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 №4

на тему

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

Выполнил студент гр.153502 Леоненко А.О.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Теоретические сведения 3](#_Toc148709461)

[2 Формулировка задачи 4](#_Toc148709462)

[3 Описание функций программы 5](#_Toc148709463)

[3.1 Интерфейс программы 5](#_Toc148709464)

[Список использованных источников 6](#_Toc148709465)

[Приложение А 7](#_Toc148709466)

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Создает поток для выполнения в виртуальном адресном пространстве вызывающего процесса.

Количество потоков, которые может создать процесс, ограничено доступной виртуальной памятью. По умолчанию каждый поток имеет один мегабайт пространства стека. Поэтому невозможно создать 2048 или более потоков в 32-разрядной системе без /*3GB* *boot*.*ini* параметра. Если уменьшить размер стека по умолчанию, можно создать дополнительные потоки. Однако производительность приложения будет выше, если вы создаете по одному потоку на процессор и создаете очереди запросов, для которых приложение хранит сведения контекста. Поток будет обрабатывать все запросы в очереди перед обработкой запросов в следующей очереди.

Поток создается с приоритетом потока *THREAD*\_*PRIORITY*\_*NORMAL*. Используйте функции *GetThreadPriority* и *SetThreadPriority*, чтобы получить и задать значение приоритета потока.

Когда поток завершается, объект потока достигает сигнального состояния, удовлетворяющего всем потокам, ожидающим объекта.

Объект потока остается в системе до тех пор, пока поток не завершится, а все дескрипторы для него не будут закрыты с помощью вызова CloseHandle.

Функции *ExitProcess*, *ExitThread*, *CreateThread*, *CreateRemoteThread* и процесс, который запускается (в результате вызова метода *CreateProcess*), сериализуются друг между другом в рамках процесса. Одновременно в адресном пространстве может произойти только одно из этих событий [1].

2 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на *Win32* *API*.

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

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

1 отображение загрузки процессора в процентах;

2 отображение загрузки оперативной памяти в процентах;

3 отрисовка графиков загруженности ресурсов.

3.1 Интерфейс программы

На рисунке 3.1 изображен интерфейс программы. Интерфейс состоит из строк с отображением загруженности процессора и оперативной памяти в процентах, а также графики, отображающие ту же информацию.

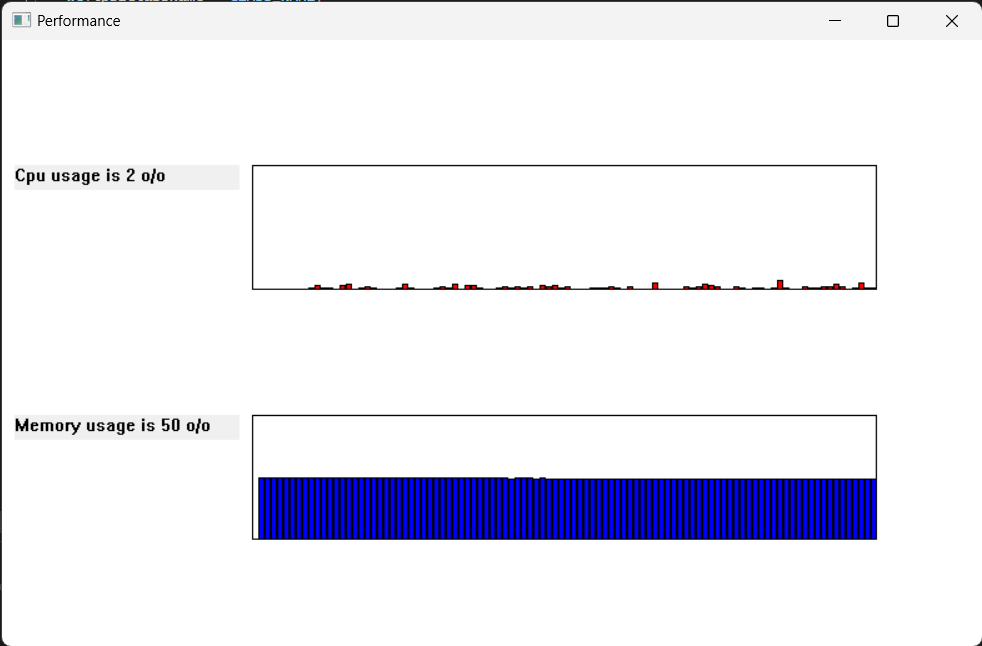


Рисунок 3.1 – Интерфейс программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функция CreateThread [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread

ПРИЛОЖЕНИЕ А

Листинг кода

Файл main.cpp

#include <windows.h>

#include <vector>

std::vector<int> cpuUsageHistory;

std::vector<int> memoryUsageHistory;

bool isStopped = false;

static float CalculateCPULoad();

static unsigned long long FileTimeToInt64();

float GetCPULoad();

ULARGE\_INTEGER lastKernelTime, lastUserTime, lastIdleTime;

HBRUSH hCpuBrush;

HBRUSH hMemoryBrush;

HBRUSH hRectBrush;

HWND hProcessor;

HWND hMemory;

int startSpace = 200;

DWORD UpdatePerformanceData(LPVOID lpParam) {

while (!isStopped) {

int cpuUsage = GetCPULoad() \* 100;

TCHAR buffer[256];

wsprintf(buffer, L"Cpu usage is %d o/o", cpuUsage);

SetWindowText(hProcessor, buffer);

MEMORYSTATUSEX memoryStatus;

memoryStatus.dwLength = sizeof(memoryStatus);

GlobalMemoryStatusEx(&memoryStatus);

auto total = memoryStatus.ullTotalPhys / 1024 / 1024;

auto available = memoryStatus.ullAvailPhys / 1024 / 1024;

int memoryUsage = ((float)total - (float)available) / (float)total \* 100;

TCHAR buffer2[256];

wsprintf(buffer2, L"Memory usage is %d o/o", memoryUsage);

SetWindowText(hMemory, buffer2);

const int maxHistorySize = 100;

if (cpuUsageHistory.size() >= maxHistorySize) {

cpuUsageHistory.erase(cpuUsageHistory.begin());

}

if (memoryUsageHistory.size() >= maxHistorySize) {

memoryUsageHistory.erase(memoryUsageHistory.begin());

}

cpuUsageHistory.push\_back(cpuUsage);

memoryUsageHistory.push\_back(memoryUsage);

Sleep(1000);

}

ExitThread(0);

return 1;

}

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

static HBRUSH hBackgroundBrush = CreateSolidBrush(RGB(255, 255, 255));

switch (uMsg) {

case WM\_CREATE:

SetTimer(hwnd, 1, 1000, NULL);

hCpuBrush = CreateSolidBrush(RGB(255, 0, 0));

hMemoryBrush = CreateSolidBrush(RGB(0, 0, 255));

hRectBrush = CreateSolidBrush(RGB(255, 255, 255));

hProcessor = CreateWindow(L"STATIC", L"", WS\_CHILD | WS\_VISIBLE, 10, 100, 180, 20, hwnd, (HMENU)1001, NULL, NULL);

hMemory = CreateWindow(L"STATIC", L"", WS\_CHILD | WS\_VISIBLE, 10, 300, 180, 20, hwnd, (HMENU)1002, NULL, NULL);

break;

case WM\_TIMER:

InvalidateRect(hwnd, NULL, TRUE);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

SelectObject(hdc, hRectBrush);

Rectangle(hdc, startSpace, 200, startSpace + 500, 100);

Rectangle(hdc, startSpace, 400, startSpace + 500, 300);

// CPU

SelectObject(hdc, hCpuBrush);

for (int i = 1; i < cpuUsageHistory.size(); ++i) {

Rectangle(hdc, i \* 5 + startSpace, 200, (i + 1) \* 5 + startSpace, 200 - cpuUsageHistory[i]);

}

// Memory

SelectObject(hdc, hMemoryBrush);

for (int i = 1; i < memoryUsageHistory.size(); ++i) {

Rectangle(hdc, i \* 5 + startSpace, 400, (i + 1) \* 5 + startSpace, 400 - memoryUsageHistory[i]);

}

EndPaint(hwnd, &ps);

}

break;

case WM\_DESTROY:

isStopped = true;

KillTimer(hwnd, 1);

DeleteObject(hCpuBrush);

DeleteObject(hMemoryBrush);

DeleteObject(hCpuBrush);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

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

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

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"Performance",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

return 0;

}

ShowWindow(hwnd, nCmdShow);

CreateThread(

NULL, // default security attributes

0, // use default stack size

UpdatePerformanceData, // thread function name

NULL, // argument to thread function

0, // use default creation flags

NULL);

MSG msg = { };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

static float CalculateCPULoad(unsigned long long idleTicks, unsigned long long totalTicks)

{

static unsigned long long \_previousTotalTicks = 0;

static unsigned long long \_previousIdleTicks = 0;

unsigned long long totalTicksSinceLastTime = totalTicks - \_previousTotalTicks;

unsigned long long idleTicksSinceLastTime = idleTicks - \_previousIdleTicks;

float ret = 1.0f - ((totalTicksSinceLastTime > 0) ? ((float)idleTicksSinceLastTime) / totalTicksSinceLastTime : 0);

\_previousTotalTicks = totalTicks;

\_previousIdleTicks = idleTicks;

return ret;

}

static unsigned long long FileTimeToInt64(const FILETIME& ft)

{

return (((unsigned long long)(ft.dwHighDateTime)) << 32) | ((unsigned long long)ft.dwLowDateTime);

}

float GetCPULoad()

{

FILETIME idleTime, kernelTime, userTime;

return GetSystemTimes(&idleTime, &kernelTime, &userTime) ? CalculateCPULoad(FileTimeToInt64(idleTime), FileTimeToInt64(kernelTime) + FileTimeToInt64(userTime)) : -1.0f;

}