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 №4

на тему

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

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 процессами и потоками на *Windows*, научиться создавать, завершать, изменять приоритеты процессов и потоков, а также изучить организацию и контроль исследование эффективности. Для достижения цели будет создано графическое приложение в виде стандартной игры змейка.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе *Windows* является важной частью организации работы приложений и ресурсов компьютера.

Процесс в операционной системе *Windows* представляет собой независимую программу, выполняющуюся в собственном адресном пространстве памяти. Процесс обладает собственными системными ресурсами, такими как память и дескрипторы файлов. Поток легковесный исполняемый агент внутри процесса. Потоки внутри одного процесса могут совместно использовать ресурсы этого процесса, включая память и файлы. Один процесс может содержать несколько потоков, которые выполняются параллельно.

Для порождения новых процессов в *Windows* можно использовать функции, такие как *CreateProcess*. Это позволяет запускать новые исполняемые файлы в отдельных процессах. Для порождения новых потоков внутри процесса можно использовать функции, такие как *CreateThread*. Это позволяет создавать параллельные задачи внутри процесса. Завершение процессов и потоков может осуществляться с помощью функций *ExitProcess* и *ExitThread*. Эти функции завершают выполнение соответствующих процессов и потоков.

В *Windows* можно изменять приоритеты процессов и потоков для управления их доступом к ресурсам *CPU*. Высший приоритет обеспечивает больше времени *CPU* для выполнения задачи. Приоритеты процессов можно устанавливать с помощью функции *SetPriorityClass*, а приоритеты потоков - с помощью функции *SetThreadPriority*.

Для оценки эффективности работы процессов и потоков в *Windows* можно использовать различные инструменты мониторинга и профилирования, такие как *Windows* *Performance* *Monitor* и *Visual* *Studio* *Profiler*. Эти инструменты позволяют анализировать использование *CPU*, памяти и других ресурсов, а также выявлять узкие места в процессах и потоках для оптимизации производительности.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано графическое приложение в виде стандартной игры – змейка. В данном случае было добавлено отображение частоты обновления игры для исследования эффективности нашей игры (Рисунок 1).



Рисунок 1 – Окно приложения с отображением частоты обновления игры

Для управления змейкой используется обработка нажатия клавиш *VK\_LEFT*, *VK\_RIGHT*, *VK\_UP*, *VK\_DOWN.* В случае, если змейка врезается в край игровой области – игра заканчивается (Рисунок 2).

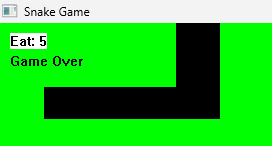


Рисунок 2– Выход за игровое поле

Для отслеживания статистики съеденных яблок есть счётчик яблок в левом верхем игру, а также окно уведомлений, предупреждающее пользователя при съедании новых 5 яблок (Рисунок 3).

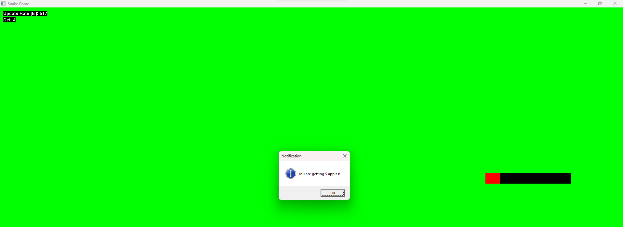


Рисунок 3– Уведомление пользователя о собранном числе яблок

При нажатии кнопки *R*, *S*, *L* воспроизводится рестарт игры, либо сохранение или выгрузка в файл соответственно. Загрузка в файл происходит в бинарном формате. Записываются след параметры: размер змейки, координатор первого, второго, третьего и четвертого сегмента соответственно.

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены и применены важные аспекты управления процессами и потоками в операционной системе *Windows* с использованием *Win32* *API*. Управление памятью и вводом-выводом: в рамках приложения были реализованы функции для сохранения и загрузки игрового состояния в файл. Это позволяет пользователям сохранять текущий прогресс игры и восстанавливать его в будущем. Операции считывания и записи данных в файлы предоставили пример работы с вводом-выводом в *Windows*. 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, такими как информация 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, не блокируя пользовательский интерфейс приложения. Это обеспечивает непрерывное взаимодействие игрока с игрой даже во время выполнения этих операций. Отображение файлов в память: для хранения и управления данными о состоянии игры использовалась работа с файлами, что позволяет сохранять информацию на диске и взаимодействовать с ней через *API* *Windows*. Данное приложение представляет собой игровой проект, который наглядно демонстрирует применение указанных аспектов программирования в контексте разработки многозадачных и многопоточных приложений под операционной системой *Windows*. Это позволяет эффективно управлять процессами и потоками, обеспечивая отзывчивость и производительность приложения.

# 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

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab3.cpp

#include <windows.h>

#include <deque>

#include <fstream>

#include <thread>

#include <mutex>

#include <string>

#include <iostream>

#include <chrono>

#include <fstream>

#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.bin";

const wchar\_t\* sharedMemoryName = L"MySharedMemory";

std::thread gameUpdateThread;

HWND hWnd;

HWND restartButton;

HHOOK g\_hook = NULL;

struct SnakeSegment {

int x, y;

SnakeSegment() : x(0), y(0) {}

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

};

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

deque<SnakeSegment> snake;

int foodX, foodY;

bool gameOver = false;

int direction = 1;

int foodEaten = 0;

mutex snakeMutex;

HANDLE hMapFile;

LPCTSTR pBuf;

void CreateMemoryMapping() {

hMapFile = CreateFileMapping(

INVALID\_HANDLE\_VALUE,

NULL,

PAGE\_READWRITE,

0,

sizeof(SnakeSegment),

sharedMemoryName);

if (hMapFile == NULL) {

cout << "Failed to create file display object" << endl;

return;

}

pBuf = (LPCTSTR)MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(SnakeSegment));

if (pBuf == NULL) {

cout << "Failed to display file view" << endl;

CloseHandle(hMapFile);

return;

}

}

void CloseMemoryMapping() {

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

}

void SaveGame() {

lock\_guard<mutex> lock(snakeMutex);

ofstream file(saveFileName, ios::binary);

if (file.is\_open()) {

int size = static\_cast<int>(snake.size());

file.write(reinterpret\_cast<char\*>(&size), sizeof(int));

for (const SnakeSegment& segment : snake) {

file.write(reinterpret\_cast<const char\*>(&segment), sizeof(SnakeSegment));

}

file.close();

}

SnakeSegment segment;

if (!snake.empty()) {

segment = snake.front();

}

memcpy((LPVOID)pBuf, &segment, sizeof(SnakeSegment));

}

void LoadGame() {

ifstream file(saveFileName, ios::binary);

if (file.is\_open()) {

int size;

file.read(reinterpret\_cast<char\*>(&size), sizeof(int));

deque<SnakeSegment> newSnake;

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

SnakeSegment segment;

file.read(reinterpret\_cast<char\*>(&segment), sizeof(SnakeSegment));

newSnake.push\_back(segment);

}

file.close();

SnakeSegment segment;

memcpy(&segment, pBuf, sizeof(SnakeSegment));

{

lock\_guard<mutex> lock(snakeMutex);

snake = newSnake;

snake.push\_front(segment);

}

}

}

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

int cellWidth = screenWidth / gridSize;

int cellHeight = screenHeight / gridSize;

HBRUSH brush = CreateSolidBrush(color);

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

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

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

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

SelectObject(hdc, oldBrush);

SelectObject(hdc, oldPen);

DeleteObject(brush);

DeleteObject(pen);

}

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':

SaveGame();

break;

case 'L':

LoadGame();

break;

}

}

void UpdateGame(HWND hWnd) {

auto start = std::chrono::high\_resolution\_clock::now();

if (snake.empty()) {

return;

}

int headX = snake.front().x;

int headY = snake.front().y;

switch (direction) {

case 0:

headX--;

break;

case 1:

headY--;

break;

case 2:

headX++;

break;

case 3:

headY++;

break;

}

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

gameOver = true;

return;

}

lock\_guard<mutex> lock(snakeMutex);

for (const SnakeSegment& segment : snake) {

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

gameOver = true;

return;

}

}

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

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

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

foodEaten++;

if (foodEaten % 5 == 0) {

MessageBox(hWnd, L"You are getting 5 apples!", L"Notification", MB\_OK | MB\_ICONINFORMATION);

}

}

else {

snake.pop\_back();

}

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

InvalidateRect(hWnd, NULL, TRUE);

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

// Отображение частоты обновления в верхней части окна

wstring updateRateText = L"Update Rate (μs): " + to\_wstring(duration.count());

TextOut(GetDC(hWnd), 10, 10, updateRateText.c\_str(), static\_cast<int>(updateRateText.length()));

}

void PaintGame(HDC hdc) {

RECT clientRect;

GetClientRect(hWnd, &clientRect);

HBRUSH greenBrush = CreateSolidBrush(RGB(0, 255, 0));

FillRect(hdc, &clientRect, greenBrush);

DeleteObject(greenBrush);

auto start = std::chrono::high\_resolution\_clock::now();

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

COLORREF backgroundColor = RGB(0, 0, 0);

COLORREF textColor = RGB(255, 255, 255);

SetBkColor(hdc, backgroundColor);

SetTextColor(hdc, textColor);

TextOut(hdc, 10, 30, foodEatenText.c\_str(), static\_cast<int>(foodEatenText.length()));

DrawCell(hdc, foodX, foodY, RGB(255, 0, 0)); // for apple

lock\_guard<mutex> lock(snakeMutex);

for (const SnakeSegment& segment : snake) {

DrawCell(hdc, segment.x, segment.y, RGB(0, 0, 0));

}

if (gameOver) {

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

}

auto end = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

wstring updateRateText = L"Update Rate (μs): " + to\_wstring(duration.count());

TextOut(hdc, 10, 10, updateRateText.c\_str(), static\_cast<int>(updateRateText.length()));

}

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);

}

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) {

RestartGame();

}

break;

default:

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

}

return 0;

}

void CreateGameUpdateThread();

int WINAPI WinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPSTR lpCmdLine, \_In\_ int nCmdShow) {

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

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;

}

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 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);

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

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

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

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

if (g\_hook == NULL) {

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

}

CreateMemoryMapping();

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);

InvalidateRect(hWnd, NULL, TRUE);

}

Sleep(100);

}

}

CloseMemoryMapping();

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);

}

void CreateGameUpdateThread() {

gameUpdateThread = std::thread([] {

while (true) {

if (!gameOver) {

UpdateGame(hWnd);

InvalidateRect(hWnd, NULL, TRUE);

}

Sleep(100);

}

});

}

Листинг 2 – Файл global\_defines.h

#define IDC\_RESTART\_BUTTON 101