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

ОТЧЕТ

к лабораторной работе №4

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ**

**И ПОТОКАМИ (WINDOWS)**

Студент И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Исследовать и овладеть методами управления процессами и потоками в операционной системе *Windows*, включая порождение, завершение и изменение приоритетов процессов и потоков, а также провести анализ и оценку эффективности данных операций с целью оптимизации работы системы и ресурсов.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе *Windows* является фундаментальной частью обеспечения эффективной работы приложений. Процесс представляет собой изолированное исполняемое окружение, включающее в себя код, данные и системные ресурсы, такие как файлы и память. Внутри процессов существуют потоки, которые представляют собой независимые исполняемые единицы в рамках процесса. Потоки могут выполнять параллельные задачи, что повышает отзывчивость и производительность приложений. [1]

Порождение процессов и потоков в *Windows* позволяет создавать новые задачи для выполнения. Это может быть полезно, например, при запуске новых приложений или при одновременном выполнении разных частей одного приложения.

Завершение процессов и потоков важно для освобождения ресурсов и предотвращения утечек памяти. Ошибки в управлении завершением могут привести к нежелательным последствиям, таким как "зависание" приложений или даже сбои операционной системы. Создание процессов: в *Windows* процессы создаются с помощью функции *CreateProcess*, которая запускает новое приложение в отдельном процессе. Это позволяет приложениям работать независимо друг от друга. [2]

Потоки создаются с помощью функции *CreateThread*. Каждый поток имеет свою собственную функцию выполнения, что позволяет параллельно выполнять различные задачи в рамках одного процесса.

*CreateToolhelp32Snapshot* это функция *WinAPI*, используемая для создания снимка текущего состояния процессов, потоков и модулей в системе. Она позволяет получить информацию о запущенных процессах и их связанных компонентах.

В *Windows* каждому процессу и потоку назначается приоритет выполнения. Это позволяет операционной системе распределять процессорное время между процессами и потоками. Приоритеты включают в себя высокий, нормальный, низкий и другие уровни.

Приоритеты могут быть изменены во время выполнения приложения. Это может быть полезным для управления использованием процессорного времени и ресурсов. Повышение приоритета может ускорить выполнение задачи, а понижение - предоставить больше ресурсов другим задачам.

Правильное управление процессами и потоками обеспечивает отзывчивость системы и возможность параллельного выполнения задач. Это важно для многозадачных приложений и систем с множеством пользователей.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Было создано приложение для отслеживания и управления процессами в системе, позволяющее завершать процессы. (рисунок 1).

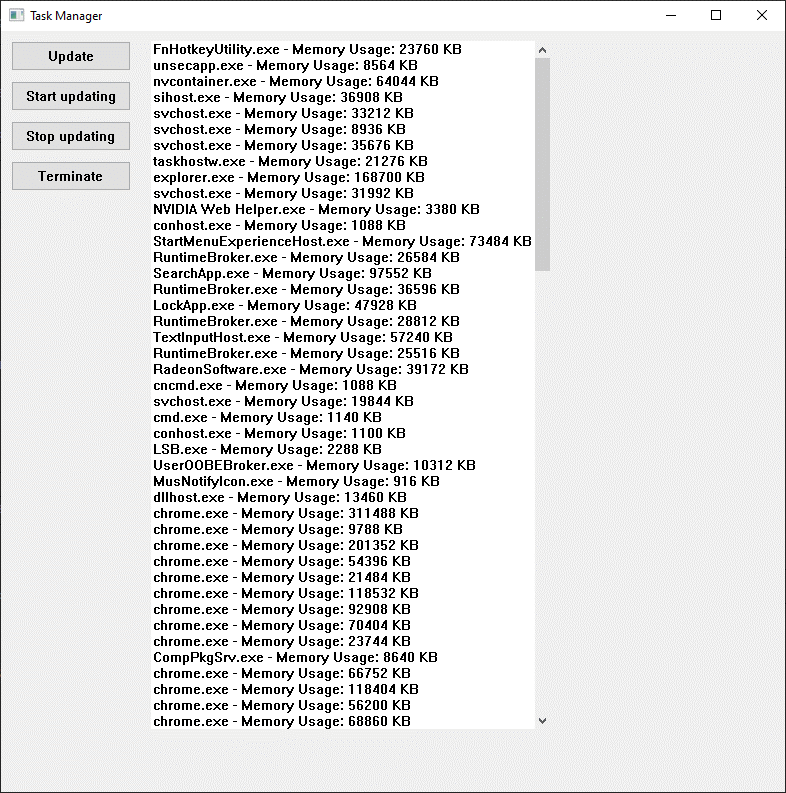


Рисунок 1 – Окно приложения

Для завершения процесса следует выбрать из списка процесс и нажать кнопку “*Terminate*”. И в зависимости от сложившейся ситуации процесс может быть завершен, или же пользователь будет уведомлен что данного процесса уже нет.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения данной лабораторной работы было разработано и рассмотрено приложение, предназначенное для мониторинга процессов и управления ими. Приложение демонстрирует использование основных принципов управления процессами в операционной системе Windows, а также применение механизмов управления приоритетами потоков и обеспечения отзывчивости системы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/>.
2. Процессы и потоки [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/\_processthreadsapi/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Source.cpp*

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#include <windows.h>

#include <tchar.h>

#include <vector>

#include <string>

#include <tlhelp32.h>

#include <iostream>

#include <windows.h>

#include <tchar.h>

#include <process.h>

#include <Psapi.h>

#define ID\_TERMINATE 1001

#define ID\_UPDATE 1002

#define ID\_STOP\_UPDATING 1003

#define ID\_START\_UPDATING 1004

#define ID\_PROCESS\_LIST 1005

#define UPDATE\_TIME 1000

HANDLE g\_hThread = NULL;

HANDLE g\_hThreadMutex = NULL;

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);

void UpdateProcessList(HWND hwnd);

void TerminateProcces(HWND hwnd);

void StartUpdateThread(HWND hwnd);

void StopUpdateThread();

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

MSG msg;

HWND hwnd;

WNDCLASSEX wc;

wc.cbSize = sizeof(WNDCLASSEX);

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW);

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

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpfnWndProc = WndProc;

wc.lpszClassName = L"TaskManager";

wc.lpszMenuName = NULL;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

if (!RegisterClassEx(&wc))

{

MessageBox(NULL, L"Failed to register window class.", L"Error", MB\_ICONERROR | MB\_OK);

return EXIT\_FAILURE;

}

hwnd = CreateWindow(wc.lpszClassName, L"Task Manager", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 800, NULL, NULL, wc.hInstance, NULL);

if (hwnd == INVALID\_HANDLE\_VALUE)

{

MessageBox(NULL, L"Failed to create a window.", L"Error", MB\_ICONERROR);

return EXIT\_FAILURE;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

switch (msg)

{

case WM\_CREATE:

{

HWND hUpdateButton = CreateWindow(L"BUTTON", L"Update", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 10, 10, 120, 30, hwnd, (HMENU)ID\_UPDATE, NULL, NULL);

HWND hStartUpdating = CreateWindow(L"BUTTON", L"Start updating", WS\_VISIBLE | WS\_CHILD, 10, 50, 120, 30, hwnd, (HMENU)ID\_START\_UPDATING, NULL, NULL);

HWND hStopUpdating = CreateWindow(L"BUTTON", L"Stop updating", WS\_VISIBLE | WS\_CHILD, 10, 90, 120, 30, hwnd, (HMENU)ID\_STOP\_UPDATING, NULL, NULL);

HWND hTerminateButton = CreateWindow(L"BUTTON", L"Terminate", WS\_VISIBLE | WS\_CHILD, 10, 130, 120, 30, hwnd, (HMENU)ID\_TERMINATE, NULL, NULL);

HWND hProcessList = CreateWindow(L"LISTBOX", L"", WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | LBS\_DISABLENOSCROLL | LBS\_NOTIFY, 150, 10, 400, 700, hwnd, (HMENU)ID\_PROCESS\_LIST, NULL, NULL);

StartUpdateThread(hwnd);

g\_hThreadMutex = CreateMutex(NULL, FALSE, NULL);

break;

}

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case ID\_STOP\_UPDATING:

{

StopUpdateThread();

break;

}

case ID\_START\_UPDATING:

{

StartUpdateThread(hwnd);

break;

}

case ID\_UPDATE:

{

UpdateProcessList(hwnd);

break;

}

case ID\_TERMINATE:

{

TerminateProcces(hwnd);

break;

}

}

break;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

default:

{

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

}

}

return 0;

}

DWORD WINAPI UpdateProcessListThread(LPVOID lpParam) {

while (true)

{

UpdateProcessList((HWND)lpParam);

Sleep(UPDATE\_TIME);

}

return 0;

}

void StartUpdateThread(HWND hwnd) {

WaitForSingleObject(g\_hThreadMutex, INFINITE);

if (g\_hThread == NULL)

{

g\_hThread = CreateThread(NULL, 0, UpdateProcessListThread, (LPVOID)hwnd, 0, NULL);

}

ReleaseMutex(g\_hThreadMutex);

}

void StopUpdateThread() {

WaitForSingleObject(g\_hThreadMutex, INFINITE);

if (g\_hThread != NULL) {

TerminateThread(g\_hThread, 0);

CloseHandle(g\_hThread);

g\_hThread = NULL;

}

ReleaseMutex(g\_hThreadMutex);

}

void TerminateProcces(HWND hwnd) {

HWND listBox = GetDlgItem(hwnd, ID\_PROCESS\_LIST);

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

if (selectedIndex == LB\_ERR) {

MessageBox(hwnd, L"No process selected.", L"Error", MB\_ICONERROR);

return;

}

TCHAR processName[MAX\_PATH];

SendMessage(listBox, LB\_GETTEXT, selectedIndex, (LPARAM)processName);

TCHAR\* spacePos = \_tcschr(processName, ' ');

if (spacePos != nullptr) {

\*spacePos = '\0';

}

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

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

MessageBox(hwnd, L"Error creating process snapshot.", L"Error", MB\_ICONERROR);

return;

}

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

DWORD targetPID = 0;

bool processFound = false;

if (Process32First(hProcessSnap, &pe32)) {

do {

if (\_tcscmp(pe32.szExeFile, processName) == 0) {

targetPID = pe32.th32ProcessID;

processFound = true;

break;

}

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

}

CloseHandle(hProcessSnap);

if (!processFound) {

MessageBox(hwnd, L"Process not found.", L"Error", MB\_ICONERROR);

return;

}

HANDLE hProcess = OpenProcess(PROCESS\_TERMINATE, FALSE, targetPID);

if (hProcess == NULL) {

MessageBox(hwnd, L"Error opening process.", L"Error", MB\_ICONERROR);

return;

}

if (TerminateProcess(hProcess, 0)) {

MessageBox(hwnd, L"Process terminated successfully.", L"Success", MB\_ICONINFORMATION);

}

else {

MessageBox(hwnd, L"Error terminating process.", L"Error", MB\_ICONERROR);

}

CloseHandle(hProcess);

}

void UpdateProcessList(HWND hwnd) {

// Очистка listBox

HWND listBox = GetDlgItem(hwnd, ID\_PROCESS\_LIST);

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

// Создание снимка (TH32CS\_SNAPPROCESS включает в снимок все процессы в системе)

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

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

MessageBox(hwnd, L"Error creating process snapshot.", L"Error", MB\_ICONERROR);

return;

}

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(hProcessSnap, &pe32)) {

do {

// Открытие процесса

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, pe32.th32ProcessID);

if (hProcess != NULL) {

PROCESS\_MEMORY\_COUNTERS pmc;

if (GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc))) {

TCHAR buffer[512];

\_stprintf\_s(buffer, \_T("%s - Memory Usage: %I64u KB"), pe32.szExeFile, pmc.WorkingSetSize / (1024));

SendMessage(listBox, LB\_ADDSTRING, 0, (LPARAM)buffer);

//InvalidateRect(listBox, NULL, TRUE);

}

CloseHandle(hProcess);

}

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

}

CloseHandle(hProcessSnap);

}