Д. Выкочко, А. И. Кучеров

(ГГУ им. Ф. Скорины, Гомель)

РАЗРАБОТКА КЛИЕНТСКОГО МОДУЛЯ СЛЕЖЕНИЯ ЗА ПОЛЬЗОВАТЕЛЯМИ

Клиентский модуль сбора статистики должен представлять из себя приложение, которое никак не влияя на работу пользователя будет собирать статистику активности последнего. Затем эти данные будут переданы серверной части для дальнейшей обработки и анализа.

Выбор среды Visual Studio 6.0 и языка Visual C++ не случаен. Он обусловлен тем, что ни одно другое средство разработки не предоставляет требуемых возможностей. Для примера приведем расшареный сегмент памяти в dll-библиотеках. Была сделана попытка реализовать вышесказанное в среде Borland 6.0, как одной из наиболее авторитетных, которая успехом не увенчалась.

Принцип слежения прост: каждые n секунд получается снимок всех запущенных в системе процессов. На основании этих снимков делается подсчет времени работы в каждом приложении. Список всех процессов получается вызовом функции CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0), далее вызывается Process32First(). Этой функцией получается первый процесс, при помощи которого можно пройтись по всему списку последовательными вызовами Process32Next. Функции для работы с процессами описаны в заголовочном файле tlhelp32.h. В этом же классе определен метод форматированного вывода накопившейся статистики. Вывод статистики представляет из себя xml-документ. Формат xml был выбран исключительно из-за удобства дальнейшего сбора и анализа полученной информации.

Для сбора статистики работы пользователя с клавиатурой и мышью необходимо определить функции для установки/удаления хуков. Также нужно определить функции для запуска опроса процессов и получения полной статистики работы пользователя в системе. Они будут определены в dll как внешние, т.е. могут быть вызваны из другого приложения, которое эту библиотеку загрузит. Для определения таких функций требуется добавить в проект специальный файл. Это обычный текстовый фал с таким же именем, как и у библиотеки и расширением def, в котором перечислены экспортируемые функции. Формат следующий: EXPORTS и далее с новой строки имена функций. Полная статистика получается при помощи вызова соответствующей функции в описанном классе и слияние этой информации с информацией об использовании клавиатуры и мыши. В результате получается корректный xml-файл.

Следует отметить еще одну особенность этой библиотеки, а именно то, что в памяти этой dll хранится получаемая статистика, для чего в ней имеется специальный сегмент, данные в котором используются всеми приложения совместно. Специальный сегмент требуется по тому, что без него каждый процесс, который эту dll подгрузит будет собирать статистику для себя лично, т.е. у него, грубо говоря, будет храниться информации о том, сколько раз пользователь кликнул на это окошко. Для системы в общем информация не особо полезная, поэтому требуется механизм обеспечения доступа всем процессам к одному хранилищу. В качестве такого хранилища может выступать база данных или даже простой файл, но, во-первых, работа с базой данных потребует более сложного кодирования, а во-вторых скажется на производительности всей системы, подгрузит сеть и потребует для себя сервера. С фалами ситуация еще хуже: такое решение не очень сложно в реализации, но программа будет требовать значительно больше системных ресурсов.

Следующим проектом или приложением является по своей сути win32 программой. Эта программа непосредственно взаимодействует с вышеупомянутой библиотекой. Т.е. здесь

содержится логика всего в общем модуля. Основная функция которую выполняет эта часть приложения – это конечно запуск цикла опроса процессов, для чего делаются постоянные вызовы из соответствующей библиотеки. Тут же устанавливаются хуки и записывается собранная статистика в файл. Также предусмотрено простое логирование для определения некоторых наиболее вероятных проблем, как то невозможность загрузить библиотеку или невозможность вызова функций.

Данное приложение отвечает за настройку. Настройками определяются интервалы считывания процессов, записи полученных данных на диск и путь для записи этих данных. Первые два параметра требуются для более точной настройки производительности и точности получаемых результатов. На медленных машинах, соответственно, может потребоваться больший интервал между считываниями процессов, на машинах с медленным жестким диском – больший интервал записи на диск. Таким образом уменьшится влияние на общую производительность, правда, за счет точности собираемой статистики. В случае неудачной загрузки приняты значения 5 и 10 для считывания и записи соответственно, в качестве папки для сбора логов в этом случае примется корень диска C.

Третий проект не связан с остальными, но выполняет часть функционала всего пакета. Это также dll-библиотека в которой определены функции-триггеры. Эти функции связываются с определенными событиями в системе и позволяют наблюдать за пользователем. Логируются следующие действия: вход/выход из системы, запуск/остановка скринсейвера и бликировка/разблокировка сеанса. Функции-триггеры также экспортируются при помощи def-файла и имеют следующее описание extern "C" void __stdcall foo(PWLX_NOTIFICATION_INFO pInfo). В такой функции вызывается функция записи в лог, соответственно, с параметром-именем события. В лог попадает строка вида <время>-<имя_события>. Далее при анализе такая информация позволит легко определить время работы пользователя в системе, а также то, сколько пользователь отсутствовал на рабочем месте.

Чтобы связать вызов определенной функции с событием в системе, требуется некоторая настройка реестра. В разделе HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows NT\ \CurrentVersion\Winlogon\Notify создаем ветку. Например, Watcher, имя значения не имеет. В этом разделе требуется создать несколько параметров, которые будут указывать на dll-библиотеку и на соответствующие функции в этой библиотеке. Принцип работы прост: Windows при загрузке системы подгружает все библиотеки, описанные в подразделах ветки Notify. Затем при наступлении события вызываются функции из всех библиотек, которые соответствуют этому событию.

В ходе разработки пришлось столкнуться с такой распространенной проблемой, как утечка памяти. Язык C++ не защищен от этого как, например, Java. Хотя следует отметить, что, даже программируя на Java можно, при особо “умелом” подходе, добиться неконтролируемого расхода ресурсов. Чтобы избежать такой проблемы в C++ от программиста требуется очень большая внимательность. Что касается этого приложения, то после первого же основательного тестирования пришлось пересматривать весь код в поисках «таких» мест. С памятью работа ведется довольно часто и поэтому невнимательность на начальных этапах разработки стоила достаточно дорого. Следующая проблема с которой пришлось столкнуться – это проблема запуска программ от имени другого пользователя. При работающем сканере приложения просто зависают. И тут, кажется, виноваты CreateToolhelp32Snapshot и Process32First. Каким образом они мешают сказать сложно, но больше думать не на что. Как видно работая в системе на таком низком уровне сложно добиться полной стабильности.