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

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты](#_1fob9te) 5

[Вывод](#_4q3p755gzys6) 7

[Список использованных источников](#_4q3p755gzys6) 8

[Приложение А (обязательное) Листинг кода](#_4q3p755gzys6) 9

**1 ЦЕЛИ РАБОТЫ**

1 Изучить управление процессами и потоками (Windows).

2 Изучить порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности.

3 Разработать приложение для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе Windows играет важную роль в разработке приложений. Это включает в себя порождение (создание), завершение, изменение приоритетов процессов и потоков, а также исследование эффективности выполнения процессов и потоков. Рассмотрим основные концепции и методы, связанные с этой темой, и их применение в разрабатываемом приложении.

Для порождения процессов в Windows используется функция *CreateProcess*. Она создает новый процесс и запускает в нем указанную исполняемую программу.

При порождении процесса можно указать различные параметры, такие как путь к исполняемому файлу, командную строку, структуры с информацией о процессе и т.д.

Для порождения потоков в процессе используется функция *CreateThread*. Она создает новый поток выполнения в контексте указанного процесса.

Функции Win32 API для работы с процессами предоставляют мощные возможности для управления и мониторинга процессов в операционной системе Windows. Ниже приведены подробности некоторых из наиболее часто используемых функций:

1 Функция *CreateToolhelp32Snapshot* создает снимок текущего состояния системы, включая информацию о процессах и потоках.

2 Функции *Thread32First* и *Thread32Next* позволяют перечислять потоки процесса в снимке.

3 Функции *SuspendThread* и *ResumeThread* позволяют приостанавливать и возобновлять выполнение потоков в процессе.

4 Функция *TerminateProcess* позволяет принудительно завершить указанный процесс.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Был доработан текстовый редактор из предыдущих двух лабораторных и добавлена поддержка записи и чтения файлов в двоичном формате с использованием маппинга файлов в память.

На рисунке 3.1 продемонстрирована функция приостановки процесса посредством остановки работы приложения *Google Chrome*. После нажатия на кнопку приостановки окно приложения перестало реагировать на взаимодействия.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.1 – Приостановка процесса

На рисунке 3.2 продемонстрирована функция возобновления процесса посредством возобновления работы приложения Google Chrome. После нажатия на кнопку возобновления окно приложения возобновило свою работу и стало реагировать.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.2 ­ Возобновление работы потока

На рисунке 3.3 продемонстрирована функция завершения процесса посредством завершения работы приложения *Google Chrome*. Можно заметить что после нажатия на кнопку завершения процесс *Google Chrome* пропал из диспетчера и после нажатия кнопки обновления списка процессов он так же пропадет.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.3 ­ Завершение работы процесса

.

**ВЫВОД**

В результате было разработано приложение для отслеживания и управления процессами в системе, позволяющее приостанавливать, возобновлять и завершать процессы.

Для этого был изучен теоретический материал и средства Win32 Api, используемые для этих целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Process and Thread Functions, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/procthread/process-and-thread-functions>.

[2] Процессы и потоки, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/procthread/processes-and-threads>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <windows.h>

#include <tlhelp32.h>

// Идентификаторы элементов управления

#define BTN\_REFRESH 101

#define BTN\_SUSPEND 102

#define BTN\_RESUME 103

#define BTN\_TERMINATE 104

#define LST\_PROCESSES 105

// Прототипы функций

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

void RefreshProcessList(HWND hwnd);

void SuspendProcess(HWND hwnd);

void ResumeProcess(HWND hwnd);

void TerminateProcess(HWND hwnd);

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

{

// Регистрация класса окна

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

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Создание окна

HWND hwnd = CreateWindowEx(

0, // дополнительные стили окна

CLASS\_NAME, // имя класса окна

L"Process Manager", // заголовок окна

WS\_OVERLAPPEDWINDOW, // стиль окна

CW\_USEDEFAULT, // позиция X окна

CW\_USEDEFAULT, // позиция Y окна

500, // ширина окна

400, // высота окна

NULL, // родительское окно

NULL, // дескриптор меню

hInstance, // дескриптор приложения

NULL // указатель на данные создания окна

);

if (hwnd == NULL)

{

return 0;

}

// Создание элементов управления

CreateWindow(

L"BUTTON",

L"Refresh",

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

10, 10, 100, 30,

hwnd,

reinterpret\_cast<HMENU>(BTN\_REFRESH),

hInstance,

NULL

);

CreateWindow(

L"BUTTON",

L"Suspend",

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

120, 10, 100, 30,

hwnd,

reinterpret\_cast<HMENU>(BTN\_SUSPEND),

hInstance,

NULL

);

CreateWindow(

L"BUTTON",

L"Resume",

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

230, 10, 100, 30,

hwnd,

reinterpret\_cast<HMENU>(BTN\_RESUME),

hInstance,

NULL

);

CreateWindow(

L"BUTTON",

L"Terminate",

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

340, 10, 100, 30,

hwnd,

reinterpret\_cast<HMENU>(BTN\_TERMINATE),

hInstance,

NULL

);

HWND lstProcesses = CreateWindow(

L"LISTBOX",

NULL,

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | WS\_VSCROLL,

10, 50, 460, 290,

hwnd,

reinterpret\_cast<HMENU>(LST\_PROCESSES),

hInstance,

NULL

);

// Отображение окна

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Основной цикл сообщений

MSG msg = {};

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

{

switch (uMsg)

{

case WM\_COMMAND:

{

if (HIWORD(wParam) == BN\_CLICKED)

{

switch (LOWORD(wParam))

{

case BTN\_REFRESH:

RefreshProcessList(hwnd);

break;

case BTN\_SUSPEND:

SuspendProcess(hwnd);

break;

case BTN\_RESUME:

ResumeProcess(hwnd);

break;

case BTN\_TERMINATE:

TerminateProcess(hwnd);

break;

}

}

break;

}

case WM\_MOUSEWHEEL:

{

HWND lstProcesses = GetDlgItem(hwnd, LST\_PROCESSES);

SendMessage(lstProcesses, uMsg, wParam, lParam);

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

}

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

}

void RefreshProcessList(HWND hwnd)

{

HWND lstProcesses = GetDlgItem(hwnd, LST\_PROCESSES);

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

HANDLE hProcessSnap;

PROCESSENTRY32 pe32;

hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

if (hProcessSnap == INVALID\_HANDLE\_VALUE)

{

return;

}

pe32.dwSize = sizeof(PROCESSENTRY32);

if (!Process32First(hProcessSnap, &pe32))

{

CloseHandle(hProcessSnap);

return;

}

do

{

SendMessage(lstProcesses, LB\_ADDSTRING, 0, reinterpret\_cast<LPARAM>(pe32.szExeFile));

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

CloseHandle(hProcessSnap);

}

void SuspendProcess(HWND hwnd)

{

HWND lstProcesses = GetDlgItem(hwnd, LST\_PROCESSES);

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

if (selectedIndex == LB\_ERR)

{

return;

}

wchar\_t processName[MAX\_PATH];

SendMessage(lstProcesses, LB\_GETTEXT, selectedIndex, reinterpret\_cast<LPARAM>(processName));

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

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(hSnapshot, &pe32))

{

do

{

if (wcscmp(pe32.szExeFile, processName) == 0)

{

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

THREADENTRY32 te32;

te32.dwSize = sizeof(THREADENTRY32);

if (Thread32First(hThreadSnapshot, &te32))

{

do

{

if (te32.th32OwnerProcessID == pe32.th32ProcessID)

{

HANDLE hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, te32.th32ThreadID);

if (hThread != NULL)

{

SuspendThread(hThread);

CloseHandle(hThread);

}

}

} while (Thread32Next(hThreadSnapshot, &te32));

}

CloseHandle(hThreadSnapshot);

break;

}

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

}

CloseHandle(hSnapshot);

}

void ResumeProcess(HWND hwnd)

{

HWND lstProcesses = GetDlgItem(hwnd, LST\_PROCESSES);

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

if (selectedIndex == LB\_ERR)

{

return;

}

wchar\_t processName[MAX\_PATH];

SendMessage(lstProcesses, LB\_GETTEXT, selectedIndex, reinterpret\_cast<LPARAM>(processName));

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

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(hSnapshot, &pe32))

{

do

{

if (wcscmp(pe32.szExeFile, processName) == 0)

{

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

THREADENTRY32 te32;

te32.dwSize = sizeof(THREADENTRY32);

if (Thread32First(hThreadSnapshot, &te32))

{

do

{

if (te32.th32OwnerProcessID == pe32.th32ProcessID)

{

HANDLE hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, te32.th32ThreadID);

if (hThread != NULL)

{

ResumeThread(hThread);

CloseHandle(hThread);

}

}

} while (Thread32Next(hThreadSnapshot, &te32));

}

CloseHandle(hThreadSnapshot);

break;

}

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

}

CloseHandle(hSnapshot);

}

void TerminateProcess(HWND hwnd)

{

HWND lstProcesses = GetDlgItem(hwnd, LST\_PROCESSES);

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

if (selectedIndex == LB\_ERR)

{

return;

}

wchar\_t processName[MAX\_PATH];

SendMessage(lstProcesses, LB\_GETTEXT, selectedIndex, reinterpret\_cast<LPARAM>(processName));

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

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(PROCESSENTRY32);

if (Process32First(hSnapshot, &pe32))

{

do

{

if (wcscmp(pe32.szExeFile, processName) == 0)

{

HANDLE hProcess = OpenProcess(PROCESS\_TERMINATE, FALSE, pe32.th32ProcessID);

if (hProcess != NULL)

{

TerminateProcess(hProcess, 0);

CloseHandle(hProcess);

}

break;

}

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

}

CloseHandle(hSnapshot);

}