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 №6

на тему

**СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМНОГО**

**ИСКЛЮЧЕНИЯ (WINDOWS). ИЗУЧЕНИЕ И**

**ИСПОЛЬЗОВАНИЕ СРЕДСТВ СИНХРОНИЗАЦИИ**

**И ВЗАИМНОГО ИСКЛЮЧЕНИЯ**

Студент Д. С. Кончик

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 данной лабораторной работы заключается в следующем:

1. Изучение теоретических основ синхронизации и взаимного исключения в многопоточном программировании.
2. Ознакомление с основными проблемами, возникающими при отсутствии механизмов синхронизации, а также возможными последствиями их возникновения.
3. Изучение различных типов примитивов синхронизации, доступных в операционной системе *Windows*.
4. Получение практических навыков использования примитивов синхронизации в процессе разработки приложения, использующего многопоточность.
5. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства синхронизации и взаимного исключения в операционных системах семейства *Windows*, включая *Win32 API*, являются важными механизмами для обеспечения безопасной и согласованной работы множества потоков и процессов. Они позволяют предотвратить возможные конфликты при доступе к общим ресурсам и гарантировать правильную последовательность выполнения операций [1].

Критические секции – это одно из наиболее простых и широко используемых средств синхронизации в *Win32 API*. Они позволяют ограничить доступ к участку кода только одному потоку в конкретный момент времени. Критические секции создаются с помощью функции *InitializeCriticalSection* и управляются функциями *EnterCriticalSection* и *LeaveCriticalSection*. Они обеспечивают монопольный доступ к критической секции, что позволяет избежать состязания за ресурсы внутри процесса.

Мьютексы (*Mutex*) представляют собой более общий механизм синхронизации, который может использоваться для согласования потоков и даже процессов между разными процессами. Они создаются с помощью функции *CreateMutex* и могут быть использованы для координирования доступа к разделяемым ресурсам. Мьютексы могут быть использованы как внутри процесса, так и между разными процессами.

Семафоры (*Semaphore)* представляют собой средство синхронизации, которое позволяет ограничить количество потоков, имеющих доступ к общим ресурсам. Семафоры создаются с помощью функции *CreateSemaphore* и управляются с использованием функций *WaitForSingleObject* и *ReleaseSemaphore*. Семафоры могут использоваться для регулирования доступа к пулам ресурсов или задачам в ограниченном количестве [2].

События (*Event*)представляют собой средство синхронизации, которое позволяет одному или нескольким потокам ожидать события и сигнализировать о его наступлении. События создаются с помощью функции *CreateEvent* и могут использоваться для ожидания определенных условий перед продолжением выполнения кода.

Эти средства синхронизации и взаимного исключения в *Win32 API* позволяют разработчикам создавать безопасные и надежные многозадачные приложения, где множество потоков или процессов могут взаимодействовать с общими ресурсами, не вызывая конфликтов или гонок данных. Выбор конкретного средства зависит от требований приложения и его архитектуры.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Основой является приложение стрелочных часов с возможностью менять часовой пояс по нажатию *VK\_LEFT* и *VK\_RIGHT* (рисунок 1)*.*

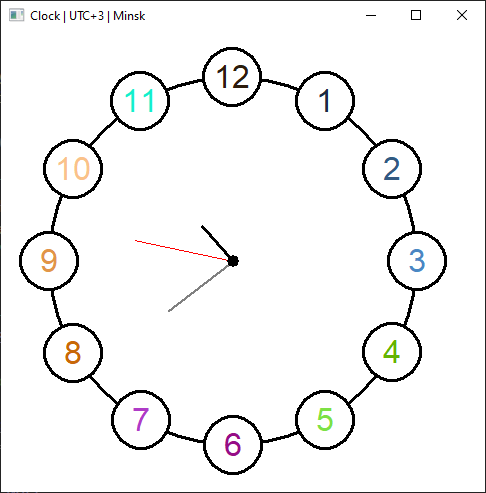


Рисунок 1 – Окно приложения

Была реализована запись логов (рисунок 2) при изменении часового пояса с помощью средств *Win32 API*. При каждой записи в файл *log.txt* создается новый поток. В качестве средства синхронизации для обеспечения доступа к записи только одному потоку в конкретный момент времени используется *Mutex*.

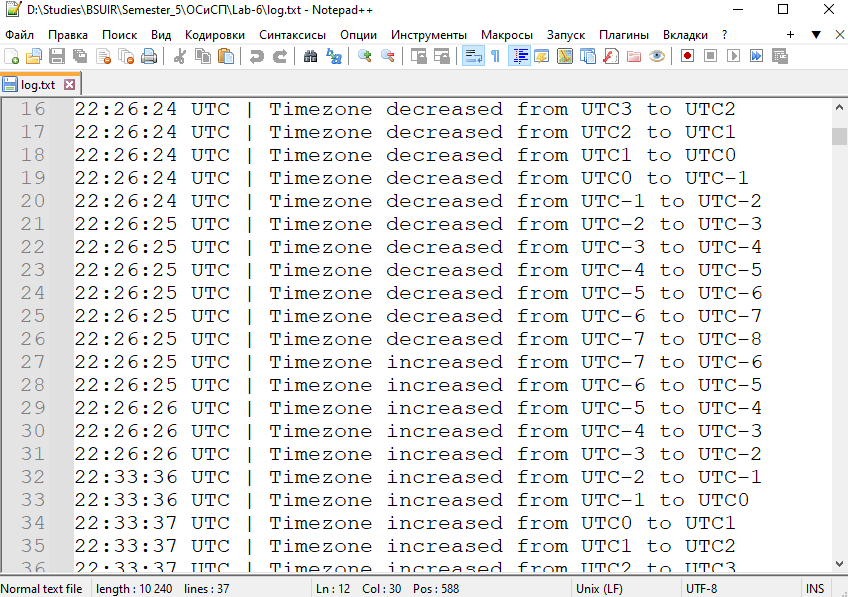


Рисунок 2– Файл *log.txt*

# ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные средства синхронизации и взаимного исключения: критические секции, мьютексы, семафоры, события. С помощью использования многопоточного программирования и инструментов синхронизации доступа разных потоков было создано логирование действий приложения (изменение часового пояса) с выводом подробной информации.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Синхронизация [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization.
2. Синхронизация процессов и потоков [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/7602858/.
3. Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/>.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Source.cpp*

#define \_USE\_MATH\_DEFINES

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <string>

#include <string>

#include <ctime>

#include <map>

#include <thread>

#include "Constants.h"

#include "FileMapping.h"

using namespace std;

int utcOffset = 3;

HHOOK keyboardHook;

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

LRESULT CALLBACK MainWindowProcedure(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

void PlaySoundAsync(wstring soundFile);

RECT reduceRect(RECT rect, const double alpha);

void DrawClock(HDC hdc, RECT rect, int hour, int minute, int second);

void DisplayTimeZone(HWND hWnd);

void UpdateClock(HWND hWnd);

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR cmdline, int ss) {

hLogMutex = CreateMutex(NULL, FALSE, NULL);

InitMapping();

memcpy(&utcOffset, pTimeZoneData, 4);

keyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, hInst, 0);

// Создание класса главного окна

WNDCLASS wc;

wc.style = NULL;

wc.lpfnWndProc = MainWindowProcedure;

wc.cbClsExtra = NULL;

wc.cbWndExtra = NULL;

wc.hInstance = hInst;

wc.hIcon = NULL;

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

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

wc.lpszMenuName = NULL;

wc.lpszClassName = L"MainWindowClass";

if (!RegisterClass(&wc))

return EXIT\_FAILURE;

// Создание главного окна

HWND hWnd = CreateWindow(

L"MainWindowClass", L"Clock",

WS\_OVERLAPPEDWINDOW,

0, 0, 500, 500,

NULL, NULL, hInst, NULL

);

if (!hWnd)

return EXIT\_FAILURE;

ShowWindow(hWnd, ss);

SetTimer(hWnd, NULL, 1000, NULL);

// Цикл обработки сообщений

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(keyboardHook);

CloseMapping();

CloseHandle(hLogMutex);

return msg.wParam;

}

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

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN) {

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

if (pKeyInfo->vkCode == VK\_SPACE) {

thread soundThread(PlaySoundAsync, L"audio/kukushka.wav");

soundThread.detach();

string\* log = new string("Кукушка");

HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)WriteLog, log, 0, NULL);

}

}

}

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

}

// Функция обработки сообщений

LRESULT CALLBACK MainWindowProcedure(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

switch (msg)

{

case WM\_SIZE:

{

UpdateClock(hWnd);

break;

}

case WM\_GETMINMAXINFO: {

MINMAXINFO\* minsize = (MINMAXINFO\*)lParam;

minsize->ptMinTrackSize.x = 400;

minsize->ptMinTrackSize.y = 400;

break;

}

case WM\_TIMER: {

UpdateClock(hWnd);

break;

}

case WM\_KEYDOWN:

switch (wParam) {

case VK\_LEFT:

if (utcOffset <= -11) {

MessageBox(NULL, L"Minimum time zone: UTC-11", L"Warning", MB\_ICONWARNING | MB\_OK);

}

else {

utcOffset -= 1;

UpdateClock(hWnd);

memcpy(pTimeZoneData, &utcOffset, 4);

string\* log = new string("Timezone decreased from UTC" + to\_string(utcOffset) + " to UTC" + to\_string(utcOffset - 1));

HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)WriteLog, log, 0, NULL);

}

break;

case VK\_RIGHT:

if (utcOffset >= 12) {

MessageBox(NULL, L"Maximum time zone: UTC+12", L"Warning", MB\_ICONWARNING | MB\_OK);

}

else {

utcOffset += 1;

UpdateClock(hWnd);

memcpy(pTimeZoneData, &utcOffset, 4);

string\* log = new string("Timezone increased from UTC" + to\_string(utcOffset - 1) + " to UTC" + to\_string(utcOffset));

HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)WriteLog, log, 0, NULL);

}

break;

}

break;

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

}

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

}

void PlaySoundAsync(wstring soundFile) {

PlaySound(soundFile.c\_str(), NULL, SND\_FILENAME);

}

// Уменьшить прямоугольик отрисовки

RECT reduceRect(RECT rect, const double alpha) {

RECT result;

// Исходные размеры

int width = rect.right - rect.left;

int height = rect.bottom - rect.top;

// Новые размеры

int newWidth = (int)(width \* alpha);

int newHeight = (int)(height \* alpha);

// Смещение по X и Y для центрирования уменьшенного прямоугольника

int xOffset = (width - newWidth) / 2;

int yOffset = (height - newHeight) / 2;

// Новые координаты всех границ прямоугольника

result.left = rect.left + xOffset;

result.top = rect.top + yOffset;

result.right = result.left + newWidth;

result.bottom = result.top + newHeight;

return result;

}

// Отрисовка часов

void DrawClock(HDC hdc, RECT rect, int hour, int minute, int second) {

// Вычисление в соответствии с размером окна

int width = rect.right - rect.left;

int height = rect.bottom - rect.top;

int diameter = min(width, height);

int radius = diameter / 2;

int centerX = rect.left + diameter / 2;

int centerY = rect.top + diameter / 2;

// Отрисовка большого круга

int penWidth = diameter / 100;

HPEN hPen = CreatePen(PS\_SOLID, penWidth, RGB(0, 0, 0));

HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);

Ellipse(hdc, rect.left, rect.top, rect.left + diameter, rect.top + diameter);

// Установка размера шрифта в зависимости от диаметра

int fontSize = diameter / 10;

HFONT hFont = CreateFont(fontSize, 0, 0, 0, FW\_NORMAL, FALSE, FALSE, FALSE, DEFAULT\_CHARSET,

OUT\_OUTLINE\_PRECIS, CLIP\_DEFAULT\_PRECIS, CLEARTYPE\_QUALITY, DEFAULT\_PITCH, L"Arial");

HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);

// Рисование цифр

for (int i = 1; i <= 12; i++) {

// Расчет координаты (x, y) для цифры

double angle = i \* (360.0 / 12.0);

double radian = angle \* M\_PI / 180.0;

int textX = centerX + (radius \* sin(radian));

int textY = centerY - (radius \* cos(radian));

// Создание кисти для заливки окружности

HBRUSH hBrush = CreateSolidBrush(RGB(255, 255, 255));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

// Отрисовка окружности для цифры

int radius = diameter \* 0.08;

Ellipse(hdc, textX - radius, textY - radius, textX + radius, textY + radius);

// Восстановка предыдущей кисти

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

// Отрисовка цифры

SetBkMode(hdc, TRANSPARENT);

wstring digit = to\_wstring(i);

SIZE textSize;

GetTextExtentPoint32W(hdc, digit.c\_str(), digit.length(), &textSize);

SetTextColor(hdc, RGB((i \* 25) % 256, (i \* 45) % 256, (i \* 65) % 256));

TextOutW(hdc, textX - textSize.cx / 2, textY - textSize.cy / 2, digit.c\_str(), digit.length());

SetBkMode(hdc, OPAQUE);

}

// Восстановление предыдущего пера

SelectObject(hdc, hOldPen);

DeleteObject(hPen);

// Восстановление предыдущего шрифта

SelectObject(hdc, hOldFont);

DeleteObject(hFont);

// Отрисовка часовой стрелки

int hourAngle = (hour % 12) \* 30 + (minute / 2);

int hourLength = radius \* 0.25;

int hx = centerX + hourLength \* sin(hourAngle \* M\_PI / 180.0);

int hy = centerY - hourLength \* cos(hourAngle \* M\_PI / 180.0);

int hourPenWidth = diameter / 100;

HPEN hHourPen = CreatePen(PS\_SOLID, hourPenWidth, RGB(0, 0, 0));

hOldPen = (HPEN)SelectObject(hdc, hHourPen);

MoveToEx(hdc, centerX, centerY, NULL);

LineTo(hdc, hx, hy);

SelectObject(hdc, hOldPen);

DeleteObject(hHourPen);

// Отрисовка минутной стрелки

int minuteAngle = minute \* 6 + (second / 10);

int minuteLength = radius \* 0.45;

int mx = centerX + minuteLength \* sin(minuteAngle \* M\_PI / 180.0);

int my = centerY - minuteLength \* cos(minuteAngle \* M\_PI / 180.0);

int minutePenWidth = diameter / 150;

HPEN hMinutePen = CreatePen(PS\_SOLID, minutePenWidth, RGB(128, 128, 128));

hOldPen = (HPEN)SelectObject(hdc, hMinutePen);

MoveToEx(hdc, centerX, centerY, NULL);

LineTo(hdc, mx, my);

SelectObject(hdc, hOldPen);

DeleteObject(hMinutePen);

// Отрисовка секунднрй стрелки

int secondAngle = second \* 6;

int secondLength = radius \* 0.55;

int sx = centerX + secondLength \* sin(secondAngle \* M\_PI / 180.0);

int sy = centerY - secondLength \* cos(secondAngle \* M\_PI / 180.0);

int secondPenWidth = diameter / 200;

HPEN hSecondPen = CreatePen(PS\_SOLID, secondPenWidth, RGB(255, 0, 0));

hOldPen = (HPEN)SelectObject(hdc, hSecondPen);

MoveToEx(hdc, centerX, centerY, NULL);

LineTo(hdc, sx, sy);

SelectObject(hdc, hOldPen);

DeleteObject(hSecondPen);

// Отрисовка круга в центре часов

HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 0));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

int r = radius / 30;

Ellipse(hdc, centerX - r, centerY - r, centerX + r, centerY + r);

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

// Отображение тайм зоны (в title окна)

void DisplayTimeZone(HWND hWnd) {

wstring new\_title = BASE\_TITLE;

if (utcOffset < 0)

new\_title += to\_wstring(utcOffset);

else if (utcOffset > 0)

new\_title += L"+" + to\_wstring(utcOffset);

new\_title += L" | " + TIME\_ZONE\_MAP[utcOffset];

SetWindowText(hWnd, new\_title.c\_str());

}

// Обновить часы

void UpdateClock(HWND hWnd) {

HDC hdc = GetDC(hWnd);

RECT clientRect;

GetClientRect(hWnd, &clientRect);

// Закрашиваем всю область окна фоновым цветом

HBRUSH bgBrush = CreateSolidBrush(GetSysColor(COLOR\_WINDOW));

FillRect(hdc, &clientRect, bgBrush);

DeleteObject(bgBrush);

// Текущее UTC время

time\_t now = time(0);

tm\* gmtm = gmtime(&now);

// Отрисовка часов

DrawClock(hdc, reduceRect(clientRect, 0.8), gmtm->tm\_hour + utcOffset, gmtm->tm\_min, gmtm->tm\_sec);

// Вывод пояса в заголовок окна

DisplayTimeZone(hWnd);

ReleaseDC(hWnd, hdc);

}

Листинг 2 – Файл *Constants.h*

#pragma once

#include <Windows.h>

#include <string>

#include <map>

using namespace std;

wstring BASE\_TITLE = L"Clock | UTC";

map<int, wstring> TIME\_ZONE\_MAP = {

{-11, L"Midway Island"},

{-10, L"Honolulu"},

{-9, L"Anchorage"},

{-8, L"Los Angeles"},

{-7, L"Denver"},

{-6, L"Chicago"},

{-5, L"New York"},

{-4, L"Caracas"},

{-3, L"Buenos Aires"},

{-2, L"South Georgia"},

{-1, L"Azores"},

{0, L"London"},

{1, L"Paris"},

{2, L"Athens"},

{3, L"Minsk"},

{4, L"Kabul"},

{5, L"Islamabad"},

{6, L"Almaty"},

{7, L"Bangkok"},

{8, L"Beijing"},

{9, L"Tokyo"},

{10, L"Sydney"},

{11, L"Solomon Islands"},

{12, L"Suva"}

};

LPCSTR timeZoneFileName = "timezone.txt";

CONST INT timeZoneFileSize = 4;

LPCSTR logFileName = "log.txt";

CONST INT logFileSize = 1024 \* 10;

Листинг 3 – Файл *FileMapping.h*

#pragma once

#include <string>

#include <Windows.h>

#include <mutex>

#include "Constants.h"

HANDLE hLogMutex;

HANDLE hTimeZoneFile;

HANDLE hTimeZoneMapFile;

LPVOID pTimeZoneData;

HANDLE hLogFile;

HANDLE hLogMapFile;

LPVOID pLogData;

void InitMapping() {

if (hTimeZoneFile = CreateFileA(timeZoneFileName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL),

hTimeZoneFile == INVALID\_HANDLE\_VALUE) {

MessageBox(NULL, L"hTimeZoneFile error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

if (hTimeZoneMapFile = CreateFileMappingA(hTimeZoneFile, NULL, PAGE\_READWRITE, 0, timeZoneFileSize, NULL),

hTimeZoneMapFile == NULL) {

MessageBox(NULL, L"hTimeZoneMapFile error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

if (pTimeZoneData = MapViewOfFile(hTimeZoneMapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, timeZoneFileSize),

pTimeZoneData == NULL) {

MessageBox(NULL, L"pTimeZoneData error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

if (hLogFile = CreateFileA(logFileName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL),

hLogFile == INVALID\_HANDLE\_VALUE) {

MessageBox(NULL, L"hLogFile error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

if (hLogMapFile = CreateFileMappingA(hLogFile, NULL, PAGE\_READWRITE, 0, logFileSize, NULL),

hLogMapFile == NULL) {

MessageBox(NULL, L"hLogMapFile error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

if (pLogData = MapViewOfFile(hLogMapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, logFileSize),

pLogData == NULL) {

MessageBox(NULL, L"pLogData error", L"Warning", MB\_ICONWARNING | MB\_OK);

return;

}

}

string NumberToStringWithLeadingZero(int number) {

if (number < 10)

return "0" + to\_string(number);

else

return to\_string(number);

}

void WriteLog(LPVOID lpParam) {

WaitForSingleObject(hLogMutex, INFINITE);

string\* str = static\_cast<string\*>(lpParam);

if (pLogData != NULL) {

// Текущее UTC время

time\_t now = time(0);

tm\* gmtm = gmtime(&now);

\*str = NumberToStringWithLeadingZero(gmtm->tm\_hour) + ":" +

NumberToStringWithLeadingZero(gmtm->tm\_min) + ":" +

NumberToStringWithLeadingZero(gmtm->tm\_sec) + " UTC | " + \*str + "\n";

// Теущая длина в памяти

int currentLength = strlen((char\*)pLogData);

if (currentLength + str->length() > logFileSize) {

memset(pLogData, 0, logFileSize);

MessageBox(NULL, L"Недостаточно места в файле лога. Файл очищен.", L"Warning", MB\_ICONWARNING | MB\_OK);

currentLength = 0;

}

CopyMemory((char\*)pLogData + currentLength, str->c\_str(), str->length());

}

else {

MessageBox(NULL, L"Память не инициализирована", L"Ошибка", MB\_ICONERROR | MB\_OK);

}

delete str;

ReleaseMutex(hLogMutex);

}

void CloseMapping() {

UnmapViewOfFile(pTimeZoneData);

CloseHandle(hTimeZoneMapFile);

CloseHandle(hTimeZoneFile);

UnmapViewOfFile(pLogData);

CloseHandle(hLogMapFile);

CloseHandle(hLogFile);

}