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 № 3

на тему

**УПРАВЛЕНИЕ ПАМЯТЬЮ И ВВОДОМ-ВЫВОДОМ, РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ВВОДА-ВЫВОДА WINDOWS ФУНКЦИИ API ПОДСИСТЕМЫ ПАМЯТИ WIN 32. ОРГАНИЗАЦИЯ И КОНТРОЛЬ АСИНХРОННЫХ ОПЕРАЦИЙ ВВОДА-ВЫВОДА. ОТОБРАЖЕНИЕ ФАЙЛОВ В ПАМЯТЬ.**

Выполнил:

студент гр. 153503

Вергасов В.М.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_Toc146883343)

[2 Краткие теоретические сведения 4](#_Toc146883344)

[3 Полученные результаты 11](#_Toc146883345)

[Выводы 12](#_Toc146883346)

[Список литературы 13](#_Toc146883347)

# 1 ЦЕЛИ РАБОТЫ

1. Изучить основные функции и возможности подсистемы памяти Win 32, такие как выделение и освобождение памяти, управление памятью, работа с указателями и массивами.
2. Освоить работу с функциями API для управления вводом-выводом, такими как *CreateFile*, *ReadFile*, *WriteFile*, *CloseHandle* и другими.
3. Научиться использовать асинхронные операции ввода-вывода с использованием функций API, таких как *CreateFileMapping*, *MapViewOfFile*, *UnmapViewOfFile* и других.
4. Освоить отображение файлов в память с использованием функций *API*, таких как *CreateFileMapping*, *MapViewOfFile*, и *UnmapViewOfFile*.
5. Изучить организацию и контроль асинхронных операций ввода-вывода, таких как создание и управление потоками, использование семафоров и событий.
6. Разработать и реализовать программу, демонстрирующую использование функций *API* для управления памятью, вводом-выводом и отображением файлов в память.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление памятью и вводом-выводом являются важными аспектами разработки программного обеспечения на платформе Windows [1]. В этом контексте функции API подсистемы памяти Win32 предоставляют базовые инструменты для управления памятью и организации ввода-вывода, а расширенные возможности ввода-вывода позволяют выполнять более сложные операции.

1 Функции *API* подсистемы памяти Windows:

- *VirtualAlloc*: выделение виртуальной памяти

- *VirtualFree*: освобождение виртуальной памяти

- *HeapCreate* и *HeapDestroy*: создание и уничтожение кучи памяти

- *malloc* и *free*: стандартная C-библиотека для выделения и освобождения памяти

2 Расширенные возможности ввода-вывода в Windows включают:

- *Asynchronous I/O*: использование асинхронного ввода-вывода для улучшения производительности и отзывчивости приложения

- *I/O Completion Ports*: механизм обработки запросов ввода-вывода с использованием многопоточности

- *Overlapped I/O*: асинхронный ввод-вывод с использованием буферов, перекрывающих несколько операций

3 Отображение файлов в память:

- *ReadFile* и *WriteFile*: функции *API* для чтения и записи данных в файл

- *CreateFileMapping* и *MapViewOfFile*: отображение файла в память для доступа к его содержимому

- *UnmapViewOfFile* и *CloseHandle*: освобождение памяти и закрытие файла после использования

Контроль асинхронных операций ввода-вывода и отображение файлов в память позволяют разработчикам создавать более эффективные и надежные приложения.

# 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Результат работы программы после запуска показан на рисунке 3.1.

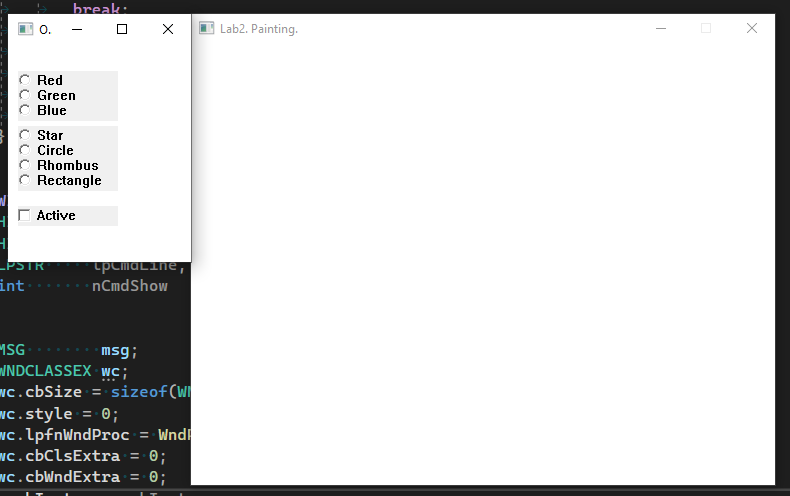


Рисунок 3.1 – Результат работы программы после запуска

После выбора фигуры, ее цвета и активации рисования – можно создавать фигуры на полотне. Пример заполненного полотна отображен на рисунке 3.2.

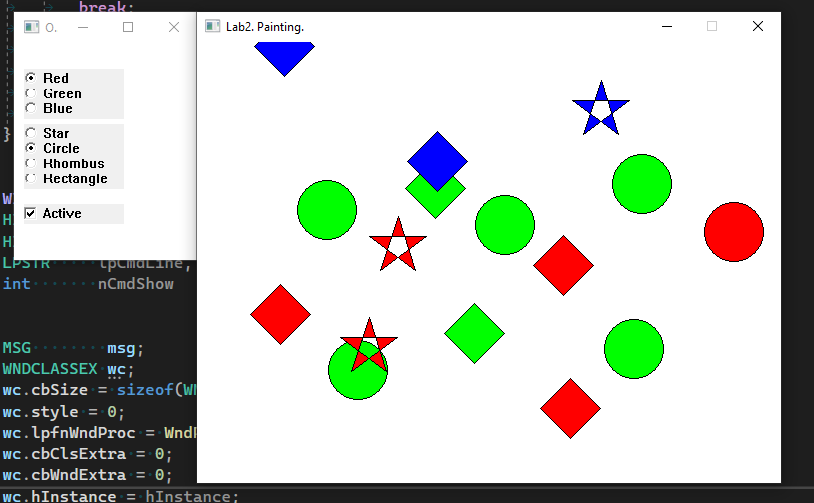


Рисунок 3.2 – Нарисованные фигуры в области для рисования

Так же с помощью отображения файла программа пишет в файл логи при срабатывании WinHook, пример можно увидеть на рисунке 3.3. Исходный код программы отображен в листинге 1.

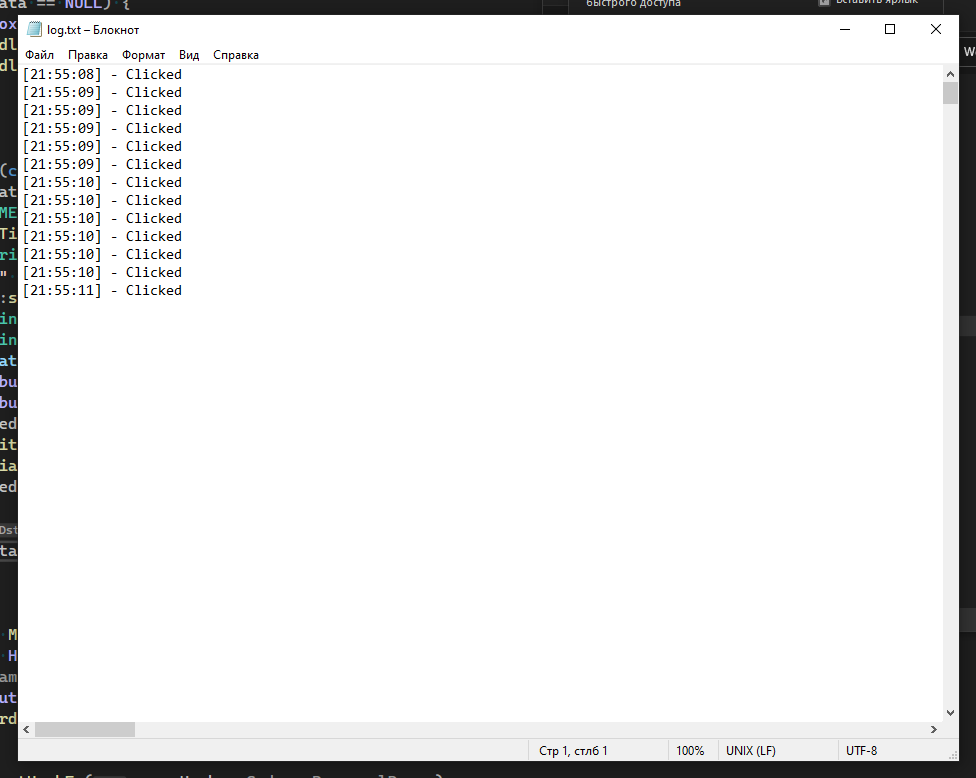


Рисунок 3.3 – логи программы

Файл будет перезаписываться при перезапуске программы. Т.к. логи пишутся из функции *WinHook*, то будут записаны все нажатия на левую клавишу мыши, пока программа запущена.

Листинг 1 **–** Код исходной программы:

#ifndef UNICODE

#define UNICODE

#define UNICODE\_WAS\_UNDEFINED

#endif

#include <Windows.h>

#include <wchar.h>

#include <tchar.h>

#include <cmath>

#include <string>

#include <sstream>

#include <iomanip>

#ifdef UNICODE\_WAS\_UNDEFINED

#undef UNICODE

#endif

#define PI acos(-1)

#define ID\_CHECK\_BOX 228

#define WS\_RADIOBUTTON\_PARAMS WS\_VISIBLE | WS\_CHILD | BS\_RADIOBUTTON

CONST WCHAR \*fileName = L"log.txt";

CONST long long FILESIZE = 1024 \* 1024;

const wchar\_t \*colors[3] = {

L"Red",

L"Green",

L"Blue"

};

const wchar\_t \*figures[4] = {

L"Star",

L"Circle",

L"Rhombus",

L"Rectangle"

};

const int colors\_id[3] = {

101,

102,

103

};

const int figures\_id[4] = {

104,

105,

106,

107

};

int RGB[3];

int x\_coord = 0, y\_coord = 0;

HANDLE hFile = NULL;

HANDLE hMapFile = NULL;

LPVOID pMappedData = NULL;

size\_t mappedDataSize = 0;

HWND check\_box,

child\_hwnd,

color\_radio\_buttons[3],

figure\_radio\_buttons[4];

HWND hwnd\_main;

HINSTANCE hInst;

HHOOK mouseHook = NULL;

void UninitializeMappingFile() {

if (pMappedData != NULL) {

UnmapViewOfFile(pMappedData);

pMappedData = NULL;

}

if (hMapFile != NULL) {

CloseHandle(hMapFile);

hMapFile = NULL;

}

if (hFile != INVALID\_HANDLE\_VALUE) {

CloseHandle(hFile);

hFile = INVALID\_HANDLE\_VALUE;

}

}

VOID InitializeMappingFile() {

hFile = CreateFile(

fileName,

GENERIC\_READ | GENERIC\_WRITE,

0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

if (hFile == INVALID\_HANDLE\_VALUE) {

MessageBox(NULL, L"CreateFile failed!", L"Error", MB\_ICONERROR);

return;

}

hMapFile = CreateFileMapping(

hFile,

NULL,

PAGE\_READWRITE,

0,

FILESIZE,

NULL

);

if (hMapFile == NULL) {

MessageBox(NULL, L"CreateFileMapping failed!", L"Error", MB\_ICONERROR);

CloseHandle(hFile);

return;

}

pMappedData = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, FILESIZE);

if (pMappedData == NULL) {

MessageBox(NULL, L"MapViewOfFile failed!", L"Error", MB\_ICONERROR);

CloseHandle(hMapFile);

CloseHandle(hFile);

return;

}

}

void RecordPress(const std::string &actionString) {

if (pMappedData != NULL) {

SYSTEMTIME currentTime;

GetLocalTime(&currentTime);

std::ostringstream os;

os << "[" << std::setw(2) << std::setfill('0') << std::to\_string(currentTime.wHour) << ":" <<

std::setw(2) << std::setfill('0') << std::to\_string(currentTime.wMinute) << ":" << std::setw(2) << std::setfill('0') << std::to\_string(currentTime.wSecond);

std::string time = os.str();

std::string keyInfo = time + "] - " + actionString + "\n";

SIZE\_T dataSize = keyInfo.size() \* sizeof(CHAR);

OutputDebugString(std::to\_wstring(dataSize).c\_str());

OutputDebugString(L"\n");

if (mappedDataSize + dataSize >= FILESIZE) {

UninitializeMappingFile();

InitializeMappingFile();

mappedDataSize = 0;

}

memcpy((CHAR \*)pMappedData + mappedDataSize, keyInfo.c\_str(), dataSize);

mappedDataSize += dataSize;

}

}

LRESULT CALLBACK MouseHookProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode == HC\_ACTION) {

if (wParam == WM\_LBUTTONDOWN) {

OutputDebugString(L"Clicked\n");

RecordPress("Clicked");

}

}

return CallNextHookEx(mouseHook, nCode, wParam, lParam);

}

void DrawFigure(HDC hdc, int type) {

HBRUSH hBrush = CreateSolidBrush(RGB(RGB[0], RGB[1], RGB[2]));

SelectObject(hdc, hBrush);

switch (type) {

case 0:

{

POINT point[5];

for (int i = 0; i < 5; ++i) {

point[i].x = x\_coord + sin(2 \* PI / 5 \* (i \* 2 % 5)) \* 30;

point[i].y = y\_coord - cos(2 \* PI / 5 \* (i \* 2 % 5)) \* 30;

}

Polygon(hdc, point, 5);

break;

}

case 1:

{

Ellipse(hdc, x\_coord - 30, y\_coord - 30, x\_coord + 30, y\_coord + 30);

break;

}

case 2:

{

POINT point[4];

point[0].x = x\_coord - 30;

point[0].y = y\_coord;

point[1].x = x\_coord;

point[1].y = y\_coord - 30;

point[2].x = x\_coord + 30;

point[2].y = y\_coord;

point[3].x = x\_coord;

point[3].y = y\_coord + 30;

Polygon(hdc, point, 4);

break;

}

case 3:

{

Rectangle(hdc, x\_coord - 30, y\_coord - 30, x\_coord + 30, y\_coord + 30);

break;

}

}

}

LRESULT CALLBACK WndProcChild(

HWND hwnd,

UINT msg,

WPARAM wParam,

LPARAM lParam

) {

switch (msg) {

case WM\_LBUTTONDOWN:

{

x\_coord = LOWORD(lParam);

y\_coord = HIWORD(lParam);

InvalidateRect(child\_hwnd, 0, 0);

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

if (SendMessage(check\_box, BM\_GETCHECK, 1, NULL)) {

for (int i = 0; i < 3; ++i) {

if (SendMessage(color\_radio\_buttons[i], BM\_GETCHECK, 1, 0)) {

RGB[i] = 255;

} else {

RGB[i] = 0;

}

}

for (int i = 0; i < 4; ++i) {

if (SendMessage(figure\_radio\_buttons[i], BM\_GETCHECK, 1, NULL)) {

DrawFigure(hdc, i);

}

}

}

EndPaint(hwnd, &ps);

break;

}

case WM\_CLOSE:

{

SendMessage(hwnd\_main, msg, 0, NULL);

break;

}

default:

{

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

}

}

}

LRESULT CALLBACK WndProc(

HWND hwnd,

UINT msg,

WPARAM wParam,

LPARAM lParam

) {

switch (msg) {

case WM\_CREATE:

{

WNDCLASS wc;

memset(&wc, 0, sizeof(WNDCLASS));

wc.style = 0;

wc.hInstance = hInst;

wc.lpfnWndProc = WndProcChild;

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

wc.lpszClassName = L"C Windows";

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

RegisterClass(&wc);

child\_hwnd = CreateWindow(

L"C Windows", L"Lab2. Painting.",

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_THICKFRAME | WS\_MINIMIZEBOX,

CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 480, NULL, NULL, hInst, NULL

);

ShowWindow(child\_hwnd, SW\_NORMAL);

UpdateWindow(child\_hwnd);

if (child\_hwnd == NULL) {

MessageBox(hwnd, L"Can't create child window.", L"Error!", MB\_ICONERROR | MB\_OK);

}

for (size\_t i = 0; i < 3; ++i) {

color\_radio\_buttons[i] = CreateWindow(

L"BUTTON", colors[i], WS\_RADIOBUTTON\_PARAMS,

10, 10 + 15 \* (i + 1), 100, 20, hwnd, (HMENU)colors\_id[i], NULL, NULL

);

}

for (size\_t i = 0; i < 4; ++i) {

figure\_radio\_buttons[i] = CreateWindow(

L"BUTTON", figures[i], WS\_RADIOBUTTON\_PARAMS,

10, 65 + 15 \* (i + 1), 100, 20, hwnd, (HMENU)figures\_id[i], NULL, NULL

);

}

check\_box = CreateWindow(

L"BUTTON", L"Active", WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

10, 160, 100, 20, hwnd, (HMENU)ID\_CHECK\_BOX, NULL, NULL

);

break;

}

case WM\_COMMAND:

{

int target\_id = LOWORD(wParam);

if (target\_id == ID\_CHECK\_BOX) {

bool status = SendDlgItemMessage(hwnd, ID\_CHECK\_BOX, BM\_GETCHECK, 0, 0);

SendMessage(check\_box, BM\_SETCHECK, status ^ true, NULL);

} else if (target\_id >= 101 && target\_id <= 103) {

for (int i = 0; i < 3; ++i) {

SendMessage((HWND)color\_radio\_buttons[i], BM\_SETCHECK, 0, NULL);

}

SendMessage(color\_radio\_buttons[target\_id - 101], BM\_SETCHECK, 1, NULL);

} else if (target\_id >= 104 && target\_id <= 107) {

for (int i = 0; i < 4; ++i) {

SendMessage((HWND)figure\_radio\_buttons[i], BM\_SETCHECK, 0, NULL);

}

SendMessage((HWND)figure\_radio\_buttons[target\_id - 104], BM\_SETCHECK, 1, NULL);

}

break;

}

case WM\_CLOSE:

{

DestroyWindow(hwnd);

break;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

default:

{

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

}

}

}

void SetKeyboardHook() {

mouseHook = SetWindowsHookEx(WH\_MOUSE\_LL, MouseHookProc, NULL, 0);

if (mouseHook == NULL) {

MessageBox(NULL, L"Failed to set mouse hook", L"Error", MB\_ICONERROR);

}

}

void UnhookKeyboardHook() {

if (mouseHook != NULL) {

UnhookWindowsHookEx(mouseHook);

mouseHook = NULL;

}

}

int WINAPI WinMain(

HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow

) {

MSG msg;

WNDCLASSEX wc;

wc.cbSize = sizeof(WNDCLASSEX);

wc.style = 0;

wc.lpfnWndProc = WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hInstance;

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

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

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

wc.lpszMenuName = NULL;

wc.lpszClassName = L"VadVergasovWindowClass";

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

if (!RegisterClassEx(&wc)) {

MessageBox(NULL, L"Window registration is failed!", L"Error!", MB\_ICONWARNING | MB\_OK);

return 0;

}

hwnd\_main = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"VadVergasovWindowClass",

L"Operating Environments and Operating Systems, Lab: 2",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 180, 250,

NULL, NULL, hInstance, NULL

);

if (hwnd\_main == NULL) {

MessageBox(NULL, L"Window creation is failed!", L"Error!", MB\_ICONWARNING | MB\_OK);

return 0;

}

hInst = hInstance;

InitializeMappingFile();

SetKeyboardHook();

ShowWindow(hwnd\_main, nCmdShow);

UpdateWindow(hwnd\_main);

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookKeyboardHook();

UninitializeMappingFile();

return msg.wParam;

}

# ВЫВОДЫ

В результате выполнения лабораторной работы были изучены расширенные возможности оконного интерфейса *Win32*, включая формирование сложных изображений и создание различных элементов управления. Реализована возможность записи логов в файл через отображение файла в виртуальную память.

Также были разработана программа, демонстрирующая возможности расширенного оконного интерфейса, *WinHook* и *GDI*, а также создан простой графический редактор с возможностью записи логов в файл.

# СПИСОК ЛИТЕРАТУРЫ

1. Programming reference for the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ – Дата доступа: 21.09.2023.