItem(hwnd, ID\_MAIN\_LIST);

int itemCount = SendMessage(hwndList, LVM\_GETITEMCOUNT, 0, 0);

LVITEM lvi;

lvi.mask = LVIF\_TEXT;

lvi.iItem = itemCount;

lvi.iSubItem = 0;

lvi.pszText = path;

SendMessage(hwndList, LVM\_INSERTITEM, itemCount, (LPARAM)&lvi);

lvi.iSubItem = 1;

std::wstring filename = GetFileName(path);

lvi.pszText = const\_cast<LPWSTR>(filename.c\_str());

SendMessage(hwndList, LVM\_SETITEMTEXT, itemCount, (LPARAM)&lvi);

lvi.iSubItem = 2;

std::wstring processIdStr = std::to\_wstring(processInfos.back().dwProcessId);

lvi.pszText = const\_cast<LPWSTR>(processIdStr.c\_str());

SendMessage(hwndList, LVM\_SETITEMTEXT, itemCount, (LPARAM)&lvi);

}

else {

MessageBox(hwnd, L"Не получилось запустить процесс :(",

L"Что-то пошло не так", MB\_OK | MB\_ICONWARNING);

}

mtx.unlock();

}

return 0;

}

case MENU\_ABOUT: {

MSGBOXPARAMS msgbox = { 0 };

msgbox.cbSize = sizeof(MSGBOXPARAMS);

msgbox.hwndOwner = hwnd;

msgbox.hInstance = appInstance;

msgbox.lpszText = PROGRAM\_DESCRIPTION;

msgbox.lpszCaption = L"О программе";

msgbox.dwStyle = MB\_OK | MB\_USERICON;

msgbox.lpszIcon = MAKEINTRESOURCE(MY\_INFO\_ICON);

MessageBoxIndirect(&msgbox);

//MessageBox(hwnd, PROGRAM\_DESCRIPTION, L"О программе", MB\_OK );

return 0;

}

}

break;

}

case WM\_SIZE: {

HWND hwndList = GetDlgItem(hwnd, ID\_MAIN\_LIST);

RECT rc;

GetClientRect(hwnd, &rc);

MoveWindow(hwndList, 0, 0, rc.right, rc.bottom, TRUE);

int totalWidth = rc.right;

double k[3] = { 0.5, 0.3, 0.2 };

int newWidth[3]{};

int sum = 0;

for (int i = 0; i < 2; ++i) {

newWidth[i] = k[i] \* totalWidth;

sum += newWidth[i];

}

newWidth[2] = totalWidth - sum;

for (int i = 0; i < 3; i++) {

ListView\_SetColumnWidth(hwndList, i, newWidth[i]);

}

return 0;

}

case WM\_GETMINMAXINFO: {

MINMAXINFO\* mmi = (MINMAXINFO\*)lparam;

mmi->ptMinTrackSize.x = MIN\_WIDTH;

mmi->ptMinTrackSize.y = MIN\_HEIGHT;

return 0;

}

case WM\_NOTIFY: {

switch (LOWORD(wparam)) {

case ID\_MAIN\_LIST: {

switch (((LPNMHDR)lparam)->code) {

case LVN\_ITEMACTIVATE: {

HWND hListView = GetDlgItem(hwnd, ID\_MAIN\_LIST);

int index = ListView\_GetNextItem(hListView, -1, LVNI\_SELECTED);

if (index != -1) {

// Если процесс уже завершен, то выведем сообщение об этом

DWORD exitCode;

if (GetExitCodeProcess(processInfos[index].hProcess, &exitCode)) {

if (exitCode != STILL\_ACTIVE) {

MessageBox(hwnd, L"Процесс уже был завершен, подождите пока обновится диспетчер",

L"Терпения мой друг", MB\_OK | MB\_ICONINFORMATION);

return 0;

}

}

if (TerminateApplication(index)) {

SendMessage(hListView, LVM\_DELETEITEM, index, 0);

}

else {

MessageBox(hwnd, L"Не получилось завершить процесс :(",

L"Что-то пошло не так", MB\_OK | MB\_ICONWARNING);

}

}

return 0;

}

}

break;

}

}

break;

}

case WM\_CLOSE: {

if (!processInfos.empty()) {

int result = MessageBox(hwnd, L"Закрыть приложение?", L"Подтверждение", MB\_YESNO | MB\_ICONQUESTION);

if (result == IDNO) {

return 0;

}

}

running = false;

checker.join();

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

return 0;

}

}

return DefWindowProc(hwnd, msg, wparam, lparam);

}

BOOL OpenFileName(HINSTANCE hInstance, HWND hwnd, LPWSTR buffer, int sizeBuffer) {

OPENFILENAME ofn = {};

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.hInstance = hInstance;

ofn.lpstrFilter = L"Приложение (\*.exe)\0\*.exe\0\0";

ofn.nFilterIndex = 1;

ofn.lpstrCustomFilter = NULL;

ofn.lpstrFile = buffer;

ofn.lpstrFile[0] = '\0';

ofn.nMaxFile = sizeBuffer;

ofn.lpstrFileTitle = NULL;

ofn.lpstrInitialDir = NULL;

ofn.lpstrTitle = L"Выберите приложение";

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

return GetOpenFileName(&ofn);

}

void InitListView(HINSTANCE hInstance, HWND hwnd) {

HWND hwndList = CreateWindowEx(NULL, WC\_LISTVIEW, L"",

WS\_CHILD | WS\_VISIBLE | LVS\_REPORT | LVS\_SINGLESEL,

0, 0, MIN\_WIDTH, MIN\_HEIGHT,

hwnd, (HMENU)ID\_MAIN\_LIST, hInstance, NULL);

SendMessage(hwndList, LVM\_SETEXTENDEDLISTVIEWSTYLE, 0,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_TWOCLICKACTIVATE);

LVCOLUMN lvc;

lvc.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

lvc.fmt = LVCFMT\_CENTER;

lvc.cx = 150;

lvc.pszText = const\_cast<LPWSTR>(L"Путь к файлу");

SendMessage(hwndList, LVM\_INSERTCOLUMN, 0, (LPARAM)&lvc);

lvc.cx = 100;

lvc.pszText = const\_cast<LPWSTR>(L"Название программы");

SendMessage(hwndList, LVM\_INSERTCOLUMN, 1, (LPARAM)&lvc);

lvc.cx = 100;

lvc.pszText = const\_cast<LPWSTR>(L"ID процесса");

SendMessage(hwndList, LVM\_INSERTCOLUMN, 2, (LPARAM)&lvc);

}

BOOL LaunchApplication(LPWSTR path) {

STARTUPINFO si = {};

PROCESS\_INFORMATION pi = {};

si.cb = sizeof(si);

// Запускаем приложение

BOOL ok = CreateProcess(

path, // путь к исполняемому файлу

NULL, // командная строка

NULL, // процесс не наследует дескрипторы

NULL, // поток не наследует дескрипторы

FALSE, // не наследовать дескрипторы

0, // дополнительные флаги

NULL,

NULL, // использовать текущую директорию

&si, // информация о запуске

&pi // информация о процессе

);

if (!ok) {

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

processInfos.push\_back(pi);

WaitForInputIdle(pi.hProcess, INFINITE);

}

return ok;

}

BOOL TerminateApplication(int index) {

mtx.lock();

PROCESS\_INFORMATION pi = processInfos[index];

BOOL ok = TerminateProcess(pi.hProcess, 0);

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

processInfos.erase(processInfos.begin() + index);

mtx.unlock();

return ok;

}

void CheckProcesses(HWND hwnd) {

HWND hwndList = GetDlgItem(hwnd, ID\_MAIN\_LIST);

while (running) {

mtx.lock();

for (int i = 0; i < processInfos.size(); ) {

DWORD exitCode;

if (GetExitCodeProcess(processInfos[i].hProcess, &exitCode)) {

if (exitCode != STILL\_ACTIVE) {

CloseHandle(processInfos[i].hProcess);

CloseHandle(processInfos[i].hThread);

processInfos.erase(processInfos.begin() + i);

SendMessage(hwndList, LVM\_DELETEITEM, i, 0);

continue;

}

}

++i;

}

mtx.unlock();

std::this\_thread::sleep\_for(std::chrono::milliseconds(200));

}

}

std::wstring GetFileName(LPWSTR filePath) {

std::wstring path(filePath);

size\_t lastSlash = path.find\_last\_of(L"\\/");

return path.substr(lastSlash + 1);

}