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 №8

на тему

Некоторые служебные и технологические задачи

Студент П. С. Павлюткин

Преподаватель С. И. Сиротко

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 отдельные задачи, связанные с конфигурированием, мониторингом, управлением системой, а также технологические аспекты: библиотеки и сборки.Для достижения цели будет создано приложение для протоколирования оконных сообщений.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Подходы к хранению и использованию конфигураций. Реестр *Windows*: назначение; структура; типы и характер хранимых данных; интерфейс, группы функций *API*; порядок использования реестра и типичные задачи. Журналирование: назначение, решаемые задачи, основные используемые подходы. Журналы *Windows*: структура; виды хранимых данных; интерфейс, группы функций *API*; порядок использования журналов. Механизм *WinHook* – перехват и обработка оконных сообщений Windows. Библиотеки: назначение, место в технологическом процессе программирования; виды библиотек; использование библиотек. Динамические библиотеки (*DLL*) Windows: структура модуля; динамическое и статическое подключение, явный и неявный импорт; импортируемые объекты (символы); типовой каркас библиотеки *DLL*; порядок использования библиотек *DLL*, типичные применения и проблемы. Смешанные сборки (*mixed assembly*) *Windows*.

Логирование представляет собой процесс регистрации информации о работе программы в виде логов.

Отладка - процесс поиска и исправления ошибок в программном коде. Инструменты, позволяющие шагать по коду, устанавливать точки останова и анализировать значения переменных. Профилирование – сбор данных о производительности программы с целью оптимизации.

Инструменты для сбора данных о времени выполнения кода, использовании памяти и других характеристик. Тестирование программы с целью проверки ее корректности и соответствия требованиям. Процесс автоматизации компиляции, сборки и развертывания программного продукта.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было реализовано приложение для протоколирования оконных сообщений. (Рисунок 1).



Рисунок 1 – Открытие блокнота

Запускает новый процесс *Notepad* (`notepad.exe`) с использованием функции `*CreateProcessW*`. Создаются процесс и его поток, и запускается *Notepad*.

Даёт *Notepad* некоторое время (2 секунды) для полной инициализации.

Используется функция `*FindWindowW*` для поиска окна *Notepad*. Окно *Notepad* ищется по классу, который, вероятно, является `"*Notepad*"`.

Если окно *Notepad* найдено, вызывается `*SetHook*`, который устанавливает глобальный хук `*WH\_CALLWNDPROC*` с помощью `*SetWindowsHookEx*`. Этот хук позволяет перехватывать оконные сообщения.

Функция обратного вызова хука. Когда происходит оконное сообщение, она записывает его данные (номер сообщения, *wParam*, *lParam*) в файл лога (`*WindowMessages.log`*).

По завершении программы или её завершении, приложение закрывает файл лога и снимает установленный хук.

Таким образом, приложение создает процесс Notepad, устанавливает хук для перехвата оконных сообщений этого процесса и записывает эти сообщения в файл лога. Это может быть использовано для отслеживания и анализа сообщений, которые обрабатывает Notepad.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были рассмотрены методы взаимодействия между процессами с использованием глобального хука (*WinHook*) в среде Windows, при помощи библиотеки *WinAPI*, основанной на языке программирования *C++*. Работа включала в себя разработку программы, способной перехватывать оконные сообщения в заданном процессе, а также записывать эти сообщения в лог-файл для дальнейшего анализа.

Проект состоит из двух основных компонентов: приложения, запускающего процесс *Notepad* (Блокнот), и хука, устанавливающегося для перехвата оконных сообщений в данном процессе. После запуска *Notepad* и обнаружения его окна, устанавливается глобальный хук для отслеживания оконных событий. Полученные сообщения записываются в лог-файл "*WindowMessages.log*".

Применение глобального хука позволяет перехватывать сообщения, поступающие в указанное окно, и предоставляет дополнительные возможности для отладки и мониторинга оконных событий. Полученный опыт реализации таких служебных и технологических задач с использованием *WinAPI* может быть полезен для разработчиков, работающих над приложениями, требующими более глубокого взаимодействия с оконными процессами в операционной системе *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

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab8.cpp

#include <windows.h>

#include <string>

#include <fstream>

HHOOK g\_hook;

HWND g\_targetWindow;

std::wofstream g\_logFile;

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

if (nCode >= 0) {

CWPSTRUCT\* msg = reinterpret\_cast<CWPSTRUCT\*>(lParam);

if (msg->hwnd == g\_targetWindow) {

std::wstring message = L"Message: " + std::to\_wstring(msg->message) +

L", wParam: " + std::to\_wstring(msg->wParam) +

L", lParam: " + std::to\_wstring(msg->lParam) + L"\n";

g\_logFile << message;

}

}

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

}

HWND FindNotepadWindow() {

return FindWindowW(L"Notepad", nullptr);

}

void SetHook(HWND targetWindow) {

g\_targetWindow = targetWindow;

g\_hook = SetWindowsHookEx(WH\_CALLWNDPROC, HookCallback, nullptr, GetCurrentThreadId());

if (g\_hook == nullptr) {

MessageBoxW(nullptr, L"Failed to set hook.", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

g\_logFile.open(L"WindowMessages.log", std::ios::out | std::ios::app);

if (!g\_logFile.is\_open()) {

MessageBoxW(nullptr, L"Failed to open log file.", L"Error", MB\_OK | MB\_ICONERROR);

UnhookWindowsHookEx(g\_hook);

return;

}

}

void Unhook() {

if (g\_logFile.is\_open()) {

g\_logFile.close();

}

UnhookWindowsHookEx(g\_hook);

}

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

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

if (CreateProcessW(L"C:\\Windows\\System32\\notepad.exe", nullptr, nullptr, nullptr, FALSE, 0, nullptr, nullptr, &si, &pi)) {

Sleep(2000);

HWND targetWindow = FindNotepadWindow();

if (targetWindow != nullptr) {

SetHook(targetWindow);

MSG msg;

while (GetMessageW(&msg, nullptr, 0, 0) > 0) {

TranslateMessage(&msg);

DispatchMessageW(&msg);

}

Unhook();

}

else {

MessageBoxW(nullptr, L"Failed to find Notepad window.", L"Error", MB\_OK | MB\_ICONERROR);

}

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

MessageBoxW(nullptr, L"Failed to create process.", L"Error", MB\_OK | MB\_ICONERROR);

}

return 0;

}