Моделирование и анализ информационных систем. Т. 23, №6 (2016), с. 673–687
Modeling and Analysis of Information Systems. Vol. 23, No 6 (2016), pp. 673–687

DOI: 10.18255/1818-1015-2016-6-673-687

УДК 004.04

Имитационное моделирование для анализа выполнимости приложений реального времени

Баранов С. Н.[1], **Никифоров В.В.**

получена 15 марта 2016

Аннотация. Описывается развиваемый авторами подход к проверке выполнимости многозадачных приложений реального времени в различных сочетаниях дисциплины планирования и протокола доступа к разделяемым общим информационным ресурсам при исполнении данного приложения на многоядерной вычислительной платформе. Структура приложения задается в виде простого формализованного профиля, состоящего из сегментов трех видов, и описывающего доступ задач приложения к разделяемым информационным ресурсам; для каждого сегмента дается оценка необходимого ему объема вычислительного ресурса процессора. В основе данного подхода лежит введенное авторами понятие плотности программного приложения, которое характеризует потенциальную эффективность использования вычислительного ресурса приложением с определенным профилем. Значение эффективности определяется путем оценки выполнимости приложения с заданным профилем в зависимости от производительности процессора. Практическим инструментом для такой оценки служит разработанная авторами программа имитационного моделирования, обеспечивающая более точные, по сравнению с известными аналитическими методами, оценки. Приводится архитектура этого инструмента и общие сведения по его двум разнородным реализациям, а также представленные графиками результаты проведенных с их помощью экспериментов на ряде эталонных примеров, включая конфигурации Лю-Лейланда многозадачного приложения реального времени, вместе с их анализом и объяснением. Предложенный подход позволяет находить и выбирать оптимальное сочетание дисциплины планирования и протокола доступа для многозадачного приложения с заданным профилем.

Ключевые слова: имитационное моделирование, реальное время, плотность приложений, выполнимость приложений

Для цитирования: Баранов С. Н., Никифоров В.В., "Имитационное моделирование для анализа выполнимости приложений реального времени", *Моделирование и анализ информационных систем*, **23**:6 (2016), 673–687.

Об авторах:
Баранов Сергей Николаевич, orcid.org/0000-0001-6692-8432, д-р. физ.-мат. наук, проф.,
СПИИРАН, университет ИТМО,
Кронверкский пр., 49, г. Санкт-Петербург, 197101 Россия, e-mail: SNBaranov@gmail.com

Никифоров Виктор Викентьевич, orcid.org/0002-0007-1896-0876, д-р. тех. наук, проф.,
СПИИРАН, 14-я линия 39, г. Санкт-Петербург, 199178 Россия, e-mail: nik@iias.spb.su

Благодарности:
[1]Работа выполнена при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01).

Введение

Программные приложения для систем реального времени (СРВ) обычно строятся как совокупности *задач* с их *приоритетами*, каждая из которых считается последовательной программой, замкнутой в себе по передачам управления. Во время своего исполнения такие задачи могут разделять *общие системные ресурсы*, как *исполнительные* – процессор или процессорные ядра в случае многоядерной платформы, так и *информационные* – глобальные массивы данных, интерфейсные регистры периферийных устройств, элементы человеко-машинного интерфейса и т.п. Доступ к исполнительным ресурсам управляется применяемой *дисциплиной планирования*, а доступ к информационным ресурсам – выбранным *протоколом доступа*.

Каждая задача τ_i приложения СРВ характеризуется своим *периодом* T_i, *предельным сроком* ее выполнения D_i и *абсолютным весом* W_i. Период и предельный срок задаются в абсолютных единицах времени (например, миллисекундах) и обычно рассматриваются как «внешние ограничения» данного приложения, тогда как его абсолютный вес считается «внутренним ограничением» – он характеризует «объем вычислительной работы» (и, таким образом, количество вычислительного ресурса), необходимой данной задаче для выполнения своей функции, и задается в виде числа эталонных машинных операций в данной реализации данной задачи. При заданной *производительности процессора* P в виде числа тех же эталонных операций в единицу времени абсолютный вес W_i задачи τ_i может быть преобразован в ее *относительный вес* C_i, выраженный в единицах времени:

$$C_i = \frac{W_i}{P}. \tag{1}$$

Поведение приложения состоит в параллельном исполнении ряда экземпляров $\tau_i^{(j)}$ его задач τ_i, называемых *заданиями*, которые порождаются с периодичностью T_i. Поскольку задания $\tau_i^{(j)}$ конкурируют между собой за общие системные ресурсы, то исполнение любого из них может быть приостановлено и затем возобновлено через какой-то промежуток времени. *Временем отклика* R_i задачи τ_i является максимальный из интервалов времени между моментами порождения и завершения заданий $\tau_i^{(j)}$, которые называются *интервалами существования* этих заданий.

Ключевым требованием к программному приложению СРВ является его *выполнимость*, формулируемая как $\forall i\ (D_i \geq R_i)$ для всех допустимых сценариев взаимодействия данного приложения с внешней средой. Выполнимость приложения может быть установлена путем аналитической оценки времени отклика для каждой задачи приложения или путем имитационного моделирования поведения данного приложения при заданном сценарии его взаимодействия с внешней средой с помощью соответствующего программного инструмента.

Для приложений СРВ, предназначенных к исполнению на одноядерном процессоре, точные аналитические оценки их выполнимости существуют еще с начала 1970-х годов [1–3], однако для многоядерных процессоров такие оценки до сих пор не известны, а предлагаемые грубые методы дают пессимистические оценки, по сравнению с реальным поведением таких систем [4, 5]. По этой причине возникает необходимость в программном инструменте имитационного моделирования для получения оценок выполнимости приложений СРВ, предназначенных к исполнению

на многоядерных платформах при различных сочетаниях дисциплин планирования и протоколов доступа, более точных, чем оценки, какие дают известные аналитические методы. В данной работе описывается архитектура такого инструментального средства [6] и результаты ряда экспериментов, выполненных с его помощью.

Данная работа дополняет результаты исследований [7]. В разделах 1 и 2 приводятся введенные в [7] понятия и обозначения, необходимые для дальнейшего изложения (плотность приложения, общая схема работы и архитектура программного инструментального комплекса). Отсутствующее в [7] описание понятия профиля многозадачного приложения и способа его представления введено в разделе 3, что позволило в разделе 4 дополнить результаты проведенных экспериментов точным указанием параметров исследуемых примеров приложений и выводами по особенностям полученных результатов измерений.

1. Плотность приложения

Производной характеристикой поведения задачи τ_i при заданной производительности P многоядерного процессора является его *полезная нагрузка*:

$$u_i = \frac{C_i}{T_i}, \tag{2}$$

вычисляемая как доля в периоде задачи, которая уходит на собственно вычислительную работу для данной задачи; таким образом, *общая полезная нагрузка* U многозадачного приложения из n задач может быть вычислена как

$$U = \frac{1}{k} \times \sum_{i=1}^{n} u_i, \tag{3}$$

где k – количество ядер в процессоре на данной платформе, каждое с производительностью P операций в единицу времени. Величина $1 - U$ задает долю процессорного времени, которая не используется данным приложением (процессор либо простаивает, либо занят вычислениями, не связанными с работой данной СРВ). Увеличение производительности процессора ведет к увеличению доли его простоя для данного программного приложения СРВ.

Рассмотрим вспомогательную характеристику задачи – ее *жесткость*:

$$H_i = \frac{T_i}{D_i}. \tag{4}$$

Если $H_i \geq 1$, то интервалы существования двух последовательных экземпляров одной и той же задачи τ_i – заданий $\tau_i^{(j)}$ и $\tau_i^{(j+1)}$ – не пересекаются. Противоположное условие $H_i < 1$ означает, что интервалы существования таких заданий могут пересекаться. Если все задачи приложения имеют одну и ту же жесткость H, то эта величина H называется жесткостью всего приложения. Часто более удобным оказывается рассматривать обратную к жесткости величину H^{-1}.

В работе [8] понятие *плотности приложения* определяется как максимальное значение его общей полезной нагрузки: $Dens = \max_P\{U(P)\}$ для всех значений P

производительности процессора, при которых данное приложение выполнимо. Очевидно, что для любого конкретного приложения найдется столь малое число ϵ, что данное приложение выполнимо при $P > \epsilon^{-1}$ и не выполнимо при $P < \epsilon$. Также очевидно, что если приложение выполнимо при значениях производительности P_1 и P_2, где $P_1 < P_2$, то оно выполнимо и для любого $P \in [P_1, P_2]$. Таким образом, существует значение плотности $Dens$, соответствующее минимальной производительности P_0, при которой данное приложение все еще выполнимо: для всех $P < P_0$ приложение не выполнимо, а для всех $P \geq P_0$ выполнимо.

Приведенное выше рассуждение приводит к экономичному алгоритму «поимки льва в пустыне» для вычисления этого значения $Dens$. Начиная с интервала $[a, b]$ для значений производительности, где $a = 0$ и $b = P_{max}$, причем P_{max} настолько велико, что выполнимость приложения при такой производительности обеспечивается, выполняется имитационное моделирование поведения приложения с производительностью процессора $P = (b-a)/2$. Следующим рассматриваемым интервалом будет либо $[a, P]$, если сеанс имитационного моделирования подтверждает выполнимость приложения, либо $[P, b]$ в противном случае. Цикл завершается на интервале $[P-\epsilon, P+\epsilon]$, где P – результирующее значение этой минимальной производительности с точностью до заданного значения $\epsilon > 0$. Плотность приложения, определяемая внешними факторами и структурными характеристиками приложения, а также выбором сочетания дисциплины планирования с протоколом доступа к разделяемым информационным ресурсам, может использоваться как объективный критерий экономичности данного сочетания при заданных характеристиках приложения.

Целью данного исследования было нахождение и анализ зависимостей между жесткостью приложения и его плотностью для различных сочетаний дисциплин планирования и протоколов доступа. Для этой цели в рамках единой программной архитектуры были параллельно разработаны два разнородных и независимых прототипа инструментального средства для оценки выполнимости приложений CPB: один на языке C# [9] объемом 1,4 KLOC, другой объемом 0,9 KLOC на языке Форт [10] в стандарте Forth 2012 [11], одновременно с набором эталонных приложений. Результаты имитационного моделирования поведения указанных эталонных приложений на этих двух инструментах различались менее чем на 0,1%, что является сильным аргументом в пользу их достоверности.

2. Проверка выполнимости

Общая схема работы обоих симуляторов для определения выполнимости приложений представлена на рис. 1. Работа начинается с инициализации, которая состоит в выборе желаемого сочетания дисциплины планирования и протокола доступа, установки режима наследования заданиями приоритетов, задания соответствующих констант, считывания файла с описанием структуры приложения и формирования соответствующих этому внутренних объектов – ресурсов и задач. Затем формируется начальный список *EventList* системных событий, состоящий из активизации всех задач в моменты системного времени, определяемые их фазовыми сдвигами относительно начала отсчета системного времени. Счетчики максимального време-

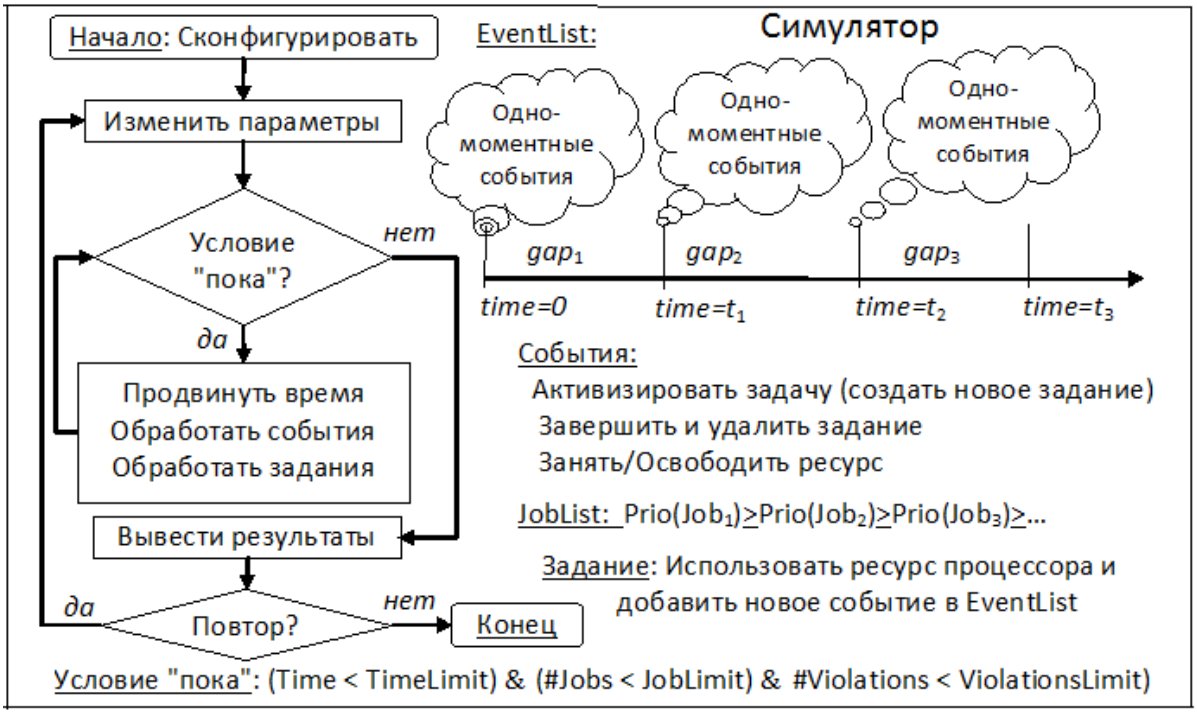

Рис. 1. Общая схема работы симулятора
Fig. 1. The overall structure of the simulator workflow

ни отклика для каждой задачи устанавливаются в нуль и все ресурсы переводятся в состояние «свободен».

В главном цикле моделирования рассматривается первая группа одномоментных событий в упорядоченном списке *EventList*, системное время симулятора устанавливается на этот момент, и по очереди обрабатываются все события из этой первой группы в соответствии с их типом.

1. Если очередное обрабатываемое событие – активизация задачи, то создается новое задание (экземпляр данной задачи), которое добавляется в список *JobList* активных заданий со своим приоритетом и запланированным моментом начала, равным текущему значению системного времени, а в список событий *EventList* добавляется новое событие – активизировать следующий экземпляр данной задачи (новое задание) в момент времени, не меньший чем текущее значение системного времени плюс период T_i данной задачи.

2. При завершении какого-либо задания обновляется время отклика соответствующей задачи как максимум из его текущего значения и полученного интервала существования данного задания. Если величина этого интервала превышает заданный предельный срок D_i данной задачи, то фиксируется нарушение ее выполнимости. Завершаемое задание удаляется из списка *JobList*.

3. В случае занятия/освобождения ресурса, если занимается уже занятый ресурс, то соответствующее задание переносится из списка *JobList* активных заданий в упорядоченный по приоритетам список заданий, ожидающих осво-

бождения данного ресурса; в противном случае данный ресурс занимается данным заданием. При освобождении ресурса, если список заданий, ожидающих его освобождения, не пуст, то первое задание из этого возвращается в список *JobList* активных заданий в соответствии со своим приоритетом и данный ресурс занимается этим заданием; в противном случае ресурс освобождается.

По завершении обработки события оно удаляется из списка *EventList*. После того, как будут обработаны все одномоментные события данной группы, рассматривается список *JobList* активных заданий (он мог измениться в результате обработки событий). Если этот список не пуст, то выбирается его первый элемент и счетчик процессорного времени, затраченного данным заданием на выполнение своей вычислительной работы, увеличивается на соответствующую величину. При этом возможно порождение нового события – завершение данного задания. После этого повторяется основной цикл симулятора. Условием завершения основного цикла (Условие «пока» на рис. 1) является либо исчерпание установленного лимита системного времени на данную сессию имитационного моделирования, либо достижение заданного числа созданных в процессе моделирования заданий, либо выявление заданного числа зарегистрированных нарушений выполнимости.

Результаты сеанса имитационного моделирования – максимальное время отклика задач, количество зафиксированных нарушений предельных сроков их выполнения, плотность приложения и другие данные – выводятся на экран. Также может быть выведен системный журнал моделирования, в котором регистрируются все возникшие системные события вместе с моментом времени их возникновения и другими сопроводительными данными. Вся эта информация может быть легко перенесена в MS Excel для удобного и наглядного представления полученных результатов и записей в системном журнале.

3. Профиль многозадачного приложения

В рассматриваемых ниже моделях программных приложений код каждой задачи представляется в виде последовательности сегментов. Каждый сегмент содержит фрагмент программного кода, замыкаемый оператором, связанным с каким-либо системным событием. Используются три типа сегментов, отличающиеся разновидностью замыкающего сегмент системного события:

- финальный сегмент – его замыкающим оператором является оператор завершения исполняемого задания;
- сегмент с запросом ресурса – его замыкающим оператором является оператор запроса доступа задания к одному из разделяемых ресурсов;
- сегмент с освобождением ресурса – его замыкающим оператором является оператор завершения одного из занятых заданием разделяемых ресурсов (оператор освобождения ресурса).

Задачи, не содержащие операторов доступа к разделяемому ресурсу, состоят из единственного (финального) сегмента.

Текстовое представление сегмента начинается спецификатором типа сегмента и завершается заданием его абсолютного веса. Спецификаторы сегментов имеют вид:

- «E_» – спецификатор финального сегмента;
- «L_» – спецификатор сегмента типа *lock* (запрос ресурса);
- «U_» – спецификатор сегмента типа *unlock* (освобождение ресурса).

Вес сегмента представляется целым числом, отражающим объем вычислений, исполняемых в рамках сегмента. Условимся задавать этот объем (т.е. оценку веса сегмента) в виде эквивалентного количества эталонных операций (эталоном такого рода может выступать, например, арифметическая операция с плавающей запятой).

Представление финального сегмента содержит два элемента: спецификатор и абсолютный вес. Например, последовательность символов «E_20» представляет финальный сегмент с абсолютным весом $w = 20$ эталонных операций.

В представлении *lock*-сегмента между спецификатором и значением абсолютного веса помещается целое число, соответствующее номеру запрашиваемого разделяемого ресурса – например, символы «L_1_10» представляют сегмент с абсолютным весом $w = 10$, завершающийся оператором запроса доступа к разделяемому ресурсу с номером 1. Аналогично символы «U_1_40» представляют *unlock*-сегмент с абсолютным весом $w = 40$, завершающийся оператором освобождения ранее занятого разделяемого ресурса с номером 1.

Для представления значений периодов задач в формальной модели приложения используется конструкция типа «T=1000», означающая, что период данной задачи равен 1000 единиц физического времени (например, миллисекунд). Значения предельных сроков D_i представляются аналогичными конструкциями типа «D=1000».

Текстовое представление модели задачи имеет вид строки, начинающейся описаниями ее параметров, за которыми следует описание последовательности сегментов ее кода. Описания отдельных параметров и сегментов разделяются пробелами. Например, для периодической задачи τ_i, представляемой строкой:

$$\text{T=1000 D=1000 L\_1\_10 U\_1\_40 E\_20}$$

ее экземпляры – задания $\tau_i^{(j)}$ $(j = 1, 2, ...)$ – порождаются каждую секунду (T=1000 миллисекунд), причем допустимая продолжительность существования каждого из них не превышает одной секунды (D=1000 миллисекунд), абсолютный вес W_i кода задачи τ_i составляет 10+40+20=70 эталонных операций. Задача содержит три сегмента. Первый сегмент с абсолютным весом 10 эталонных операций заканчивается операцией запроса информационного ресурса g_1. Второй сегмент весом 40 эталонных операций заканчивается операцией освобождения информационного ресурса g_1. Третий сегмент весом 20 эталонных операций заканчивается операцией завершения исполнения задачи.

Относительный вес C_i одного задания типа τ_i (т.е. выраженный в единицах физического времени объем ресурса процессора, требуемый для его исполнения) зависит от производительности P процессора, задаваемой числом эталонных операций за единицу физического времени (например, миллисекунду) по формуле (1). Величина u_i в формуле (2) характеризует полезную нагрузку – долю процессорного времени,

необходимого задаче τ_i. Сумма U в формуле (3) дает общую полезную нагрузку на каждое ядро процессора от данного многозадачного приложения. Таким образом, описание пяти задач $\tau_1, ..., \tau_5$ в правом столбце матрицы (5)

τ_1	T=1000 D=1000 E_150
τ_2	T=1330 D=1330 E_198
τ_3	T=1150 D=1150 E_168
τ_4	T=1520 D=1520 E_218
τ_5	T=1750 D=1750 E_260

(5)

представляет характеристики приложения из этих пяти независимых (без сегментов типа *lock/unlock*) задач. Следующий пример (6) представляет приложение с пятью задачами, разделяющими четыре ресурса $g_1, ..., g_4$:

τ_1	T=1000 D=1000 L_1_0001 U_1_0147 E_0002
τ_2	T=1150 D=1150 L_2_0001 L_3_0004 U_2_0002 U_3_0002 E_0159
τ_3	T=1330 D=1330 L_3_0001 L_4_0004 U_3_0002 U_4_0002 E_0189
τ_4	T=1520 D=1520 L_2_0001 L_4_0004 U_2_0002 U_4_0002 E_0219
τ_5	T=1750 D=1750 L_1_0001 U_1_0257 E_0002

. (6)

Данные о составе и параметрах задач, последовательности и параметрах сегментов каждой из задач, составляют *профиль* данного программного приложения. Данные о профиле приложения, дополненные указанием используемой дисциплины планирования и применяемого протокола доступа к разделяемым ресурсам, составляют *конфигурацию* исполнения приложения.

4. Результаты экспериментов

Для выполнения экспериментов по имитационному моделированию должны быть заданы конкретные конфигурации исследуемых программных приложений. Применяемые в практике промышленного программирования информативные и ценные конфигурации приложений реального времени, как правило, являются конфиденциальной информацией их владельцев и не публикуются; поэтому описываемые эксперименты проводились с использованием приложений, представляющих в большей степени методологический интерес – в частности, с приложениями, профиль которых отвечает следующим условиям:

- значения нагрузки одинаковы для всех задач: $\forall i, j\ (u_i = u_j)$ – приложения, отвечающие этим условиям, будем называть сбалансированными по нагрузкам, или просто сбалансированными;

- периоды задач распределены логарифмически (см. рис. 2, левая часть).

Приложения отвечающие этим условиям, впервые были рассмотрены в работе [1] — такие приложения будем называть приложениями с профилем Лю-Лейланда.

Профиль (7) дает пример приложения из 10 задач с профилем Лю-Лейланда:

τ_1	T=1000 D=1000 E_72		τ_6	T=1410 D=1410 E_102
τ_2	T=1070 D=1070 E_77		τ_7	T=1510 D=1510 E_109
τ_3	T=1140 D=1140 E_83		τ_8	T=1650 D=1650 E_116
τ_4	T=1230 D=1230 E_89		τ_9	T=1740 D=1740 E_124
τ_5	T=1320 D=1320 E_95		τ_{10}	T=1870 D=1870 E_132

. (7)

В левой части рис. 2 показана зависимость значений периодов T_i задач τ_i в приложении с профилем (7) от номера i задачи – в логарифмическом масштабе эта зависимость отображается прямой линией. В правой части того же рис. 2 показана зависимость плотности $Dens$ приложения от жесткости $H = T_i/D_i$ (фактически H^{-1}) ее задач для двух классических дисциплин планирования: «темпо-монотонной» (Rate Monotonic – RM) и «монотонной по срочности» (Earliest Deadline First – EDF) при исполнении на одноядерном процессоре. Значение H^{-1}, обратное величине жесткости H, изменяется на графике рис. 2 от 0,4 до 1,0 (оно предполагается одинаковым для всех 10 задач; т.е., фактически является жесткостью всего приложения).

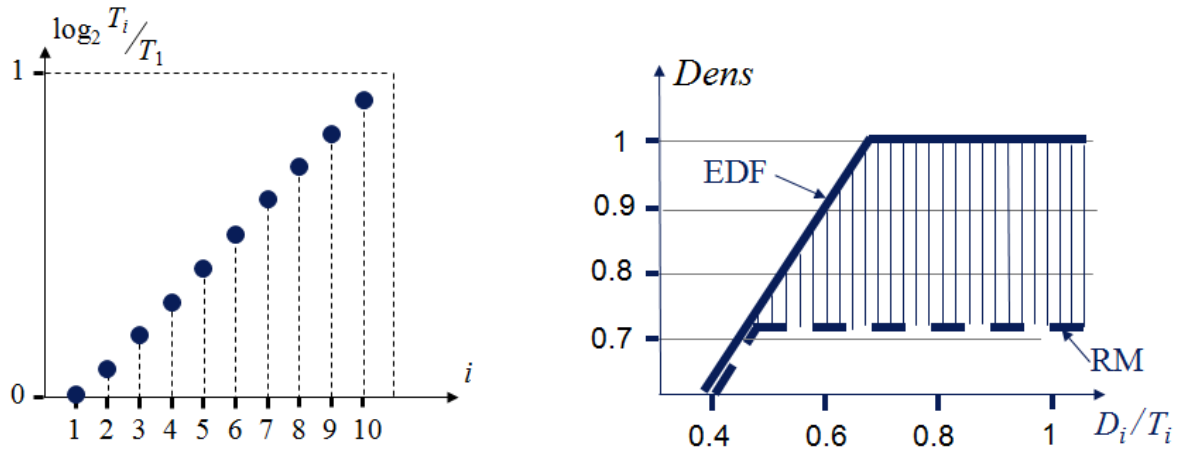

Рис. 2. Конфигурация Лю-Лейланда из 10 задач на одноядерной платформе
Fig. 2. The Liu-Layland configuration of 10 tasks on a single-core platform

Как видим, обе дисциплины планирования – EDF и RM – демонстрируют одинаковое поведение приложения, когда предельный срок существенно меньше периода (т.е. жесткость велика или, что то же самое, обратная величина к жесткости мала). Затем, с уменьшением жесткости (т.е. с увеличением ее обратной величины $H^{-1} = H_i^{-1} = D_i/T_i$) при дисциплине планирования EDF достигается максимально возможная плотность приложения, равная 1, тогда как при дисциплине RM плотность не превышает значения 0,72.

На рис. 3 представлены результаты моделирования поведения приложения из 5 независимых задач с профилем (5) с теми же двумя дисциплинами планирования при исполнении на одно- (слева) и двуядерной (справа) платформе. Профиль (5), как и профиль (7), отвечает требованиям сбалансированности нагрузок и логарифмического распределения периодов задач. Поэтому характер графиков плотности при дисциплинах планирования EDF и RM повторяет характер аналогичных графиков на рис. 2. Однако в случае использования дисциплины планирования EDF

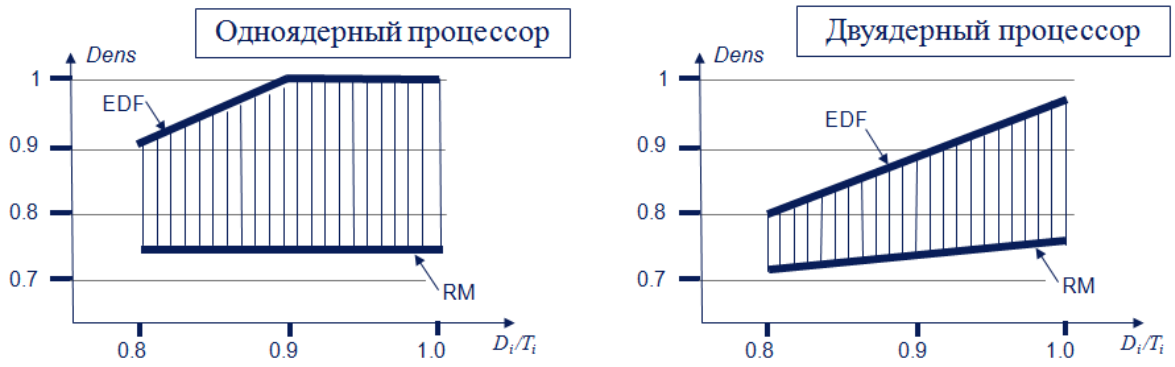

Рис. 3. Исполнение приложения из 5 независимых задач с профилем Лю-Лейланда
Fig. 3. Execution of an application of 5 independent tasks with the Liu-Layland profile

потенциальная эффективность 100% достигается лишь при значении $H^{-1} = 0,9$. Для приложений с профилем (7) такая эффективность достижима при существенно более жестких требованиях к срочности D_i (до величины $H^{-1} = 0,7$) – см. рис. 2.

Можно заметить, что при оптимальной производительности процессора P применение дисциплины планирования EDF с плотностью приложения, близкой к 1, обеспечивает 100% загрузку процессора ($Dens = 1$) на одноядерной платформе, тогда как на двуядерном процессоре не только дисциплина RM, но даже и EDF не может обеспечить такой экономичности.

На рис. 4 представлены результаты моделирования исполнения приложения из пяти задач с профилем (6) на одноядерной платформе. Здесь задачи $\tau_1, ..., \tau_5$ уже не являются независимыми – они используют 4 разделяемых информационных ресурса $g_1, ..., g_4$ с применением мьютексов для охраны соответствующих критических интервалов.

Рис. 4. Конфигурация из 5 зависимых задач на одноядерной платформе
Fig. 4. Configuration of 5 dependent tasks on a single-core platform

На графиках рис. 4 представлена зависимость плотности приложения от жесткости при дисциплинах планирования RM и EDF как с наследованием приоритетов

(что позволяет, как правило, ускорить выполнение приложения), так и без него. Стоит отметить два неожиданных факта, обнаруженных в данном случае:

- при отсутствии наследования приоритетов значение плотности для RM и EDF в точности совпадают;
- при добавлении наследования приоритетов плотности оказываются такими же, как для независимых задач для обеих дисциплин планирования.

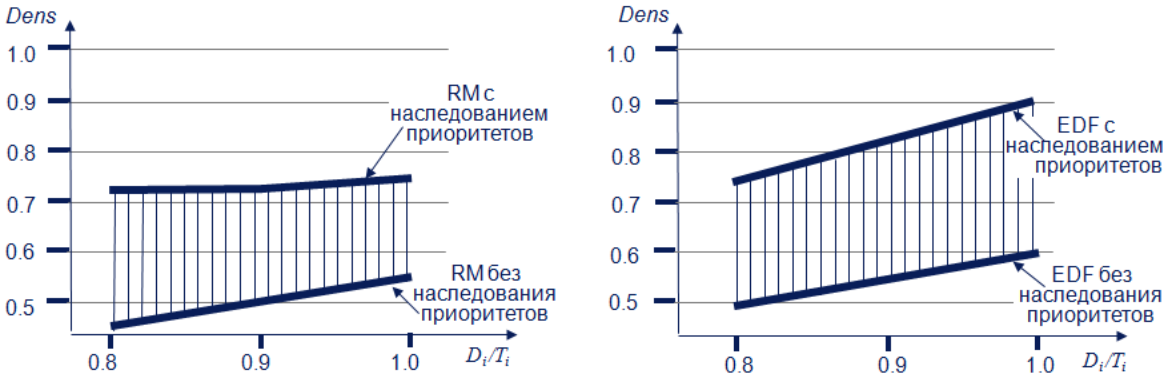

Рис. 5. Исполнение приложения из 5 зависимых задач на двуядерной платформе
Fig. 5. Execution of an application of 5 dependent tasks on a two-core platform

Когда та же конфигурация с добавленным механизмом наследования приоритетов исполняется на двуядерном процессоре, то преимущества дисциплины планирования EDF перед RM становятся менее значительными, в сравнении с тем, что наблюдалось на одноядерном процессоре (см. рис. 5).

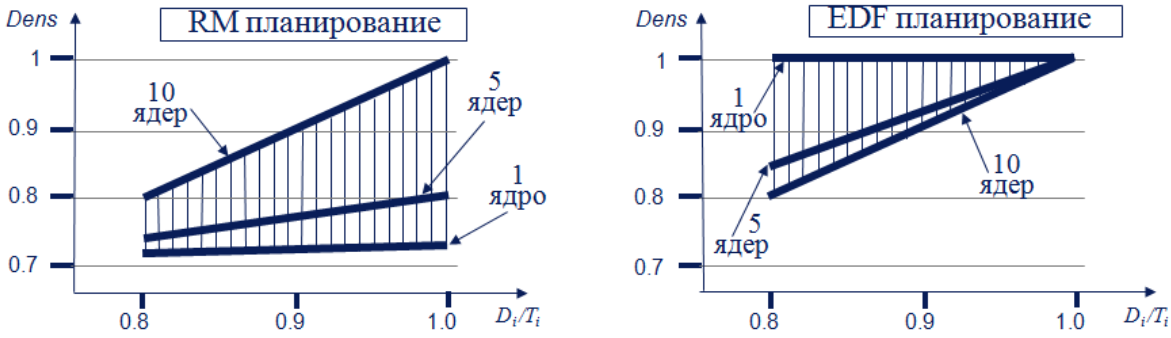

Рис. 6. Исполнение приложения из 10 независимых задач на многоядерной платформе
Fig. 6. Execution of an application of 10 independent tasks on a multi-core platform

На рис. 6 показано, как при увеличении числа ядер процессора исчезает различие между дисциплинами планирования RM и EDF для приложений с профилем (7).

На рис. 7 представлены результаты имитационного моделирования конфигурации из 4-х задач с неравномерной полезной нагрузкой для приложения из независимых задач – профиль (8) и для приложения с взаимозависимыми задачами – профиль (9).

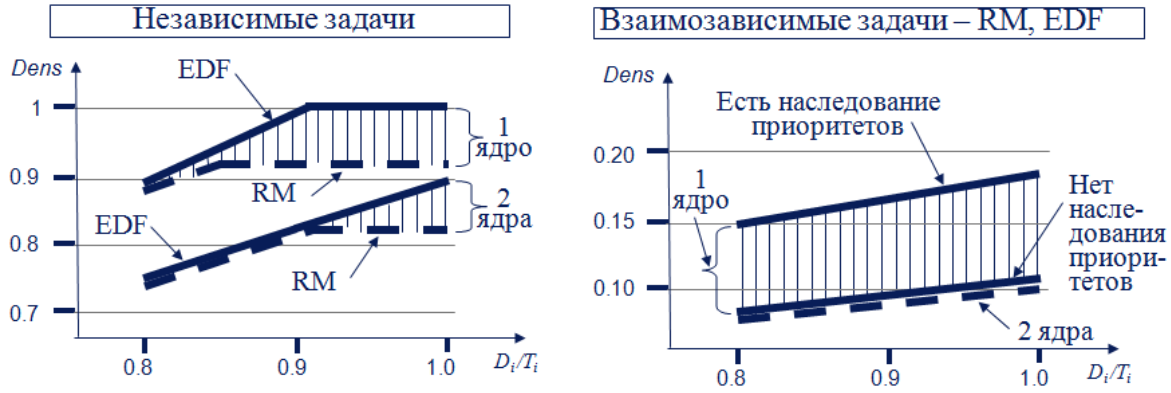

Рис. 7. Сравнение дисциплин RM и EDF для 4-х зависимых и независимых задач
Fig. 7. Scheduling modes RM vs. EDF for 4 dependent and independent tasks

Независимость задач на рис. 7 задается профилем (8):

τ_1	T=0050 D=0050 E_005
τ_2	T=0500 D=0500 E_200
τ_3	T=0550 D=0550 E_055
τ_4	T=0600 D=0600 E_240

, (8)

а их взаимозависимость в виде разделения ими 2-х информационных ресурсов (задачи τ_1 и τ_3 разделяют ресурс g_1, а задачи τ_3 и τ_4 разделяют ресурс g_2) задается профилем (9):

τ_1	T=0050 D=0050 L_1_0001 U_1_0003 E_001
τ_2	T=0500 D=0500 E_200
τ_3	T=0550 D=0550 L_1_0005 L_2_0005 U_2_0035 U_1_0005 E_005
τ_4	T=0600 D=0600 L_2_0005 U_2_0230 E_005

. (9)

Как следует из графиков на рис. 7 слева, для независимых задач с жесткостью, близкой к 1, дисциплина планирования EDF существенно более экономична, чем RM, как на одноядерной, так и на двуядерной платформе. Эта разница исчезает с уменьшением величины H^{-1}.

Анализ графиков на рис. 7 справа показывает, что:

- при наличии взаимозависимости между задачами приложения эффективность использования процессорного времени снижается в несколько раз при исполнении как на одноядерном, так и на двуядерном процессоре;
- применение дисциплины планирования EDF не приводит к повышению эффективности использования процессорного времени в сравнении с дисциплиной планирования RM;
- при исполнении на двуядерном процессоре применение механизма наследования приоритетов не приводит к повышению эффективности использования процессорного времени.

Дисциплины планирования RM и EDF оптимальны в своем классе приложений при исполнении на одноядерной платформе. Но эти дисциплины не оптимальны на многоядерных процессорах, что хорошо видно на приложениях со сдвинутой нагрузкой, т.е. при существенно неравномерном распределении общей полезной нагрузки между задачами. В приложении с профилем (10)

τ_1	T=1040 D=1040 E_840
τ_2	T=1000 D=1000 E_100
τ_3	T=1010 D=1010 E_101
τ_4	T=1020 D=1020 E_102
τ_5	T=1030 D=1030 E_103

(10)

60% от общей полезной нагрузки приходится на задачу τ_1: ее можно назвать «тяжелой», тогда как задачи $\tau_2, ..., \tau_5$ следует назвать «легкими». В таких случаях целесообразно применить дисциплину RM в следующей модификации: тяжелой задаче придается высший приоритет, а планирование легких задач следует обычной дисциплине RM (см. рис. 8).

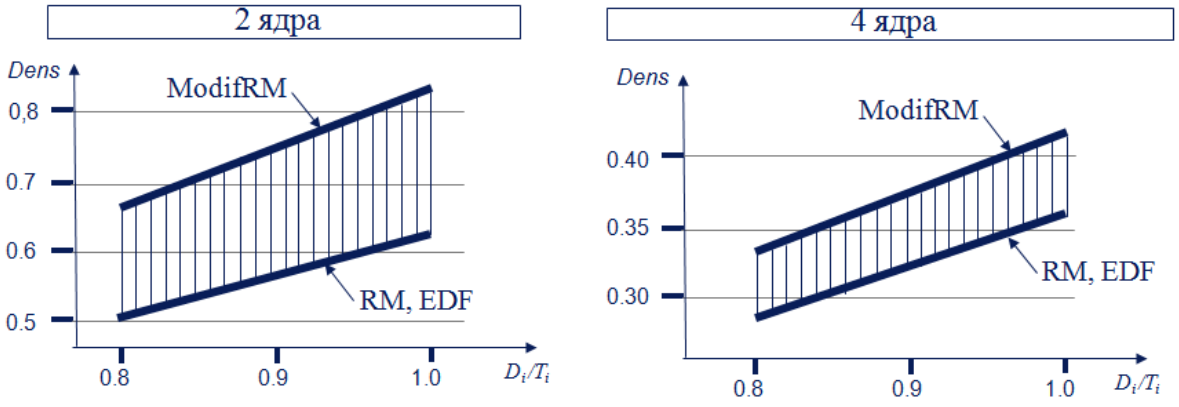

Рис. 8. Сравнение дисциплин RM и EDF для задач со сдвинутой нагрузкой
Fig. 8. Scheduling modes RM vs. EDF for tasks with shifted utility load

Графики на рис. 8 показывают преимущество этой дисциплины ModifRM (Modified Rate Monotonic) на платформе с 2 и 3 ядрами, тогда как применение дисциплин RM и EDF в этом случае одинаково неэкономично.

5. Заключение

Применение описанных методов имитационного моделирования на программном симуляторе для оценки выполнимости приложений для многоядерных платформ позволяет получить объективные данные для выбора оптимального сочетания дисциплин планирования и протоколов доступа. Точные аналитические методы для таких оценок известны только для одноядерных платформ; будучи примененными к многоядерным платформам, они дают слишком пессимистические результаты. Таким

образом, имитационное моделирование становится важным инструментом для выявления оптимальных структур приложений и платформ для многозадачных приложений реального времени. Разработанные симуляторы могут использоваться для проверки выполнимости таких приложений.

Дальнейшие исследования будут нацелены на расширение номенклатуры изучаемых дисциплин планирования, протоколов доступа и профилей приложений.

Список литературы / References

[1] Liu C., Layland J., "Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment", *Journal of the ACM*, **20**:1 (1973), 46–61.

[2] Andersson B., Baruah S., Jonsson J., "Static-Priority Scheduling on Multiprocessors", *Proc. 22nd IEEE Real-Time Systems Symposium*, 2001, 193–202.

[3] Laplante P.A., *Real-Time Systems Design and Analysis*, John Wiley & Sons, Inc., 2004.

[4] Baker T., "Multiprocessors EDF and Deadline Monotonic Schedulability Analysis", *Proc. 24th IEEE Real-Time Systems Symposium*, 2003, 120–129.

[5] Andersson B., "Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%", *Proc. 12th International Conference on Principles of Distributed Systems*, 2008, 73–88.

[6] Baranov S.N., "Real-Time Multi-Task Simulation in Forth", *Proc. 18th Conf. FRUCT Association*, 2016, 21–26.

[7] Baranov S.N., Nikiforov V.V., "Application Density and Feasibility Checking in Real-Time Systems", *System Informatics*, 2016, № 7, 1–9.

[8] Baranov S.N., Nikiforov V.V., "Density of Multi-Task Real-Time Applications", *Proc. 17th Conf. FRUCT Association*, 2015, 9–15.

[9] Никифоров В. В., *Программа ОЭКПП для оценки эффективности конфигураций программных приложений*, Свидетельство о государственной регистрации программы для ЭВМ №2016618872 от 9 августа 2016 (RU), 2016, http://www1.fips.ru/wps/portal/Registers/.
[Nikiforov V. V., *Program OEKPP for Estimating the Efficiency of Software Application Configurations*, Certificate of computer program registration number 2016618872 of 9 August 2016 (RU), 2016, http://www1.fips.ru/wps/portal/Registers/].

[10] Баранов С. Н., *Программа RTMT для имитационного моделирования исполнения многозадачных приложений*, Свидетельство о государственной регистрации программы для ЭВМ №2016613095 от 16 марта 2016 (RU), 2016, http://www1.fips.ru/wps/portal/Registers/.
[Baranov S. N., *Program RTMT for Simulation of Multi-Task Application Behavior*, Certificate of computer program registration number 2016613095 of 16 March 2016 (RU), 2016, http://www1.fips.ru/wps/portal/Registers/].

[11] Forth 200x, 2016, http://www.forth200x.org/forth200x.html.

Baranov S. N., Nikiforov V.V., "Analysis of Real-Time Applications Feasibility through Simulation", *Modeling and Analysis of Information Systems*, **23**:6 (2016), 673–687.

DOI: 10.18255/1818-1015-2016-6-673-687

Abstract. An approach to estimate feasibility of a real-time multi-task application with various combinations of the scheduling mode and the protocol of access to shared informational resources when run on a multi-core platform is described. The application structure is specified through a simple

formalized profile consisting of segments of three types and specifying access to informational resources shared among application tasks, the amount of the required computing resource being estimated for each segment. The approach is based on the notion of application density introduced by the authors, which characterizes the use of computational resource by this application and is derived from estimation of the application feasibility for various values of processor performance and the number of its cores in case of a multi-core platform. The overall structure of a simulation tool for estimation of the task response time (and therefore, application feasibility) is described, which provides more exact data compared to the known analytical methods where they are applicable. Two dissimilar implementations of this tool were developed and run on a number of benchmarks, including Liu-Layland configurations specified in the described formalism for application structure; the results in form of charts are presented along with their analysis and interpretation. The suggested approach allows to indentify an optimal combination of the scheduling mode and access protocol for the given multi-task application structure.

Keywords: simulation, real-time, application density, application feasibility

About the authors:
Sergey N. Baranov, orcid.org/0000-0001-6692-8432, Doc. Nat. Sci.,
SPIIRAS, ITMO University,
49 Kronverksky pr., St.Petersburg 197101, Russia, e-mail: SNBaranov@gmail.com

Victor V. Nikiforov, orcid.org/0009-0203-2514-7755, Doc. Nat. Sci.,
SPIIRAS, 14 liniya 39, St.Petersburg 199178, Russia,
e-mail: nik@iias.spb.su

Acknowledgments:
This work was partially financially supported by Government of the Russian Federation, Grant 074-U01.