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 А. С. Сорокин

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 является одной из фундаментальных функций операционной системы Windows, обеспечивающей многозадачность и эффективное распределение ресурсов компьютера. Процессы и потоки представляют основные строительные блоки для параллельного выполнения задач и обеспечивают отзывчивость и производительность приложений. Рассмотрим ключевые аспекты управления процессами и потоками в операционной системе Windows.

Процесс в операционной системе Windows представляет изолированное выполнение приложения. Каждый процесс имеет своё собственное виртуальное адресное пространство, независимые ресурсы и описатели файлов. Процессы обеспечивают изоляцию и безопасность, так как не имеют прямого доступа друг к другу.

Потоки – это наименьшие единицы выполнения внутри процесса. Они разделяют общее адресное пространство и ресурсы процесса. Потоки позволяют выполнять параллельные задачи в рамках одного процесса и облегчают многозадачность.

FILETIME - это структура данных, используемая в Windows для представления времени. В контексте idleTime, kernelTime и userTime, FILETIME используется для измерения времени, проведенного процессором в различных состояниях.

FILETIME предоставляет точность в микросекундах и может быть преобразован в другие форматы времени для удобства анализа и обработки данных. При использовании этих значений времени важно учитывать контекст и цель измерений, чтобы правильно интерпретировать результаты и принять соответствующие решения по управлению ресурсами и производительностью системы.

Исследование эффективности Win32 API: Эффективность Win32 API можно исследовать с помощью различных методов и метрик производительности. Одним из подходов является сравнение времени выполнения задач с разными приоритетами и анализ изменений производительности системы при изменении приоритетов. Это может включать создание нагрузочных тестов или бенчмарков, сравнение использования ресурсов и анализ системных журналов для определения влияния изменения приоритетов на производительность системы.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Было создано приложение для анализа производительности процессора, отображающее загрузку ресурсов в реальном времени (рисунок 1).



Рисунок 1 – Окно приложения

# ЗАКЛЮЧЕНИЕ

В рамках выполнения данной лабораторной работы было разработано и рассмотрено приложение, предназначенное для анализа производительности процессора в реальном времени. 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 Lab\_4*.cpp*

#include <windows.h>

#include "SoftwareDefinitions.h"

#include <string>

#include <thread>

#include <chrono>

#include <Pdh.h>

#define ID\_TIMER 1

#define WINDOW\_WIDTH 300

#define WINDOW\_HEIGHT 200

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_HAND), hInst, LoadIcon(NULL, IDI\_ERROR),

L"MainWndClass", SoftwareMainProcedure);

if (!RegisterClassW(&SoftwareMainClass)) { return -1; }

CreateWindow(L"MainWndClass", L"Lab\_4 window", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 550, 250, NULL, NULL, NULL, NULL);

MSG msg = { };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hIcon = Icon;

NWC.hCursor = Cursor;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg)

{

case WM\_COMMAND:

switch (wp) {

case OnMenuClicked1:

MessageBoxA(hWnd, "menu 1 was klicked", "Menu worked", MB\_OK);

break;

case OnCalculate:

cpuLoad = GetCPULoadPercentage();

SetWindowTextA(hEditControl, (" cpuload: " + std::to\_string(cpuLoad)).c\_str());

break;

case OnExitSoftware:

PostQuitMessage(0);

break;

}

break;

case WM\_CREATE:

MainWndAddMenus(hWnd);

MainWndAddWidgets(hWnd);

SetTimer(hWnd, ID\_TIMER, 1000, NULL); // Устанавливаем таймер на 1 секунду

break;

case WM\_TIMER:

if (wp == ID\_TIMER)

{

double cpuLoadPercentage = GetCPULoadPercentage() + 10;

// Обновляем заголовок окна с текущей загрузкой процессора

char title[50];

sprintf\_s(title, "CPU Load: %.2f%%", cpuLoadPercentage);

SetWindowTextA(hEditControl, title);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, msg, wp, lp);

}

}

void MainWndAddMenus(HWND hwnd) {

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

//HMENU SubActionMenu = CreateMenu();

//AppendMenu(SubActionMenu, MF\_STRING, OnMenuClicked3, L"Menu 3");

//AppendMenu(SubMenu, MF\_POPUP, (UINT\_PTR)SubActionMenu, L"Action");

AppendMenu(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(SubMenu, MF\_STRING, OnExitSoftware, L"Exit");

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)SubMenu, L"File");

AppendMenu(RootMenu, MF\_STRING, (UINT\_PTR)SubMenu, L"Help");

SetMenu(hwnd, RootMenu);

}

void MainWndAddWidgets(HWND hWnd) {

hStaticControl = CreateWindowA("static", "Main Window!", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 200, 5, 100, 30, hWnd, NULL, NULL, NULL);

hEditControl = CreateWindowA("edit", "", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL, 5, 40, 470, 120, hWnd, NULL, NULL, NULL);

//CreateWindowA("button", "Calculate", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 130, 200, 100, 50, hWnd, (HMENU)OnCalculate, NULL, NULL);

}

double GetCPULoadPercentage()

{

FILETIME idleTime, kernelTime, userTime;

if (GetSystemTimes(&idleTime, &kernelTime, &userTime))

{

ULONGLONG idleTimeDiff = SubtractTimes(idleTime, lastIdleTime);

ULONGLONG kernelTimeDiff = SubtractTimes(kernelTime, lastKernelTime);

ULONGLONG userTimeDiff = SubtractTimes(userTime, lastUserTime);

ULONGLONG totalTimeDiff = kernelTimeDiff + userTimeDiff;

ULONGLONG idleTimePercentage = (idleTimeDiff \* 100) / totalTimeDiff;

ULONGLONG cpuLoadPercentage = 100 - idleTimePercentage;

lastIdleTime = idleTime;

lastKernelTime = kernelTime;

lastUserTime = userTime;

return static\_cast<double>(cpuLoadPercentage);

}

return 0.0;

}

// Вспомогательная функция для вычисления разницы между двумя FILETIME значениями

ULONGLONG SubtractTimes(const FILETIME& ftA, const FILETIME& ftB)

{

LARGE\_INTEGER a, b;

a.LowPart = ftA.dwLowDateTime;

a.HighPart = ftA.dwHighDateTime;

b.LowPart = ftB.dwLowDateTime;

b.HighPart = ftB.dwHighDateTime;

return a.QuadPart - b.QuadPart;

}

Листинг 2 – Файл SoftwareDefinitions*.h*

#pragma once

#define OnMenuClicked1 1

#define OnMenuClicked2 2

#define OnMenuClicked3 3

#define OnExitSoftware 4

#define OnCalculate 5

FILETIME lastIdleTime = { 0 };

FILETIME lastKernelTime = { 0 };

FILETIME lastUserTime = { 0 };

HWND hEditControl;

HWND hStaticControl;

double cpuLoad;

double cpuusage;

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp);

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure);

void MainWndAddMenus(HWND hwnd);

void MainWndAddWidgets(HWND hwnd);

ULONGLONG SubtractTimes(const FILETIME& ftA, const FILETIME& ftB);

double GetCPULoadPercentage();