Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе №4

на тему

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

Студент: гр.153502

Кирзнер А. П.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Цель работы 3

2 Теоретические сведения 4

3 Результат выполнения программы 5

Список использованных источников 6

Приложение А 7

# 1 ЦЕЛЬ РАБОТЫ

Создать приложение для анализа производительности процессора и памяти, отображающее графики загрузки ресурсов в реальном времени.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Функция *SetRect* устанавливает координаты указанного прямоугольника. Это эквивалентно присвоению левого, верхнего, правого и нижнего аргументов соответствующим членам структуры *RECT* [1].

Функция *FillRect* заполняет прямоугольник, используя указанную кисть. Эта функция включает левую и верхнюю границы, но исключает правую и нижнюю границы прямоугольника [2].

Функция *FrameRect* рисует границу вокруг указанного прямоугольника, используя указанную кисть. Ширина и высота границы всегда представляют собой одну логическую единицу [3].

Функция *BeginPaint* подготавливает указанное окно к рисованию и заполняет структуру *PAINTSTRUCT* информацией о рисовании [4].

Функция *GetSystemTimes* получает информацию о системном времени. В многопроцессорной системе возвращаемые значения представляют собой сумму назначенного времени для всех процессоров [5].

Функция *GlobalMemoryStatusEx* получает информацию о текущем использовании системой физической и виртуальной памяти [6].

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Главное окно приложения состоит из двух графиков для отображения загруженности процессора и памяти соответственно с течением времени. Графики разбиты на 10 полос для лучшего понимания значений. Также возле каждого графика расположено процентное количество загруженности в данный момент времени (рисунок 1).

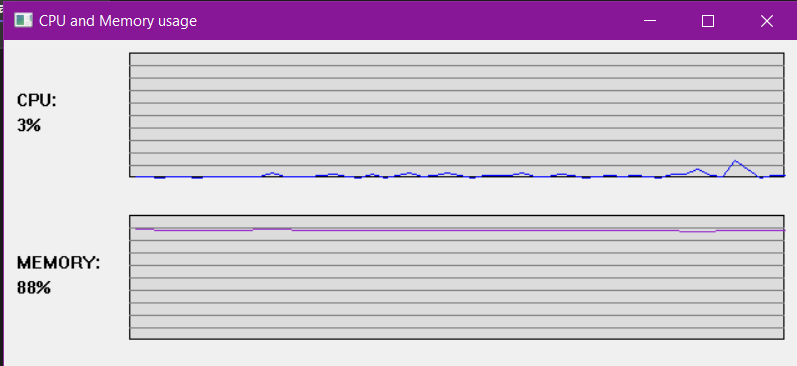


Рисунок 1 – Графический интерфейс программы

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функция SetRect (winuser.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setrect>.

[2] Функция FillRect (winuser.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-fillrect>.

[3] Функция FrameRect (winuser.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru/windows/win32/api/winuser/nf-winuser-framerect>.

[4] Функция BeginPaint (winuser.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-beginpaint>.

[5] Функция GetSystemTimes (processthreadsapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-getsystemtimes>.

[6] Функция GlobalMemoryStatusEx (sysinfoapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex>.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл Lab4.h**

#pragma once

HANDLE handleThread;

HWND cpuText;

HWND memoryText;

double cpu = -1;

double memory = -1;

int cpuArray[400];

int memoryArray[400];

FILETIME idleTime;

FILETIME kernelTime;

FILETIME userTime;

FILETIME last\_idleTime;

FILETIME last\_kernelTime;

FILETIME last\_userTime;

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);

DWORD ThreadProc(LPVOID lpParam);

**Файл Lab4.cpp**

#include <Windows.h>

#include <thread>

#include <string>

#include "Lab4.h"

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

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInst, LoadIcon(NULL, IDI\_APPLICATION), L"MainWndClass", SoftwareMainProcedure);

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

MSG SoftwareMainMessage = { 0 };

HWND hWnd = CreateWindow(L"MainWndClass", L"CPU and Memory usage", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 500, 250, NULL, NULL, NULL, NULL);

ShowWindow(hWnd, ncmdshows);

UpdateWindow(hWnd);

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

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

return 0;

}

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

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

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) {

HPEN hpen, hpenCPUGraph, hpenMemGraph;

RECT rect, rectCPUGraph, rectMemGraph, rectWindow;

switch (msg) {

case WM\_CREATE: {

for (int i = 400 - 1; i >= 0; i--) {

cpuArray[i] = -1;

memoryArray[i] = -1;

}

handleThread = CreateThread(NULL, 0, ThreadProc, NULL, NULL, NULL);

CreateWindow(

L"STATIC",

L"CPU:",

WS\_VISIBLE | WS\_CHILD,

10,

40,

80,

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

cpuText = CreateWindow(

L"STATIC",

L"",

WS\_VISIBLE | WS\_CHILD,

10,

60,

80,

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

CreateWindow(

L"STATIC",

L"MEMORY:",

WS\_VISIBLE | WS\_CHILD,

10,

170,

80,

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

memoryText = CreateWindow(

L"STATIC",

L"",

WS\_VISIBLE | WS\_CHILD,

10,

190,

80,

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

break;

}

case WM\_PAINT: {

GetClientRect(hWnd, &rectWindow);

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

SetRect(&rectCPUGraph, 100, 10, rectWindow.right - 10, 110);

SetRect(&rectMemGraph, 100, 140, rectWindow.right - 10, 240);

FillRect(hdc, &rectCPUGraph, CreateSolidBrush(RGB(220, 220, 220)));

FillRect(hdc, &rectMemGraph, CreateSolidBrush(RGB(220, 220, 220)));

FrameRect(hdc, &rectCPUGraph, CreateSolidBrush(RGB(0, 0, 0)));

FrameRect(hdc, &rectMemGraph, CreateSolidBrush(RGB(0, 0, 0)));

hpen = CreatePen(PS\_SOLID, 0, RGB(128, 128, 128));

SelectObject(hdc, hpen);

for (int i = 10; i < 100; i += 10) {

MoveToEx(hdc, rectCPUGraph.left, rectCPUGraph.bottom - i, 0);

LineTo(hdc, rectCPUGraph.right, rectCPUGraph.bottom - i);

MoveToEx(hdc, rectMemGraph.left, rectMemGraph.bottom - i, 0);

LineTo(hdc, rectMemGraph.right, rectMemGraph.bottom - i);

}

DeleteObject(hpen);

hpenCPUGraph = CreatePen(PS\_SOLID, 1, RGB(0, 0, 255));

SelectObject(hdc, hpenCPUGraph);

for (int i = 1, j = 10; i < 400; i++, j += 10)

{

if ((cpuArray[i + 1] >= 0) && ((rectWindow.right - j) > rectCPUGraph.left + 10))

{

MoveToEx(hdc, rectWindow.right - j, 110 - cpuArray[i], 0);

LineTo(hdc, rectWindow.right - j - 10, 110 - cpuArray[i + 1]);

}

}

DeleteObject(hpenCPUGraph);

hpenMemGraph = CreatePen(PS\_SOLID, 1, RGB(153, 50, 204));

SelectObject(hdc, hpenMemGraph);

for (int i = 0, j = 10; i < 400; i++, j += 10)

{

if ((memoryArray[i + 1] >= 0) && ((rectWindow.right - j) > rectMemGraph.left + 10))

{

MoveToEx(hdc, rectWindow.right - j, rectMemGraph.bottom - memoryArray[i], 0);

LineTo(hdc, rectWindow.right - j - 10, rectMemGraph.bottom - memoryArray[i + 1]);

}

}

DeleteObject(hpenMemGraph);

EndPaint(hWnd, &ps);

break;

}

case WM\_GETMINMAXINFO:

{

LPMINMAXINFO lpMMI = (LPMINMAXINFO)lp;

lpMMI->ptMinTrackSize.x = 650;

lpMMI->ptMinTrackSize.y = 300;

lpMMI->ptMaxTrackSize.x = 650;

lpMMI->ptMaxTrackSize.y = 300;

break;

}

case WM\_DESTROY:

CloseHandle(handleThread);

PostQuitMessage(0);

break;

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

}

}

DWORD ThreadProc(LPVOID lpParam)

{

GetSystemTimes(&last\_idleTime, &last\_kernelTime, &last\_userTime);

for (;;)

{

if (GetSystemTimes(&idleTime, &kernelTime, &userTime) != 0)

{

double usr = userTime.dwLowDateTime - last\_userTime.dwLowDateTime;

double ker = kernelTime.dwLowDateTime - last\_kernelTime.dwLowDateTime;

double idl = idleTime.dwLowDateTime - last\_idleTime.dwLowDateTime;

double sys = ker + usr;

last\_idleTime = idleTime;

last\_userTime = userTime;

last\_kernelTime = kernelTime;

if (sys != 0) {

cpu = (sys - idl) / sys \* 100;

for (int i = 400 - 1; i > 0; i--) {

cpuArray[i] = cpuArray[i - 1];

}

cpuArray[0] = cpu;

int cpuValue = static\_cast<int>(cpu);

std::wstring cpuString = std::to\_wstring(cpuValue) + L"%";

SetWindowText(cpuText, cpuString.c\_str());

}

}

MEMORYSTATUSEX statex;

statex.dwLength = sizeof(statex);

GlobalMemoryStatusEx(&statex);

memory = statex.dwMemoryLoad;

for (int i = 400 - 1; i > 0; i--) {

memoryArray[i] = memoryArray[i - 1];

}

memoryArray[0] = memory;

int memoryValue = static\_cast<int>(memory);

std::wstring memoryString = std::to\_wstring(memoryValue) + L"%";

SetWindowText(memoryText, memoryString.c\_str());

InvalidateRect(FindWindow(L"MainWndClass", NULL), NULL, TRUE);

Sleep(1000);

}

}