**УДК** 004.04 **DOI** 10.15622/sp.52.7

В.В. Никифоров, С.Н. Баранов

# СТАТИЧЕСКАЯ ПРОВЕРКА КОРРЕКТНОСТИ РАЗДЕЛЕНИЯ РЕСУРСОВ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

*Никифоров В.В., Баранов С.Н.* **Статическая проверка корректности разделения ресурсов в системах реального времени.**

**Аннотация.** В ряду вопросов, возникающих в ходе разработки программных комплексов для СРВ, необходимо решать как общие для многозадачных систем вопросы обеспечения логической корректности создаваемой системы (сохранение целостности информационных ресурсов, исключения возможности взаимного блокирования задач), так и специфические для СРВ вопросы динамической корректности (своевременности исполнения задач). Решение этих вопросов в конечном счете сводится к проверке корректности размещения в теле каждой из задач синхронизирующих операторов, обеспечивающих согласованное исполнение задач. Такая проверка корректности осуществляется статически. С этой целью строятся модели, отражающие размещение синхронизирующих операторов в задачах приложения.

В настоящей статье предлагаются методы обработки таких моделей посредством построения специальных многодольных графов — графов зависимостей синхронизирующих операторов. Представляются две разновидности таких графов: а) графы связок, обеспечивающие проверку логической корректности многозадачных приложений, (корректность пересечений пар критических интервалов); и б) графы связок и критических интервалов, обеспечивающие проверку динамической корректности приложений для СРВ.

**Ключевые слова:** системы реального времени, модели многозадачных приложений, выполнимость задач, протоколы доступа, разделяемые ресурсы.

**1. Введение.** Ключевое свойство программных приложений реального времени состоит в том, что они работают в «структуре времени», определяемой ходом внешних процессов. Такое соответствие между ходом исполнения компонентов программных приложений системы реального времени (СРВ) и ходом внешних процессов обусловлено наличием более или менее жестких временных рамок для информационных обменов с внешними процессами.

Требование соответствия между структурой времени, отражающей ход внешних процессов, и структурой времени исполнения программных компонентов является одним из проявлений общего требования к архитектуре рационально построенной СРВ — требования соблюдения принципа структурного соответствия. В самом общем виде этот принцип формулируется следующим образом: для рационально построенной СРВ структура комплекса программных объектов и особенности их исполнения должны представлять собой преломленное отображение структуры и движения множества внешних объектов [1]. Данный принцип предлагается в качестве руководящего положения

при разработке архитектуры программных приложений для СРВ. Одним из главных следствий этого принципа является требование организации программного приложения СРВ в виде комплекса кооперативных задач $\tau_1$, $\tau_2$, …, $\tau_n$.

В ходе разработки программных приложений СРВ приходится искать способы разрешения противоречия между требованиями снижения объема выделяемых вычислительных ресурсов и требованиями сохранения выполнимости отдельных задач и программного приложения в целом. Разрешение этого противоречия обеспечивает увеличение эффективности использования вычислительных ресурсов при сохранении своевременности исполнения системой возлагаемых на нее функций. Исследования, посвященные разработке эффективных дисциплин планирования для СРВ на базе одиночных одноядерных процессоров, начались более сорока лет назад [2, 3]. Впоследствии они были продолжены для многопроцессорных систем и многоядерных процессоров. Исследования в этой области продолжаются по сей день [4, 5]. Особенно активно ведутся исследования в области разработки эффективных дисциплин планирования и оценки выполнимости приложений, исполняемых на многоядерных процессорах [6, 7]. На этой базе развиваются методы повышения эффективности использования вычислительных ресурсов многоядерных процессоров [8], в частности для программных приложений СРВ с нетривиальной структурой задач [9, 10, 11] и для программных приложений, ориентированных на использование в специальных прикладных областях [12]. Следует отметить, что аналогичные проблемы выполнимости исследуются как при проектировании чипов [13], так и при специфицировании программных систем с применением компонентного подхода [14], так что рассматриваемый здесь метод может применяться более широко.

Для СРВ с задачами, имеющими доступ к разделяемым глобальным информационным ресурсам, разработаны специальные протоколы обработки запросов на доступ к таким ресурсам. В случае применения таких протоколов требуется специальная статическая (на этапе проектирования программного приложения) обработка моделей, отражающих структуру межзадачных связей в многозадачном программном комплексе СРВ [15].

В ряду вопросов, возникающих в ходе разработки программных комплексов для СРВ, необходимо решать, как общие для многозадачных систем вопросы обеспечения *логической* корректности создаваемой системы (сохранение целостности информационных ресурсов и исключения возможности взаимного блокирования задач [16]), и специфические для СРВ вопросы *динамической* корректности (своевре-

менности исполнения задач [2, 17]). Решение этих вопросов сводится в конечном счете к проверке корректности размещения в теле каждой задачи *синхронизирующих операторов*, обеспечивающих согласованное исполнение задач. Такая проверка корректности осуществляется статически. С этой целью строятся модели, отражающие размещение синхронизирующих операторов в задачах приложения. В настоящей статье предлагаются методы обработки таких моделей посредством построения специальных многодольных графов — *графов зависимостей синхронизирующих операторов*. Представляются две разновидности таких графов:

— *графы связок* (пар пересекающихся критических интервалов), обеспечивающие проверку логической корректности многозадачных приложений;

— *графы связок и критических интервалов*, обеспечивающие проверку динамической корректности приложений для СРВ.

**2. Логическая корректность многозадачных программных приложений.** В ходе работы многозадачного программного комплекса составляющие его кооперативные задачи разделяют общие системные ресурсы: исполнительные ресурсы (в первую очередь — процессоры, ядра многоядерных процессоров) и информационные ресурсы (глобальные массивы данных, интерфейсные регистры периферийных устройств, элементы человеко-машинного интерфейса и т.п.). Для обеспечения согласованности исполнения, взаимодействующие задачи обмениваются данными и синхронизирующими (сигнальными) информационными сообщениями. Корректность организации взаимодействии задач состоит в обеспечении: а) *целостности* разделяемых информационных ресурсов; б) отсутствия опасности возникновения *взаимного блокирования* задач.

Стандартный подход к обеспечению целостности разделяемых информационных ресурсов сводится к использованию *мьютексов*. Для каждого из разделяемых ресурсов $g_k$ формируется программный объект мьютекс `mut_k` — синхронизирующий элемент, фиксирующий занятость ресурса $g_k$. Участок кода, в рамках которого задача $\tau_i$ имеет доступ к ресурсу $g_k$ (*критический интервал* по доступу к $g_k$), ограничивается синхронизирующими операторами `lock(mut_k)` и `unlock(mut_k)`, означающими, соответственно, захват и освобождение ресурса $g_k$.

**2.1. Маршрутные сети**. При разработке программных комплексов с разделяемыми ресурсами для проверки логической корректности межзадачных синхронизирующих связей используются разного рода

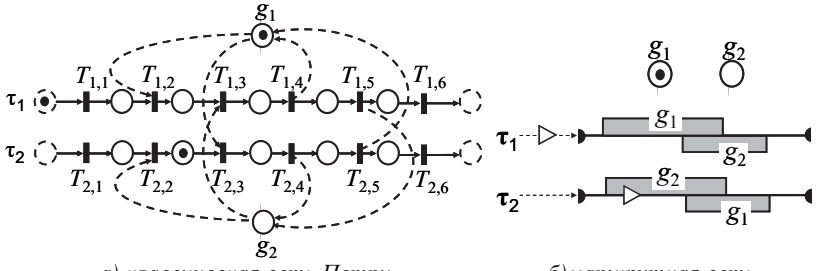
сетевые модели, в частности модели, основанные на классических сетях Петри [18]. В настоящей статье с этой целью используются маршрутные сети — подмножество раскрашенных сетей Петри, ориентированное на представление структуры межзадачных синхронизирующих связей в многозадачном программном комплексе.

Маршрутная сеть, представляющая приложение из $n$ задач $\tau_1$, $\tau_2, \ldots, \tau_n$, содержит $n$ позиций-источников, $n$ позиций-стоков, $n$ непересекающихся маршрутов; последовательность $\{T_{i,1}, T_{i,2}, \ldots\}$ переходов $i$-го маршрута, соответствующая последовательности синхронизирующих операторов в теле задачи $\tau_i$, каждый из маршрутов $\{T_{i,1}, T_{i,2}, \ldots\}$ начинается позицией-источником. Любой из переходов, содержащихся в маршрутной сети, принадлежит одному из маршрутов $\{T_{i,1}, T_{i,2}, \ldots\}$; то есть переходы, не включенные в какой-либо из маршрутов $\{T_{i,1}, T_{i,2}, \ldots\}$, в сети отсутствуют. Каждая из позиций, не принадлежащая ни одному из маршрутов $\{T_{i,1}, T_{i,2}, \ldots\}$, моделирует синхронизирующий элемент. В настоящей статье рассматриваются только синхронизирующие элементы типа мьютексов.

На рисунке 1*а* средствами сетей Петри изображена структура программного приложения из двух задач, разделяющих два ресурса $g_1$ и $g_2$. Два маршрута, моделирующие задачи $\tau_1$ и $\tau_2$, связаны с разделяемыми ресурсами $g_1$ и $g_2$ пунктирными дугами. Каждая из позиций маршрута, заключенная между двумя его переходами, соответствует сегменту кода задачи, заключенному между соседними синхронизирующими операторами. Размещение меток в позициях сети представляет текущее состояние процесса ее функционирования. Наличие метки в позиции между переходами $T_{2,2}$ и $T_{2,3}$ означает, что в текущий момент времени задача $\tau_2$ выполняет вычисления, предписанные тем сегментом ее кода, который заключен между синхронизирующими операторами, соответствующими переходам $T_{2,2}$ и $T_{2,3}$. Наличие метки в переходе-источнике маршрута $\tau_1$ означает, что задача $\tau_1$ уже активизирована, но еще не инициализирована. Наличие метки в позиции $g_1$ означает, что ресурс $g_1$ свободен. Отсутствие метки в позиции $g_2$ означает, что ресурс $g_2$ занят.

На рисунке 1*б* та же сеть Петри в том же состоянии изображена специальными средствами маршрутных сетей [19]. Прямоугольники

представляют критические интервалы по доступу к ресурсам $g_1$ и $g_2$, положение треугольников на маршрутах $\tau_1$ и $\tau_2$ указывает текущее состояние соответствующих задач.




*а) классическая сеть Петри* *б)маршрутная сеть*

Рис. 1. Приложение из двух задач с двумя разделяемыми ресурсами

Текущее состояние ресурсов отражается размещением меток в позициях $g_1$ и $g_2$. Вариант представления маршрутной сети на рисунке 1*б* более компактен и лучше обозрим в сравнении с вариантом на рисунке 1*а*. Вместе с тем содержательно эти варианты эквивалентны — по изображению специальными средствами маршрутных сетей классическое изображение сети Петри восстанавливается однозначно.

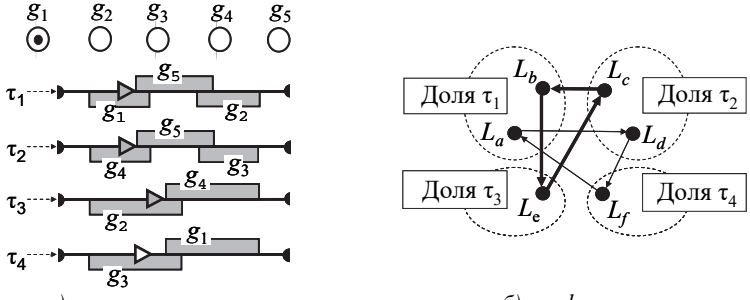
**2.2. Протоколы доступа к разделяемым ресурсам**. Взаимное блокирование задач парализует работу программного приложения либо полностью (тупик — все задачи $\tau_1$, $\tau_2$, . . . , $\tau_n$ переходят в состояние бесконечного ожидания), либо частично (клинч — часть задач переходит в состояние бесконечного ожидания, остальные задачи продолжают выполняться). При работе СРВ со структурой приложения на рисунке 1 возможно возникновение тупика. Из состояния на рисунке 1 тупик может быть достигнут выполнением следующих двух шагов.

Шаг 1. Срабатывает переход $T_{1,1}$: задача $\tau_1$ инициализируется и приступает к выполнению вычислений, соответствующих первому сегменту ее кода.

Шаг 2. Срабатывает переход $T_{1,2}$: задача $\tau_1$ захватывает ресурс $g_1$ и приступает к выполнению вычислений, соответствующих второму сегменту кода.

Дальнейшее продолжение исполнения задач $\tau_1$ и $\tau_2$ невозможно, поскольку переход $T_{1,3}$ не может сработать ввиду занятости ресурса $g_2$ задачей $\tau_2$, а переход $T_{2,3}$ не сможет сработать ввиду занятости ресурса $g_1$ задачей $\tau_1$. Задачи $\tau_1$ и $\tau_2$ попали в состояние взаимного блокирования.

На рисунке 2*а* приведено состояние системы из четырех задач с пятью разделяемыми ресурсами, в котором задачи $\tau_1$, $\tau_2$, $\tau_3$ попали в клинч, а задача $\tau_4$ может беспрепятственно завершить вычисления и далее активизироваться и завершаться произвольное число раз.




*а) маршрутная сеть* *б) граф связок*

Рис. 2. Приложение из четырех задач с пятью разделяемыми ресурсами

Стандартный подход к обеспечению корректности реализации приложений с задачами, использующими разделяемые ресурсы, опирается на оснащение синхронизирующих механизмов типа мьютексов специальными протоколами доступа к разделяемым информационным ресурсам. Протоколы различаются условиями входа в критический интервал по доступу к разделяемому ресурсу и/или действиями, выполняемыми при выполнении синхронизирующих операций [20].

Для простейшего протокола (ПП) единственное условие входа в критический интервал состоит в том, что требуемый ресурс свободен. В условиях применения ПП возможна инверсия приоритетов, ведущая к нарушению своевременности исполнения высокоприоритетных задач. Протокол наследования приоритетов (ПНП) обеспечивает устранение возможности возникновения инверсии приоритетов, но не гарантирует решения проблемы взаимного блокирования (как в случае приложения на рисунке 2).

При реализации протокола пороговых приоритетов (ППП) наряду с механизмом наследования приоритетов выполняется дополнительная проверка возможности занять запрашиваемый ресурс. С каждым ресурсом (т.е., с каждым мьютексом) связывается параметр, называемый *пороговым приоритетом* ресурса [21]. Пороговый приоритет равен высшему уровню базового (статического) приоритета задачи, которая может занять ресурс. Задаче разрешается занять ресурс, только если ее базовый приоритет выше, чем пороговые приоритеты всех ресурсов, уже занятых в настоящее время другими

задачами. Для определения значений пороговых приоритетов разделяемых ресурсов требуется выполнение статической обработки моделей приложения.

Обеспечиваемое применением ППП предотвращение возможности взаимного блокирования задач сопряжено со снижением эффективности реализации межзадачных синхронизирующих связей (из-за усложнения реализации, означающего увеличение расходов процессорного времени), а также с тем, что низкоприоритетное задание, занявшее ресурс с высоким уровнем порогового приоритета, может блокировать исполнение средне-приоритетной задачи даже в условиях, когда запрашиваемый ресурс свободен.

Более существенное снижение эффективности реализации приложения с применением ППП (до десятков процентов) связано с тем, что ППП применим только в условиях использования дисциплин планирования со статическими приоритетами задач. Применение ППП не допускает использования более эффективных дисциплин планирования с динамическими приоритетами задач.

Если структура программного приложения исключает возникновение взаимного блокирования задач, то допустимо применение ПП и ПНП, совместимых с использованием эффективных дисциплин планирования. Отсюда следует актуальность построения метода статической обработки модели синхронизирующих связей программного приложения, обеспечивающего проверку возможности возникновения взаимного блокирования задач. Для систем на мьютексах такой метод опирается на построение графа зависимостей связок критических интервалов — разновидности графа зависимостей синхронизирующих межзадачных связей.

**2.3. Граф связок**. Два критических интервала задачи $\tau$ по ресурсам $g$ и $g^*$ назовем *связанными* (образующими *связку* $L =< \tau, g, g^* >$), если они пересекаются; то есть содержат общие сегменты кода задачи. Каждая связка $L =< \tau, g, g^* >$ состоит из трех участков. На начальном (*головном*) участке связки $L =< \tau, g, g^* >$ задача $\tau$ имеет доступ к *головному ресурсу* $g$ связки. На центральном участке задача $\tau$ имеет доступ к обоим ресурсам связки — головному $g$ и *дополнительному* $g^*$. На завершающем участке один из ресурсов связки уже освобожден задачей $\tau$. Центральный участок состоит из сегментов, образующих пересечение связанных критических интервалов.

В задаче $\tau_1$ на рисунке $2a$ содержится две связки — $L_a =< \tau_1, g_1, g_5 >$ и $L_b =< \tau_1, g_5, g_2 >$. В связке $L_a$ общим участком критических интервалов по головному ресурсу $g_1$ и дополнительному

ресурсу $g_5$ является сегмент между операторами `lock(mut_5)` и `unlock(mut_1)`. В связке $L_b$ пересечением связанных критических интервалов является сегмент между операторами `lock(mut_2)` и `unlock(mut_5)`. В задаче $\tau_2$ имеются связки $L_c =< \tau_2, g_4, g_5 >$ и $L_d =< \tau_2, g_5, g_3 >$. Задача $\tau_3$ содержит одну связку $L_e =< \tau_4, g_2, g_4 >$, задача $\tau_4$ — связку $L_f =< \tau_4, g_3, g_1 >$.

Необходимым условием возникновения взаимного блокирования является наличие в программном приложении таких пар связок, для которых имеет место следующее отношение зависимости. Связка $L_x =< \tau_i, g_a, g_b >$ является *зависимой* от связки $L_y =< \tau_k, g_c, g_d >$, если $\tau_i$ и $\tau_k$ — различные задачи и $g_b \equiv g_c$. Другими словами, связка $L_x$ является зависимой от $L_y$, если $L_x$ и $L_y$ принадлежат различным задачам и головной ресурс связки $L_y$ совпадает с дополнительным ресурсом связки $L_x$.

Факт зависимости связки $L_x$ от связки $L_y$ обозначим символом '$\rightarrow$' ($L_x \rightarrow L_y$). Во введенных обозначениях для модели на рисунке 2*а* имеют место зависимости $L_a \rightarrow L_d$, $L_b \rightarrow L_e$, $L_c \rightarrow L_b$, $L_d \rightarrow L_f$, $L_e \rightarrow L_c$ и $L_f \rightarrow L_a$. Графически эти зависимости можно отразить построением многодольного ориентированного *графа зависимостей связок критических интервалов* (или просто *графа связок*): каждая связка представляется вершиной графа; наличие дуги из вершины $L_x$ в вершину $L_y$ означает, что связка $L_x$ зависит от связки $L_y$. Каждой задаче соответствует отдельная доля построенного графа.

**2.4. Междольные маршруты и контуры**. При статическом анализе свойств системы с взаимозависимыми задачами большое значение имеет выделение междольных маршрутов и контуров в графе связок. *Маршрутом* в ориентированном графе является такая последовательность дуг, что:

1) две непосредственно следующие друг за другом дуги (смежные дуги — предшествующая и последующая) имеют общую вершину, предшествующая дуга является входящей дугой общей вершины, последующая — исходящей дугой общей вершины;

2) каждая из вершин маршрута встречается в нем только один раз (маршрут не имеет самопересечений, начальная и конечная вершины не совпадают).

В многодольном графе маршрут является *междольным*, если в нем нет двух вершин, принадлежащих одной и той же доле.

*Контуром* ориентированного графа является замкнутая последовательность смежных дуг без самопересечений (первая дуга является исходящей для той же вершины, для которой последняя дуга является входящей).

В многодольном графе контур является *междольным контуром*, если в нем нет двух вершин, принадлежащих одной и той же доле.

**2.5. Необходимое и достаточное условие взаимного блокирования задач.** Анализ структуры графа зависимостей связок важен потому, что имеет место следующий факт: возможность возникновения в многозадачной системе взаимных ожиданий задач существует в том и только в том случае, если в соответствующем графе зависимостей связок критических интервалов имеются междольные контуры.

Граф зависимостей связок на рисунке 2*б* имеет два междольных контура. Каждый из них представляет достижимую разметку, при которой часть задач попала в состояние взаимного блокирования. Так, междольному контуру $L_b \to L_e \to L_c \to L_b$ (выделен на рисунке 2*б* жирными дугами) соответствует кольцо взаимных ожиданий, связывающее задачи $\tau_1$, $\tau_3$, и $\tau_2$ в состоянии системы, представленном на рисунке 2*а*. В графе связок на рисунке 2*а* имеется еще один междольный контур — контур $L_a \to L_c \to L_e \to L_a$. Этот междольный контур соответствует возможному варианту следования системных событий, приводящему к взаимному блокированию задач $\tau_1$, $\tau_2$ и $\tau_4$.

Если граф связок программного приложения не содержит междольных контуров, то это приложение может быть реализовано без риска возникновения тупиков и клинчей не только с использованием ППП, но и с использованием ПНП или ПП. При использовании ПНП значение фактора блокирования может оказаться меньшим, чем при использовании ППП. Это относится к реализации систем как на классических одноядерных, так и на многоядерных процессорах. В любом случае возможность отказа от применения ППП позволяет использовать более эффективные дисциплины планирования.

Отсюда следует, что использование графов связок при статическом анализе приложений СРВ может заметно повысить эффективность использования ресурсов процессора.

**3. Динамическая корректность программных приложений.** При разработке СРВ наряду с решением вопросов верификации, касающихся логической корректности программных приложений, необходимо решать и вопросы верификации, касающиеся их *динамической*

корректности, в частности вопросы обеспечения *выполнимости* — гарантий своевременности выполнения задач, входящих в состав программного приложения.

Наиболее точный подход к оценке выполнимости программных приложений СРВ опирается на оценку $R_i$ — максимально возможного значения времени отклика для каждой из задач $\tau_i$, входящих в состав приложения. В случае классических одноядерных систем алгоритмы оценки значений времени отклика дают абсолютно точный результат, поскольку основаны на анализе порядка функционирования системы в условиях заведомо наихудших (*критических*) сценариев системных событий. В этом случае подстановка вычисленной оценки времени отклика в неравенство $R_i \le D_i$ дает необходимое и достаточное условие выполнимости задачи $\tau_i$.

**3.1. Составляющие времени отклика задач**. Для независимых задач величина времени отклика $R_i$ равна сумме двух составляющих:

$$R_i = C_i + I_i,$$

где $C_i$ — фактор веса задачи $\tau_i$ (максимальный объем процессорного времени, используемого для однократного исполнения задачи $\tau_i$); $I_i$ — фактор приоритета задачи $\tau_i$ (максимальная продолжительность пребывания задачи $\tau_i$ в состоянии ожидания освобождения исполнительного ресурса, занятого более приоритетными задачами).

Для классических одноядерных процессоров вклад в $I_i$ каждой задачи $\tau_j$, более приоритетной чем $\tau_i$, равен $C_j \left\lceil R_i / T_j \right\rceil$, где $\lceil x \rceil$ — ближайшее сверху к $x$ целое число. В этом случае значение фактора приоритета равно сумме $I_i = \sum_{j<i} C_j \left\lceil R_i / T_j \right\rceil$. Подставляя эту сумму в выражение для $R_i$, получаем рекуррентное уравнение. Решение этого уравнения ищется методом последовательных приближений [17].

Обобщение метода оценки времени отклика на случай систем с многоядерными процессорами (например, предлагаемые в работах [22, 23]) могут давать несколько завышенные оценки значений $R_i$, то есть могут отличаться некоторой долей пессимизма. Это обусловлено тем, что для таких систем универсальный способ построения критического сценария системных событий не известен.

В случае систем с взаимозависимыми задачами, использующими разделяемые ресурсы, выражение для времени отклика $R_i$ должно быть дополнено фактором блокирования $B_i$, отражающим увеличение длины интервала существования для заданий типа $\tau_i$ за счет возможного пребывания задачи в состоянии «ожидание»: $R_i = C_i + I_i + B_i$ [24]. Величина $B_i$ соответствует максимально возможной продолжительности ожидания задачей момента освобождения требуемого задаче $\tau_i$ ресурса, занятого менее приоритетной задачей. Оценка величины $B_i$ выполняется различно для различных протоколов доступа к ресурсам [19]. При использовании ППП для систем с непересекающимися критическими интервалами подход к оценке $B_i$ определяется тезисом: «Задание может быть блокировано на протяжении исполнения не более чем одного критического интервала». Это дает основание полагать фактор $B_i$ равным максимальной продолжительности критических интервалов, принадлежащих менее приоритетным (в сравнении с $\tau_i$) задачам с ресурсами, у которых значение порогового приоритета не ниже приоритета $\tau_i$.

В работе [19] приведен подход к оценке фактора $B_i$ для систем с задачами, содержащими связки критических интервалов. В любом случае использование ППП гарантирует не более чем однократное блокирование исполнения $\tau_i$ критическим интервалом, принадлежащим менее приоритетной задаче (или связкой критических интервалов). В работе [25] показано, что в случае реализации на многоядерных процессорах в ходе оценки значения $B_i$ необходимо учитывать возможность возникновения *составного блокирования*, когда исполнение $\tau_i$ блокируется неоднократно. Использование ПНП допускает возникновение составного блокирования и при исполнении на одноядерном процессоре.

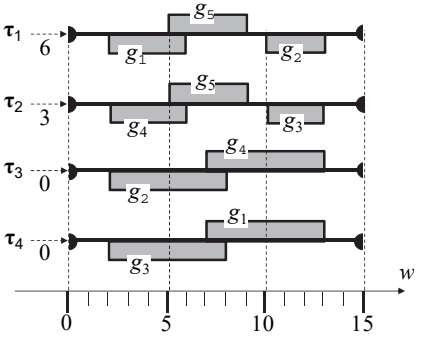
В работе [26] показано, что при использовании ПНП как на классических одноядерных, так и на многоядерных процессорах возможно возникновение цепного блокирования, когда отдельный запрос занятого ресурса блокируется со стороны нескольких активных заданий, связанных цепочкой зависимостей критических интервалов. Для оценки значения фактора блокирования в таких условиях может использоваться еще одна разновидность графов зависимостей синхронизирующих операторов.

**3.2. Граф связок и критических интервалов.** На рисунке 3*а* изображена структура системы из четырех задач, отличающаяся от системы на рисунке 2*а* тем, что**:**

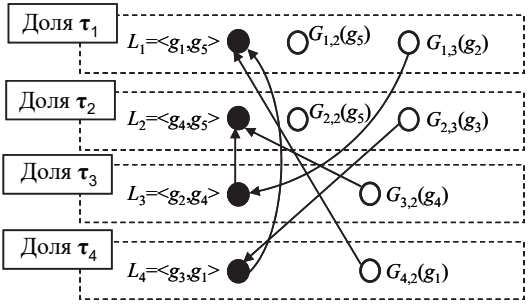
— в задаче $\tau_1$ оператор lock(mut_2) вынесен за пределы критического интервала по ресурсу $g_5$;

— в задаче $\tau_2$ оператор lock(mut_3) вынесен за пределы критического интервала по ресурсу $g_5$.

В результате такой модификации структуры системы происходит разрыв каждого из междольных контуров графа связок — система становится свободной от возможности возникновения взаимного блокирования (следовательно, допустимо ее исполнение под управлением более экономичных протоколов ПНП или ПП).




*а) маршрутная сеть*




*б) граф связок и критических интервалов*

Рис. 3. Приложение, при исполнении которого возможно цепное блокирование

Изображение маршрутной сети на рисунке 2*а* и на рисунке 3*а* служит различным целям. Рисунок 2*а* демонстрирует конкретное состояние маршрутной сети, поэтому содержит элементы разметки (представлены текущие состояния задач и ресурсов). Данные, отраженные на рисунке 3*а*, используются для рассмотрении сценария, приводящего к цепному блокированию, поэтому на нем отражены (в

масштабе) значения весов $w$ сегментов кода задач ($w$ — максимальный объем процессорного времени, используемого для исполнения сегмента при монопольном владении процессором) и значения моментов времени первой активизации каждой из $\varphi(\tau_1) = 6$, $\varphi(\tau_2) = 3$, $\varphi(\tau_3) = 0$, $\varphi(\tau_4) = 0$.

Порядок возникновения системных событий при исполнении приложения, представленного маршрутной сетью рисунка 3*а*, приведен в таблице 1.

Таблица 1. Сценарий системных событий, вызывающих цепное блокирование

| Момент времени | Событие | Изменения состояний задач и ресурсов |
|---|---|---|
| $t$=0 | Активизация задач $\tau_3$, $\tau_4$. | Задача $\tau_3$ захватывает процессор, $\tau_4$ ждет освобождения процессора. |
| $t$=2 | Захват задачей $\tau_3$ ресурса $g_2$. | Доступ к ресурсу $g_2$ блокируется задачей $\tau_3$. |
| $t$=3 | Активизация задачи $\tau_2$. | Задача $\tau_2$ захватывает процессор, $\tau_3$ и $\tau_4$ ожидают освобождения процессора. |
| $t$=5 | Захват задачей $\tau_2$ ресурса $g_4$. | Доступ к ресурсу $g_4$ блокируется задачей $\tau_2$. |
| $t$=6 | Активизация задачи $\tau_1$. | Задача $\tau_3$ захватывает процессор, задачи $\tau_2$, $\tau_3$ и $\tau_4$ ожидают освобождения процессора. |
| $t$=8 | Захват задачей $\tau_1$ ресурса $g_1$. | Доступ к ресурсу $g_1$ блокируется задачей $\tau_1$. |
| $t$=11 | Захват задачей $\tau_1$ ресурса $g_5$. | Доступ к ресурсу $g_5$ блокируется задачей $\tau_1$. |
| $t$=14 | Освобождение ресурса $g_1$ задачей $\tau_1$ | Ресурс $g_1$ свободен, задача $\tau_1$ продолжает владеть процессором. |
| $t$=17 | Освобождение ресурса $g_5$ задачей $\tau_1$. | Ресурс $g_5$ свободен, задача $\tau_1$ продолжает владеть процессором. |
| $t$=18 | Запрос задачей $\tau_1$ доступа к ресурсу $g_2$, занятому задачей $\tau_1$. | Задача $\tau_1$ переходит в состояние ожидания момента освобождения ресурса $g_2$. Процессор предоставляется задаче $\tau_3$, которая наследуя приоритет задачи $\tau_1$, завладевает процессором. |
| $t$=22 | Запрос задачей $\tau_3$ доступа к ресурсу $g_4$, занятому задачей $\tau_2$. | Задача $\tau_3$ переходит в состояние ожидания освобождения ресурса $g_4$. Процессор предоставляется задаче $\tau_2$. Возникает цепное блокирование: $\tau_1$ блокируется не только непосредственно задачей $\tau_3$, но и косвенно задачей $\tau_2$, занимающей ресурс $g_4$, нужный задаче $\tau_3$. |

Рассмотрение представленного в таблице 1 порядка возникновения системных событий показывает, что запрос задачей $\tau_1$ ресурса $g_2$ приводит к блокированию $\tau_1$ сначала непосредственно задачей $\tau_3$, владеющей требуемым ресурсом, а затем (косвенно, в момент времени $t = 22$) — задачей $\tau_2$, владеющей ресурсом, который требуется не самому $\tau_1$, а блокирующему его $\tau_3$. Таким образом, критические интервалы задач $\tau_3$ и $\tau_2$ образуют два звена цепи, блокирующей исполнение задачи $\tau_1$.

Возникает вопрос, как много активных задач может быть вовлечено в цепочки блокирования. Ответ на этот вопрос может дать анализ графа связок и критических интервалов. Ориентированный многодольный граф этого типа отличается тем, что каждая из его долей, соответствующая конкретной задаче $\tau_i$, состоит из вершин, соответствующих:

— критическим интервалам, входящим в состав имеющихся в коде $\tau_i$ связок;

— свободным критическим интервалам (имеющимся в коде $\tau_i$ критическим интервалам, не входящим в состав связок).

Дуги графа связок и критических интервалов строятся по следующим правилам.

*Правило 1.* Две вершины $L_a$ и $L_b$ (соответствующие связкам $L_a$ и $L_b$), соединяются дугой, ведущей из $L_a$ в $L_b$, в том случае, если $L_a$ и $L_b$ принадлежат разным долям графа и головной ресурс связки $L_b$ совпадает с дополнительным ресурсом $L_a$.

*Правило 2.* Вершина $G(g)$ соединяется дугой, ведущей из нее в вершину $L$, в том случае, если $L$ и $G(g)$ принадлежат разным долям графа и головной ресурс $L$ совпадает с ресурсом $g$.

*Правило 3.* Вершина $L$ соединяется дугой, ведущей из нее в вершину $G(g)$, если $L$ и $G(g)$ принадлежат разным долям графа и ресурс $g$ совпадает с дополнительным ресурсом связки $L$.

*Правило 4.* Вершина $G(g_x)$, принадлежащая *i*-ой доле графа, соединяется дугой, ведущей из нее в вершину $G(g_y)$, принадлежащую *j*-ой доле графа в том случае, если $i \neq j$ и $g_x \equiv g_y$.

На рисунке 3*б* приведен фрагмент графа связок и критических интервалов для программного приложения с конфигурацией на рисунке 3*а*

для дисциплины планирования с фиксированными приоритетами задач (приоритеты снижаются с ростом значения индекса задачи). Нагрузим каждую дугу графа связок и критических интервалов параметром $W$, который будем называть весом дуги. Вес дуги, ведущей в вершину $G(g)$, равен объему вычислений, требуемых для выполнения критического интервала, соответствующего вершине $G(g)$. Вес дуги, ведущей в вершину $L$, равен объему вычислений, требуемых для выполнения головного критического интервала связки, соответствующей вершине $L$.

Для того чтобы выяснить, какие варианты цепного блокирования возможны на входе в *k*-ый критический интервал задачи $\tau_i$, следует для вершины $G$, соответствующей этому критическому интервалу в графе связок и критических интервалов, построить блокирующие маршруты — междольные маршруты, не выходящие за рамки нижней (по отношению к $\tau_i$) части графа (т.е., не выходящий за рамки части графа, соответствующей задачам с меньшим, чем у $\tau_i$, значением базового приоритета). Вес маршрута равен сумме весов составляющих его дуг. Цепное блокирование возможно в том случае, если блокирующий маршрут содержит более одной дуги.

Для вершины $G_{1,3}$ графа на рисунке 2 имеются два блокирующих маршрута — маршрут $(G_{1,3}, L_3)$ и маршрут $(G_{1,3}, L_3, L_2)$. Маршрут $(G_{1,3}, L_3)$ содержит дугу с весом $W(G_{1,3}, L_3) = 6$. Маршрут $(G_{1,3}, L_3, L_2)$ содержит две дуги с суммарным весом $W(G_1, L_3) + W(L_3, L_2) = 10$. Следовательно, на входе в критический интервал по ресурсу $g_2$ для задачи $\tau_1$ возможно цепное блокирование с максимальной продолжительностью до 10 единиц времени.

Заметим, что междольный маршрут $(G_{2,3}, L_4, L_1)$ не является блокирующим маршрутом, так как выходит за рамки части графа, соответствующего задачам с меньшим, чем у $\tau_2$, приоритетом.

В случае, если для задачи $\tau_i$ возможно составное блокирование, значение фактора блокирования для $\tau_i$ можно представлять оценкой $B_i$, равной сумме весов блокирующих маршрутов, исходящих из соответствующих вершин *i*-ой доли графа связок и критических интервалов. При этом оценка $B_i$ может оказаться пессимистической, так как в разных учитываемых маршрутах возможны повторения блокирующих задач. В таком случае для получения точной оценки фактора блокирования следует исключить подобные повторы.

**4. Заключение**. При разработке программных приложений СРВ наряду с решением вопросов верификации, касающихся корректности отдельных задач, необходимо решать и вопросы, касающиеся логической и динамической корректности системы в целом. К числу вопросов логической корректности относится необходимость проверки гарантий невозможности возникновения таких некорректных ситуаций, когда задачи, взаимосвязанные операциями приема сигнальных сообщений, попадают в состояния бесконечного ожидания (тупики и клинчи). Статическая проверка таких гарантий может быть выполнена путем построения и анализа графа связок, относящегося к одной из разновидностей многодольных ориентированных графов зависимостей синхронизирующих операторов.

К числу вопросов динамической корректности СРВ относится необходимость обеспечения эффективного использования вычислительных ресурсов при сохранении гарантий своевременности выполнения задач, входящих в состав программных приложений. При этом необходимо учитывать возможную продолжительность блокирования высокоприоритетных задач ожиданием освобождения информационных ресурсов, занятых низкоприоритетными задачами. В случае систем, допускающих цепное блокирование задач, для оценки продолжительности блокирования целесообразно построение и использование многодольного ориентированного графа связок и критических интервалов, относящегося к другой разновидности многодольных ориентированных графов зависимостей синхронизирующих операторов.

## Литература


1. *Давиденко К.Я.* Технология программирования АСУТП. Проектирование систем реального времени, параллельных и распределенных приложений // М.: Энергоатомиздат. 1985. 183 с.
2. *Liu C., Layland J*. Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment // Journal of the ACM. 1973. vol. 20. no. 1. pp. 46–61.
3. *Сорокин С.В*. Системы реального времени: операционные системы // Современные технологии автоматизации. 1997. №2. С. 22-31.
4. *Buttazo G.C., Bertogna M., Yao G*. Limited Preemptive Scheduling for Real-Time Systems // IEEE Transactions on Industrial Informatics. 2013. vol. 9. no 1. pp. 3–15.
5. *Грюнталь А.И*. Планирование систем с асинхронным стартом // Информационные технологии и вычислительные системы. 2012. № 1. С. 32–51.
6. *Sun Y., Lipari G., Guan N., Yi W*. Improving the response time analysis of global fixed-priority multiprocessor scheduling // Proc. of 20 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2014. pp. 1–9.
7. *Guan N. et al*. Bounding Carry-in Interference to Improve Fixed-Priority Global Multiprocessor Scheduling Analysis // Proc. of 21 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2015. pp. 11–20.
8. *Andersson B*. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38% // Proceedings of the 12th International Conference on Principles of Distributed Systems. 2008. pp. 73–88.

9. *Baruah S., Bonifaci V., Marchetti-Spaccamela A*. The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks // Proc. of 27 Euromicro Conference on Real-Time Systems (ECRTS). 2015. pp. 222–231.
10. *Guan N., Chuancai G., Stigge M., Deng Q*. Approximate Response Time Analysis of Real-Time Task Graphs // Proc. of 35 IEEE Real-Time Systems Symposium (RTSS). 2014. pp. 304–313.
11. *Докучаев А.Н*. Особенности диспетчеризации сверхлегких задач в мультипроцессорных вычислительных системах реального времени // Информационные технологи. 2012. №2. С. 14–18.
12. *Колесов Н.В., Скородумов Ю.М., Толмачева М.В.* Алгоритм независимого назначения иерархических заданий на процессоры в системе реального времени // Вестник компьютерных и информационных технологий. 2013. № 6. С.28–33.
13. *Pomeranz I.* Design-for-Testability for Functional Broadside Tests under Primary Input Constraints // ACM Transactions on Design Automation of Electronic Systems (TODAES). 2016. vol. 21. Issue 2. Article No. 35.
14. *Bujtor F., Sorokin L., Vogler W*. Testing Preorders for dMTS: Deadlock-and the New Deadlock-/DivergenceTesting // ACM Transactions on Embedded Computing Systems (TECS). 2016. vol. 16. Issue 2. Article No. 41.
15. *Данилов М.В., Никифоров В.В*. Статическая обработка спецификаций программных систем реального времени // Программные продукты и системы. 2000. №4. С. 13–19.
16. *Dijkstra E.W*. Hierarchical Ordering of Sequential Processes // Acta Informatica. 1971. vol. 1. no. 2. pp. 115–138.
17. *Liu J.W.S*. Real-Time Systems // NJ: Prentice Hall. 2000. 590 p.
18. *Котов В.Е*. Сети Петри // М.: Физматгиз. 1984. 160 с.
19. *Никифоров В.В., Шкиртиль В.И*. Маршрутные сети — графический формализм представления структуры программных приложений реального времени // Труды СПИИРАН. 2010. Вып. 14. С. 5–28.
20. *Audsley N.C*. Resource Control for Hard Real-Time Systems: A Review // Real-Time Systems Research Group. Department of Computer Science. University of York. UK. 1991.
21. *Sha L., Rajkumar R., Lehoczky J.P*. Priority Inheritance Protocols: An Approach to Real-Time Synchronization // IEEE Trans. on Computers. 1990. vol. 39. no 9. pp. 1175–1185.
22. *Andersson B., Jonsson J*. Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or Not to Partition // Proceedings of the 7th International Conference on Real-Time Systems and Applications. 2000. pp. 337–346.
23. *Baker T.P., Cirinei M., Bertogna M*. EDZL scheduling analysis // Real-Time Systems. 2008. vol. 40. no. 3. pp. 264–289.
24. *Докучаев А.Н*. К оценке эффективности диспетчеризации мульти-процессорных систем реального времени с учетом влияния длительных блокировок // Программная инженерия. 2012. №9. С.2–7.
25. *Никифоров В.В., Шкиртиль В.И*. Составное блокирование взаимосвязанных задач в системах на многоядерных процессорах // Известия высших учебных заведений. Приборостроение. 2012. № 2. С. 25–31.
26. *Nikiforov V.V., Baranov S.N*. Multi-Partite Graphs and Verification of Software Applications for Real-Time Systems // Cybernetics and Information Technologies. 2016. vol. 16. no. 2. pp. 85–96.

**Никифоров Виктор Викентьевич** – д-р техн. наук, профессор, главный научный сотрудник лаборатории информационно-вычислительных систем и технологий программирования, Федеральное государственное бюджетное учреждение науки Санкт-Петербургского института информатики и автоматизации Российской академии наук (СПИИРАН). Область научных интересов: системы реального времени, встроенные системы, операционные системы. Число научных публикаций – 120. nik@iias.spb.su; 14-я линия В.О., д. 39, Санкт-Петербург, 199178; р.т.: +7(812)3280887.

**Баранов Сергей Николаевич** – д-р физ.-мат. наук, профессор, главный научный сотрудник лаборатории информационно-вычислительных систем и технологий программирования, Федеральное государственное бюджетное учреждение науки Санкт-Петербургского института информатики и автоматизации Российской академии наук (СПИИРАН), профессор, международная научная лаборатория Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики (ИТМО). Область научных интересов: технология программирования, формальные методы. Число научных публикаций – 120. snbaranov@gmail.com; 14-я линия В.О., д. 39, Санкт-Петербург, 199178; р.т.: +7(812)328-0887.

**DOI** 10.15622/sp.52.7

V.V. NIKIFOROV, S.N. BARANOV

# STATIC VERIFICATION OF TASK ACCESS TO SHARED RESOURCES IN REAL-TIME SYSTEMS

*Nikiforov V.V., Baranov S.N.* **Static Verification of Task Access to Shared Resources in Real-Time Systems.**

**Abstract.** Among issues which arise when developing software complexes for real-time systems (RTS) one should resolve common multi-task system issues of ensuring *logical* correctness of the system being created (preserving the integrity of informational resources, eliminating the possibility of mutual task blocking), as well as issues of ensuring *dynamic* correctness, specific for RTS (feasibility of the application tasks). In the long run, resolving these issues is reduced to checking the correctness of how *synchronizing operators* which ensure consistent execution of application tasks are scattered in the task bodies. In order to perform such checking statically, models which represent the scattering of synchronizing operators in application tasks are constructed.

In this paper several methods of processing such models are proposed which are based on constructing special multi-partite graphs — *graphs of synchronizing operator dependencies*. Two varieties of such graphs are resented: a) *graphs of bundles*, which ensure verification of logical correctness of multi-task applications (correctness of intersections of critical interval pairs); and b) *graphs of bundles and critical intervals*, which ensure verification of dynamic correctness of RTS applications.

**Keywords:** real-time systems, multi-task application models, task feasibility, access protocols, shared resources.

**Nikiforov Victor Vikentievich** – Ph.D., Dr. Sci., professor, chief researcher of computing and information systems and programming technologies laboratory, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: real-time systems, embedded systems, operating systems. The number of publications – 120. nik@iias.spb.su; 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone: +7(812)3280887

**Baranov Sergey Nikolaevich** – Ph.D., Dr. Sci., professor, chief researcher of computing and information systems and programming technologies laboratory, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS), professor, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University). Research interests: software engineering, formal methods in software development. The number of publications – 120. snbaranov@gmail.com; 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone: +7(812)328-0887.

## References

1. Davidenko K.Ya. *Tehnologija programmirovanija ASUTP. Proektirovanie sistem real'nogo vremeni, parallel'nyh i raspredelennyh prilozhenij* [A Technology for Programming Industrial Control Systems. Designing Real-Time Systems, Parallel and Distributed Applications]. Moscow: Energoatomizdat. 1985. 183 p. (In Russ.).
2. Liu C., Layland J. Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment. *Journal of the ACM*. 1973. vol. 20. no. 1. pp. 46–61.
3. Sorokin S.V. [Real-Time Systems: Operating Systems]. *Sovremennye tehnologii avtomatizatsii – Modern Automation Technologies*. 1997. vol. 2. pp. 22–31. (In Russ.).
4. Buttazo G.C., Bertogna M., Yao G. Limited Preemptive Scheduling for Real-Time Systems. *IEEE Transactions on Industrial Informatics*. 2013. vol. 9. no 1. pp. 3–15.
5. Grüntal A.I. [Scheduling of Systems with an Asynchronous Start]. *Informatsionnye tehnologii i vychislitelnye sistemy – Informational Technologies and Computing Systems*. 2012. vol. 1. pp. 32–51. (In Russ.).

6. Sun Y., Lipari G., Guan N., Yi W. Improving the response time analysis of global fixed-priority multiprocessor scheduling. Proc. of 20 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2014. pp. 1–9.
7. Guan N. et al. Bounding Carry-in Interference to Improve Fixed-Priority Global Multiprocessor Scheduling Analysis. Proc. of 21 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2015. pp. 11–20.
8. Andersson B. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. Proceedings of the 12th International Conference on Principles of Distributed Systems. 2008. pp. 73–88.
9. Baruah S., Bonifaci V., Marchetti-Spaccamela A. The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks. Proc. of 27 Euromicro Conference on Real-Time Systems (ECRTS). 2015. pp. 222–231.
10. Guan N., Chuancai G., Stigge M., Deng Q. Approximate Response Time Analysis of Real-Time Task Graphs. Proc. of 35 IEEE Real-Time Systems Symposium (RTSS). 2014. pp. 304–313.
11. Dokuchaev A.N. [Special Features of Scheduling of Super-Light Tasks in Multi-Processor Real-Time Computing Systems]. *Informatsionnye tehnologii – Informational Technologies*. 2012. vol. 2. pp. 14–18. (In Russ.).
12. Kolesov N.V., Skorodumov Yu.M.,Tolmacheva M.V. [An Algorithm of Independent Assigning of Hierarchical Tasks to Processors in a Real-Time System]. *Vestnik kompyuternyh i informatsionnyh tehnologiy – Herald of Computer and Informational Technologies*. 2013. vol. 6. pp. 28–33. (In Russ.).
13. Pomeranz I. Design-for-Testability for Functional Broadside Tests under Primary Input Constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*. 2016. vol. 21. no. 2. Article No. 35.
14. Bujtor F., Sorokin L., Vogler W. Testing Preorders for dMTS: Deadlock-and the New Deadlock-/DivergenceTesting. *ACM Transactions on Embedded Computing Systems (TECS)*. 2016. vol. 16. Issue 2. Article No. 41.
15. Danilov M.V., Nikiforov V.V. [Static Processing of Real-Time Software System Specifications]. *Programmnye produkty i sistemy – Software Products and Systems*. 2000. vol. 4. pp. 13–19. (In Russ.).
16. Dijkstra E.W. Hierarchical Ordering of Sequential Processes. *Acta Informatica*. 1971. vol. 1. no. 2. pp. 115–138.
17. Liu J.W.S. Real-Time Systems. NJ: Prentice Hall. 2000. 590 p.
18. Kotov V.Ye. *Seti Petri* [Petri Nets]. Moscow: Fizmatgiz. 1984. 160 p. (In Russ.).
19. Nikiforov V.V., Shkirtil V.I. [Route Nets – Graphical Form for Structural Representation of Real-Time Software Applications]. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2010. vol. 14. pp. 5–28. (In Russ.).
20. *Audsley N.C.* Resource Control for Hard Real-Time Systems: A Review. Real-Time Systems Research Group. Department of Computer Science. University of York. UK. 1991.
21. Sha L., Rajkumar R., Lehoczky J.P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. on Computers*. 1990. vol. 39. no 9. pp. 1175–1185.
22. Andersson B., Jonsson J. Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or Not to Partition. Proceedings of the 7th International Conference on Real-Time Systems and Applications. 2000. pp. 337–346.
23. Baker T.P., Cirinei M., Bertogna M. EDZL scheduling analysis. *Real-Time Systems*. 2008. vol. 40. no. 3. pp. 264–289.
24. Dokuchaev A.N. To [Estimation of Scheduling Efficiency of Multi-Processor Real-Time Systems with Impact of Long-Time Blocking Taken into Account]. *Programmnaya inzheneriya – Software Engineering*. 2012. vol. 9. pp. 2–7. (In Russ.).
25. Nikiforov V.V., Shkirtil V.I. Compound Blocking of Interdependent Tasks in Systems on Multicore Processors. *Izvestiya Vyschih Uchebnyh Zavedeniy. Priborostroenie – Journal of Instrument Engineering*. 2012. vol. 2. pp. 25–31. (In Russ.).
26. Nikiforov V.V., Baranov S.N. Multi-Partite Graphs and Verification of Software Applications for Real-Time Systems. *Cybernetics and Information Technologies*. 2016. vol. 16. no. 2. pp. 85–96.