УДК 681.31

А.В. КЛИМЕНКО, К.Д. АЛЕКСЕЕВ

ОБЗОР АППАРАТНЫХ СРЕДСТВ И API-СЕРВИСОВ ОПРЕДЕЛЕНИЯ ВРЕМЕНИ В ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Abstract: *Main needs of multimedia and time-sensitive applications, overview of timer chipsets, analysis and estimations of system time services are given in the article. Research is aimed to use in multimedia and time-sensitive applications.*
Key words: *multimedia, timer, timer chipset, time service.*

Анотація: *У статті викладені основні потреби мультимедійних та чутливих до часу програм, проведено огляд системних таймерів, аналіз та запропоновано оцінки системних сервісів визначення часу. Дослідження призначені для побудови мультимедійних та чутливих до часу програм.*
Ключові слова: *мультимедіа, таймер, таймерний пристрій, сервіс визначення часу.*

Аннотация: *В статье излагаются основные потребности мультимедийных и чувствительных ко времени приложений, проведен обзор системных таймеров, анализ и предложены оценки системных сервисов определения времени. Исследования предназначены для построения мультимедийных и чувствительных ко времени программ.*
Ключевые слова: *мультимедиа, таймер, таймерное устройство, сервис определения времени.*

1. Введение

Сегодня перед разработчиками стоит множество задач, связанных с воспроизведением видео, звука и мультимедиа в целом. Это значит, что нужно работать с большими массивами и потоками данных, обрабатывать их, кодировать/декодировать и воспроизводить. Поэтому ключевым моментом при воспроизведении мультимедийного контента является своевременность, т.е. синхронность. Современные персональные компьютеры по производительности в состоянии работать с видео высокого разрешения. Нет сложности в том, чтобы считать необходимое количество данных с носителя, декодировать их и воспроизвести. Ключевая проблема – сделать это все вовремя.

Данная статья предлагает некоторую теоретическую информацию о системных таймерах Windows, потребностях современных, чувствительных ко времени, приложений и практические исследования точности разных таймеров.

2. Обзор системных таймеров

Системные таймеры – это таймеры, к которым обращается операционная система для отслеживания времени дня (даты) и обработки запланированных, зависящих от времени событий.

С развитием компьютерных технологий процессорные скорости увеличились на несколько порядков, а приложения стали гораздо более сложными. Такая эволюция предъявила новые требования к архитектуре таймеров, учитывая разрешающую способность таймера и скорость отклика. Попытки удовлетворить эти требования на сегодняшних системах заставили разработчиков пожертвовать качеством приложений и производительностью системы в целом для преодоления ограничений сегодняшних таймерных архитектур. Для удовлетворения потребностей приложений и поддержки сложных приложений в будущем системные таймеры должны предоставить разрешающую способность как минимум в 1 миллисекунду.

В персональных компьютерах на данный момент для разных задач используют четыре системных таймера, каждый из которых имеет собственный набор атрибутов, что делает работу с

ISSN 1028-9763. Математичні машини і системи, 2009, № 3

ними сложной и запутанной.

8254 Programmable Interval Timer (PIT) был представлен в персональном компьютере IBM в 1981 году. Он имеет разрешающую способность в 1 миллисекунду и поддерживает как периодический, так и апериодический режим. Поскольку считывание и запись на это устройство требуют коммуникации через порт ввода/вывода, использование его для операционной системы очень ресурсоемко. Из-за этого апериодическая функциональность данного таймера не используется на практике. По этой же причине данный таймер используется только в периодическом режиме для осуществления периодических прерываний в однопроцессорных системах.

В 1984 г. IBM ввели первый таймер реального времени real-time clock (RTC) в дополнение к 8254. Как и 8254, RTC имеет максимальное разрешение в 1 миллисекунду и поддерживает периодический и апериодический режимы. Как и в случае с 8254, коммуникация с устройством происходит через порт ввода/вывода, и, соответственно, эта процедура ресурсоемка. В дополнение к этим ограничениям RTC был разработан для генерации прерываний Advanced Configuration and Power Interface (ACPI) на системах, поддерживающих ACPI, что снижает производительность системы. RTC используется в периодическом режиме для осуществления прерываний системного профиля на одно-процессорных системах и для прерываний часов в мультипроцессорных системах.

У Advanced Programmable Interrupt Controller (APIC)-таймера есть много возможностей таймера рельного времени и он был разработан для использования в синхронизации многих процессоров, но имеет плохое разрешение, а его чип бывает медлительным. Поскольку многие системы не имеют APIC-таймера, он не используется по умолчанию.

ACPI-таймер, известный как PM Clock, был добавлен в архитектуру для предоставления надежных временных отпечатков, зависящих от процессорной скорости. В связи с этим таймер не несет никакой другой функциональной нагрузки.

Системы нуждаются в апериодической функциональности и более высокой точности. Без этих двух пунктов системы будут не в состоянии удовлетворить потребности будущих мультимедийных и чувствительных ко времени приложений. Новый таймер, который предоставит такую функциональность, заменит таймер 8254, таймер реального времени, PM Clock и APIC-таймер и позволит убрать устаревшие чипы из архитектуры.

Microsoft Windows XP использует периодические прерывания для отслеживания времени, срабатывания таймерных объектов и уменьшения квантов времени, выделенных на работу тредов. Когда Windows XP загружается, по умолчанию период прерывания составляет 10 миллисекунд, а иногда 15 на некоторых системах. Это означает, что каждые 10 миллисекунд операционная система получает прерывание от встроенного в архитектуру таймера.

Когда срабатывает часовое прерывание, Windows совершает два основных действия: обновляет счетчик тиков, если прошел полный тик с предыдущего обновления, и проверяет, не истек ли срок годности запланированных таймерных объектов. Тик – это абстрактная единица времени, которую использует Windows для отслеживания времени дня и завершенности кванта треда. На большинстве систем тик составляет 10 миллисекунд, на некоторых – 15. По умолчанию

часовое прерывание происходит каждый тик, но операционная система или приложения могут изменять период часовых прерываний, в то время, как период таймерного тика изменяться не может. Итак, тик всегда постоянен, вне зависимости от периода часовых прерываний.

В дополнение к отсчету тиков во время часовых прерываний система проверяет, не истек ли срок таймерного объекта, и планирует отложенный процедурный вызов (deferred procedure call), если такой найден. Таймерные объекты используются системой для отслеживания сроков и оповещения приложений о достижении этих сроков. Например, мультимедийное приложение, проигрывающее синхронизированные аудио и видео, может запросить, чтобы часть кода (например, проигрывание звука) была исполнена через 233 миллисекунды. Windows отслеживает этот запрос запуском таймерного объекта. Впоследствии, когда генерируются часовые прерывания, проверяются таймерные объекты на достижение нужного времени. Очевидно, что при периоде часовых прерываний в 10 миллисекунд операционная система проиграет звук с 7-миллисекундным опозданием. Если операционная система проверяет таймерные объекты в 230 миллисекунд и видит, что ни один из них не истек, следующая проверка происходит в 240 миллисекунд, и тут обнаруживается запоздавший запрос. В данном случае расстояние между часовыми прерываниями слишком велико и конечный пользователь может заметить проблемы синхронизации аудио/видео.

Очевидным недостатком такой имплементации является большой период часовых прерываний, который может вылиться в большие задержки. Для того чтобы приложения могли получать сообщения на сегодняшних системах с большей точностью, они должны запрашивать меньший период часовых прерываний. При рассмотрении предыдущего примера мультимедийное приложение должно запросить часовые прерывания с периодичностью в одну миллисекунду для того, чтобы код был выполнен точно вовремя.

Это позволит мультимедийному приложению выполнить свой код и проиграть звук вовремя, но понизит производительность системы в целом. Тесты компании Microsoft показали, что понижение размера тика до 2 миллисекунд не производит должного эффекта на производительность системы, в то время, как понижение размера тика до менее 2 миллисекунд значительно снизит производительность системы в целом. На более быстрых системах цена понижения периода часовых прерываний менее, чем до 2 миллисекунд, может быть доступной, но незначительный эффект повышения частоты прерываний на содержимое кэшей и управление питанием может быть нежелательным.

В предыдущем примере период часовых прерываний был понижен до 1 миллисекунды для того, чтобы проиграть звук точно вовремя. Это вылилось в 10 сгенерированных прерываний между временем в 230 миллисекунд и 240 миллисекунд в то время, как требовалось всего одно прерывание. Лучшим решением этой проблемы было бы планирование апериодического прерывания непосредственно тогда, когда должно быть выполнено некоторое задание. Тогда таймер сработает только один раз в 233 миллисекунды и система не будет тратить время на ненужные прерывания.

В дополнение к выигрышу в производительности апериодическая функциональность также положительно отразится на продолжительности работы батареи на мобильных платформах. Мобильные системы находятся в энергосберегающем режиме, когда нет системной активности,

таким образом продлевая жизнь батарейки. Для продления жизни батареи в мобильных платформах разработчики архитектур пытаются увеличить период системных часов. Это позволит системе находиться в энергосберегающем режиме больше времени, но это ещё более задержит запуск запланированных событий. Компромиссом между потребностями управления питанием и приложениями будет полная неактивность системы, когда нет никаких заданий и есть возобновление работы только для запланированных заданий или срочных сигналов. Апериодическая функциональность предлагает оптимальное решение.

Существующие таймеры имеют теоретическую максимальную разрешающую способность в 1 миллисекунду и достижимую разрешающую способность в 2 миллисекунды в силу ограничений производительности. В дополнение существующие таймеры могут иметь запаздывание в 1 миллисекунду, а это означает, что лучшее разрешение, на которое может полагаться приложение, составляет 3 миллисекунды. Мультимедийные приложения, тем не менее, могут иметь потребность в точности не менее 1 миллисекунды. Таким образом, в то время, как существующие таймеры теоретически отвечают потребностям современных мульмедийных приложений, в действительности они не в состоянии эти потребности удовлетворить.

Для удовлетворения мультимедийных потребностей на системах с вышеописанными таймерными устройствами разработчики были вынуждены изобрести трюки и уловки такие, как большие буферы проигрывания и загруженное ожидание, которые теряют в производительности и качестве для того, чтобы скрыть несовершенность таймерных устройств. Из-за этих трюков и уловок мультимедийные приложения на общедоступных системах с вышеописанными таймерными устройствами никогда не достигнут высокой производительности и высокого уровня удовлетворения ожиданий пользователя от персональных компьютеров.

Проигрывание и синхронизация видео высокого разрешения требуют таймерного разрешения в 1 миллисекунду для декодирования и синхронизации видеокадров и аудиопотока. Сложные графические пакеты требуют гранулярности в 1 микросекунду для корректного анимирования движения губ, совпадающего с произносимыми словами. Для удовлетворения потребностей мультимедийных и других чувствительных ко времени приложений необходим новый таймер, который сможет генерировать прерывания минимум каждую миллисекунду без торможения и генерации прерываний, запаздывающих на несколько миллисекунд [1].

3. Анализ системных сервисов определения времени

Посмотрим на проблему с точки зрения разработчика программного обеспечения. В его распоряжении есть множество системных сервисов определения времени, но мы исследуем следующие из них: QueryPerformanceCounter()/QueryPerformanceFrequency(), timeGetTime(), GetTickCount(), GetSystemTimeAsFileTime() и внутренний счетчик Pentium _emit 0x0F. Исследовать их нужно на скорость вызова, точность и надежность.

Для начала объясним, что из себя представляет каждый из этих сервисов. QueryPerformanceCounter() возвращает содержимое переменной процессора, являющейся счетчиком высокого разрешения, в количестве тиков. QueryPerformanceFrequency() возвращает частоту счетчика в количестве тиков в секунду.

Функция timeGetTime() возвращает время, прошедшее с начала запуска Windows в миллисекундах. Этот сервис является мультимедийным сервисом Windows. GetTickCount() также возвращает время, прошедшее с начала запуска Windows в миллисекундах, но при этом её точность не может регулироваться функциями timeBeginPeriod() и timeEndPeriod() в отличие от предыдущей.

GetSystemTimeAsFileTime() возвращает системное время и дату в UTC-формате с теоретической заявленной точностью в 100 наносекунд.

Инструкция _emit 0x0F работает только на семействе процессоров Intel Pentium, возвращает количество тиков. Неудобство использования этого счетчика заключается в необходимости калибровки на каждой отдельной машине. То есть количество тиков этого счетчика в одной секунде неизвестно. Калибровка же добавит к ошибке точности полученных результатов ошибку точности таймера, которым будет производиться оценка частоты тиков.

Все исследования проводились на двух разных компьютерах: MacBook Pro с процессором Intel Core 2 Duo с частотой 2,4 Ghz (будем называть станция 1), ПК с процессором Intel Pentium D с частотой 3,2 Ghz (станция 2).

При проведении первого эксперимента исследуем скорость вызова функций QueryPerformanceCounter(), timeGetTime(), GetTickCount() и инструкции _emit 0x0F [2]. В ходе эксперимента в циклах на 100, 500, 1000, 10000 итераций вызываются соответствующие функции. Засекается время перед циклом и после цикла, разница во времени делится на количество итераций. Результаты показаны в табл. 1 и 2, скорость вызова представлена в микросекундах.

Таблица 1. Результаты первого эксперимента для станции 1

Кол-во итераций/сервис	100	500	1000	10000
QueryPerformanceCounter()	3,4836	2,5494	2,4598	2,5094
timeGetTime()	0,0782	0,0458	0,0430	0,0410
GetTickCount()	0,0307	0,0128	0,0100	0,0077
_emit 0x0F	0,1089	0,0893	0,0866	0,0842

Таблица 2. Результаты первого эксперимента для станции 2

Кол-во итераций/сервис	100	500	1000	10000
QueryPerformanceCounter()	0,3204	0,3121	0,3148	0,3116
timeGetTime()	0,0657	0,0582	0,0579	0,0576
GetTickCount()	0,0096	0,0056	0,0053	0,0050
_emit 0x0F	0,0350	0,0317	0,0314	0,0311

По результатам экспериментов на обеих станциях видно, что время вызова QueryPerformanceCounter() наибольшее и составляет приблизительно 2,5 – 3 мкс на станции 1 и 0,3 мкс на станции 2, в то время, как время вызова GetTickCount() наименьшее: 0,008 – 0,03 мкс на станции 1 и 0,0050 – 0,0096 мкс на станции 2.

При проведении второго эксперимента попытаемся определить точность возвращаемых

результатов функциями QueryPerformanceCounter(), timeGetTime(), GetTickCount() и GetSystemTimeAsFileTime(). В ходе эксперимента вызываем соответствующие функции в циклах по 10 000 000 итераций (QueryPerformanceCounter()) и по 1 000 000 000 итераций (timeGetTime(), GetTickCount() и GetSystemTimeAsFileTime()). Находим максимальную разницу между результатами на текущей и предыдущей итерациях. Результаты представлены в миллисекундах и показаны в следующей табл. 3.

Таблица 3. Результаты второго эксперимента

Станция/сервис	Станция 1	Станция 2
QueryPerformanceCounter()	4,583	0,381
timeGetTime()	5,000	1,000
GetTickCount()	16,000	16,000
GetSystemTimeAsFileTime()	15,625	15,625

Видно, что GetTickCount() на самом деле имеет точность в 16 мс, GetSystemTimeAsFileTime() – 15,6 мс, timeGetTime() – 5 мс на станции 1 и 1 мс на станции 2, а QueryPerformanceCounter() – 4,6 мс и 0,4 мс соответственно.

Исследуем надежность наиболее точных сервисов QueryPerformanceCounter() и timeGetTime(). Поскольку timeGetTime() возвращает время в миллисекундах, оценим количество итераций, в которых отклонение равно 1 мс, и количество итераций, в которых отклонение больше 1 мс (табл. 4).

Таблица 4. Результаты исследования надежности сервиса timeGetTime()

Отклонение	Станция 1	Станция 2
Равно 1 мс	16328	58245
Больше 1 мс	46	0

На станции 1 на 16328 изменений значения счетчика 46 отклонений, превышающих 1 мс, а на станции 2 на 58245 изменений 0 отклонений, превышающих 1 мс. Вышеприведенные результаты говорят о том, что на станции 1 мы можем рассчитывать на точность timeGetTime() в 1 мс в 99,7% случаев, а на станции 2 – в 100% случаев (табл. 5).

Таблица 5. Результаты исследования надежности сервиса QueryPerformanceCounter()

Отклонение	Станция 1	Станция 2
10-100 мкс	2316	214
100-1000 мкс	69	2
Более 1 мс	30	0

Попытаемся оценить надежность QueryPerformanceCounter(). Для QueryPerformanceCounter() посчитаем количество отклонений в пределах 10–100 мкс, количество отклонений в пределах 100-1000 мкс и количество отклонений более 1 мс.

На станции 1 на 10 000 000 вызовов 99 отклонений больше 100 мкс, а на станции 2, на то же количество вызовов, 2 отклонения больше 100 мкс. Данные говорят о том, что мы можем

рассчитывать на точность QueryPerformanceCounter() в 100 мкс на станции 1 в 99,9991% случаев, а на станции 2 – в 99,9999 %.

Исследования показали, что использование сервиса _emit 0x0F не практично из-за архитектурных ограничений: не будет работать на AMD или других x86 процессорах, сервисы GetTickCount() и GetSystemTimeAsFileTime() имеют очень низкую точность, а QueryPerformanceCounter() и timeGetTime() имеют довольно высокую точность и надежность. Авторами статьи для дальнейших исследований был выбран QueryPerformanceCounter() как самый точный и имеющий высокий уровень надежности.

4. Выводы

Подытоживая вышесказанное, отметим, что сервисы определения времени, как и аппаратные таймеры, – узкое место современных персональных компьютеров (ПК). Необходимо разработать и внедрить в архитектуру ПК новое таймерное устройство, обладающее более высокой разрешающей способностью, чем существующее на сегодняшний день. Поскольку же разработчик в данный момент времени ограничен возможностями старых аппаратных таймеров, имеем сервис QueryPerformanceCounter(), позволяющий получать данные с точностью до 100 мкс. На такую точность мы и должны рассчитывать при написании программного обеспечения.

СПИСОК ЛИТЕРАТУРЫ

1. http://www.microsoft.com/whdc/system/sysinternals/mm-timer.mspx.
2. http://developer.nvidia.com/object/timer_function_performance.html.

Стаття надійшла до редакції 04.06.2009