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 №1

на тему

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (WINDOWS). ПОРОЖДЕНИЕ, ЗАВЕРШЕНИЕ, ИЗМЕНЕНИЕ ПРИОРИТЕТОВ ПРОЦЕССОВ И ПОТОКОВ, ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ.

Выполнил студент гр.153502 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc148040559)

[2 Теоретические сведения 4](#_Toc148040560)

[3 Описание функций программы 5](#_Toc148040561)

[3.1 Получение информации о ресурсах системы в реальном времени 5](#_Toc148040562)

[Список использованных источников 6](#_Toc148040563)

[Приложение А 7](#_Toc148040564)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Управление процессами и потоками (Windows). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности.

В качестве задачи необходимо создать приложение для анализа производительности процессора и памяти, отображающее графики загрузки ресурсов в реальном времени.

# **2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Многопоточность в C++ позволяет выполнить несколько потоков выполнения (threads) внутри одного процесса. Каждый поток выполняет код независимо от других потоков, что может повысить производительность и эффективность многозадачных приложений.

Поток ­ это наименьшая единица выполнения внутри процесса. В многопоточном приложении можно создать и управлять несколькими потоками.

В C++, для создания потока можно использовать стандартную библиотеку C++ (std::thread) или библиотеку для работы с потоками операционной системы (например, Windows API или POSIX threads).

В многопоточных приложениях несколько потоков могут работать с общими ресурсами, и это может привести к состояниям гонки и другим проблемам.

Для обеспечения синхронизации потоков используются мьютексы, условные переменные, семафоры и другие механизмы синхронизации.

Если несколько потоков обращаются к общим данным, необходимо обеспечить безопасность доступа к этим данным. Это может быть достигнуто с использованием мьютексов или других механизмов синхронизации.

Deadlock возникает, когда два или более потока ожидают друг друга, чтобы освободить ресурсы, и, таким образом, ни один из них не может продолжить выполнение.

Проектирование и обработка deadlock ­ это важная часть работы с многопоточностью.

В Windows API (WinAPI) для работы с потоками используются функции и структуры, предоставляемые операционной системой Windows.

Для создания нового потока в WinAPI используется функция CreateThread. Эта функция принимает указатель на функцию, которая будет выполняться в потоке, и другие параметры.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Получение информации о ресурсах системы в реальном времени.

3.1 Получение информации о ресурсах системы в реальном времени

На экране есть 2 индикатора ­ загруженности процессора и оперативной памяти. А под ними находятся числовые показатели (рисунок 1).

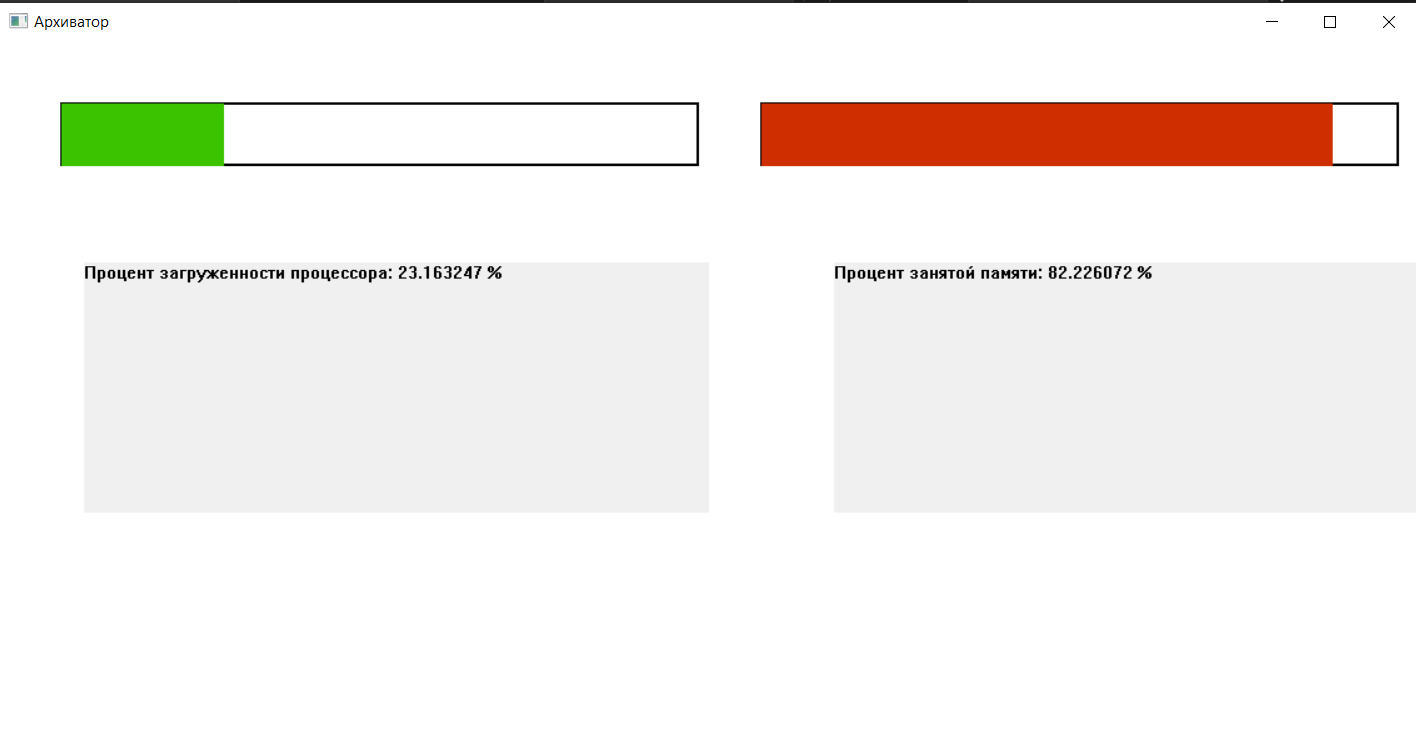


Рисунок 1 – Интерфейс программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32>

[2] C++ WINDOWS API – Электронные данные. – Режим доступа: <https://www.youtube.com/@egordeveloper3794>

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Lab4.h

#pragma once

#include "resource.h"

#include <Windows.h>

#include <string>

void startMonitoring(int msRefresh);

SystemInfo systemInfo;

HWND workloadText;

HWND mainHwnd;

HWND memoryText;

RECT rectWorkloadOutline;

RECT rectMemoryOutline;

RECT rectWorkload;

RECT rectMemory;

bool needToRedrawRects = true;

double workload = 0.0;

double memory = 0.0;

LPCWSTR string\_to\_lpcwstr(std::string s);

DWORD WINAPI startMonitoring(LPVOID lpParam);

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

Файл Lab4.cpp

#ifndef UNICODE

#define UNICODE

#endif

#if defined(UNICODE)

#define \_tstring wstring

#else

#define \_tstring string

#endif

#include <windows.h>

#include <thread>

#include <chrono>

#include "SystemInfo.h"

#include "Lab4.h"

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Register the window class.

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window.

HWND hwnd = CreateWindowEx(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Архиватор", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL)

{

return 0;

}

mainHwnd = hwnd;

workloadText = CreateWindowW(

L"Static", // Класс элемента управления "Static"

L"Процент загруженности процессора: ", // Текст надписи

WS\_CHILD | WS\_VISIBLE, // Стиль окна

100, 100, // Положение (x, y)

300, 200, // Ширина и высота

hwnd, // Родительское окно

(HMENU)NULL, // Идентификатор элемента управления (может быть NULL)

GetModuleHandle(NULL), // Дескриптор экземпляра приложения

NULL // Дополнительные данные (может быть NULL)

);

memoryText = CreateWindowW(

L"Static", // Класс элемента управления "Static"

L"Процент занятой памяти процессора: ", // Текст надписи

WS\_CHILD | WS\_VISIBLE, // Стиль окна

200, 50, // Положение (x, y)

300, 50, // Ширина и высота

hwnd, // Родительское окно

(HMENU)NULL, // Идентификатор элемента управления (может быть NULL)

GetModuleHandle(NULL), // Дескриптор экземпляра приложения

NULL // Дополнительные данные (может быть NULL)

);

ShowWindow(hwnd, nCmdShow);

HANDLE monitoringThread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)startMonitoring, (LPVOID)100, 0, NULL);

if (monitoringThread == NULL) {

// Обработка ошибки

return 1;

}

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

CloseHandle(monitoringThread);

return 0;

}

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

{

switch (uMsg)

{

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

case WM\_PAINT:

{

int x1 = 50;

int x2 = 560;

int y1 = 50;

int y2 = 100;

int spacing = 50;

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

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

// Создаем перо (краску) для черной обводки

HPEN blackPen = CreatePen(PS\_SOLID, 2, RGB(0, 0, 0)); // RGB(0, 0, 0) представляет черный цвет

HGDIOBJ oldPen = SelectObject(hdc, blackPen);

// Рисуем прямоугольник с черной обводкой

rectWorkloadOutline = { x1, y1, x2, y2 }; // Координаты и размеры прямоугольника

Rectangle(hdc, rectWorkloadOutline.left, rectWorkloadOutline.top, rectWorkloadOutline.right, rectWorkloadOutline.bottom);

// Рисуем прямоугольник с черной обводкой

rectMemoryOutline = { x1 + (x2 - x1) + spacing, y1, x2 + (x2 - x1) + spacing, y2}; // Координаты и размеры прямоугольника

Rectangle(hdc, rectMemoryOutline.left, rectMemoryOutline.top, rectMemoryOutline.right, rectMemoryOutline.bottom);

// Восстанавливаем предыдущее перо и освобождаем созданное

SelectObject(hdc, oldPen);

DeleteObject(blackPen);

HBRUSH greenBrush = CreateSolidBrush(RGB(workload \* 2.55, 255 - workload \* 2.55, 0));

rectWorkload = { x1, y1, x1 + lround(workload \* 5.6), y2 };

FillRect(hdc, &rectWorkload, greenBrush);

// Освобождаем кисть

DeleteObject(greenBrush);

greenBrush = CreateSolidBrush(RGB(memory \* 2.55, 255 - memory \* 2.55, 0));

rectMemory = { x1 + (x2 - x1) + spacing, y1, x1 + (x2 - x1) + spacing + lround(memory \* 5.6), y2 };

FillRect(hdc, &rectMemory, greenBrush);

// Освобождаем кисть

DeleteObject(greenBrush);

EndPaint(hwnd, &ps);

break;

}

case WM\_SIZE:

{

int width = LOWORD(lParam);

int height = HIWORD(lParam);

SetWindowPos(workloadText, NULL, width / 2 - 500, height / 2 - 100, 500, 200, SWP\_NOZORDER);

SetWindowPos(memoryText, NULL, width / 2 + 100, height / 2 - 100, 500, 200, SWP\_NOZORDER);

}

case WM\_COMMAND:

{

switch (wParam)

{

default:

{

break;

}

}

}

default:

{

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

}

}

}

// Функция потока

DWORD WINAPI startMonitoring(LPVOID lpParam)

{

int msRefresh = (int)lpParam;

int iters = 0;

while (true) {

iters++;

double workload\_ = workload;

double memory\_ = memory;

HANDLE t1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)SystemInfo::GetWorkloadInfo, (LPVOID)&workload\_, 0, NULL);

HANDLE t2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)SystemInfo::GetMemoryInfo, (LPVOID)&memory\_, 0, NULL);

// Ожидание завершения потоков

WaitForSingleObject(t1, INFINITE);

WaitForSingleObject(t2, INFINITE);

// Закрытие дескрипторов потоков

CloseHandle(t1);

CloseHandle(t2);

workload = workload\_;

memory = memory\_;

std::\_tstring workloadStr(L"Процент загруженности процессора: " + std::to\_wstring(workload) + L" %");

std::\_tstring memoryStr(L"Процент занятой памяти: " + std::to\_wstring(memory) + L" %");

SetWindowTextW(workloadText, workloadStr.c\_str());

SetWindowTextW(memoryText, memoryStr.c\_str());

if (iters == 10)

{

needToRedrawRects = true;

InvalidateRect(mainHwnd, NULL, TRUE);

needToRedrawRects = false;

iters = 0;

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(msRefresh));

}

}

Файл SystemInfo.h

#pragma once

class SystemInfo

{

public:

SystemInfo();

void static GetWorkloadInfo(double& value);

void static GetMemoryInfo(double& value);

};

Файл SystemInfo.cpp

#include "SystemInfo.h"

#include "Windows.h"

#include "TCHAR.h"

SystemInfo::SystemInfo() { }

void SystemInfo::GetWorkloadInfo(double &value)

{

// Получение информации о загруженности процессора

FILETIME idleTime, kernelTime, userTime;

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

ULONGLONG idleTime64 = ((ULONGLONG)idleTime.dwHighDateTime << 32) + idleTime.dwLowDateTime;

ULONGLONG kernelTime64 = ((ULONGLONG)kernelTime.dwHighDateTime << 32) + kernelTime.dwLowDateTime;

ULONGLONG userTime64 = ((ULONGLONG)userTime.dwHighDateTime << 32) + userTime.dwLowDateTime;

value = 100.0 - (idleTime64 \* 100.0 / (kernelTime64 + userTime64));

}

}

void SystemInfo::GetMemoryInfo(double& value)

{

MEMORYSTATUSEX memoryStatus;

memoryStatus.dwLength = sizeof(memoryStatus);

if (GlobalMemoryStatusEx(&memoryStatus)) {

DWORDLONG totalPhysicalMemory = memoryStatus.ullTotalPhys;

DWORDLONG freePhysicalMemory = memoryStatus.ullAvailPhys;

value = 100.0 - (freePhysicalMemory \* 100.0 / totalPhysicalMemory);

}

}