**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра информатики и прикладной математикаи**

Лабораторная работа 6

Перехват прерываний в ОС Windows

Выполнил: Гхази Даниэль

Группа P3218

Преподаватель: Зыков А.Г.

2017 г.

**Задание**

Разработать программу, по функциональности альтернативную 6 лабораторной работе из прошлого семестра (по резидентным программам), выполняющую обработку требуемого события и осуществляющую при этом необходимое действие по крайней мере двумя способами.

• С использованием стандартной библиотеки Си (hotkey)

• С использованием средств WinApi (hook)S

**Код программы**

#include <stdio.h>

#include <Windows.h>

#include <string.h>

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <time.h>

#include <thread>

#include <conio.h>

#include <string>

#include <vector>

using namespace std;

HHOOK \_hook;

KBDLLHOOKSTRUCT \_eventInfo;

void showErrorMessage() {

cout << "Usage : main hook|hotkey" << endl;

}

void showHookSetupFailed() {

cout << "Hook setting failed" << endl;

}

void showHookSetupSuccessful() {

cout << "Hook setting done" << endl;

}

void showHookReleaseFailed() {

cout << "Hook releasing failed" << endl;

}

void showHookReleaseSuccessful() {

cout << "Hook releasing done" << endl;

}

void showHotkeySetupFailed() {

cout << "Hotkey setting failed" << endl;

}

void showHotkeySetupSuccessful() {

cout << "Hotkey setting done" << endl;

}

void showHotkeyReleaseFailed() {

cout << "Hotkey releasing failed" << endl;

}

void showHotkeyReleaseSuccessful() {

cout << "Hotkey releasing done" << endl;

}

void releaseHook() {

if (UnhookWindowsHookEx(\_hook)) {

showHookReleaseSuccessful();

}

else {

showHookReleaseFailed();

}

}

LRESULT \_\_stdcall hookCallback(int hookHandlingCode, WPARAM actionIdentifier, LPARAM eventInfo)

{

if (hookHandlingCode >= 0)

{

\_eventInfo = \*((KBDLLHOOKSTRUCT\*)eventInfo);

if (((actionIdentifier == WM\_KEYDOWN) || (actionIdentifier == WM\_SYSKEYDOWN)) && (GetAsyncKeyState(160) != 0))

{

if (\_eventInfo.vkCode == 160)

{

std::cout << "Entering hook mode, to exit of it, press \"esc\" button. " << std::endl;

std::cin.ignore(100000, '\n');

std::vector<char> symbols;

std::vector<int> numberOfPresses;

char key;

\_getch();

do

{

key = \_getch();

if (key == 27) break;

bool found = false;

for (int i = 0; i < symbols.size(); i++)

{

if (key == symbols[i])

{

numberOfPresses[i]++;

std::cout << "Symbol \"" << key << "\" was pressed " << numberOfPresses[i] <<

" times." << std::endl;

found = true;

break;

}

}

if (found) continue;

symbols.push\_back(key);

numberOfPresses.push\_back(1);

std::cout << "Symbol \"" << key << "\" was pressed 1 time." << std::endl;

} while (true);

std::cout << "Exiting hook mode." << std::endl;

PostQuitMessage(0);

ofstream myfile;

myfile.open("output.txt");

for (int i = 0; i < symbols.size(); i++)

{

myfile << symbols[i] << " - " << numberOfPresses[i] << endl;

}

myfile.close();

}

else if (\_eventInfo.vkCode == VK\_MENU)

{

releaseHook();

}

}

}

return CallNextHookEx(\_hook, hookHandlingCode, actionIdentifier, eventInfo);

}

void setHook() {

if (!(\_hook = SetWindowsHookEx(WH\_KEYBOARD\_LL, hookCallback, NULL, 0)))

{

showHookSetupFailed();

}

else

{

showHookSetupSuccessful();

}

}

void registerHotKey()

{

if (RegisterHotKey(NULL, 1, MOD\_SHIFT | MOD\_NOREPEAT, 'C'))

{

showHotkeySetupSuccessful();

}

else

{

showHotkeySetupFailed();

}

}

int main()

{

std::string mode;

std::cout << "1. hook\n2. hotkey\n";

std::cin >> mode;

int menuItem = mode[0] - '0';

switch (menuItem)

{

case 1:

cout << "Hook selected" << endl;

setHook();

break;

case 2:

cout << "Hotkey selected" << endl;

registerHotKey();

}

MSG msg;

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

{

if (msg.message == WM\_HOTKEY)

{

std::cout << "Entering hotkey mode, to exit of it, press \"esc\" button. " << std::endl;

std::vector<char> symbols;

std::vector<int> numberOfPresses;

char key;

do

{

key = \_getch();

if (key == 27) break;

bool found = false;

for (int i = 0; i < symbols.size(); i++)

{

if (key == symbols[i])

{

numberOfPresses[i]++;

std::cout << "Symbol \"" << key << "\" was pressed " << numberOfPresses[i] <<

" times." << std::endl;

found = true;

break;

}

}

if (found) continue;

symbols.push\_back(key);

numberOfPresses.push\_back(1);

std::cout << "Symbol \"" << key << "\" was pressed 1 time." << std::endl;

} while (true);

std::cout << "Exiting hotkey mode." << std::endl;

PostQuitMessage(0);

ofstream myfile;

myfile.open("output.txt");

for (int i = 0; i < symbols.size(); i++)

{

myfile << symbols[i] << " - " << numberOfPresses[i] << endl;

}

myfile.close();

}

}

\_getch();

return 0;

}

**Вывод**

Таким образом, были исследованы простейшие способы регистрации событий для выполнения действий по нажатии горячих клавиш в Windows. Использование SetHookWindowsEx позволяет делегировать некоторой функции обработку сразу нескольких сочетаний горячих клавиш, применение же RegisterHotKey сопряжено с некоторыми ограничения, основное из которых – невозможность определения нескольких комбинаций клавиш в рамках конкретного процесса, в связи с чем появляются основания считать использование SetHookWindowsEx предпочтительным в общем случае, хотя и несколько более сложным.