УДК 004.04
doi:10.15217/issn1684-8853.2017.4.59

АЛГОРИТМЫ ПРОВЕРКИ ПРИМЕНИМОСТИ ПРОТОКОЛОВ ДОСТУПА К РЕСУРСАМ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

В. В. Никифоров[a], *доктор техн. наук, профессор*
С. А. Подкорытов[a], *PhD in Computer Science, младший научный сотрудник*
[a]*Санкт-Петербургский институт информатики и автоматизации РАН, Санкт-Петербург, РФ*

Введение: *разработка многозадачных приложений требует организации разделения доступа отдельных задач к общим ресурсам. Для этого принято использовать синхронизирующие элементы типа мьютексов. Использование мьютексов может приводить к взаимному блокированию задач, для предотвращения которого применяются специальные протоколы доступа к ресурсам, усложняющие выполнение операций над мьютексами (запрос/освобождение ресурса) за счет введения дополнительных условий и (или) действий. Для приложений, работающих в реальном времени, соответствующее увеличение времени отклика может оказаться существенным. Ранее было предложено решение проблемы возникновения взаимного блокирования на основе статической обработки моделей программных приложений реального времени, представляемых средствами графического формализма типа маршрутных сетей. Это решение опирается на построение специального многодольного ориентированного графа — графа зависимостей связок критических интервалов.* ***Цель исследования:*** *разработать алгоритмы, реализующие предложенную ранее обработку, и дать оценку сложности их исполнения.* ***Результаты:*** *разработаны алгоритмы, позволяющие анализировать программные продукты на возможность возникновения в них взаимного блокирования задач. Анализ проводится в три этапа. На первом этапе строится граф связок критических интервалов. После этого в построенном графе выделяются междольные контуры. Наконец, обнаруженные на предыдущем этапе междольные контуры проверяются на дизъюнктность. Оценка сложности показала, что время построения графа связок линейно относительно суммы числа связок и их зависимостей. Сложность построения перечня междольных контуров оценивается как O((n + e)(c + 1)), где c — число контуров в графе связок; n — число вершин; e — число ребер. Сложность проверки графа связок на дизъюнктность междольных контуров линейно зависит от суммы длин всех этих контуров.* ***Практическая значимость:*** *разработанные алгоритмы позволяют на раннем этапе разработки принимать решения о перестройке структуры приложения для устранения возможности взаимной блокировки задач и о выборе оптимального с точки зрения производительности протокола доступа.*

Ключевые слова *— системы реального времени, модели многозадачных приложений, протоколы доступа, разделяемые ресурсы.*

Введение

Программное приложение для систем реального времени, как правило, строится в виде комплекса кооперативных (действующих для достижения общих стоящих перед системой целей) задач τ_1, τ_2, ..., τ_n [1]. Для обеспечения анализа особенностей структуры многозадачных приложений реального времени строятся модели, отражающие размещение синхронизирующих операторов в задачах приложения, строятся методы обработки таких моделей, позволяющие проверить возможность возникновения аномальных ситуаций в ходе исполнения приложения. Примером аномальных ситуаций является взаимное блокирование задач, когда две или более задач оказываются связанными замкнутой цепочкой ожиданий поступления синхронизирующих посылок [2].

Известны подходы к построению таких моделей и методов их обработки, направленные на предотвращение возможностей возникновения взаимного блокирования задач [3]. Подход, представленный в работах [4, 5], опирается на представление структуры многозадачного программного приложения средствами маршрутных сетей [6] и обработку таких моделей с помощью специальных многодольных графов — графов связок критических интервалов. При этом для проверки возможности возникновения взаимного блокирования предлагается выполнять проверку наличия в графе связок междольных контуров. Однако в указанных работах не представлены алгоритмы:

— построения перечня связок критических интервалов;

— построения графа связок, соответствующего конкретному экземпляру маршрутной сети;

— построения перечня имеющихся в графе связок междольных контуров;

— проверки наличия в графе связок пересекающихся междольных контуров.

В настоящей статье приведены такие алгоритмы, выполнена оценка их сложности.

Протоколы доступа к разделяемым ресурсам

Большинство реальных программных приложений реального времени содержит взаимозависимые задачи, для которых не исключается возможность попадания в состояние ожидания сигнальных сообщений от других задач. Одной из разновидностей причин возникновения взаимной

зависимости задач является необходимость обеспечения корректного доступа различных задач к общим глобальным информационным ресурсам, что, в свою очередь, должно обеспечивать сохранение их целостности. Общепринятым методом обеспечения целостности разделяемых информационных ресурсов является использование мьютексов [2]. Для каждого из разделяемых информационных ресурсов g_1, g_2, ... формируются мьютексы m_1, m_2, ... — синхронизирующие интерфейсные элементы, контролирующие доступ к этим ресурсам. Имеющиеся в коде задачи критические интервалы по доступу к ресурсу g начинаются операторами lock(m) — операторами запроса на вход в критический интервал по доступу к этому ресурсу. Каждый критический интервал по доступу к ресурсу g завершается оператором unlock(m) — оператором освобождения ресурса g. Перечень условий предоставления запрашиваемого ресурса при обращении к оператору lock(m) зависит от применяемого протокола доступа [7]. При применении простейшего протокола доступа (Primitive Protocol — PP) единственное условие предоставления запрашиваемого ресурса состоит в том, что в текущий момент ресурс свободен. Если ресурс занят, то задание, запрашивающее разрешения на доступ к нему, переводится в состояние ожидания момента его освобождения.

Средствами маршрутных сетей представлена структура приложения, состоящего из двух задач τ_1 и τ_2 (рис. 1), каждая из которых содержит пересекающиеся критические интервалы по доступу к ресурсам g_1 и g_2: в задаче τ_1 критические интервалы являются сцепленными, в задаче τ_2 — вложенными. Два критических интервала задачи τ по ресурсам g и g^* назовем связанными (образующими связку $L = \langle\tau, g, g^*\rangle$), если они пересекаются, т. е. содержат общие сегменты кода задачи. На начальном (головном) участке связки $L = \langle\tau, g, g^*\rangle$ задача τ имеет доступ к головному ресурсу g связки. На центральном участке задача τ имеет доступ к обоим ресурсам связки — головному g и дополнительному g^*. На завершающем участке один из ресурсов связки уже освобожден задачей τ.

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

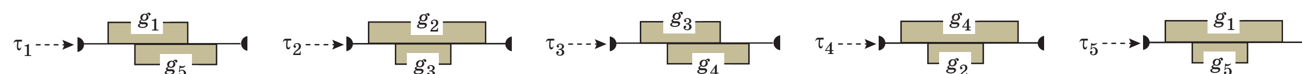
■ ***Fig. 1.*** An application with two tasks and two shared resources

Факт наличия связок критических интервалов является необходимым условием возможности возникновения взаимного блокирования задач. Однако сам по себе этот факт не означает, что взаимное блокирование действительно возможно. Так, в приложении со структурой рис. 1, несмотря на наличие связок критических интервалов, взаимное блокирование невозможно. А для приложения со структурой рис. 2 возможно попадание в состояние взаимного блокирования задач τ_2, τ_3, τ_4.

В одном из способов предотвращения взаимного блокирования применяется протокол пороговых приоритетов (Priority Ceiling Protocol — PCP) [3]. Отличие PCP от PP состоит в том, что в дополнение к проверке, свободен ли запрашиваемый ресурс, выполняется дополнительная проверка текущего состояния других разделяемых ресурсов. За гарантию предотвращения взаимного блокирования путем применения PCP приходится платить снижением эффективности исполнения приложения (оцениваемой, например, путем определения значения «плотности приложения» [8]). Плотность приложения может снижаться, в частности из-за возможности составного [9] и цепного [10] блокирования задач, а также из-за того, что некоторые протоколы не допускают использования высокоэффективных дисциплин планирования с динамическим переназначением приоритетов задач. В этой связи важно иметь способ проверки структуры приложения на возможность возникновения взаимного блокирования. Такая возможность предоставляется путем построения специального многодольного ориентированного графа — графа связок критических интервалов (или просто графа связок) [4].

Граф связок

Граф связок является многодольным графом, отражающим отношения зависимости связок. Связка $L_x = \langle\tau_i, g_a, g_i\rangle$ является зависимой от связки $L_y = \langle\tau_k, g_i, g_d\rangle$, если τ_i и τ_k — различные задачи и $g_b = g_c$. 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.



■ ***Рис. 2.*** Приложение, допускающее возникновение взаимного блокирования задач

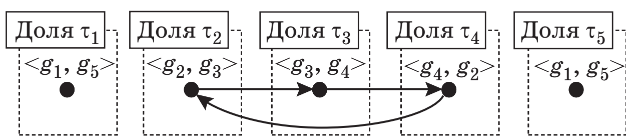
■ ***Fig. 2.*** An application where deadlocks are possible

Граф связок для приложения со структурой рис. 2 изображен на рис. 3.

Поскольку зависимые связки не могут принадлежать одной и той же задаче, граф связок является многодольным графом (по одной доле на задачу). Имеющийся в графе рис. 3 контур является междольным контуром в том смысле, что все его вершины принадлежат разным долям графа. Именно наличие такого контура означает возможность возникновения взаимного блокирования задач. Для наличия возможности возникновения в ходе работы приложения взаимного блокирования необходимо и достаточно, чтобы в соответствующем графе связок присутствовал междольный контур.

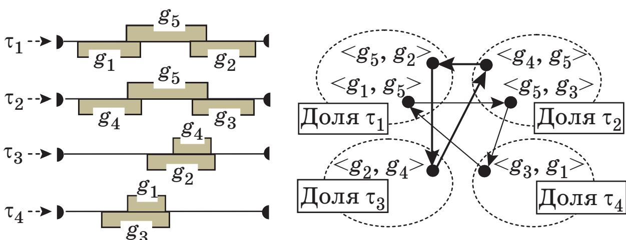
В приложении со структурой рис. 1 нет зависимых связок, соответствующий граф связок не содержит дуг (и, следовательно, междольных контуров). При работе такого приложения может применяться PP, допускающий использование эффективных дисциплин планирования. В случае программного приложения рис. 2 применение PP могло бы привести к взаимному блокированию — здесь необходимо применять протокол, специально ориентированный на предотвращение взаимного блокирования, например, PCP.

В работе [11] представлен протокол междольных контуров (Interparty Contours Protocol — ICP), который совмещает достоинства PP и PCP, допускает использование дисциплин планирования с динамически назначаемыми приоритетами заданий и вместе с тем предотвращает взаимное блокирование. Применение этого протокола допустимо при произвольном числе междольных контуров в соответствующем графе связок, например, в случае приложения со структурой рис. 4.



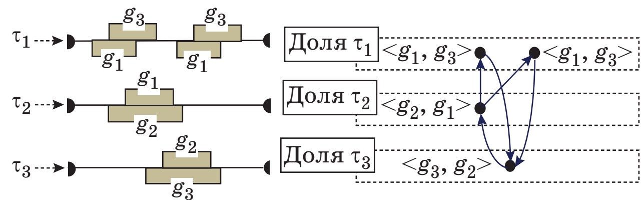
■ ***Рис. 3.*** Граф связок для приложения со структурой рис. 2

■ ***Fig. 3.*** Graph of bundles corresponding to the application from fig. 2



■ ***Рис. 4.*** Граф связок с двумя непересекающимися междольными контурами

■ ***Fig. 4.*** Graph of bundles with two disjoint circuits



■ ***Рис. 5.*** Граф связок с пересекающимися междольными контурами

■ ***Fig. 5.*** Graph of bundles with intersecting circuits

Единственное ограничение на применение ICP состоит в том, что междольные контуры графа связок не должны пересекаться. О том, что возможность таких пересечений в общем случае существует, свидетельствует рис. 5.

Существующая версия ICP непригодна для применения в приложениях, которым соответствует граф связок с пересекающимися междольными контурами. В этих условиях для приложения на рис. 5 потребуется применение PCP и, следовательно, использование менее эффективных дисциплин планирования со статическими приоритетами. Отсюда следует актуальность разработки следующих алгоритмов:

— построения графа связок по данным о структуре приложения, представляемой, например, средствами маршрутных сетей;

— составления перечня междольных контуров, имеющихся в графе связок;

— проверки наличия в графе связок пересекающихся междольных контуров.

Также актуальна оценка сложности перечисленных алгоритмов.

Алгоритм построения графа связок

Построение графа связок предлагается выполнять в два этапа. На первом этапе по данным о структуре приложения составляется список связок. На втором этапе по составленному списку строится граф связок.

Алгоритм построения перечня вершин графа связок

Для каждой задачи необходимо построить список связок, соответствующих пересечению ее критических интервалов. Для этого необходимо организовать перебор сегментов каждой задачи в цикле. При переборе сегментов необходимо хранить набор занятых ресурсов. Оптимальной структурой данных для этого является хеш-таблица, поскольку она обеспечивает константное время добавления и удаления элементов. Тело цикла, перебирающего сегменты, будет выглядеть следующим образом.

Если сегмент занимает ресурс g:

1) построить список пар вида $<g_i, g>$, где g_i — имена занятых ресурсов, хранимых во вспомогательной хеш-таблице;

2) добавить эти пары к списку связок данной задачи;

3) добавить имя ресурса g в хеш-таблицу.

Для сегмента, освобождающего ресурс g, достаточно просто удалить имя ресурса g из хеш-таблицы.

Поскольку операции добавления и удаления элементов занимают константное время, самой трудоемкой частью вышеприведенного алгоритма является составление списка пар $<g_i, g>$. В худшем случае (все ресурсы последовательно занимаются, а затем освобождаются) на n-м шаге приходится добавлять n пар. Таким образом, максимальное время работы $O(n^2)$. Однако на практике количество одновременно пересекающихся интервалов ограничено сверху, а в таком случае время исполнения $O(n)$.

Алгоритм построения ребер графа связок

В результате выполнения представленного алгоритма сформирован состав долей графа связок и для каждой доли — состав принадлежащих ей вершин. Если в какой-либо задаче есть однотипные связки (такие связки, у которых совпадают и головные, и дополнительные ресурсы), то доля графа связок, представляющая задачу τ_i, включает соответствующее число однотипных вершин.

Для построения всех ребер, исходящих из вершины $L = <\tau, g, g'>$, следует использовать вспомогательную структуру данных, содержащую все связки, для которых ресурс g является головным. В качестве такой структуры целесообразно использовать «словарь» (map), позволяющий хранить пары (*ключ*, *значение*). Словари, использующие в основе хеш-таблицы, обеспечивают доступ к хранимым значениям по значению хеш-функции ключа за константное время. Заполнение такого словаря потребует линейного времени от количества связок.

После этого добавление ребер в граф может быть осуществлено следующим образом:

1) для каждой вершины $<g_i, g_j>$ получим из словаря список всех вершин вида $<g_j, g>$, для которых ресурс g_j является головным;

2) для каждой вершины вида $<g_j, g>$ добавим в граф соответствующее ребро, если связки $<g_i, g_j>$ и $<g_j, g>$ соответствуют разным задачам.

Таким образом, граф связок может быть построен за линейное относительно суммы количества связок и количества добавляемых ребер время.

Алгоритм построения перечня контуров в ориентированном графе

Алгоритм построения перечня междольных контуров может быть основан на базе алгоритма построения перечня всех контуров в ориентированном графе. В этом разделе приводится описание такого алгоритма, предложенного Джонсоном в [12]. Данный алгоритм основывается на алгоритмах, предложенных в работах [13] и [14], однако превосходит их по скорости исполнения. Алгоритм Джонсона является лучшим на данный момент с точки зрения скорости исполнения [15].

Алгоритм Джонсона

Ниже приведены некоторые определения из общей теории графов, которые используются в описании алгоритма построения перечня контуров.

Граф $G' = (U, W)$ называется *подграфом* графа $G = (E, V)$, если $U \subset E$, $W \subset V$. Говорят, что подграф G' *порожден* множеством вершин W, если U содержит всевозможные ребра из V, соединяющие вершины из W. Подграф G' называется *сильным*, если для любых вершин $u, v \in W$ существуют маршруты из вершины u в вершину v и обратно. Подграф G' называется *максимальным*, если любой подграф G'', содержащий в себе G', не является сильным.

Алгоритмы построения максимально сильного подграфа за линейное время описаны в работах [16, 17].

Пусть вершины графа пронумерованы в произвольном порядке $L_1 < L_2 < ... < L_n$. Для каждой вершины, начиная с L_1, рассматривается максимальный сильный подграф, порожденный данной и последующей вершинами.

Алгоритм конструирует маршруты для каждого максимально сильного подграфа, начиная с вершины с наименьшим номером. Для хранения текущего маршрута используется стек. Чтобы избежать повторного обхода, посещенные вершины помечаются как заблокированные. Посещенная вершина остается заблокированной до тех пор, пока все смежные с ней вершины остаются заблокированными.

Рассмотрим для примера участок некоторого графа: две вершины L, L' и ребро из L в L'. Пусть в момент посещения вершины L вершина L' уже заблокирована. В этом случае вершина L останется блокированной до тех пор, пока не будет разблокирована вершина L'.

Чтобы обеспечить своевременную разблокировку, для каждой вершины хранится вспомогательный список вершин, которые должны быть разблокированы вместе с ней.

Обход графа организован следующим образом. Вершина L помещается в пустой стек и по-

мечается как блокированная. Для вершины, находящейся на вершине стека, рассматриваются смежные к ней вершины:

1) если начальная вершина L является одной из смежных, то текущее состояние стека и стартовая вершина добавляются к списку найденных контуров;

2) если смежная вершина не является заблокированной, то она помещается в стек и помечается как заблокированная, после чего начинают рассматривать смежные с ней вершины;

3) если контур не был обнаружен, то текущая вершина снимается со стека и добавляется в списки, соответствующие всем смежным с ней вершинам.

Необходимо отметить, что блокировка не снимается с вершин при снятии их со стека. Это гарантирует, что вершина не будет посещена повторно до того, как будет обнаружен контур или завершен обход всего подграфа.

При обнаружении контура блокировка снимается с текущей вершины, а сама вершина — со стека. При снятии блокировки с каждой вершины снимается также блокировка и со всех вершин из соответствующего списка.

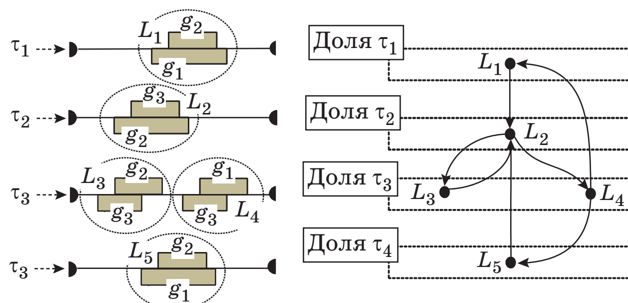
Когда в стеке заканчиваются вершины, строится следующий максимальный сильный подграф, порожденный вершинами $\{L_2, L_3, ..., L_n\}$, и процесс обхода повторяется.

Время работы между нахождением двух контуров оценивается как $O(n + e)$, где n — количество вершин, а e — количество ребер. Общее время работы алгоритма оценивается как $O((n + e)\times(c + 1))$, где c — количество контуров в графе.

Пример работы алгоритма построения перечня контуров

Ниже приведен пример работы алгоритма построения перечня контуров для графа, изображенного на рис. 6.

Пусть вершины графа связок упорядочены следующим образом: $L_1 < L_2 < L_3 < L_4 < L_5$.



■ ***Рис. 6.*** Граф связок для примера приложения из четырех задач с тремя ресурсами

■ ***Fig. 6.*** Graph of bundles for an application with four tasks and three shared resources

Изначально алгоритм рассматривает максимальный подграф, порожденный связками L_1 и следующими за ней. Такой подграф содержит все связки. Ниже приведена последовательность шагов, иллюстрирующая обработку алгоритмом данного подграфа.

1. Связка L_1 помещается на вершину стека и блокируется.
2. Единственная смежная связке L_1 связка L_2 помещается на вершину стека и блокируется.
3. Первая смежная связке L_2 связка L_3 помещается на вершину стека и блокируется.
4. Единственная смежная связке L_3 связка L_2 уже заблокирована, поэтому связка L_3 снимается со стека и добавляется в список, соответствующий связке L_2.
5. Вторая смежная связке L_2 связка L_4 помещается на вершину стека и блокируется.
6. Первая смежная связке L_4 связка L_1 является начальной, потому текущее состояние стека (L_1, L_2, L_4) и стартовая связка L_1 добавляются к списку контуров.
7. Вторая смежная связке L_4 связка L_5 помещается на вершину стека и блокируется.
8. Единственная смежная связке L_5 связка L_2 уже заблокирована, поэтому связка L_5 снимается со стека и добавляется в список, соответствующий связке L_2.
9. Связка L_5 снимается со стека, но остается заблокированной.
10. Связка L_4 снимается со стека и разблокируется.
11. Связка L_2 снимается со стека и разблокируется.
12. Разблокируются связки L_5 и L_3 из вспомогательного списка, соответствующего связке L_2.
13. Снимается со стека последняя содержащаяся в нем связка L_1.

Состояние стека, списка заблокированных вершин, а также вспомогательных списков для каждой вершины при обработке алгоритмом данного подграфа приведено в таблице. Звездочкой выделено состояние стека в момент обнаружения контура.

Далее алгоритм рассматривает максимальный подграф, порожденный связками L_2 и последующими. Такой подграф содержит связки L_2, L_3, L_4, L_5. Последовательность шагов, иллюстрирующая обработку этого подграфа, следующая.

1. На вершину пустого стека помещается связка L_2.
2. Первая смежная связке L_2 связка L_3 помещается на вершину стека и блокируется.
3. Единственная смежная связке L_3 связка L_2 является начальной, потому текущее состояние стека (L_2, L_3) и стартовая связка L_2 добавляются к списку контуров.
4. Связка L_3 снимается со стека и разблокируется.

■ Состояние стека и списка заблокированных вершин

■ Stack contents and corresponding blocked vertices list

Стек	Блок	Вспомогательные списки при				
		L_1	L_2	L_3	L_4	L_5
L_1	L_1	—	—	—	—	—
L_1, L_2	L_1, L_2	—	—	—	—	—
L_1, L_2, L_3	L_1, L_2, L_3	—	—	—	—	—
L_1, L_2	L_1, L_2, L_3	—	L_3	—	—	—
L_1, L_2, L_4*	L_1, L_2, L_3, L_4	—	L_3	—	—	—
L_1, L_2, L_4, L_5	L_1, L_2, L_3, L_4, L_5	—	L_3	—	—	—
L_1, L_2, L_4	L_1, L_2, L_3, L_4, L_5	—	L_3, L_5	—	—	—
L_1, L_2	L_1, L_2, L_3, L_5	—	L_3, L_5	—	—	—
L_1	L_1	—	—	—	—	—

5. Вторая смежная связке L_2 связка L_4 помещается на вершину стека и блокируется.

6. Единственная в данном подграфе смежная связке L_4 связка L_5 помещается на вершину стека и блокируется.

7. Единственная смежная связке L_5 связка L_2 является начальной, потому текущее состояние стека (L_2, L_4, L_5) и стартовая связка L_2 добавляются к списку контуров.

8. Связка L_5 снимается со стека и разблокируется.

9. Связка L_4 снимается со стека и разблокируется.

10. Последняя в стеке связка L_2 снимается со стека и разблокируется.

Последующие подграфы, рассматриваемые алгоритмом, тривиальны (содержат только одну вершину) и не содержат контуров.

Алгоритмы построения перечня и проверки наличия междольных контуров в графе связок

Алгоритм Джонсона, описанный в предыдущем разделе, может быть модифицирован для построения списка междольных контуров. Оригинальный алгоритм блокирует посещаемые вершины, чтобы избежать повторного обхода одних и тех же участков графа. Модифицированный алгоритм, помимо блокировки вершин, блокирует также и соответствующие доли.

При помещении очередной связки на вершину стека соответствующая доля блокируется. Связка не может быть помещена на вершину стека, если заблокирована она или соответствующая ей доля. При снятии вершины со стека соответствующая доля разблокируется, хотя отдельные вершины, относящиеся к данной доле, могут оставаться заблокированными.

Блокировка и разблокировка долей может быть организована за константное время. Поскольку на этапе построения графа связок для каждой связки сохранялось название соответствующей задачи, то проверка, относится ли данная связка к заблокированной доле, тоже занимает константное время.

Заметим, что блокировка долей в дополнение к блокировке отдельных вершин хоть и может уменьшить количество перебираемых алгоритмом маршрутов, существуют случаи, в которых подобный подход не уменьшает общего количества шагов. Примером может служить граф, целиком состоящий из одного или нескольких непересекающихся междольных контуров, подобный приведенному на рис. 4.

Таким образом, модифицированный алгоритм, строящий список междольных контуров, работает за то же время, что и оригинальный алгоритм, строящий список всех контуров.

Алгоритм проверки наличия пересекающихся контуров принимает на вход набор контуров, где каждый контур представлен в виде списка вершин. Поиск общих вершин можно осуществить за один обход всех контуров. Выделим хеш-таблицу для хранения уже посещенных вершин. Для каждой вершины, встреченной при обходе контура, проверим ее наличие в таблице посещенных вершин. Если вершина не принадлежит ни к одному из уже посещенных контуров, то добавим ее в таблицу и перейдем к обработке следующей вершины. Встретив же уже посещенную вершину, алгоритм останавливается, возвращая соответствующий результат. Если же при обходе всех контуров не было обнаружено ни одной вершины, встречающейся более одного раза, то и пересечений между контурами не существует.

Поскольку добавление и поиск элемента в хеш-таблице занимает константное время, общее вре-

мя работы алгоритма поиска пересекающихся контуров равно $O(n)$, где n — сумма длин всех контуров.

Заключение

Представленные алгоритмы позволяют выполнять анализ структуры программных приложений систем реального времени на возможность применения тех или иных протоколов доступа к разделяемым ресурсам. В общем случае анализ выполняется в три этапа.

На первом этапе строится многодольный граф связок критических интервалов по доступу задач к разделяемым ресурсам. Число долей в этом графе равно числу задач в программном приложении. Сложность построения графа связок линейно зависит от суммы числа вершин и числа ребер графа связок.

Второй этап состоит в построении перечня междольных контуров графа связок. Сложность построения перечня междольных контуров оценивается как $O((n + e)(c + 1))$.

Факт отсутствия междольных контуров в графе связок означает, что взаимное блокирование задач невозможно и, следовательно, при реализации приложения возможно применение любого протокола доступа к разделяемым ресурсам, начиная с простейшего протокола, допускающего использование эффективных процедур планирования с динамически переопределяемыми приоритетами задач.

При обнаружении в графе связок междольных контуров возможны следующие варианты реализации приложения:

— применение протоколов пороговых приоритетов, защищающих приложение от возникновения взаимного блокирования задач, однако применение этих протоколов исключает возможность использования наиболее эффективных дисциплин планирования;

— перестройка структуры задач таким образом, чтобы разорвать имеющиеся в графе связок междольные контуры, тогда необходимость применения протоколов пороговых приоритетов отпадает, открывается возможность использования самых эффективных дисциплин планирования;

— если такая перестройка структуры задач не удается, то может оказаться целесообразным выполнение третьего этапа анализа, состоящего в проверке наличия в графе связок пересечений междольных контуров. Если таких пересечений нет, защиту от возникновения взаимного блокирования обеспечит применение протокола междольных контуров, не накладывающего ограничений на варианты используемых дисциплин планирования.

Сложность проверки графа связок на отсутствие пересекающихся междольных контуров линейно зависит от суммы длин всех этих контуров.

Литература

1. **Давиденко К. Я.** Технология программирования АСУТП. Проектирование систем реального времени, параллельных и распределенных приложений. — М.: Энергоатомиздат, 1985. — 183 с.
2. **Liu J. W. S.** Real-Time Systems. — NJ: Prentice Hall, 2000. — 590 p.
3. **Sha L., Rajkumar R., Lehoczky J. P.** Priority Inheritance Protocols: An Approach to Real-Time Synchronization // IEEE Trans. on Computers. 1990. N 39(9). P. 1175–1185.
4. **Никифоров В. В., Баранов С. Н.** Статическая проверка корректности разделения ресурсов в системах реального времени // Тр. СПИИРАН. 2017. № 3(52). С. 17–21.
5. **Nikiforov V. V., Baranov S. N.** Multi-Partite Graphs and Verification of Software Applications for Real-Time Systems // Cybernetics and Information Technologies. 2016. N 16(2). P. 85–96.
6. **Никифоров В. В., Шкиртиль В. И.** Маршрутные сети — графический формализм представления структуры программных приложений реального времени // Тр. СПИИРАН. 2010. № 14. С. 5–28.
7. **Данилов М. В., Никифоров В. В.** Статическая обработка спецификаций программных систем реального времени // Программные продукты и системы. 2000. № 4. С. 13–19.
8. **Baranov S. N., Nikiforov V. V.** Density of Multi-Task Real-Time Applications // Proc. 17th Conf. FRUCT, Yaroslavl, 2015. С. 9–15.
9. **Никифоров В. В., Шкиртиль В. И.** Составное блокирование взаимосвязанных задач в системах на многоядерных процессорах // Изв. вузов. Приборостроение. 2012. № 7. С. 25–31.
10. **Никифоров В. В., Шкиртиль В. И.** Цепное блокирование задач в системах реального времени // Информационно-измерительные и управляющие системы. 2013. № 7. С. 17–21.
11. **Никифоров В. В.** Протокол предотвращения взаимного блокирования задач в системах реального времени // Изв. вузов. Приборостроение. 2014. № 57(12). С. 21–27.
12. **Johnson D.** Finding All the Elementary Circuits of a Directed Graph // SIAM Journal on Computing. 1975. N 4(1). P. 77–84. doi:10.1137/0204007/issn0097-5397
13. **Tarjan R.** Enumeration of the Elementary Circuits of a Directed Graph // SIAM Journal on Computing. 1973. N 2(3). P. 211–216. doi:10.1137/0202017/issn 0097-5397
14. **Tiernan J. C.** An Efficient Search Algorithm to Find the Elementary Circuits of a Graph // Communica-

tions of the ACM. 1970. N 13(12). P. 722–726. doi:10.1145/362814.362819
15. **Mateti P., Deo N.** On Algorithms for Enumerating all Circuits of a Graph // SIAM Journal on Computing. 1976. N 5(1). P. 90–99. doi:10.1137/0205007
16. **Tarjan R.** Depth-first Search and Linear Graph Algorithms // SIAM Journal on Computing. 1972. N 1(2). P. 146–160.
17. **Dijkstra E.** A Discipline of Programming. — NJ: Prentice Hall, 1976. — 217 p.

UDC 004.04
doi:10.15217/issn1684-8853.2017.4.59

Algorithms for Checking Applicability of Resource Access Protocols in Real-Time Systems

Nikiforov V. V.[a], Dr. Sc., Tech., Professor, nikifor_sergei@mail.ru
Podkorytov S. A.[a], PhD, Junior Researcher, podkorytovs@gmail.com
[a]Saint-Petersburg Institute for Informatics and Automation of the RAS, 39, 14 Line, V. O., 199178, Saint-Petersburg, Russian Federation

Introduction: Developing multi-task applications often requires that access to their common resources is shared between multiple tasks. For this purpose, synchronizing elements like mutexes are often used. However, using mutexes can lead to deadlocks. To avoid them, you can use special access protocols with additional steps on top of mutex operations («request»/»free resource»). In real-time systems, these steps could lead to a significant increase in response time for high priority tasks. Previously, a solution based on static analysis of real-time application models was proposed. The applications are modeled using graphical formalism of route nets. This solution relies on building a special multipartite oriented graph (a graph of bundles of critical intervals). **Purpose:** The goal is to develop algorithms which would implement the static analysis proposed earlier, and to evaluate their complexity. **Results:** The algorithms developed in this study can determine whether the necessary and sufficient conditions for deadlocks are found in a given application. The analysis is performed in three steps. The first algorithm builds a graph of bundles of critical intervals. This algorithm is linear with respect to number of bundles and their dependencies. The second algorithm enumerates the interparty circuits. This algorithm takes $O((n+e)(c+1))$, where c is the number of the circuits, n is number of the vertices, and e is the number of the edges. Finally, the third algorithm searches for intersections between the interparty circuits. The third algorithm takes linear time with respect to the total length of all the interparty circuits. These algorithms allow developers to modify an application early in its development to prevent deadlocks or chose an access protocol providing the best performance.

Keywords — Real-Time Systems, Multi-Task Application Models, Access Protocols, Shared Resources.

References

1. Davidenko K. Ya. *Tekhnologiia programmirovaniia ASUTP. Proektirovanie sistem real'nogo vremeni, parallel'nykh i raspredelennykh prilozhenii* [A Technology for Programming Industrial Control Systems. Designing Real-Time Systems, Parallel and Distributed Applications]. Moscow, Energoatomizdat Publ., 1985. 183 p. (In Russian).
2. Liu J. W. S. *Real-Time Systems*. NJ, Prentice Hall, 2000. 590 p.
3. Sha L., Rajkumar R., Lehoczky J. P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. on Computers*, 1990, no. 39(9), pp. 1175–1185.
4. Nikiforov V. V., Baranov S. N. Static Verification of Task Access to Shared Resources in Real-Time Systems. *Trudy SPIIRAN*, 2017, vol. 3(52), pp. 138–157 (In Russian).
5. Nikiforov V. V., Baranov S. N. Multi-Partite Graphs and Verification of Software Applications for Real-Time Systems. *Cybernetics and Information Technologies*, 2016, no. 16(2), pp. 85–96.
6. Nikiforov V. V., Shkirtil V. I. Route Nets — Graphical Form for Structural Representation of Real-Time Software Applications. *Trudy SPIIRAN*, 2010, no. 14, pp. 5–28 (In Russian).
7. Danilov M. V., Nikiforov V. V. Static Processing of Real-Time Software System Specifications. *Programmnye produkty i sistemy*, 2000, no. 4, pp. 13–19 (In Russian).
8. Baranov S. N., Nikiforov V. V. Density of Multi-Task Real-Time Applications. *Proc. 17th Conf. FRUCT*, Yaroslavl, 2015, pp. 9–15.
9. Nikiforov V. V., Shkirtil V. I. Compound Blocking of Dependent Tasks in Real-Time Systems at Multi-Core Processors. *Izvestiia vuzov. Priborostroenie*, 2012, no. 1, pp. 25–31 (In Russian).
10. Nikiforov V. V., Shkirtil V. I. Chained Task Blocking in Real-Time Systems. *Informatsionno-izmeritel'nye i upravliaiushchie sistemy*, 2013, no. 7, pp. 17–21 (In Russian).
11. Nikiforov V. V. Protocol for Avoiding of Mutual Blocking in Real-Time Systems. *Izvestiia vuzov. Priborostroenie*, 2014, no. 12, pp. 21–27 (In Russian).
12. Johnson D. Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing*, 1975, no. 4(1), pp. 77–84. doi:10.1137/0204007/issn0097-5397
13. Tarjan R. Enumeration of the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing*, 1973, no. 2(3), pp. 211–216. doi:10.1137/0202017/issn:0097-5397
14. Tiernan J. C. An Efficient Search Algorithm to Find the Elementary Circuits of a Graph. *Communications of the ACM*, 1970, no. 13(12), pp. 722–726. doi:10.1145/362814.362819
15. Mateti P., Deo N. On Algorithms for Enumerating all Circuits of a Graph. *SIAM Journal on Computing*, 1976, no. 5(1), pp. 90–99. doi:10.1137/0205007
16. Tarjan R. Depth-first Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1972, no. 1(2), pp. 146–160.
17. Dijkstra E. *A Discipline of Programming*. NJ, Prentice Hall, 1976. 217 p.