КОМП'ЮТЕРНІ ЗАСОБИ, МЕРЕЖІ ТА СИСТЕМИ

S. Zinchenko

RESEARCH OF PERFORMANCE OF KNOWLEDGE ORIENTED INTELLECTUAL SOFT REAL-TIME SYSTEMS

This paper studies the performance of distributed soft real-time systems that use standard components with various scheduling algorithms and ways to improve them.
Key words: soft real-time, distributed systems, deadline assignment, priority assignment, scheduling.

Вивчається продуктивність розподілених м'яких систем реального часу, які використовують стандартні компоненти з різними алгоритмами планування і пропонуються шляхи їх покращення.
Ключеві слова: м'який реальний час, розподілені системи, терміни призначення, пріоритетні задачі, планування.

Изучается производительность распределенных мягких систем реального времени, которые используют стандартные компоненты с различными алгоритмами планирования и предлагаются пути их улучшения.
Ключевые слова: мягкое реальное время, распределенные системы, сроки назначения, приоритетные задания, планирование.

УДК 681.3

С.В. ЗИНЧЕНКО

ИССЛЕДОВАНИЕ ХАРАКТЕРИСТИК ЗНАНИЕ ОРИЕНТИРОВАННЫХ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ МЯГКОГО РЕАЛЬНОГО ВРЕМЕНИ

Введение. Распределенные знание ориентированные интеллектуальные системы (ЗОИС) [1, 2], например, системы научных исследований, позволяют собрать, отфильтровать и сохранить в БД сервера информацию, полученную с различных источников по определенной теме, которую затем можно использовать для организации и проведения новых исследований [3].

Основная часть. Анализ подобных систем позволяет выделить их особенности. Во-первых, они состоят из множества компонентов $S = \bigcup_{i=1}^{m} s_i$ (источники информации, потоки данных, сеть, сервер БД, ввод/вывод, экспертные системы, процессоры и т. п.). Во-вторых, решение глобальных задач состоят из нескольких этапов, которые используют разное подмножество компонентов $S_k \in S$. Такие задачи связаны с мягкими сроками выполнения, например, процедура обновления данных должна быть выполнена в течение нескольких минут с момента их получения. Поэтому ЗОИС можно отнести к распределенным системам мягкого реального времени (СРВ) [4 – 6].

Проблемы проектирования. Распределенные ЗОИС можно проектировать с "нуля" (программировать компоненты в соответствии с алгоритмами и функциями будущих приложений) или создавать из существующих хорошо проверенных и стандартных компонентов (коммерческие серверы БД, сети и т. п.). Стоимость развития и поддержание ЗОИС в первом случае будет высокой, и цикл разработки будет длительным. С точки

зрения стоимости и времени разработки целесообразным является второй подход. Его недостатком является то, что стандартные компоненты, как правило, не предназначены для выполнения задач в РВ. Например, коммерческие БД имеют неопределенное время соединительных операций при запросе, что может привести к приоритетным инверсиям [7]. Но даже если стандартные компоненты РВ доступны, такие как коммерческие операционные системы РВ (ОС РВ) [7, 8], они могут варьировать свойства задач в РВ, а их планировщики не могут обеспечить внешний контроль. Это означает, что при планировании выполнение глобальных задач в РВ необходимо согласовывать взаимодействия и координации между локальными планировщиками ($P = \bigcup_{i=1}^{m} p_i$).Учитывая это существует проблема создания распределенных ЗОИС из множества компонентов как РВ так и не РВ с учетом реальных требований/ограничений к СРВ, где гарантируются выполнения "почти всех" ограничений [9].

Концептуальный подход. Предлагается использовать результаты экспериментов, которые проясняют эти вопросы, иллюстрируют издержки в виде пропущенных сроков, и которые позволяют предложить стратегии по снижению затрат проектирования ЗОИС как мягких СРВ.

Предполагаем, что ЗОИС состоит из множества разных компонент $S = \bigcup_{i=1}^{m} s_i$, где $m \gg 1$ и $\forall s_i \in S$ использует свою собственную политику планирования. Исследуем, как локальные планировщики $p_i, i = 1,2, \dots m$ влияют на способность системы удовлетворить сроки выполнения задач, а также способы повышения ее производительности, когда некоторые из компонент $s_i \in S$, не могут выполнять расписания в режиме РВ.

Основная цель мягких СРВ – удовлетворение, сколько это возможно, сроков, и в отличие от жестких СРВ они не гарантируют соблюдения всех сроков [7, 8]. Ориентация ЗОИС на мягкое РВ объясняется такими причинами. Во-первых, задачи СРВ непростые, их выполнение требует участия нескольких уровней обработки, а это означает, что сложно получить точные оценки времени работы, которые необходимы для планирования жесткого РВ. Во-вторых, в большинстве случаев невозможно оценить верхний предел нагрузки на систему. Обе эти проблемы означают то, что планирование жесткого РВ невозможно.

Известно, что $\forall p_i \in P$ является независимым, и не существует глобального планировщика, который мог бы управлять всеми p_i системы. Более того, $\forall p_i \in P$ принимает решения, основываясь исключительно на задачах, которые ему необходимо выполнить, без согласования с другими $\forall p_j \in P$, где $i \neq j$. В ЗОИС, которая создается из существующих компонент, каждый компонент имеет свою собственную политику планирования и неспособен подчинятся/согласовывать свои решения с другим планированием. Планирование в режиме мягкого РВ наиболее подходит для ЗОИС, где тип заданий или/и их длительность, как правило, неизвестны заранее. Особенно, когда система обеспечивает прозрачность распределения (например, является частью локальных/удаленных данных), задачи, которые должны быть созданы, неизвестны до начала их выполнения, и, следовательно, выполнить предварительный их анализ невозможно.

В ЗОИС $\forall s_i \in S$ является уникальным, и задачи выполнятся на конкретной компоненте s_k, что приводит к несбалансированной нагрузке. Более того, перегруженный компонент s_k не может передать задачу на другие компоненты, т. е. $s_k \neq s_j$, где $j \neq k$.

Цель исследования. Цель исследования – определение: производительности системы в зависимости от компонент, которые являются задачами FCFS; преимуществ, которые можно получить, если компоненты предоставляют статический приоритет планирования не в РВ.

В исследовании используются простые модели глобальной задачи и распределенной системы, позволяющие понять и оценить основные компромиссы, которые необходимо использовать при проектировании СРВ [9]. Последовательность исследований такая: описывается модель ЗОИС; обсуждается случай, когда все компоненты системы используются в режиме планирования EDF (earliest deadline first); изучается вопрос снижения производительности системы, когда некоторые компоненты являются задачами FCFS; обсуждаются преимущества, которые можно получить, если не в РВ компонент выполняет статический приоритет планирования (RMS, rate monotonic scheduling).

Модель задач. Предполагаем, что имеется последовательность глобальных задач T, которые выполняются на нескольких компонентах системы. Каждая глобальная задача состоит из n подзадач $T = \bigcup_{i=1}^{n} T_i$, которые выполняются последовательно при условии, что подзадача $T_i, (i > 1)$ не может быть выполнена до завершения подзадачи T_{i-1}. $\forall T_i$ или T характеризуется такими атрибутами: $ar(T)$ – время поступления; $dl(T)$ – крайний срок завершения (время завершения); $sl(T)$ – резерв времени. Для $\forall T_i$ определяются стадии, чтобы они имели позиции в глобальной задаче. Например, если $T = \bigcup_{i=1}^{3} T_i$, то этап ($T_2$) = 2. Это означает, что подзадача T_i ранних стадий выполняется раньше, чем другая подзадача T_j, если они входят в одну и ту же глобальную задачу и стадия (T_i) < стадии (T_j).

Всегда существует проблема несвоевременных задач или политики управления перегрузкой. В ситуации, когда задача T уже пропустила свой срок и не завершила выполнение, для нее возможен только один вариант выполнения – отмена, в предположении, что все, что она делает далее бесполезно.

Модель системы. Модель распределенной ЗОИС включает компоненты, управляющие различными ресурсами, такими как БД, экспертные системы, вычислители, сеть и т. п. (рис. 1). Отметим, что сеть может быть включена в качестве одного/нескольких компонентов. Например, прямую связь между двумя сайтами можно рассматривать как один из ресурсов, а всю сеть можно считать другим ресурсом. Порядок выполнения целевой функции планируется в локальных планировщиках каждого из компонентов, которые являются независимыми и не взаимодействуют между собой. Единственное, что может влиять на планирование решения в режиме РВ – это атрибуты каждой из задач.

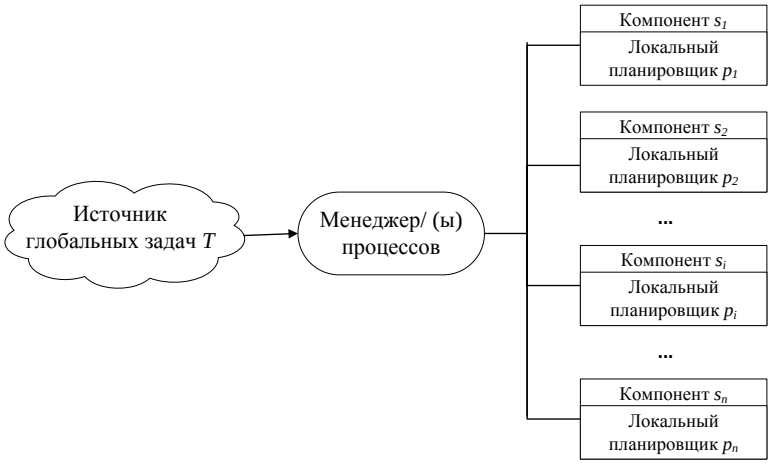

РИС. 1. Модель системы

Вновь созданные глобальные задачи сначала обрабатывает диспетчер процессов (process manager) [10]. На самом деле может быть не один, а несколько диспетчеров процессов, которые управляют одной/несколькими глобальными задачами, и при этом определяют приоритеты, начало и конец выполнения, все крайние сроки подзадач. Основной функцией диспетчера процессов является представление подзадачи соответствующего компонента для исполнения, контроль соблюдения ограничения и очередности среди подзадач глобальной задачи. Часто диспетчер процессов должен генерировать информацию для планирования выполнения подзадач до их поступления. Например, если s_i принимает задачу в соответствии с приоритетами pr_{T_i}, а s_j – задачи в зависимости от их сроков, диспетчер процессов должен убедиться, что T_i представлены компонентам в соответствии со значениями их атрибутов. Отметим, что и сам диспетчер процессов потребляет системные ресурсы, например, решая задачу установления связи между ядром диспетчера и компонентой, и это потребление можно смоделировать как выполнение дополнительных подзадач в системе. В результате можно получить модель требований к ресурсу диспетчера процессов.

Имитационная модель. Для изучения поведения системы в присутствии множества локальных алгоритмов планировщика можно воспользоваться имитационной моделью, реализованной на языке DeNet (Danish Information Network) [11]. Каждый имитационный эксперимент генерирует одну точку данных продолжительностью 10^{-6} с, за один проход создаются $> 10^5$ задач, что намного больше максимальной нагрузки на реальную систему.

Имитационная модель системы состоит из k компонент, где каждый из которых поддерживает график выполнения своих задач с помощью таких типов локальных планировщиков: с ближайшим сроком завершения (EDF), со статическим приоритетом (RMS – rate monotonic scheduling), поочередного обслужива-

ния вызовов (FCFS). Прерывание планировщика во всех случаях запрещено. Глобальные задачи однородны, и создаются как единый поток пуассоновского типа со средним интервалом времени между требованиями $1/\lambda_g$. Каждая глобальная задача состоит из m подзадач, время выполнения которых соответствуют экспоненциальному распределению со средним значением $1/\mu_s$ (с). Общее время выполнения глобальных задач, которое соответствует *m*-стадиям Эрланга [12], распределено со средним значением m/μ_s. Поэтому, скорость выполнения глобальных задач будет равна $m\lambda_g/\mu_s$. Считаем, что узлы в равной степени используются в качестве исполнительных узлов подзадач. Пространство резервного времени глобальных задач равномерно распределено в интервале [S_{min}, S_{max}].

Нормированная нагрузка на систему определяется, как отношение генерируемой скорости к общей мощности переработки системы, т. е.:

$$l = \frac{m\lambda_g}{k\mu_s}$$

Для стабильной системы $0 \le l < 1$. Использованы такие базовые настройки значений параметров: политика перегрузки управления – без прерывания; $\mu_s = 1.0$; $k = 6$; $m = 4$; $l = 0{,}5$; $[S_{min}; S_{max}] = [5, 200]$.

Планировщики задач. Предполагается, что все компоненты выполняют EDF планирование. Сначала подзадаче представляется компонент, и диспетчер процессов назначает сроки их выполнения, так что компонент не знает как их планировать. Одним из простых способов присвоения сроков подзадачам является наследование ими (из конца в конец) сроков своих глобальных задач. Эту стратегию называют стратегией максимального предельного срока (Ultimate Deadline – UD).

В работах [13, 14] показано, что UD эффективна, если глобальные задачи системы, состоят из подзадач одинаковой длины и их времена выполнения почти одинаковы. Длинные глобальные задачи страдают высокой степенью пропусков сроков. Причиной такого выполнения глобальной задачи в присутствии коротких подзадач является то, что на многочисленных этапах подзадачи, как правило, имеют более поздний предельный срок, чем конечный срок. На ранней стадии глобальная задача свои подзадачи T_i принимает с конца срока, и местные EDF планировщики будут обмануты, полагая, что T_i имеют существенный запас для выполнения. Время, должно быть зарезервировано для выполнения подзадач T_i на более поздних этапах, и поэтому, EDF планировщики дают очень низкий приоритет T_i. Это приводит к существенной задержке в выполнении T_i и, следовательно, глобальная задача будет длительной и пропустит сроки.

В работе [13] предложены методы снижения пропущенных сроков обслуживания глобальных задач. Метод равных запасов (Equal Slack – EQS), оценивает временной запас выполнения глобальной задачи и делит его поровну между оставшимися подзадачами. Например, глобальная задача поступает в момент времени 0 со сроком времени 12 и состоит из четырех подзадач, каждая из которых выполняется за 1 единицу времени. Эта глобальная задача имеет запас в 8 еди-

ниц. EQS выделит 2 его единицы для T_1 и резервирует 6 единиц от T_3 до T_1 подзадачи на более поздних стадиях к началу от срока времени. Как показано в [13], EQS приводит к существенному снижению пропусков сроков глобальной задачей. Далее предполагается, что когда подзадаче представляется компонент с EDF планированием, то ей назначается суб-срок в соответствии со стратегией EQS.

FCFS планировщики. Отметим, что при создании мягкой СРВ, можно как использовать, так и не использовать специализированные компоненты РВ, например, сети РВ или БД, планирующие запросы ввода/вывода в зависимости от сроков. Очевидно, что компоненты РВ решают проблему распределения системных конечных сроков проще, но, к сожалению, это гораздо дороже и не всегда доступно. Например, драйверы сети Ethernet и Token Ring не поддерживают трафик в РВ и они, как правило, доставляют сообщения в соответствии со стратегией FCFS.

В качестве другого примера можно назвать дисковые контроллеры, в которых используются так называемые лифтовые алгоритмы [15]. Расписание дисковых запросов блокирует дисковую операцию вместо того, чтобы запросить сроки выполнения подзадач. Обычные алгоритмы планирования РВ как EDF не выполняются эффективно для диска, поскольку они имеют длительное среднее время поиска и плохую пропускную способность [16, 17].

Практически при создании ЗОИС как мягких СРВ возникают следующие вопросы. Если нет возможности использовать компоненты РВ, то, как это влияет на производительность системы в РВ? Насколько важно, чтобы каждый компонент системы понимал сроки? Является ли компонент не РВ узким местом сроков выполнения? Можно задержку задачи не РВ компенсировать во время использования компоненты в режиме РВ? Чтобы ответить на эти вопросы и исследуется деградация распределенных мягких СРВ, когда компоненты выполняют FCFS планирование вместо EDF.

Из рис. 2 видно, что глобальная задача увеличивает пропуски с увеличением нагрузки на систему ($MD = f(l, N_{fcfs,} = fix)$). При низкой нагрузке, система имеет достаточную мощность для обработки задач, и тип локального планировщика FCFS /EDF является несущественным. По мере увеличения нагрузки система начинает пропускать больше сроков, и разница между этими двумя планировщиками становится значимой. Например, при высокой нагрузке $l = 0{,}65$, система с 6 локальными FCFS планировщиками пропускает примерно на 1,5 срока больше, чем система с 6 локальными EDF планировщиками. Отметим, что в нормальных условиях мягкая СРВ работает с малой нагрузкой и, поэтому, имеет минимальное число пропусков. Однако, система может быть перегруженной, и именно для таких режимов требуются планировщики, обеспечивающие наименьшее количество пропущенных сроков. Для оценки работы системы в таких режимах предлагается следующая математическая модель [18]:

$$MD_{N_{fcfs}=fix} = 0{,}01849 - 0{,}28126 * l + 1{,}72007 * l^2 - 4{,}31222 * l^3 + 4{,}38587 * l^4$$

Не удивительно, что все планировщики FCFS системы работают хуже в сравнении с EDF планировщиками, а переход от компонента EDF к FCFS вызывает незначительную потерю производительности. Производительность системы имеет линейную деградацию по числу компонентов $N_{\mathbf{FCFS}}$ при увеличении нагрузки на систему, что и показано на рис. 2. $MD = f(N_{fcfs}, l = fix)$. Для оценки такой потери производительности предложена такая математическая модель:

$$MD_{l=fix} = 0{,}01013 + 0{,}0026 * N_{fcfs,} - 9{,}5119E - 5 * N_{fcfs}^2.$$

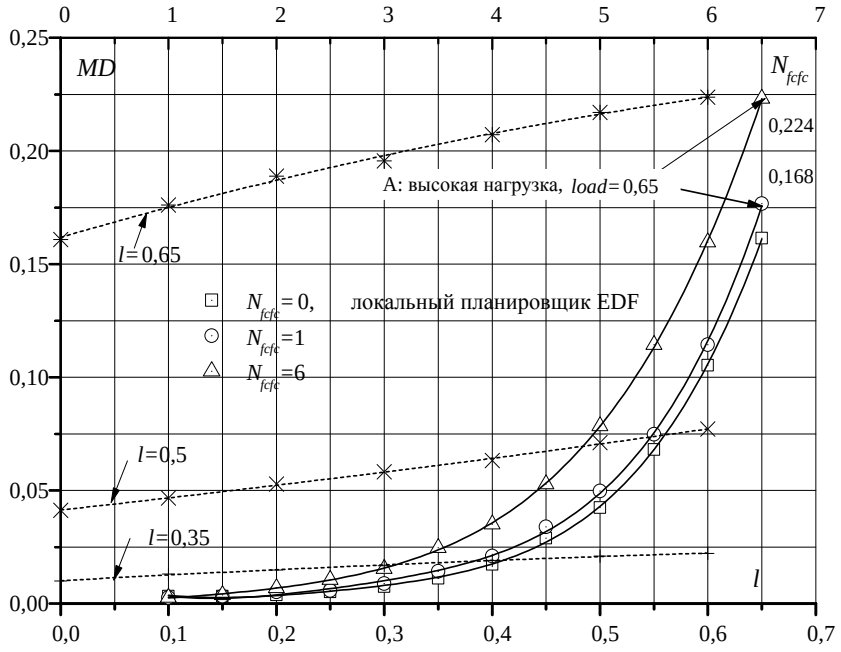

РИС. 2. Доля пропущенных сроков в зависимости от загрузки системы, числа и типа локальных планировщиков

Для изучения поведения системы при различной нагрузке были изменены условия эксперимента так, чтобы все стадии 1 подзадач исключительно выполнялись на компоненте 1, при этом компоненты 2 – 6 обрабатывали остальные подзадачи. Например, каждая глобальная задача распределенной мягкой СРВ использует сетевой компонент, но не все задачи запрашивают доступ к серверу данных. Так как каждая глобальная задача состоит с четырех подзадач, то 25 % нагрузки будет обрабатываться компонентой 1, а другие 5 компонент будут обрабатывать остальные 75 % загрузки системы.

На рис. 3 показаны два случая переключения локальных планировщиков EDF и FCFS один за другим, когда компонент 1 включается первым и последним, что соответствуют двум уровням нагрузки системы ($l = \{0{,}5; 0{,}35\}$). Наблюдаем, что при переходе с EDF на FCFS в основном загружен компонент 1, и

количество пропусков увеличивается на ~3,5 %, и при переходе на другие 5 компонентов происходит постепенное линейное увеличение MD на 0,36 % (до 13,5 %). Очевидно, что несмотря на то, что 2 – 6 компоненты совместно отвечают за 75 % общей нагрузки на систему, запуск 5 EDF планировщиков на этих компонентах менее эффективен, чем запуск 1 EDF планировщика на компоненте 1.

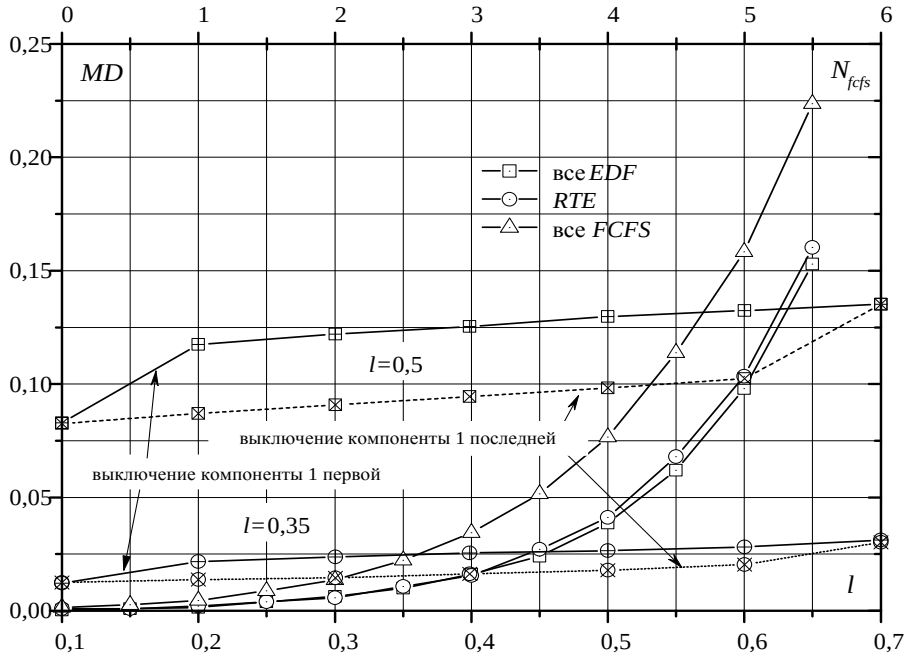

РИС. 3. Доля пропущенных сроков в зависимости от числа локальных планировщиков FCFS и типов локальных планировщиков выполнение (FCFS, EDF,RTE)

Приведенные данные позволяют сделать вывод, что потеря производительности FCFS компонента вместо EDF связана с нагрузкой этого компонента. Когда все компоненты системы загружаются однотипно и используют приблизительно одинаковое количество задач, то потери производительности почти линейны от числа компонент FCFS. Это показывает то, что эффективно использовать два FCFS компонента, если допустима незначительная потеря производительности. Если же это не так, то необходимо сосредоточить внимание на тяжело нагруженных компонентах системы. Для обеспечения более высокой производительности эти компоненты должны использовать планировщики в режиме РВ или должны быть продублированы. В некоторых случаях увеличить производительность можно путем использования компонент планирования EDF.

Статический приоритет планирования. Вышепоказано, как производительность системы снижается, когда в режиме РВ компоненты используют FCFS планирование. Стандартные не РВ компоненты бывают более эффективными, чем FCFS в случае использования статического планирования задач, например, в

POSIX UNIX и в кольцевых сетях с маркерным доступом. Рассмотрим вопрос возможного использования статических приоритетов планирования в распределенных мягких СРВ и способы возможного улучшения производительности FCFS планировщиков.

Идея состоит в эмуляции алгоритма планирования режима РВ (real time emulation – RTE) с использованием статического приоритета. В работе [19] предложены способы отображения атрибутов задач РВ (срок, время прихода и т. п.) в уровень приоритета для однопроцессорной системы. Перед выполнением подзадачи T диспетчер процессов компонента S вычисляет для T суб-срок. Затем к началу срока T вычисляется его уровень приоритета в соответствии с линейным отображением:

$$pr_{T_i} = dl(T_i) - ar(T_i) = t_{s_j},$$

где t_{sj} – настройка параметров для компонента S_j.

На рис. 4 показаны результаты оценки пропусков сроков глобальной задачей для таких случаев использования компонент: все FCFS; все EDF; все статического приоритета планирования с 4 уровнями приоритетов. Видно, что даже с 4 уровнями приоритета, статический приоритет планирования с RTE выполняет почти также как с EDF. Только при высокой нагрузке RTE незначительно превосходит EDF. Причина того, что RTE пропускает меньше сроков в том, что при высокой нагрузке, EDF, как правило, отдают предпочтение более поздним задачам. Из-за этого в режиме перегрузки RTE не может прогнозировать сроки для всех задач должным образом, по существу RTE не дает шанс закончить запоздалые задачи до их заданных сроков.

Недостаток RTE подхода проявляется в том, что его производительность чувствительна к значению ts_j и распределению допусков для задач. При равномерном распределении линейное отображение работает хорошо. В других условиях необходимы другие отображения, которые будут равномерно распределять задачи по приоритетам, для обеспечения хорошей производительности. Это приводит к усложнению диспетчера процессов. Однако, когда для ЗОИС компоненты РВ отсутствуют/недоступны, использование RTE является эффективным и экономичным способом снижения пропущенных сроков.

Выводы. В работе исследованы системы мягкого РВ, где используются обычные не РВ компоненты. Конечно, в системах жесткого РВ, никто не будет использовать компоненты не РВ. Однако, есть много приложений, таких как ЗОИС, в которых сроки не столь жесткие, что и является причиной использования доступных не РВ компонент против компонент РВ.

Показано, как производительность системы снижается, когда EDF планировщики заменяются FCFS планировщиками. В общем, при низкой нагрузке на систему, использование не РВ компонентов несущественно увеличивает пропуски. Экономически оправдано использовать стандартные компоненты не РВ, так как специализированные компоненты РВ являются дорогими и порой недоступными. В системах, которые имеют высоконагруженные компоненты важно,

чтобы они содержали планировщики РВ. В случае отсутствия планировщиков РВ, необходимо увеличить мощность компонентов системы путем дублирования компонент FCFS, а в случае статического планировщика необходимо использовать эмуляции алгоритмов РВ, устанавливающие сроки с учетом приоритетов. Результаты показывают, что последний метод выполнятся почти как реальный EDF планировщик. Для оценки доли пропущенных сроков можно воспользоваться предложенными математическими моделями.

1. *Зинченко С.В.* Онтологически управляемые информационные системы // Открытые информационные и компьютерные интегрированные технологии. – Харьков: Гос. Аэроком. Ун-т "ХАИ", 2004. – Вып. № 19. – С. 256 – 262.
2. *Зинченко С.В.* Элементы структурирования знаний: понятия, атрибуты и произвольные отношения // Открытые информационные и компьютерные интегрированные технологии. – Харьков: Гос. Аэроком. Ун-т "ХАИ", 2004. – Вып. № 23. – С. 84 – 89.
3. *Зинченко В.П., Зинченко С.В.* Архитектура и организация системы удаленного доступа к информации микроспутника // Комп'ютерні засоби, мережі та системи. –2011. – № 10. – С. 56– 67.
4. *Зінченко В.П., Ходаковський М.І., Зінченко С.В.,Татулашвілі Т.І.* Онтологічний підхід до проектування автоматизованих навчаючих систем // Комп'ютерні засоби, мережі та системи. – 2009. – № 8. – С. 94 – 101.
5. *Гераймчук М.Д., Зінченко В.П., Лапінський В.В., Зінченко С.В. та ін.* Інформаційні технології в освіті: методи та засоби. – Монографія. – К.: НТУУ "КПІ", 2009. – 89 с.
6. *Гераймчук І.М., Зінченко С.В.* Проблеми представлення знань в інформаційних технологіях. – Монографія. – К.: НТУУ «КПІ», 2010. – 240 с.
7. *Блискавицкий А.А., Кабаев С.В.* Операционные системы реального времени (обзор) // Средства и системы компьютерной автоматизации. http://www.asutp.ru.
8. *Египко В.М., Зинченко В.П.* Метод и инструментальные средства обработки данных эксперимента в реальном времени. – Киев, 1995. – 24 с. (Препр. / АН Украины ИК им. В.М. Глушкова; 95 – 20).
9. *Kao B., Garcia-Molinay H., Adelbergz B.* On Building Distributed Soft Real-Time Systems Technical Report, Stanford University, 1994. – 12 p.
10. *Олифер В.Г., Олифер Н.А.* Сетевые операционные системы. – СПб.: Питер, 2002. – 544 с.
11. *Livny M.* DeNet user 's guide. Technical report, University of Wisconsin-Madison, 1990. –30 p.
12. *Бусленко Н.П.* Моделирование сложных систем. – М.: Наука, 1978. – 356 с.
13. *Kao B., Garcia-Molina H.* Deadline assignment in a distributed soft real-time system. In Proceedings of the 13th International Conf. on Distributed Computing Systems, 1993. – P. 428 – 437.
14. *Pang H., Livny M., Carey M.J.* Transaction scheduling in multiclass real-time database systems. In Proceedings of IEEE Real-Time Systems Symposium, 1992. – P. 23 – 34.
15. *Peterson J.L., Silberschatz A.* Operating System concepts. Addison-Wesley, 1985. – 968 p.
16. *Carey M.J., Jauhari R., Livny M.* Priority in DBMS resource scheduling. In Proceedings of the 15th VLDB Conf. 1989. – P. 397 – 410.
17. *Abbott R., Garcia-Molina H.* Scheduling II O requests with deadlines: a performance evaluation. In Proceedings of IEEE Real-Time Systems Symposium, 1990. – P. 124 – 133.
18. *Зинченко В.П., Зинченко С.В., Муха И.П.* Алгоритм полиномиальной регрессии и его реализация в среде Mathcad // Нові комп'ютерні засоби, обчислювальні машини та мережі. Том 1. – К.: Ін-т кібернетики ім. В.М. Глушкова НАН України, 2001. – С. 115 – 123.
19. *Adelberg B., Garcia-Molina H., Kao B.* Emulating soft real-time scheduling using traditional operating system schedulers. In IEEE Real-Time System Symposium, 1994. – 25 p.

Получено 11.10.2012