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.**

Студент И. А. Тиханёнок

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель работы ­ изучить и применить на практике знания об управлении памятью и вводом-выводом, научиться использовать расширенные возможности ввода-вывода Windows, использовать функции API подсистемы памяти Win 32, а также изучить организацию и контроль асинхронных операций ввода-вывода и отображение файлов в память. Для достижения цели будет создано графическое приложение в виде стандартной игры - змейка.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление памятью в Win32 предоставляет разработчикам множество функций и средств для управления памятью. Основные функции включают `VirtualAlloc`, `VirtualFree`, `VirtualProtect` и `MapViewOfFile`. Эти функции позволяют выделять и освобождать виртуальную память, изменять атрибуты доступа и отображать файлы в память.

Расширенные возможности ввода-вывода в Windows включают функции `ReadFile` и `WriteFile`, которые позволяют считывать и записывать данные в файлы и другие устройства. Кроме того, Win32 поддерживает асинхронный ввод-вывод, что позволяет выполнять операции ввода-вывода параллельно, не блокируя главный поток выполнения программы.

Функции API подсистемы памяти в Win32 включают в себя `VirtualAlloc`, `VirtualFree`, `VirtualProtect` и `MapViewOfFile`, которые позволяют программистам управлять виртуальной памятью, выделять и освобождать участки памяти, изменять атрибуты доступа и отображать файлы в виртуальную память.

Организация и контроль асинхронных операций ввода-вывода также доступны в Win32. Это достигается с использованием механизма перекрывающихся операций (`Overlapped I/O`). Функции `ReadFile`, `WriteFile` и `DeviceIoControl` позволяют выполнять асинхронные операции ввода-вывода, что позволяет программам продолжать выполнение других задач, не ожидая завершения операций ввода-вывода.

Отображение файлов в память реализуется с помощью функции `CreateFileMapping`, которая позволяет отобразить файл в виртуальную память процесса. Это создает объект "отображения файла", который может быть использован для обмена данными между процессами и обеспечивает быстрый доступ к содержимому файла без явного чтения и записи.

Для отладки и мониторинга операций с памятью и вводом-выводом в Win32 существует набор инструментов, включая отладчики и профилировщики. Эти инструменты позволяют отслеживать процессы, мониторить использование памяти и анализировать производительность приложений. Управление памятью и вводом-выводом - это важные аспекты при разработке приложений под Windows, и правильное использование функций и механизмов Win32 позволяет создавать эффективные и надежные программы.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано графическое приложение в виде стандартной игры – змейка (рисунок 1).

Рисунок 1 – Окно приложения

Для изменения часового пояса используется обработка нажатия клавиш *VK\_LEFT* и *VK\_RIGHT*. В случае, если часовой пояс выходит за указанный диапазон, то приложение выводит окно с ошибкой (рисунок 2).

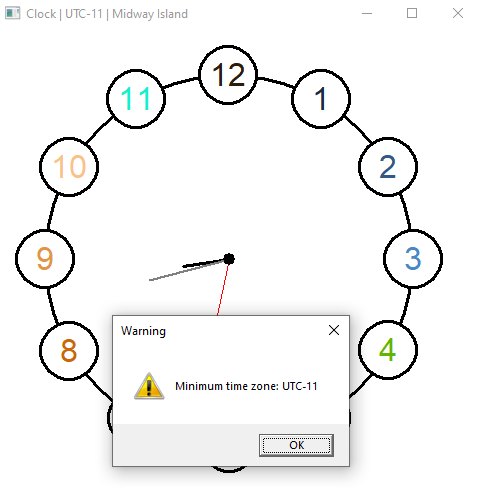


Рисунок 2– Выход за диапазон часового пояса

При нажатии пробела *VK\_SPACE* воспроизводится звук кукушки, нажатие обрабатывается с помощью хука.

# ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные принципы работы, которые иллюстрирует применение нескольких важных аспектов программирования под операционной системой Windows с использованием Win32 API: управление памятью и вводом-выводом: в приложении реализованы функции для сохранения и загрузки игрового состояния в файл. Это позволяет пользователям сохранять текущий прогресс игры и восстанавливать его в будущем. Данные считываются из файла и записываются в него, что представляет собой пример работы с вводом-выводом. Расширенные возможности ввода-вывода Windows: приложение реагирует на нажатия клавиш на клавиатуре и обновляет состояние игры в соответствии с этими действиями. Это демонстрирует обработку ввода с использованием Windows API. Функции API подсистемы памяти Win32: для обеспечения безопасности при работе с общими данными (например, информацией о змейке) в разных потоках, был использован мьютекс (mutex). Мьютекс позволяет синхронизировать доступ к данным и предотвращать гонки данных, что является важной частью работы с подсистемой памяти Win32. Организация и контроль асинхронных операций ввода-вывода: загрузка и сохранение игрового состояния выполняются асинхронно. Это означает, что они не блокируют пользовательский интерфейс приложения и позволяют игроку взаимодействовать с игрой во время выполнения этих операций. Отображение файлов в память: для хранения и управления данными о состоянии игры используется работа с файлами. Это позволяет сохранять информацию в файле на диске и взаимодействовать с ней через API Windows. В итоге данное приложение представляет собой игровой проект, который наглядно демонстрирует применение указанных аспектов программирования в контексте разработки многозадачных и многопоточных приложений под операционной системой Windows.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 30.09.2023

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Source.cpp

#include <windows.h>

#include <deque>

#include <fstream>

#include <thread>

#include <mutex>

#include <string>

#include "global\_defines.h"

using namespace std;

const int gridSize = 25;

const int screenWidth = 1100;

const int screenHeight = 800;

const wchar\_t\* saveFileName = L"D:\\Study\\OSiSP\\LR1\\LR1\\snake\_save.txt";

HWND hWnd; // Хранит хендл (дескриптор) главного окна приложения

HWND restartButton; // Хендл кнопки перезапуска игры

HHOOK g\_hook = NULL; // Хендл глобального хука

// Объявление функции LoadGameAsync

void LoadGameAsync();

// Объявление функции SaveGameAsync

void SaveGameAsync();

// Объявление функции RestartGame

void RestartGame();

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);

struct SnakeSegment {

int x, y;

SnakeSegment() : x(0), y(0) {} // Конструктор по умолчанию

SnakeSegment(int \_x, int \_y) : x(\_x), y(\_y) {}

};

deque<SnakeSegment> snake;

int foodX, foodY;

bool gameOver = false;

int direction = 1; // 0 - влево, 1 - вверх, 2 - вправо, 3 - вниз

int foodEaten = 0; // Счетчик съеденной еды

mutex snakeMutex; // Мьютекс для защиты доступа к змейке

void DrawCell(HDC hdc, int x, int y, COLORREF color) {

int cellWidth = screenWidth / gridSize;

int cellHeight = screenHeight / gridSize;

// Создаем кисть с заданным цветом

HBRUSH brush = CreateSolidBrush(color);

// Задаем цвет для контура ячейки

HPEN pen = CreatePen(PS\_SOLID, 1, color);

// Выбираем кисть и перо в контексте устройства

HBRUSH oldBrush = (HBRUSH)SelectObject(hdc, brush);

HPEN oldPen = (HPEN)SelectObject(hdc, pen);

// Рисуем прямоугольник

Rectangle(hdc, x \* cellWidth, y \* cellHeight, (x + 1) \* cellWidth, (y + 1) \* cellHeight);

// Восстанавливаем оригинальные кисть и перо

SelectObject(hdc, oldBrush);

SelectObject(hdc, oldPen);

// Удаляем созданные кисть и перо

DeleteObject(brush);

DeleteObject(pen);

}

// Функция обработки нажатия клавиш

void HandleInput(WPARAM wParam) {

switch (wParam) {

case VK\_LEFT:

if (direction != 2) direction = 0;

break;

case VK\_UP:

if (direction != 3) direction = 1;

break;

case VK\_RIGHT:

if (direction != 0) direction = 2;

break;

case VK\_DOWN:

if (direction != 1) direction = 3;

break;

case 'S':

SaveGameAsync();

break;

case 'L':

LoadGameAsync();

break;

}

}

// Функция обновления игры

void UpdateGame(HWND hWnd) {

// Обновляем позицию змейки в зависимости от направления

int headX = snake.front().x;

int headY = snake.front().y;

switch (direction) {

case 0: // Влево

headX--;

break;

case 1: // Вверх

headY--;

break;

case 2: // Вправо

headX++;

break;

case 3: // Вниз

headY++;

break;

}

// Проверяем столкновение с границей экрана

if (headX < 0 || headX >= screenWidth / gridSize || headY < 0 || headY >= screenHeight / gridSize) {

gameOver = true;

return;

}

// Проверяем столкновение с самой собой

lock\_guard<mutex> lock(snakeMutex);

for (const SnakeSegment& segment : snake) {

if (segment.x == headX && segment.y == headY) {

gameOver = true;

return;

}

}

// Проверяем, съела ли змейка еду

if (headX == foodX && headY == foodY) {

// Генерируем новую позицию для еды

foodX = rand() % (screenWidth / gridSize);

foodY = rand() % (screenHeight / gridSize);

foodEaten++; // Увеличиваем счетчик съеденной еды

// Проверяем, сколько яблок съедено, и выводим уведомление каждые 5 яблок

if (foodEaten % 5 == 0) {

MessageBox(hWnd, L"Вы съели 5 яблок!", L"Уведомление", MB\_OK | MB\_ICONINFORMATION);

}

}

else {

// Удаляем хвост змейки

snake.pop\_back();

}

// Добавляем новую голову

snake.push\_front(SnakeSegment(headX, headY));

// Обновляем экран

InvalidateRect(hWnd, NULL, TRUE);

}

void PaintGame(HDC hdc) {

// Очищаем экран и рисуем фон зеленым цветом

RECT clientRect;

GetClientRect(hWnd, &clientRect);

HBRUSH greenBrush = CreateSolidBrush(RGB(0, 255, 0)); // Зеленый цвет для фона

FillRect(hdc, &clientRect, greenBrush);

DeleteObject(greenBrush);

// Отображаем счетчик съеденной еды

wstring foodEatenText = L"Eat: " + to\_wstring(foodEaten);

TextOut(hdc, 10, 10, foodEatenText.c\_str(), foodEatenText.length());

// Рисуем еду красным цветом

SetBkColor(hdc, RGB(0, 255, 0)); // Задаем зеленый цвет текста (фона текста)

SetTextColor(hdc, RGB(0, 0, 0)); // Задаем черный цвет текста

DrawCell(hdc, foodX, foodY, RGB(255, 0, 0)); // Красный цвет для еды

// Рисуем змейку черным цветом

SetBkColor(hdc, RGB(0, 255, 0)); // Задаем зеленый цвет текста (фона текста)

SetTextColor(hdc, RGB(0, 0, 0)); // Задаем черный цвет текста

lock\_guard<mutex> lock(snakeMutex);

for (const SnakeSegment& segment : snake) {

DrawCell(hdc, segment.x, segment.y, RGB(0, 0, 0)); // Черный цвет для змейки

}

if (gameOver) {

// Выводим сообщение об окончании игры

TextOut(hdc, 10, 30, L"Game Over", 9);

}

}

void SaveGameAsync() {

lock\_guard<mutex> lock(snakeMutex);

ofstream file(saveFileName); // Открываем файл в текстовом режиме

if (file.is\_open()) {

file << snake.size() << endl; // Записываем размер змейки

for (const SnakeSegment& segment : snake) {

file << segment.x << "," << segment.y << endl; // Записываем каждый сегмент змейки

}

file.close();

}

}

void LoadGameAsync() {

ifstream file(saveFileName); // Открываем файл в текстовом режиме

if (file.is\_open()) {

int size;

file >> size; // Читаем размер змейки

deque<SnakeSegment> newSnake;

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

SnakeSegment segment;

char comma;

file >> segment.x >> comma >> segment.y; // Читаем каждый сегмент змейки

newSnake.push\_back(segment);

}

file.close();

lock\_guard<mutex> lock(snakeMutex);

snake = newSnake;

}

}

void RestartGame() {

lock\_guard<mutex> lock(snakeMutex);

snake.clear();

snake.push\_back(SnakeSegment(5, 5));

foodX = rand() % (screenWidth / gridSize);

foodY = rand() % (screenHeight / gridSize);

direction = 1;

gameOver = false;

foodEaten = 0;

InvalidateRect(hWnd, NULL, TRUE); // Добавляем это для перерисовки экрана

}

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

switch (message) {

case WM\_KEYDOWN:

HandleInput(wParam);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

PaintGame(hdc);

EndPaint(hWnd, &ps);

}

break;

case WM\_CLOSE:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

LRESULT CALLBACK RestartButtonProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_COMMAND:

if (LOWORD(wParam) == IDC\_RESTART\_BUTTON) {

// Нажата кнопка "Restart"

RestartGame();

}

break;

default:

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

}

return 0;

}

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

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Регистрируем класс окна

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, IDI\_APPLICATION);

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

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

wcex.lpszMenuName = NULL;

wcex.lpszClassName = L"SnakeGame";

wcex.hIconSm = LoadIcon(wcex.hInstance, IDI\_APPLICATION);

if (!RegisterClassEx(&wcex)) {

return 0;

}

// Создаем главное окно приложения

hWnd = CreateWindow(L"SnakeGame", L"Snake Game", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0, 400, 400, NULL, NULL, hInstance, NULL);

if (!hWnd) {

return 0;

}

restartButton = CreateWindow(L"BUTTON", L"Restart", WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | BS\_PUSHBUTTON, 10, 50, 100, 30, hWnd, (HMENU)IDC\_RESTART\_BUTTON, hInstance, NULL);

if (!restartButton) {

MessageBox(hWnd, L"Failed to create the restart button!", L"Error", MB\_ICONERROR | MB\_OK);

return 0;

}

WNDPROC oldButtonProc = (WNDPROC)SetWindowLongPtr(restartButton, GWLP\_WNDPROC, (LONG\_PTR)RestartButtonProc);

if (!oldButtonProc) {

return 0;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

// Инициализация змейки

snake.push\_back(SnakeSegment(5, 5));

foodX = rand() % (screenWidth / gridSize);

foodY = rand() % (screenHeight / gridSize);

// Устанавливаем глобальный хук для клавиши "R"

g\_hook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, NULL, 0);

if (g\_hook == NULL) {

MessageBox(hWnd, L"Failed to install the keyboard hook!", L"Error", MB\_ICONERROR | MB\_OK);

}

// Главный цикл программы

MSG msg;

while (true) {

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE)) {

if (msg.message == WM\_QUIT) {

break;

}

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else {

if (!gameOver) {

UpdateGame(hWnd); // Вызываем функцию обновления игры

InvalidateRect(hWnd, NULL, TRUE);

}

Sleep(100); // Задержка для контроля скорости змейки

}

}

// Удаляем глобальный хук

if (g\_hook != NULL) {

UnhookWindowsHookEx(g\_hook);

}

return (int)msg.wParam;

}

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

if (nCode == HC\_ACTION && (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN)) {

KBDLLHOOKSTRUCT\* pKB = (KBDLLHOOKSTRUCT\*)lParam;

if (pKB->vkCode == 'R') {

RestartGame();

}

}

return CallNextHookEx(g\_hook, nCode, wParam, lParam);

}

Листинг 2 – Файл global\_defines.h

#define IDC\_RESTART\_BUTTON 101