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 З. Э. Карась

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, которое показывает состояние батареи (рисунок 1)*.*

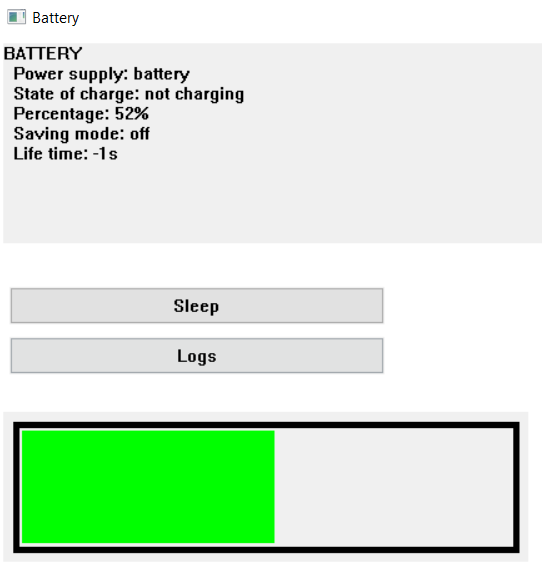


Рисунок 1 – Окно приложения

Была реализована запись логов (рисунок 2) при изменении состоянии батареи. 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*.

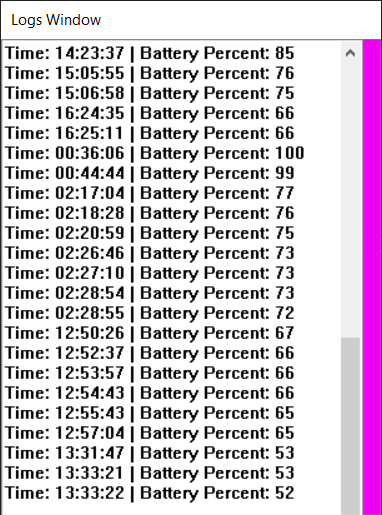


Рисунок 2– Файл *log.txt*

# 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 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*

#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;

HWND logsWindow;

int startPercent = 0;

using namespace Gdiplus;

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

void\* MapFileToMemory(const char\* fileName, size\_t& fileSize)

{

// Открываем файл для чтения

HANDLE fileHandle = CreateFileA(fileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (fileHandle == INVALID\_HANDLE\_VALUE)

{

// Обработка ошибки открытия файла

return nullptr;

}

// Получаем размер файла

fileSize = GetFileSize(fileHandle, NULL);

// Создаем отображение файла в память

HANDLE fileMappingHandle = CreateFileMapping(fileHandle, NULL, PAGE\_READONLY, 0, 0, NULL);

if (fileMappingHandle == NULL)

{

// Обработка ошибки создания отображения файла

CloseHandle(fileHandle);

return nullptr;

}

// Отображаем файл в память

void\* mappedFile = MapViewOfFile(fileMappingHandle, FILE\_MAP\_READ, 0, 0, 0);

if (mappedFile == NULL)

{

// Обработка ошибки отображения файла в память

CloseHandle(fileMappingHandle);

CloseHandle(fileHandle);

return nullptr;

}

// Закрываем дескрипторы файла и отображения

CloseHandle(fileMappingHandle);

CloseHandle(fileHandle);

return mappedFile;

}

void SaveBatteryStateToFile(int percent)

{

// Отображаем файл в память

size\_t fileSize = 0;

void\* mappedFile = MapFileToMemory("battery\_state.txt", fileSize);

if (mappedFile == nullptr)

{

// Обработка ошибки отображения файла в память

return;

}

// Открываем файл для записи

FILE\* file = nullptr;

if (fopen\_s(&file, "battery\_state.txt", "a") != 0)

{

// Обработка ошибки открытия файла для записи

UnmapViewOfFile(mappedFile);

return;

}

// Записываем данные в файл

// Получаем текущее время

std::time\_t currentTime = std::time(nullptr);

std::tm localTime;

localtime\_s(&localTime, &currentTime);

// Записываем время и проценты батареи в файл

fprintf(file, "Time: %02d:%02d:%02d | Battery Percent: %d\n",

localTime.tm\_hour, localTime.tm\_min, localTime.tm\_sec, percent);

// Закрываем файл и освобождаем отображение файла

fclose(file);

UnmapViewOfFile(mappedFile);

}

void SaveBatteryStateToEventLog(int percent)

{

HANDLE hEventLog = RegisterEventSource(NULL, L"YourApplicationName");

if (hEventLog != NULL)

{

std::wstring message = L"Battery Percent: " + std::to\_wstring(percent);

const wchar\_t\* strings[1] = { message.c\_str() };

ReportEvent(hEventLog, EVENTLOG\_INFORMATION\_TYPE, 0, 0, NULL, 1, 0, strings, NULL);

DeregisterEventSource(hEventLog);

}

}

void WriteRegistryStringValue(int percent)

{

HKEY hKey;

// Создать/открыть ключ

LONG result = RegCreateKeyEx(HKEY\_CURRENT\_USER, L"Software\\BatteryStateLab", 0, nullptr, 0, KEY\_WRITE, nullptr, &hKey, nullptr);

if (result != ERROR\_SUCCESS) {

MessageBox(nullptr, L"Ошибка при создании/открытии ключа в Реестре", L"Ошибка", MB\_ICONERROR | MB\_OK);

return;

}

// Записать значение

result = RegSetValueEx(hKey, L"Percent", 0, REG\_DWORD, reinterpret\_cast<const BYTE\*>(&percent), sizeof(DWORD));

if (result != ERROR\_SUCCESS) {

MessageBox(nullptr, L"Ошибка при записи значения в Реестр", L"Ошибка", MB\_ICONERROR | MB\_OK);

}

RegCloseKey(hKey);

}

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{};

HWND hwnd{};

WNDCLASSEX wc{ sizeof(WNDCLASSEX) };

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);

UpdateWindow(hwnd);

while (GetMessage(&msg, nullptr, 0, 0))

{

TranslateMessage(&msg);

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

}

TerminateThread(readThread, 0);

GdiplusShutdown(gdiplusToken);

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

}

void CreateLogsWindow()

{

SetWindowTextA(logsWindow, " ");

std::ifstream file("battery\_state.txt");

if (file.is\_open())

{

std::string content((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

SetWindowTextA(logsWindow, content.c\_str());

file.close();

}

}

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

{

switch (uMsg)

{

case WM\_CREATE:

{

logsWindow = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", nullptr, WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL, 0, 0, 290, 400, hWnd, nullptr, nullptr, nullptr);

CreateLogsWindow();

break;

}

case WM\_DESTROY:

{

break;

}

// Обработка других сообщений

}

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

}

void CreateLogsWindow(HWND hWndParent)

{

WNDCLASSEX wc{ sizeof(WNDCLASSEX) };

wc.hInstance = GetModuleHandle(nullptr);

wc.lpfnWndProc = LogsWndProc;

wc.lpszClassName = L"Logs";

wc.hbrBackground = CreateSolidBrush(RGB(240, 0, 240));

RegisterClassEx(&wc);

HWND childwnd = CreateWindow(wc.lpszClassName, L"Logs Window", WS\_OVERLAPPED | WS\_CLIPCHILDREN | ES\_MULTILINE, 500, 500, 320, 420, hWndParent, nullptr, nullptr, nullptr);

if (childwnd == nullptr)

{

MessageBox(hWndParent, L"Failed to create logss window", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

ShowWindow(childwnd, SW\_SHOW);

UpdateWindow(childwnd);

}

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);

}

case 1338:

{

//MessageBox(hWnd, L"Hello", L"Message", MB\_OK);

CreateLogsWindow(hWnd);

}

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);

CreateWindow(L"BUTTON", L"Logs", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 10, 240, 300, 30, hWnd, HMENU(1338), nullptr, nullptr);

}

DWORD WINAPI Power(LPVOID lpParameter)

{

while (isThreading) {

Battery battery;

if (battery.getPercent() != startPercent)

{

SaveBatteryStateToFile(battery.getPercent());

SaveBatteryStateToEventLog(battery.getPercent());

WriteRegistryStringValue(battery.getPercent());

CreateLogsWindow();

InvalidateRect(childWindow, nullptr, TRUE);

}

startPercent = battery.getPercent();

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

Sleep(1000);

}

return 0;

}

Листинг 2 – 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);

};

Листинг 3 – 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();

}