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 №2

на тему

РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN 32 И GDI. ФОРМИРОВАНИЕ СЛОЖНЫХ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ, ОБРАБОТКА РАЗЛИЧНЫХ СООБЩЕНИЙ, МЕХАНИЗМ ПЕРЕХВАТА СООБЩЕНИЙ (WINHOOK).

Выполнил студент гр.153503 Карась З.Э.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc146873457)

[2 Теоретические сведения 4](#_Toc146873458)

[3 Описание функций программы 5](#_Toc146873459)

[Список использованных источников 6](#_Toc146873460)

[ПРИЛОЖЕНИЕ А 7](#_Toc146873461)

1 Формулировка задачи

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, позволяющим отработать такие навыки, как: расширенное использование оконного интерфейса Win 32 и GDI, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook).

В качестве задачи необходимо разработать необходимо создание оконного приложения, которое будет отображать информацию о состоянии батареи и отображать информацию графически с помощью GDI.

2 Теоретические сведения

В представленном коде на C++/WinAPI реализована простая программная оболочка для отображения уровня заряда батареи и кнопки для перевода компьютера в режим сна.

– GDI+ (Graphics Device Interface Plus) — это набор функций и классов для работы с графикой в операционных системах Windows. В коде используется для отрисовки прямоугольников и заполнения их цветом.

– Структура WNDCLASSEX (Window Class Extended) используется для регистрации класса окна. В коде она определяет различные атрибуты окна, такие как обработчик оконных сообщений, идентификатор класса, стиль окна и другие.

– Window Procedure (оконная процедура) — это функция, которая обрабатывает все сообщения, отправленные окну. В коде определена функция WndProc, которая обрабатывает различные сообщения, такие как создание окна, закрытие окна и команды от кнопки.

– Функция CreateThread используется для создания нового потока, который выполняет функцию Power. Функция TerminateThread используется для принудительного завершения потока перед закрытием программы.

– Класс Battery представляет сведения о состоянии батареи компьютера. Он использует функции API для получения информации о заряде батареи и передачи этой информации в главное окно приложения.

3 Описание функций программы

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Графическое отображение батареи.

1. **Графическое отображение заряда батареи**

При Запуске программы, снизу вы увидите графическое отображение батареи.

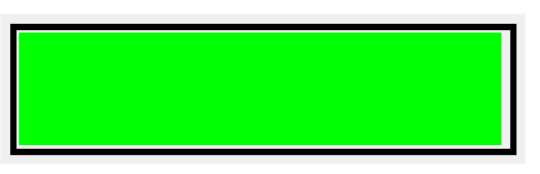


Рисунок 1 – Графическое отображение батареи

Список использованных источников

[1] Rich Edit [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/controls/rich-edit-controls

[2] Win32 API Programming Tutorial [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.winprog.org/tutorial/

[3] Разработка приложений с помощью WinAPI. – Электронные данные. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Battery.h

#pragma once

#include "libraries.h"

using namespace std;

class Battery

{

private:

string powerSupply;

string savingMode;

string stateCharge;

int percent;

int lifeTime;

public:

Battery();

string getPowerSupply(SYSTEM\_POWER\_STATUS status);

string getStateCharge(SYSTEM\_POWER\_STATUS status);

string getSavingMode(SYSTEM\_POWER\_STATUS status);

int getPercent();

friend string BatteryStatus(Battery battery);

};

Файл Libraries.h

#pragma comment(lib, "PowrProf.lib")

#pragma comment (lib, "setupapi.lib")

#include <Windows.h>

#include <iostream>

#include <powrprof.h>

#include <thread>

#include "conio.h"

#include <Poclass.h>

#include <Setupapi.h>

#include <devguid.h>

#include <sstream>

#include <objidl.h>

#include <gdiplus.h>

Файл Battery.cpp

#include "Battery.h"

Battery::Battery() {

SYSTEM\_POWER\_STATUS status;

GetSystemPowerStatus(&status);

this->powerSupply = getPowerSupply(status);

this->percent = status.BatteryLifePercent;

this->lifeTime = status.BatteryLifeTime;

this->stateCharge = getStateCharge(status);

this->savingMode = getSavingMode(status);

}

string Battery::getPowerSupply(SYSTEM\_POWER\_STATUS status) {

if (status.ACLineStatus == 1) {

return "mains";

}

else if (status.ACLineStatus == 0) {

return "battery";

}

else {

return "unknown";

}

}

string Battery::getStateCharge(SYSTEM\_POWER\_STATUS status) {

switch (status.BatteryFlag) {

case 0:

return "not charging";

case 1:

return "high ( > 66% )";

case 2:

return "low ( > 20% )";

case 4:

return "critical ( < 5% )";

case 8:

return "charging";

case 128:

return "No system battery";

default:

return "unknown";

}

}

string Battery::getSavingMode(SYSTEM\_POWER\_STATUS status) {

if (status.SystemStatusFlag) {

return "on";

}

return "off";

}

int Battery::getPercent()

{

return percent;

}

string BatteryStatus(Battery battery)

{

std::ostringstream oss;

oss << "BATTERY" << std::endl

<< " Power supply: " << battery.powerSupply << std::endl

<< " State of charge: " << battery.stateCharge << std::endl

<< " Percentage: " << battery.percent << "%" << std::endl

<< " Saving mode: " << battery.savingMode << std::endl

<< " Life time: " << battery.lifeTime << "s" << std::endl;

oss << std::endl;

return oss.str();

}

Файл Source.cpp

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#include "libraries.h"

#include "Battery.h"

void MainWndAddWidgets(HWND hWnd);

void MainWndAddButton(HWND hWnd);

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

DWORD WINAPI Power(LPVOID lpParameter);

volatile bool isThreading = true;

HANDLE readThread;

HWND staticWindow;

HWND childWindow;

int startPercent = 0;

using namespace Gdiplus;

#pragma comment (lib,"Gdiplus.lib")

void OnPaint(HDC hdc)

{

Graphics graphics(hdc);

Pen pen1(Color(255, 0, 0, 0), 5);

Pen pen2(Color(0, 255, 0, 0));

SolidBrush brush(Color(0, 255, 0)); // Зеленая кисть

graphics.DrawRectangle(&pen1, 10, 10, 400, 100);

graphics.DrawRectangle(&pen2, 15, 15, 390, 90);

int greenRectWidth = static\_cast<int>(390 \* startPercent / 100);

graphics.FillRectangle(&brush, 15, 15, greenRectWidth, 90);

}

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

{

MSG msg{}; //структура которая содержит информацию о сообщении

HWND hwnd{}; //дескриптор окна, хранит информацию об окне

WNDCLASSEX wc{ sizeof(WNDCLASSEX) }; //отвечает за характеристики окна

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

// Initialize GDI+.

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

wc.cbClsExtra = 0; //доп выделение памяти в классе окна

wc.cbWndExtra = 0; //доп выделение памяти в классе окна

wc.hInstance = hInstance;

wc.hIcon = LoadIcon(nullptr, IDI\_APPLICATION);

wc.hCursor = LoadCursor(nullptr, IDC\_ARROW);

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

wc.hIconSm = LoadIcon(nullptr, IDI\_APPLICATION);

wc.lpfnWndProc = WndProc;

wc.lpszClassName = L"BatteryApplication";

wc.lpszMenuName = nullptr;

wc.style = CS\_VREDRAW | CS\_HREDRAW;

if (!RegisterClassExW(&wc))

return EXIT\_FAILURE;

hwnd = CreateWindow(wc.lpszClassName, L"Battery", WS\_OVERLAPPEDWINDOW, 0, 0, 600, 600, nullptr, nullptr, wc.hInstance, nullptr);

if (hwnd == INVALID\_HANDLE\_VALUE)

return EXIT\_FAILURE;

ShowWindow(hwnd, nCmdShow); //показать окно

UpdateWindow(hwnd); //перерисовка окна

while (GetMessage(&msg, nullptr, 0, 0)) //Цикл обработки сообщений

{

TranslateMessage(&msg); //расшифровывает сообщение

DispatchMessage(&msg); //передает сообщение в оконную процедуру

}

TerminateThread(readThread, 0);

GdiplusShutdown(gdiplusToken);

return static\_cast<int>(msg.wParam);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

switch (uMsg)

{

case WM\_CREATE:

{

MainWndAddButton(hWnd);

MainWndAddWidgets(hWnd);

readThread = CreateThread(nullptr, 0, Power, nullptr, 0, nullptr);

}

return 0;

/\*case WM\_PAINT:

{

hdc = BeginPaint(hWnd, &ps);

OnPaint(hdc);

EndPaint(hWnd, &ps);

}

return 0;\*/

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case 1337:

{

SetSuspendState(TRUE, FALSE, FALSE);

}

break;

}

}

return 0;

case WM\_DESTROY:

{

isThreading = false;

CloseHandle(readThread);

PostQuitMessage(EXIT\_SUCCESS);

}

return 0;

}

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

}

LRESULT CALLBACK ChildWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

switch (uMsg)

{

case WM\_PAINT:

{

hdc = BeginPaint(hWnd, &ps);

OnPaint(hdc);

EndPaint(hWnd, &ps);

}

return 0;

case WM\_DESTROY:

{

PostQuitMessage(EXIT\_SUCCESS);

}

return 0;

}

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

}

void MainWndAddWidgets(HWND hWnd)

{

staticWindow = CreateWindow(L"STATIC", L"Hello window", WS\_VISIBLE | WS\_CHILD | BS\_MULTILINE, 5, 5, 480, 160, hWnd, nullptr, nullptr, nullptr);

// Create child window

WNDCLASSEX wc{ sizeof(WNDCLASSEX) };

wc.hInstance = GetModuleHandle(nullptr);

wc.lpfnWndProc = ChildWndProc;

wc.lpszClassName = L"ChildWindowClass";

wc.hbrBackground = CreateSolidBrush(RGB(240, 240, 240));

RegisterClassEx(&wc);

childWindow = CreateWindow(wc.lpszClassName, L"Child Window", WS\_VISIBLE | WS\_CHILD, 5, 300, 420, 120, hWnd, nullptr, wc.hInstance, nullptr);

}

void MainWndAddButton(HWND hWnd)

{

CreateWindow(L"BUTTON", L"Sleep", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 10, 200, 300, 30, hWnd, HMENU(1337), nullptr, nullptr);

}

DWORD WINAPI Power(LPVOID lpParameter)

{

while (isThreading) {

Battery battery;

if (battery.getPercent() != startPercent)

{

//update wnd

InvalidateRect(childWindow, nullptr, TRUE);

}

startPercent = battery.getPercent();

SetWindowTextA(staticWindow, BatteryStatus(battery).c\_str());

Sleep(1000);

}

return 0;

}