Измерение накладных расходов в операционной системе для построения системы моделирования процессов реального времени

Иванов Ю.А.
Кафедра ЭВМ ДонНТУ
No0ther@yandex.ru

Аннотация

Иванов Ю.А. Измерение накладных расходов в операционной системе для построения системы моделирования процессов реального времени. *Выполнен анализ запаздывания вносимого системными функциями операционной системы при разработке инструментальных средств управления системой реального времени. Разработаны программные модели для измерения временных характеристик циклических многопоточных блоков приложений реального времени. Исследованы числовые параметры для синтеза программно-аппаратной модели планирования расписаний.*

Анотація

Іванов Ю.О. Вимір накладних витрат в операційній системі для побудови системи моделювання процесів реального часу. *Виконаний аналіз запізнювання, що вноситься системними функціями операційної системи при розробці інструментальних засобів управління системою реального часу. Розроблені програмні моделі для виміру тимчасових характеристик циклічних багатопотокових блоків програм реального часу. Досліджені числові параметри для синтезу програмно-апаратної моделі планування розкладів.*

Abstract

Ivanov Y. Measuring of overhead costs in the operating system for the development the modeling system of real time processes. *The analysis of delay of system functions of the operating system at development of management tools is completed for the real-time system. Program models are developed for measuring of temporal descriptions of cyclic multithread blocks of the real time applications. Numerical parameters are determined for the synthesis of HW/SW scheduler model.*

Введение

Системы моделирования на основе операционных систем реального времени (ОСРВ) должны удовлетворять следующим известным требованиям [1]:

- многозадачность и способность допускать вытеснение (preemptable),
- наличие механизмов приоритетов для потоков,
- поддержка предсказуемых механизмов синхронизации,
- способность обеспечивать механизм наследования приоритетов,
- поведение ОС должно быть известным и прогнозируемым (задержки обработки прерываний, задержки переключения задач, задержки драйверов и т.д.); это значит, что во всех сценариях рабочей нагрузки системы должно быть определено максимальное время отклика.

Базовым модулем, обеспечивающим функционирование жестких динамических систем реального времени (СРВ), должен быть специализированный менеджер задач реального времени. При решении СРВ-задач моделирования возникает необходимость в разработке уникальных систем для каждой аппаратной платформы и модели. Более эффективным представляется решение – использовать в данных целях модули существующих операционных систем (ОС), профилированных соответствующим образом. Основная задача при этом сводится к созданию специализированного планировщика для распространенных операционных систем класса Windows, Unix. В этой связи актуальным является исследование архитектурных возможностей по минимизации накладных расходов, возникающих в результате выполнения сервисных операций при планировании и последующем выполнении многочастотных моделей реального времени. При этом считаем, что подзадаче или группе подзадач будет соответствовать поток многопоточного приложения.

Первым шагом в решении данной задачи является исследование программных моделей, учитывающих как архитектурные особенности персональных компьютеров, так и операционной системы. Для этого необходимо определить характеристики временных задержек и прерываний как исходных данных для расчета параметров будущей системы.

Разработка программной модели

Основной причиной отказа от использования ОС класса Windows является то, что система не позволяет выполнять задержки и измерения времени, не превышающие 1 мс. Этот интервал является критичным для большинства моделей технологических процессов. Анализ публикаций [2] показывает, что существует много попыток более быстрых обработок в

стандартных версиях Windows, но они не подтверждены численными результатами:

1) Реализация посредством драйвера таймера не удовлетворяет временным условиям потому, что в Windows NT драйвер перепрограммирован на уровне ядра и вследствие этого чтение его значений не всегда может быть корректным;

2) Использование системных функций QueryPerformance-Frequency , QueryPerformanceCounter (или любых других из набора API) также ни к чему не приводит, так как в Windows NT используется разрешение 15 мс. Следовательно, невозможно использовать эти функции в рамках конкретной модели, поскольку время выполнения самих функций существенно меньше интервалов между таймерами, которые принудительно выполняются ОС;

3) Установка обработчика прерывания на IRQ0 в Interrupt Descriptor Table может обеспечивать обработку прерываний с интервалом более 1мс, что также не достаточно для целей поставленной задачи.

Поскольку большинство клонов Linux бесплатны, имеют открытый код и позволяют изменять ядро системы, для исследования была выбрана ОС Linux версии ASP Linux 11.2 Ladoga с ядром 2.6.17. Анализ показывает [3], что в данной системе выполнение кода происходит в потоке, поэтому именно поток является базовым элементом, которому выделяется процессорное время. Но для Linux каждый поток является процессом, поэтому политики планирования применяются к процессам. По умолчанию для любого стандартного процесса в nix-системах используется политика планирования SCHED_OTHER (с приоритетом 0). Стандартные модификации Linux-систем имеют две политики для приложений, критичных к времени исполнения: SCHED_FIFO и SCHED_RR. Они выполняют контроль необходимого процесса по ходу и дают гарантии на определенные кванты выполнения программы. При этом используются приоритеты от 1 до 99 [4]. Для исследования функционирования политики планирования реального времени по дисциплине SCHED_FIFO была разработана программная модель анализа одновременной работы потоков. Фрагмент программы анализа одновременной работы потоков:

```
pid = getpid();
sp.sched_priority = 99;
res = sched_setscheduler( pid, SCHED_FIFO, &sp );
…
for ( current_id = 0; current_id < N; current_id++ )
{
      pthread_create(    &thread[current_id],    NULL,    thread_func,
&tp[current_id] );
      sp.sched_priority = priority;
      res = sched_setscheduler( pid + current_id + 1, policy, &sp );
      priority += 97;
}
…
```

```
          for ( current_id = 0; current_id < N; current_id++ )
          {
              pthread_join( thread[current_id], NULL );
          }
      …
           printf( "\nResults of a work:\n" );
          for ( current_id = 0; current_id < N; current_id++ )
            printf( "Routine of thread %d works %lu us\n", tp[
            current_id].id,          tp[current_id].lsec*1000000          +
tp[current_id].lusec );
      …
      void * thread_func( void * arg )
      {
      …
          pid_t thread_id = gettid();
          pid_t group_id = getpid();
          printf( "This is %d thread with ID = %d and belongs to group
%d\n", loc_id, thread_id, group_id );
          sched_getparam( thread_id, &sp );
          res = sched_getscheduler( thread_id );
          printf(   "It   has     %s   policy   and   priority   =   %d\n",
policyNames[res], sp.sched_priority );
      …
      }
```

Определяющей особенностью модели является то, что у главной (управляющей) программы приоритет должен быть выше, чем у дочерних потоков. Иначе поток, имеющий больший приоритет, чем программа, будет получать все процессорное время на свое выполнение и сама программа получит управление только после завершения работы потока, что является последовательным выполнением. В приведенном процессе работают два потока с девяносто восьмым и первым приоритетами, сам процесс имеет девяносто девятый приоритет. По промежуточным выводам из функций обработки потока и времени выполнения (рис. 2) видно, что поток с девяносто восьмым приоритетом полностью отобрал управление у потока с первым приоритетом, что привело к их последовательному выполнению (хотя они и запустились одновременно). Результаты выполнения потоков с разными приоритетами:

```
This is 1 thread with ID = 5594 and belongs to group 5592
       It has  SCHED_FIFO policy and priority = 98
This is 0 thread with ID = 5593 and belongs to group 5592
       It has  SCHED_FIFO policy and priority = 1
                   Results of a work:
             Routine of thread 0 works 344 us
             Routine of thread 1 works 186 us
```

Кроме того, время выполнения потока с первым приоритетом гораздо больше, так как оно содержит период простоя потока, пока выполнялся поток с девяносто восьмым приоритетом. Данный эксперимент показывает, что система в режиме реального времени производит монопольное выполнение потоков с большим приоритетом, что и необходимо для целей поставленной задачи. Для дальнейшего исследования временных параметров системы необходимо выбрать способы измерения времени и выполнения задержек.

Оценка временных интервалов

При измерении временных интервалов выполнения различных процессов в вычислительной системе наиболее точным способом является использование внешнего оборудования, не нагружающего систему дополнительными накладными расходами, которые вносит механизм измерения.

Определим возможности по измерению времени, предоставляемые исследуемой программной и аппаратной средой, в случае, если не используется дополнительное внешнее оборудование. Из обеспечиваемых интерфейсом системных функций ОС была выбрана функция GETTIMEOFDAY(2), так как возвращаемый ею формат времени включает микросекунды, что является достаточным для многих конкретных задач.

Для исследования задержек, вносимых функцией GETTIMEOFDAY(2), в модели было выполнено последовательно многократное циклическое обращение к функции и получены следующие результаты (табл.1):

Таблица 1 – Задержки, обеспечиваемые функцией GETTIMEOFDAY

Текущее значение времени(мкс)	Задержка от предыдущего значения(мкс)
402868	0
402871	3
402873	2
402876	3
402878	2
402881	3
402883	2
402886	3
402888	2
402891	3
402893	2

В ходе экспериментов было получено, что максимальное значение задержки равно 3 мкс, а минимальное – 2 мкс (табл. 1). Данные задержки удовлетворяют временному диапазону исследуемой модели, поэтому использование данной функции оправдано.

Более точные результаты могут дать только данные, полученные непосредственно с использованием аппаратных средств компьютера [5]. Одним из способов полученией таких данных является использование инструкции микропроцессора rdtsc, которая возвращает значение счетчика, хранящего количество тактов с момента начала работы микропроцессора. Зная его частоту, можно получить значение времени, соответствующее одному такту данного процессора:

$$t = \frac{delta\_rdtsc}{cpu\_frequency} = \frac{1}{cpu\_frequency},$$

где t – длительность тиков в микросекундах,
delta_rdtsc – количество тактов (1 для одного такта),
cpu_frequency – частота процессора (в МГц).

Таким образом, не учитывая конвейеризации, можно заключить, что накладные задержки возникают только из-за времени выполнения одной инструкции. Оно может быть определено, поскольку эта инструкция будет выполняться одно и то же количество тактов на одной конфигурации. Для современных компьютеров время выполнения данной инструкции, определенное этим методом, меньше 1мкс (для процессоров с частотой, превышающей 200 МГц). Следовательно, инструкция rdtsc является наиболее точным средством измерения времени во временном интервале данной задачи, поэтому ее и следует использовать.

Исследование временных задержек

Следующим этапом необходимо выбрать способ реализации временных задержек, которые могут быть использованы для запланированного выполнения процессов.

Для выполнения задержек ОС, предоставляет функцию USLEEP(3). Особенностью данной функции является то, что она позволяет выполнять временные задержки с дискретностью микросекунд. Было проведено исследование данной функции. В качестве параметров в тестовой программе были переданы значения соответствующие задержке в 1 мкс. Измерение времени выполнения проводилось с помощью функции GETTIMEOFDAY(2) и получены следующие результаты (табл. 2):

Таблица 2 – Задержки, обеспечиваемые функцией USLEEP

Текущее значение времени (мкс)	Задержка от предыдущего значения (мкс)
844947	0
847680	2733
851934	4254
855730	3796
859678	3948
878756	19078

Учитывая, что задержка была установлена на 1 мкс, реальное значени задержки многократно превысило это значение, что подтверждается результатами исследования (табл. 2). При этом максимальное значение задержек превысило 4 мс (табл. 2), поэтому использование данной функции представляется невозможным для исследования.

Проанализируем использование инструкции rdtsc для выполнения задержек. Поскольку инструкция возвращает значение счетчика тиков процессора, необходимо установить соответствие между временем тика и времени в микросекундах. При этом необходимо учесть задержку, вносимую самой инструкцией. Для этого была разработана программная модель исследования времени выполнения инструкции rdtsc путем последовательного вызова нескольких инструкций. Основной результат: время выполнения инструкции соответствует 181 такту CPU. Исходя из этого, в программной модели была выполнена задержка с помощью инструкции rdtsc на 182 такта и далее на последовательно увеличивающееся количество тактов с шагом 10. Повтор каждого из значений задержки выполнялся 300 раз для увеличения точности измерений. Реальное время выполнения задержек было измерено с помощью функции GETTIMEOFDAY(2) (см п.3) (рис.1).

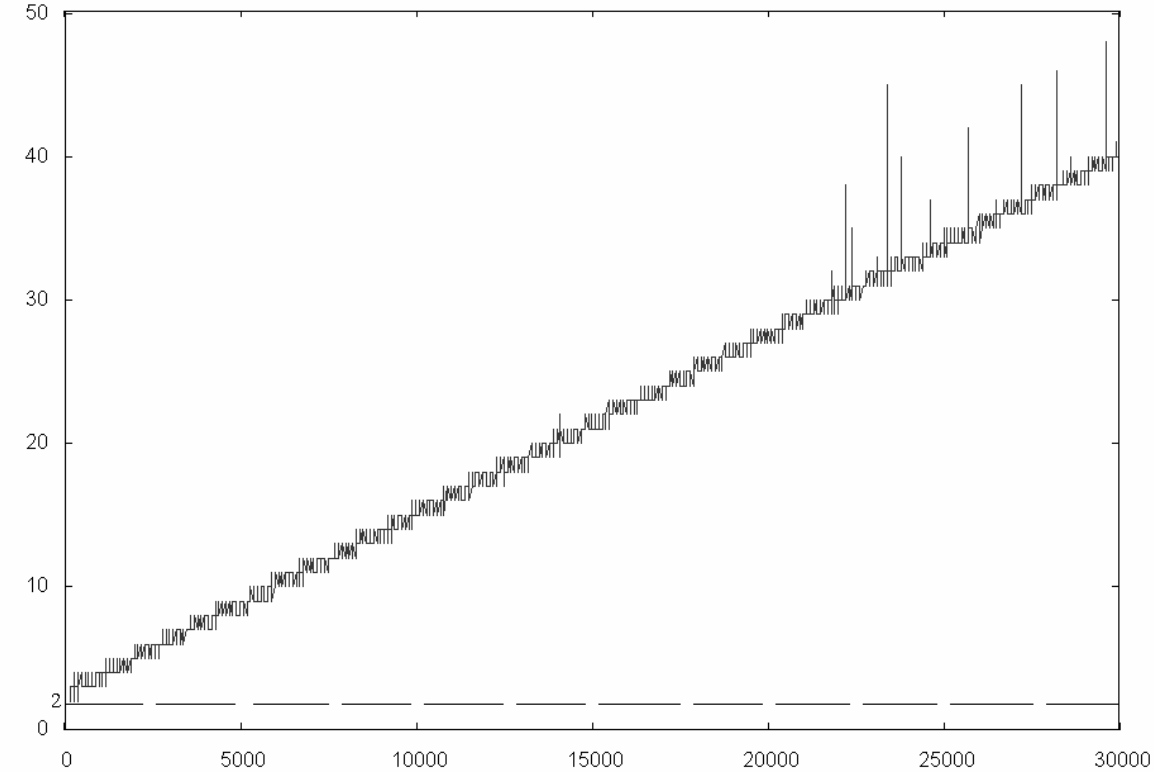

Рисунок 1 – Оценка величины задержек с помощью rdtsc

Учитывая задержки вносимые функцией GETTIMEOFDAY, из (рис.1) можно заключить, что с помощью инструкции rdtsc обеспечиваются задержки, начиная с 2 мкс.

Оценка времени переключения контекста процесса

Исследование базовых параметров позволяет перейти к определению времени переключения контекста процессов. Эта задача исследовалась в работе [6]. При данном исследовании была рассмотрена отличная от [6]

методика измерения времени переключений контекста. Время обслуживания определялось следующим образом: при остановке выполнения одного из потоков обеспечивается замер времени. При входе в функцию обработки второго потока осуществляется второй замер. Время, полученное в результате вычитания, может быть рассмотрено, как искомое время обслуживания. Данное значение времени обслуживания хранится в глобальной переменной и выводится на экран контролирующей программой при каждом переключении между потоками. Фрагмент программной модели определения времени переключения контекста:

```
          for ( i = 0; i < 100; i++ )
          {
            priority = 1;
            status = 0;

              for ( current_id = 0; current_id < N; current_id++ )
              {
                gettimeofday( &tv, &tz );
                tp[current_id].lsec = tv.tv_sec;
                tp[current_id].lusec = tv.tv_usec;
                tp[current_id].policy = policy;
                tp[current_id].priority = priority;
                pthread_create(  &thread[current_id],  NULL,  thread_func,
&tp[current_id] );
                priority += 97;
            }


            for ( current_id = 0; current_id < N; current_id++ )
            {
                pthread_join( thread[current_id], NULL );
            }

            while( status < 2 );

            printf( "Delta = %ld us\n", tp[0].delta - tp[1].delta );
            delta[i] = tp[0].delta - tp[1].delta;
          }
```

Для определения максимального и среднего времени опыт переключения контекста потоков повторялся десять тысяч раз. Ниже приведен фрагмент, в котором было получено максимальное значение для времени обслуживания (рис. 2):

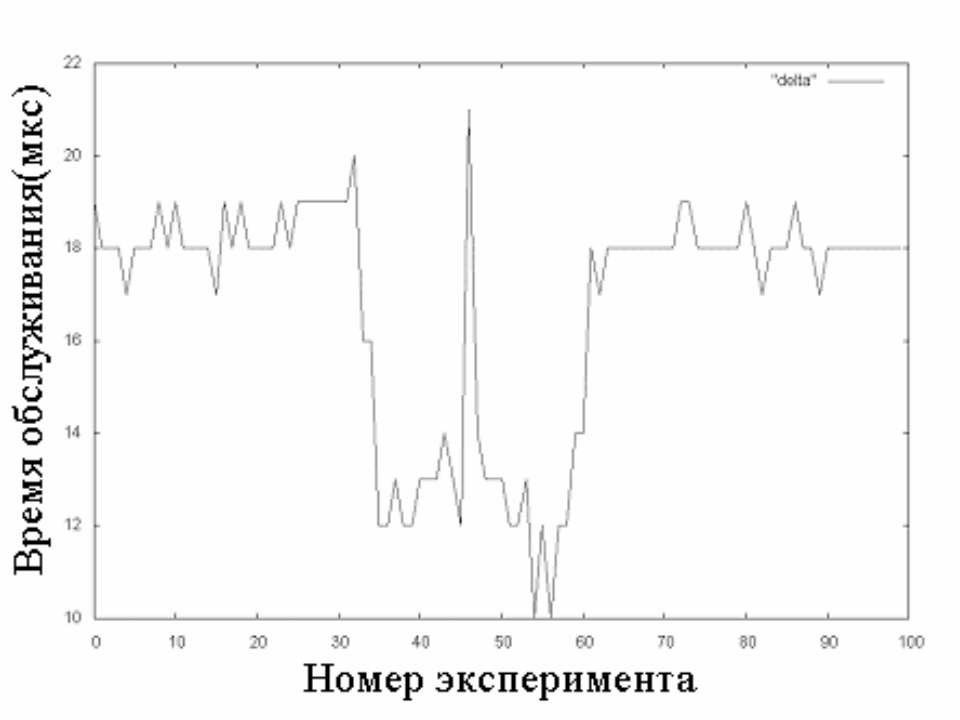

Рисунок 2 – Оценка времени переключения контекста

Наличие выбросов на диаграмме определяется особенностями алгоритмов синхронизации потоков, а также механизма обработки прерываний ядра операционной системы. Подобное поведение графика результатов времени переключения контекста, наблюдается для всех значений, полученных в ходе эксперимента. Усредненная по всем значением эксперимента величина времени обслуживания составляет 16.4257 мкс, а максимальная – 21 мкс. Данные полученные опытным путем сопоставимы с результатами исследований в [6], что подтверждает их достоверность.

Выводы

Разработаны инструментальные программные модели для измерения временных параметров. Исследования с помощью программных моделей ОС ASPLinux позволили получить интервалы изменения исходных параметров для алгоритмов планирования расписаний. Максимальное время переключения контекста при монопольном использовании процессора равно 21 мкс. В зависимости от исходного состояния операционной системы это значение может быть уменьшено до 10 мкс. Минимально возможная программно реализуемая задержка, применимая для формирования временных интервалов, составила 2 мкс. Погрешность измерения, вносимая инструкцией RDTSC, не превышает 2 мкс.

Дальнейшую оптимизацию как аппаратной (возможно многопроцессорной), так и программной модели целесообразно выполнять

для конкретной задачи реального времени совместно с профилированием алгоритма решения.

Литература

1. Timmerman Martin, Monfret Jean-Christophe. Windows NT as Real-Time OS?, Real-Time Magazine 97Q2, 1997.
2. Материалы форума, Точный тайминг, http://forum.shelek.ru/index.php/topic,2182.0.html, 2004.
3. Bovet Daniel P., Cesati Marco. Understanding the Linux Kernel, 3rd Edition, O'Reilly, 2005, p. 78–92.
4. Stevens W. R., Rago S. A., Advanced Programming in the UNIX® Environment: Second Edition, Addison Wesley Professional, 2005, p. 374-377.
5. Желтов Сергей, Братанов Станислав. Измерения производительности многоядерных систем с поддержкой технологии HT: преимущества подхода, ориентированного на потоки, Technology Intel, 2006.
6. Li Chuanpeng, Ding Cheng, Shen Kai, Quantifying the cost of context switch, ACM, 2007.

Иванов Юрий Александрович.
Магистрант ДонНТУ по специальности «Системное программирование». В 2008 году получил диплом бакалавра по специальности «Системное программирование» в ДонНТУ.
Научные интересы: моделирование, системы реального времени.

Дата надходження до редакції 12.12.2008 р.