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)**

Выполнил:

студент группы 153503

Скворцов А. В.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[3 Результаты выполнения лабораторной работы 5](#_Toc146728179)

[Выводы 7](#_Toc146728180)

[Список использованных источников 8](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 9](#_Toc146728182)

## 1 ЦЕЛЬ РАБОТЫ

## Целью лабораторной работы является исследование и оценка эффективности управления процессами и потоками в операционной системе Windows, а также проведение анализа и экспериментов, направленных на определение оптимальных стратегий порождения, завершения и изменения приоритетов процессов и потоков с целью повышения общей производительности и ресурсоэффективности системы.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Управление процессами и потоками в операционной системе Windows является важной задачей для обеспечения эффективной работы системы и оптимального использования ресурсов. В данной лабораторной работе рассматриваются следующие аспекты:

1. Завершение процессов и потоков: Завершение процессов и потоков может происходить по различным причинам, включая завершение работы программы, ошибки выполнения или запросы пользователя. Для завершения процессов и потоков в Windows используются функции TerminateProcess и TerminateThread [1]. Важно учитывать последствия завершения процессов и потоков, такие как освобождение занимаемых ресурсов и корректное завершение работы программы.
2. Приостановка процессов: Приостановка процесса означает временную остановку его выполнения без завершения. В Windows для приостановки процесса используется функция SuspendThread, которая приостанавливает все потоки, связанные с процессом. Приостановка процесса может быть полезной в таких случаях, как приоритетное выполнение других процессов или выполнение критически важных операций.
3. Возобновление процессов: Возобновление процесса позволяет продолжить его выполнение после приостановки. Для возобновления процесса в Windows используется функция ResumeThread, которая возобновляет выполнение всех потоков, связанных с процессом [2]. Возобновление процесса позволяет ему продолжить свою работу после временной приостановки.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было разработано приложение для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы. На главном экране пользователь может видеть все запущенные системные процессы (рисунок 1).

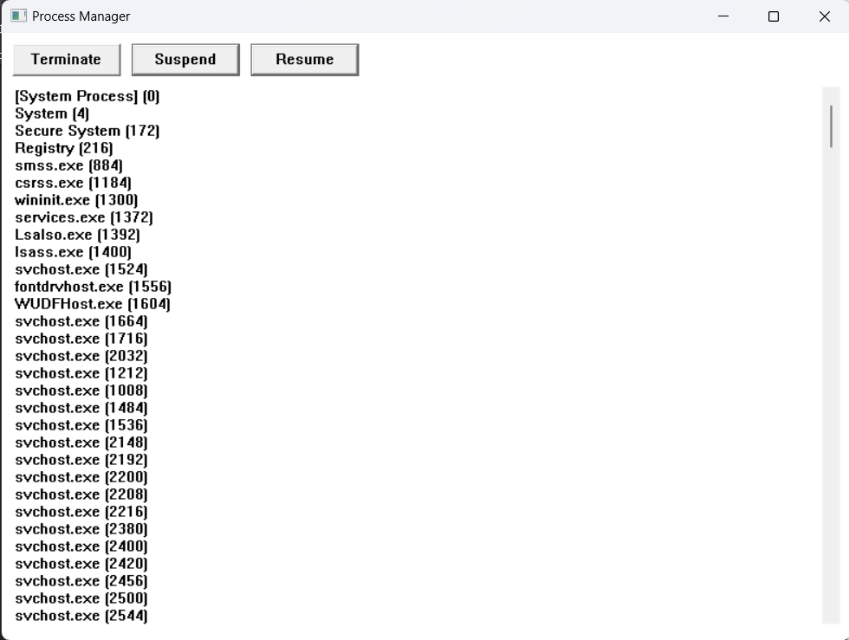


Рисунок 1 - Главный экран

Пользователь может выбрать процесс из списка для дальнейшего взаимодействия с ним (рисунок 2).

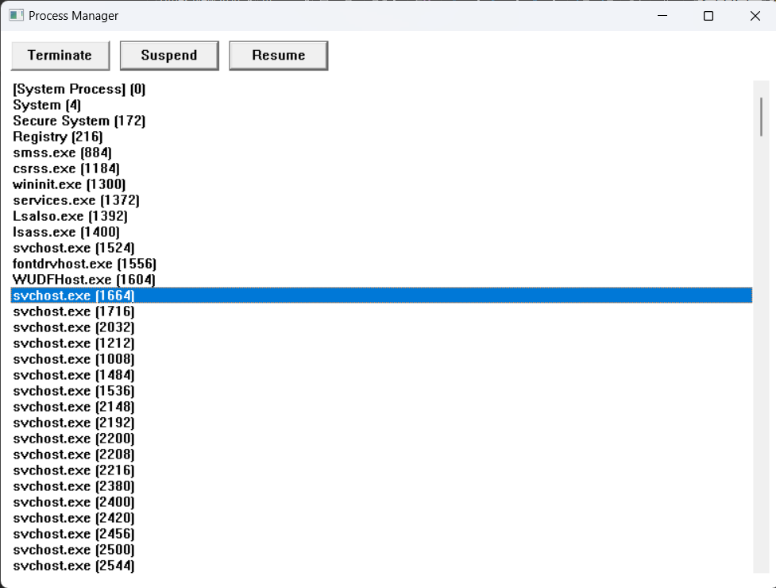


Рисунок 2 ─ Выбор процесса

Пользователю предоставляется возможность совершить одно из трех действий: завершить процесс, приостановить процесс, возобновить процесс (рисунок 3).

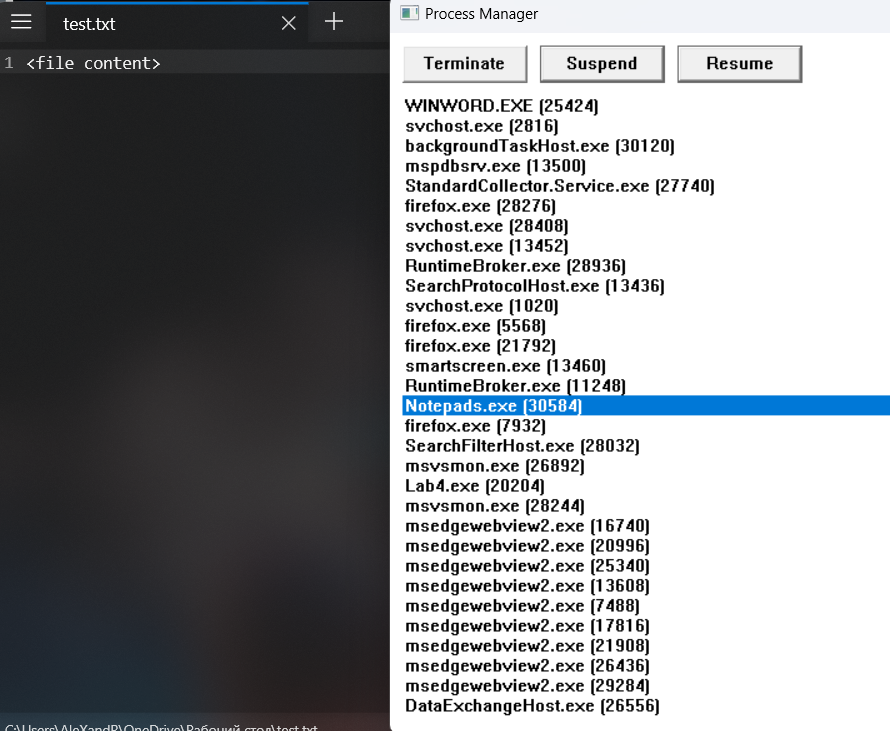


Рисунок 3 – Приостановка процесса

В приведенном примере была нажата кнопка для приостановки процесса и, соответственно, приложение «Блокнот» в данном случае перестало отвечать на какие-либо взаимодействия с интерфейсом, следовательно, процесс приостановил свое выполнение.

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было разработано приложение, предназначенное для отслеживания и управления процессами в операционной системе Windows. Целью работы было изучение основных принципов управления процессами и потоками, а также исследование эффективности различных операций, таких как порождение, завершение и изменение приоритетов процессов и потоков. В ходе разработки приложения были реализованы функции, позволяющие пользователю просматривать список активных процессов в системе, а также приостанавливать, возобновлять и завершать выбранные процессы.

Таким образом, выполнение данной лабораторной работы позволило изучить основные принципы управления процессами и потоками в операционной системе Windows, а также разработать приложение, способное эффективно управлять процессами в системе.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ил.
2. Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 19.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

Листинг 1 – Файл main.cpp

#include <windows.h>

#include <tlhelp32.h>

#include <string>

#include <regex>

#define IDC\_PROCESS\_LIST 1001

#define IDC\_TERMINATE\_BUTTON 1002

#define IDC\_SUSPEND\_BUTTON 1003

#define IDC\_RESUME\_BUTTON 1004

HWND hwndList = NULL;

HWND buttonTerminate = NULL;

HWND buttonSuspend = NULL;

HWND buttonResume = NULL;

DWORD selectedPId = -1;

std::regex processRegex("\((\\d+)\)");

VOID showErrorAndExit(const char\* errorMessage);

VOID suspendProcess(DWORD processId);

VOID resumeProcess(DWORD processId);

VOID terminateProcess(DWORD processId);

VOID getProcessList(HWND hwndList);

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

switch (uMsg) {

case WM\_CREATE:

{

buttonTerminate = CreateWindow(

L"BUTTON",

L"Terminate",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

10, 19, 100, 30, hwnd, (HMENU)IDC\_TERMINATE\_BUTTON, (HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE), NULL

);

if (buttonTerminate == NULL) {

MessageBoxA(NULL, "Failed to create terminate button", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

buttonSuspend = CreateWindow(

L"BUTTON",

L"Suspend",

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

120, 10, 100, 30,

hwnd,

(HMENU)IDC\_SUSPEND\_BUTTON,

(HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE),

NULL

);

if (buttonSuspend == NULL) {

MessageBoxA(NULL, "Failed to create suspend button", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

buttonResume = CreateWindow(

L"BUTTON",

L"Resume",

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

230, 10, 100, 30,

hwnd,

(HMENU)IDC\_RESUME\_BUTTON,

(HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE),

NULL

);

if (buttonResume == NULL) {

MessageBoxA(NULL, "Failed to create resume button", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

hwndList = CreateWindowEx(

0,

L"LISTBOX",

NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_AUTOVSCROLL | LBS\_NOTIFY | LBS\_HASSTRINGS,

10, 50, 780, 500, hwnd,

NULL,

(HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE),

NULL

);

if (hwndList == NULL) {

MessageBoxA(NULL, "Failed to create list box", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

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

EndPaint(hwnd, &ps);

return 0;

}

case WM\_COMMAND:

{

switch (HIWORD(wParam)) {

case LBN\_SELCHANGE:

{

int selectedIndex = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (selectedIndex != LB\_ERR) {

int textLen = SendMessage(hwndList, LB\_GETTEXTLEN, (WPARAM)selectedIndex, 0);

wchar\_t\* result = new wchar\_t[textLen + 1];

SendMessage(hwndList, LB\_GETTEXT, (WPARAM)selectedIndex, reinterpret\_cast<LPARAM>(result));

std::wstring res(result);

std::string res1(res.begin(), res.end());

std::cmatch m;

if (std::regex\_search(res1.c\_str(), m, processRegex)) {

selectedPId = atoi(m.str(1).c\_str());

}

}

return 0;

}

case BN\_CLICKED:

{

if ((HWND)lParam == buttonTerminate) {

if (selectedPId == -1) {

MessageBoxA(NULL, "Choose process!", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

terminateProcess(selectedPId);

} else if ((HWND)lParam == buttonSuspend) {

if (selectedPId == -1) {

MessageBoxA(NULL, "Choose process!", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

suspendProcess(selectedPId);

} else if ((HWND)lParam == buttonResume) {

if (selectedPId == -1) {

MessageBoxA(NULL, "Choose process!", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

resumeProcess(selectedPId);

}

}

}

return 0;

}

case WM\_SIZE:

{

int buttonWidth = 100;

int buttonHeight = 30;

int buttonMargin = 10;

int listBoxMarginTop = 50;

int listBoxMarginBottom = 10;

int windowWidth = LOWORD(lParam);

int windowHeight = HIWORD(lParam);

MoveWindow(buttonTerminate, buttonMargin, buttonMargin, buttonWidth, buttonHeight, true);

MoveWindow(buttonSuspend, buttonMargin + buttonWidth + buttonMargin, buttonMargin, buttonWidth, buttonHeight, true);

MoveWindow(buttonResume, buttonMargin + buttonWidth + buttonMargin + buttonWidth + buttonMargin, buttonMargin, buttonWidth, buttonHeight, true);

MoveWindow(hwndList, buttonMargin, listBoxMarginTop, windowWidth - 2 \* buttonMargin, windowHeight - listBoxMarginTop - listBoxMarginBottom, true);

return 0;

}

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

default:

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

}

}

VOID showErrorAndExit(const char\* errorMessage) {

MessageBoxA(NULL, errorMessage, "Error", MB\_OK | MB\_ICONERROR);

}

VOID suspendProcess(DWORD processId) {

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

if (hThreadSnapshot == NULL) {

showErrorAndExit("Failed to open process");

return;

}

THREADENTRY32 threadEntry;

threadEntry.dwSize = sizeof(THREADENTRY32);

Thread32First(hThreadSnapshot, &threadEntry);

do {

if (threadEntry.th32OwnerProcessID == processId) {

HANDLE hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE,

threadEntry.th32ThreadID);

if (hThread == NULL) {

showErrorAndExit("Failed to open process");

return;

}

SuspendThread(hThread);

CloseHandle(hThread);

}

} while (Thread32Next(hThreadSnapshot, &threadEntry));

CloseHandle(hThreadSnapshot);

getProcessList(hwndList);

}

VOID resumeProcess(DWORD processId) {

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

if (hThreadSnapshot == NULL) {

showErrorAndExit("Failed to open process");

return;

}

THREADENTRY32 threadEntry;

threadEntry.dwSize = sizeof(THREADENTRY32);

Thread32First(hThreadSnapshot, &threadEntry);

do {

if (threadEntry.th32OwnerProcessID == processId) {

HANDLE hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE,

threadEntry.th32ThreadID);

if (hThread == NULL) {

showErrorAndExit("Failed to open process");

return;

}

ResumeThread(hThread);

CloseHandle(hThread);

}

} while (Thread32Next(hThreadSnapshot, &threadEntry));

CloseHandle(hThreadSnapshot);

getProcessList(hwndList);

}

VOID terminateProcess(DWORD processId) {

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

if (hThreadSnapshot == NULL) {

showErrorAndExit("Failed to open process");

return;

}

THREADENTRY32 threadEntry;

threadEntry.dwSize = sizeof(THREADENTRY32);

Thread32First(hThreadSnapshot, &threadEntry);

do {

if (threadEntry.th32OwnerProcessID == processId) {

HANDLE hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE,

threadEntry.th32ThreadID);

if (hThread == NULL) {

showErrorAndExit("Failed to open process");

return;

}

TerminateThread(hThread, 0);

CloseHandle(hThread);

}

} while (Thread32Next(hThreadSnapshot, &threadEntry));

CloseHandle(hThreadSnapshot);

getProcessList(hwndList);

}

VOID getProcessList(HWND hwndList) {

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

if (hSnapshot == INVALID\_HANDLE\_VALUE) {

MessageBoxA(NULL, "Failed to create process snapshot", "Error", MB\_OK | MB\_ICONERROR);

return;

}

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

if (!Process32First(hSnapshot, &pe32)) {

CloseHandle(hSnapshot);

MessageBoxA(NULL, "Failed to retrieve process information", "Error", MB\_OK | MB\_ICONERROR);

return;

}

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

do {

std::wstring processName = std::wstring(pe32.szExeFile) + L" (" + std::to\_wstring(pe32.th32ProcessID) + L")";

INT pId = pe32.th32ProcessID;

SendMessageW(hwndList, LB\_ADDSTRING, 0, reinterpret\_cast<LPARAM>(processName.c\_str()));

} while (Process32Next(hSnapshot, &pe32));

CloseHandle(hSnapshot);

}

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

const wchar\_t CLASS\_NAME[] = L"ProcessListWindowClass";

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, 800, 600,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

MessageBoxA(NULL, "Failed to create window", "Error", MB\_OK | MB\_ICONERROR);

return 0;

}

getProcessList(hwndList);

ShowWindow(hwnd, nCmdShow);

MSG msg = {};

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}