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). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности»

Выполнил:

студент гр. 153504

Прескурел Я.Ю.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цели работы 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Полученные результаты 5](#_Toc146631500)

[Выводы 6](#_Toc146631501)

[Список использованных источников 7](#_Toc146631502)

[Приложение А 9](#_Toc146631503)

## 1 ЦЕЛИ РАБОТЫ

Изучить механизмы управления процессами в операционной системе Windows. Изучить операции по приостановке, возобновлению и завершению процессов, а также оценку их эффективности. Реализовать приложение для отслеживания и управления процессами в операционной системе Windows, что позволит осуществлять операции по приостановке, возобновлению и завершению процессов.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс Windows API (Application Programming Interface) представляет собой набор функций, предоставляемых операционной системой Windows для разработки приложений. Эти функции позволяют взаимодействовать с ОС на низком уровне, осуществляя такие действия, как управление окнами, обработка сообщений, работа с файлами и устройствами.

Порождение процессов и потоков: Для порождения нового процесса в Win32 API используется функция CreateProcess. Она создает новый процесс и возвращает его идентификатор (PID). Созданный процесс может быть независимым или связанным с родительским процессом. Для порождения нового потока внутри процесса используется функция CreateThread. Она создает новый поток и возвращает его дескриптор. Потоки внутри процесса выполняются параллельно и могут иметь различные задачи и приоритеты.

Завершение процессов и потоков: Для завершения процесса в Win32 API используется функция TerminateProcess. Она принимает дескриптор процесса и код завершения и принудительно завершает процесс. Завершение процесса может быть вызвано различными причинами, такими как успешное выполнение, ошибка или запрос пользователя. При завершении процесса освобождаются все его ресурсы. Для завершения потока внутри процесса используется функция ExitThread. Она завершает текущий поток и освобождает его ресурсы. При завершении потока можно указать код завершения, который может быть использован другими потоками или процессами для обработки.

Изменение приоритетов процессов и потоков: Для изменения приоритета процесса в Win32 API используется функция SetPriorityClass. Она принимает дескриптор процесса и новый приоритет и изменяет приоритет процесса. Приоритет процесса определяет, как много процессорного времени будет выделено данному процессу в сравнении с другими процессами. Высокий приоритет может ускорить выполнение процесса, но может также привести к ухудшению производительности других процессов. Для изменения приоритета потока внутри процесса используется функция SetThreadPriority. Она принимает дескриптор потока и новый приоритет и изменяет приоритет потока. Приоритет потока определяет, как много процессорного времени будет выделено данному потоку в сравнении с другими потоками внутри процесса.

Исследование эффективности: Для исследования эффективности процессов и потоков в Win32 API можно использовать различные функции для измерения времени выполнения, использования процессора и других ресурсов. Например, функция GetProcessTimes позволяет получить информацию о времени выполнения процесса, а функция GetThreadTimes - о времени выполнения потока. Эти функции могут быть полезны при профилировании и оптимизации процессов и потоков для достижения лучшей производительности и эффективности системы.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано приложение для отслеживания и управления процессами в операционной системе Windows, которое позволяет осуществлять операции по приостановке, возобновлению и завершению процессов. Результат работы программы показан на рисунках 3.1, 3.2 и 3.3

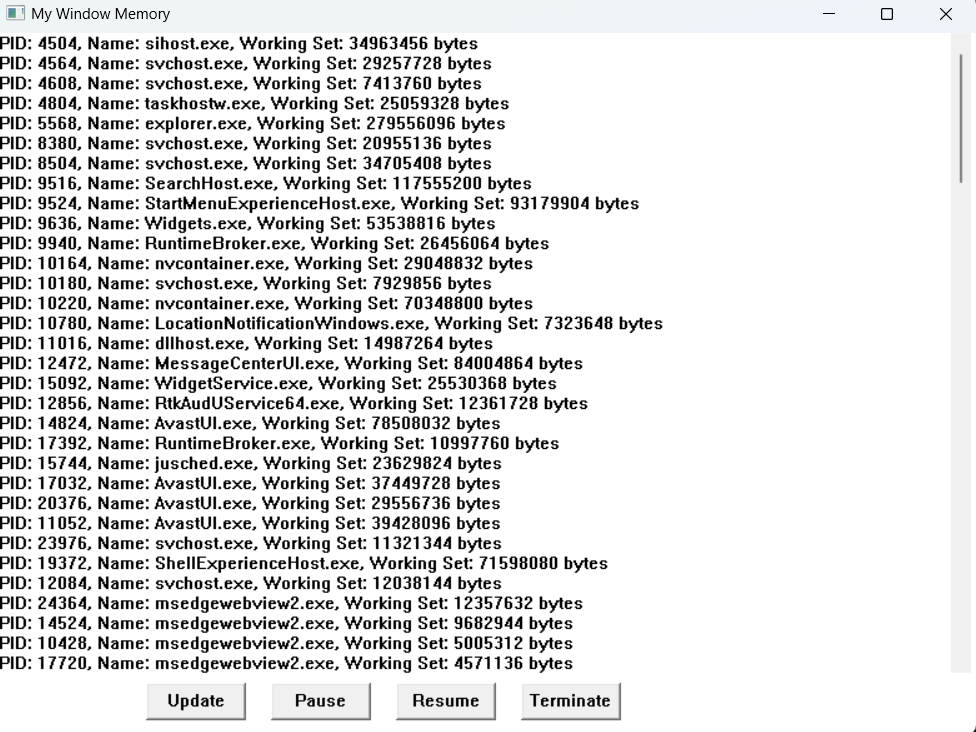


Рисунок 3.1 – Результат работы программ (1)

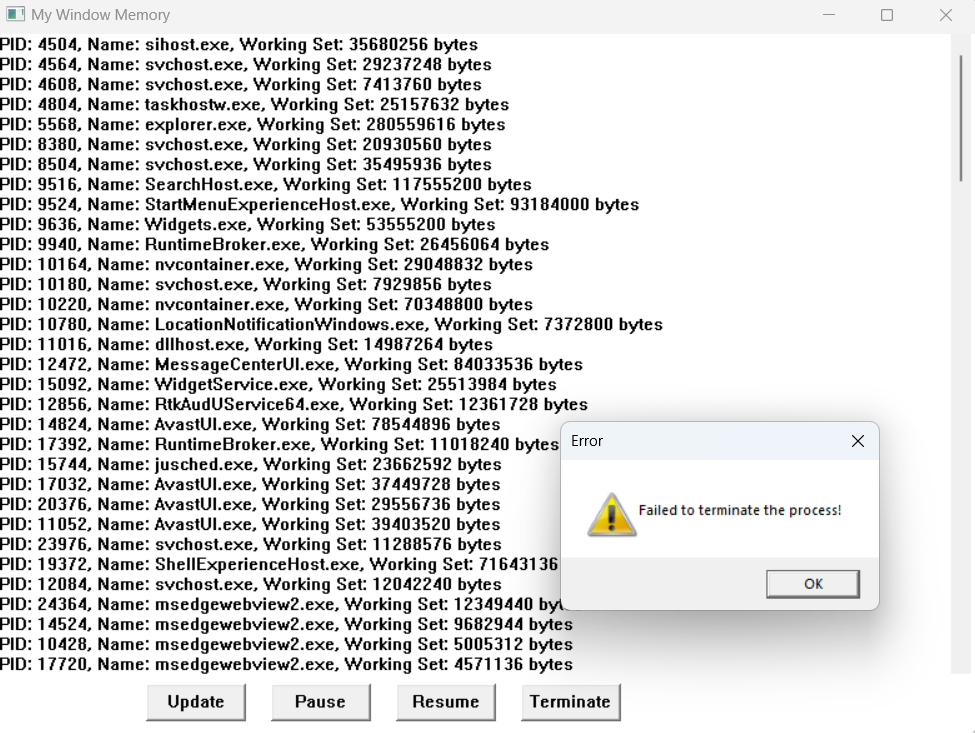


Рисунок 3.2 – Результат работы программ (2)

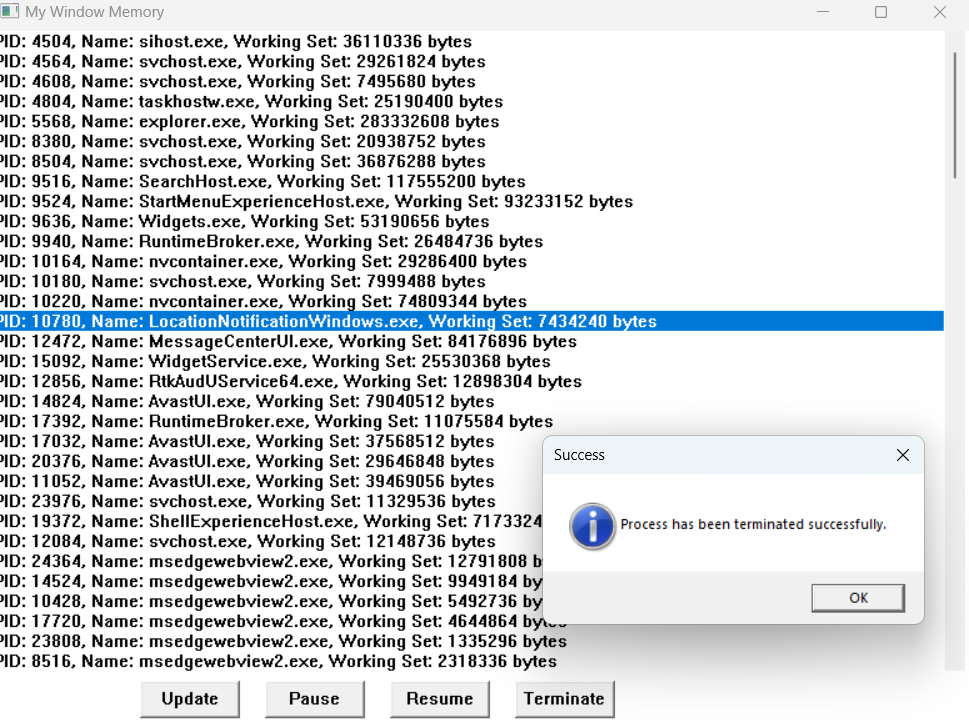


Рисунок 3.3 – Результат работы программ (3)

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены механизмы управления процессами в операционной системе Windows. Изучены операции по приостановке, возобновлению и завершению процессов, а также оценка их эффективности. Реализовано приложение для отслеживания и управления процессами в операционной системе Windows, которое позволяет осуществлять операции по приостановке, возобновлению и завершению процессов.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

[2] [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess – Дата доступа 10.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[3] [Электронный ресурс]. – Режим доступа: http://winapi.freetechsecrets.com/win32/WIN32TerminateProcess.htm – Дата доступа 10.10.2023

[4] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-resumethread – Дата доступа 23.09.2023

[5] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-suspendthread – Дата доступа 23.09.2023

[6] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/procthread/suspending-thread-execution – Дата доступа 11.10.2023

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

**Lab4.cpp**

#include <Windows.h>

#include <psapi.h>

#include <tchar.h>

#include <CommCtrl.h>

HWND hwndList;

HWND hwndButton;

HWND hwndButtonPause;

HWND hwndButtonResume;

HWND hwndButtonTerminate;

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

void UpdateProcessList(HWND hwndList);

void PauseProcess(DWORD processId);

void ResumeProcess(DWORD processId);

void TerminateProcess(DWORD processId);

void ShowMessage(LPCTSTR message, LPCTSTR title, UINT flags);

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

{

WNDCLASSEX wc = { 0 };

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

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

wc.lpszClassName = \_T("MemoryMonitorApp");

if (!RegisterClassEx(&wc))

{

ShowMessage(\_T("Window Registration Failed!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

HWND hwnd = CreateWindow(\_T("MemoryMonitorApp"), \_T("My Window Memory"), WS\_OVERLAPPEDWINDOW, 100, 100, 800, 600, NULL, NULL, hInstance, NULL);

if (!hwnd)

{

MessageBox(NULL, \_T("Window Creation Failed!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

hwndList = CreateWindowEx(0, \_T("LISTBOX"), NULL, WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | WS\_HSCROLL | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT, 0, 0, 780, 520, hwnd, NULL, GetModuleHandle(NULL), NULL);

if (!hwndList)

{

ShowMessage(\_T("Window Registration Failed!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

ShowWindow(hwnd, nCmdShow);

UpdateProcessList(hwndList);

MSG Msg;

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

{

TranslateMessage(&Msg);

DispatchMessage(&Msg);

}

return Msg.wParam;

}

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

{

switch (uMsg)

{

case WM\_CREATE:

hwndButton = CreateWindow(\_T("BUTTON"), \_T("Update"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 120, 520, 80, 30, hwnd, (HMENU)1001, NULL, NULL);

hwndButtonPause = CreateWindow(\_T("BUTTON"), \_T("Pause"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 220, 520, 80, 30, hwnd, (HMENU)1002, NULL, NULL);

hwndButtonResume = CreateWindow(\_T("BUTTON"), \_T("Resume"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 320, 520, 80, 30, hwnd, (HMENU)1003, NULL, NULL);

hwndButtonTerminate = CreateWindow(\_T("BUTTON"), \_T("Terminate"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 420, 520, 80, 30, hwnd, (HMENU)1004, NULL, NULL);

SetTimer(hwnd, 1, 5000, NULL);

break;

case WM\_COMMAND:

if (LOWORD(wParam) == 1001)

{

UpdateProcessList(hwndList);

}

else if (LOWORD(wParam) == 1002)

{

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

if (selectedIndex != LB\_ERR)

{

DWORD processId = 0;

TCHAR info[MAX\_PATH];

SendMessage(hwndList, LB\_GETTEXT, selectedIndex, (LPARAM)info);

if (\_stscanf\_s(info, \_T("PID: %d"), &processId) == 1)

{

PauseProcess(processId);

}

}

}

else if (LOWORD(wParam) == 1003)

{

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

if (selectedIndex != LB\_ERR)

{

DWORD processId = 0;

TCHAR info[MAX\_PATH];

SendMessage(hwndList, LB\_GETTEXT, selectedIndex, (LPARAM)info);

if (\_stscanf\_s(info, \_T("PID: %d"), &processId) == 1)

{

ResumeProcess(processId);

}

}

}

else if (LOWORD(wParam) == 1004)

{

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

if (selectedIndex != LB\_ERR)

{

DWORD processId = 0;

TCHAR info[MAX\_PATH];

SendMessage(hwndList, LB\_GETTEXT, selectedIndex, (LPARAM)info);

if (\_stscanf\_s(info, \_T("PID: %d"), &processId) == 1)

{

TerminateProcess(processId);

}

}

}

break;

case WM\_TIMER:

if (wParam == 1) {

UpdateProcessList(hwndList);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

void UpdateProcessList(HWND hwndList)

{

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

DWORD aProcesses[1024], cbNeeded, cProcesses;

if (EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded)) {

cProcesses = cbNeeded / sizeof(DWORD);

for (DWORD i = 0; i < cProcesses; i++) {

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, aProcesses[i]);

if (hProcess)

{

TCHAR processName[MAX\_PATH];

if (GetProcessImageFileName(hProcess, processName, MAX\_PATH) > 0)

{

// Извлекаем только имя файла из пути

TCHAR\* processNameOnly = \_tcsrchr(processName, '\\');

if (processNameOnly)

{

processNameOnly++;

}

else

{

processNameOnly = processName;

}

PROCESS\_MEMORY\_COUNTERS pmc;

if (GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc)) && pmc.WorkingSetSize > 0)

{

TCHAR info[MAX\_PATH];

\_stprintf\_s(info, \_T("PID: %d, Name: %s, Working Set: %I64d bytes\n"), aProcesses[i], processNameOnly, pmc.WorkingSetSize);

SendMessage(hwndList, LB\_ADDSTRING, 0, (LPARAM)info);

}

}

CloseHandle(hProcess);

}

}

}

}

void PauseProcess(DWORD processId)

{

HANDLE hProcess = OpenProcess(PROCESS\_SUSPEND\_RESUME, FALSE, processId);

if (hProcess)

{

DWORD result = SuspendThread(hProcess);

if (result != (DWORD)-1)

{

ShowMessage(\_T("Process has been paused successfully."), \_T("Success"), MB\_ICONINFORMATION | MB\_OK);

}

else

{

ShowMessage(\_T("Failed to pause the process!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

CloseHandle(hProcess);

}

else

{

ShowMessage(\_T("Failed to open the process for pausing!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

}

void ResumeProcess(DWORD processId)

{

HANDLE hProcess = OpenProcess(PROCESS\_SUSPEND\_RESUME, FALSE, processId);

if (hProcess)

{

DWORD result = ResumeThread(hProcess);

if (result != (DWORD)-1)

{

ShowMessage(\_T("Process has been resumed successfully."), \_T("Success"), MB\_ICONINFORMATION | MB\_OK);

}

else

{

ShowMessage(\_T("Failed to resume the process!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

CloseHandle(hProcess);

}

else

{

ShowMessage(\_T("Failed to open the process for resuming!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

}

void TerminateProcess(DWORD processId)

{

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

if (hProcess)

{

if (TerminateProcess(hProcess, 0))

{

ShowMessage(\_T("Process has been terminated successfully."), \_T("Success"), MB\_ICONINFORMATION | MB\_OK);

}

else

{

ShowMessage(\_T("Failed to terminate the process!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

CloseHandle(hProcess);

}

else

{

ShowMessage(\_T("Failed to open the process for terminating!"), \_T("Error"), MB\_ICONEXCLAMATION | MB\_OK);

}

}

void ShowMessage(LPCTSTR message, LPCTSTR title, UINT flags)

{

MessageBox(NULL, message, title, flags);

}