УДК 681.3

В.В. Никифоров, В.И. Шкиртиль

МАРШРУТНЫЕ СЕТИ — ГРАФИЧЕСКИЙ ФОРМАЛИЗМ ПРЕДСТАВЛЕНИЯ СТРУКТУРЫ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ

Никифоров В.В., Шкиртиль В.И. **Маршрутные сети — графический формализм представления структуры программных приложений реального времени.**
Аннотация. Построен графический формализм «маршрутные сети» для представления структуры программных приложений реального времени. Формализм ориентирован на проверку корректности межзадачных интерфейсов. Определены границы применимости методов анализа корректности межзадачных связей на основе использования графов состояний и переходов. На основе введения понятия профиля задачи построен метод оценки значений фактора блокирования в системах со сцепленными критическими интервалами доступа к разделяемым ресурсам.
Ключевые слова: системы реального времени, модели многозадачных программных систем, межзадачные интерфейсы, кольца взаимных ожиданий, динамическая корректность программных приложений.

Nikiforov V.V., Shkirtil V.I. **Itinarary nets — the graphical formalism for presentation of real-time sofware application strucrures.**
Abstract. A graphic formalism «itinerary nets» is expounded for representation of real–time software application structures. The formalism is oriented to verification of static and dynamic properties of task interfaces. The bounds are defined for applicability of verification that is based on using of states and transitions graphs. A method is defined for assessing the value of blocking factor in systems with concatenated critical intervals of access to shared resources. The method is based on introduce of task profile feature.
Keywords: real-time systems, models of multitask software systems, task interfaces, dynamic correctness of software applications.

1. Введение. Определяющее свойство программных приложений реального времени состоит в том, что они работают «в структуре времени, определяемой ходом внешних процессов». Такое соответствие между ходом исполнения программных компонент системы реального времени (СРВ) и ходом внешних процессов обусловлено наличием более или менее жестких временных рамок для информационных обменов с внешними процессами. Ход физического времени, определяющий течение внешних процессов, не подвержен масштабированию, его невозможно ускорить или замедлить. Компоненты СРВ должны лишь приспосабливаться к ходу реального (физического) времени.

Требование соответствия между «структурой времени», отражающей ход внешних процессов и «структурой времени» исполнения программных компонент является одним из проявлений общего требования к архитектуре рационально построенной СРВ — требования соблюдения *принципа структурного соответствия*: для рационально построенной СРВ структура комплекса программных объектов должна представлять собой преломленное отображение структуры множества внешних объектов [1]. Этот принцип предлагается в качестве руководящего положения при разработке архитектуры программной системы реального времени. Одним из главных следствий этого принципа является требование организации программного приложения СРВ в виде комплекса кооперативных задач.

Для СРВ характерно широкое разнообразие автоматизируемых узлов и агрегатов, разнообразие их взаимосвязей, различий в темпах их функционирования. В связи с этим и сообразно принципу структурного соответствия программные приложения реального времени строят в виде множества замкнутых по управлению последовательных программных компонент, называемых *задачами*. Составляющие программное приложение СРВ задачи являются кооперативными задачами в том смысле, что они служат достижению общей цели — эффективному функционированию системы в целом. В ходе работы системы конкретная задача может исполняться многократно. Отдельное исполнение задачи называют *заданием*.

Кооперативные задачи координируют свои действия, обмениваясь данными и синхронизирующими сигналами, 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 многозадачных систем актуальны проблемы обеспечения логической корректности, в первую очередь обеспечения гарантий невозможности возникновения таких некорректных ситуаций, как кольца взаимных ожиданий (тупики и «клинчи»). Наряду с обеспечением логической корректности актуальны проблемы динамической корректности — обеспечения своевременности выполнения возлагаемых на систему задач.

Проверка логической и динамической корректности структуры программных приложений СРВ достигается путем построения и анализа моделей, отражающих те и только те особенности моделируемых

приложений, представление которых необходимо и достаточно для достижения целей анализа.

Для проверки корректности программных приложений СРВ используют параметрические и структурные модели. Параметрические модели отличаются тем, что задачи, составляющие программное приложение реального времени, представляются рядом числовых характеристик (параметров), особенности структуры кода задач не рассматриваются.

Структурные модели представляют элементы внутренней организации задач, разбиение кода задачи на сегменты, разделяемые системными операторами.

2. Параметрические модели. Простейшие программные приложения состоят из ряда независимых задач τ_1, τ_2, … τ_n. Под *независимостью* задач понимается отсутствие таких межзадачных взаимосвязей (по обмену сообщениями, доступу к разделяемым информационным ресурсам), которые приводили бы к логическим ограничениям на порядок выполнения фрагментов различных заданий.

Порядок предоставления процессорного времени для обслуживания активных заданий определяется используемой дисциплиной планирования. Дисциплину планирования можно определить как способ наделения заданий целочисленными *приоритетами*: в очередной момент реализации дисциплины планирования активным заданиям присваиваются определенные значения приоритетов и ресурс процессорного времени выделяется наиболее приоритетным из активных заданий. В большинстве СРВ приоритеты задач назначаются статически. Условимся индексировать задачи τ_1, τ_2, … τ_n, составляющие приложение, в порядке снижения приоритета: τ_1 — наиболее приоритетная задача, τ_n — наименее приоритетная.

Очередная j-я активизация задачи τ_i означает порождение очередного j-го ее экземпляра — порождение *задания* $\tau_i^{(j)}$. Различные задания, порождаемые в результате активизации одной и той же задачи, являются однотипными заданиями: задача представляет собой тип заданий, порождаемых в моменты ее активизации. Порождение задания означает увеличение числа претендентов на процессорное время, и, следовательно, изменение условий распределения процессорного времени. Если в общем случае изменение условий распределения системных ресурсов называется *системным событием*, то порождение задания следует рассматривать как одну из разновидностей системных

событий. Другим примером системных событий является завершение задания — уменьшение числа претендентов на процессорное время.

В приложении с независимыми задачами задание является *активным* в некоторый конкретный момент времени, если к этому моменту оно уже порождено, но еще не завершено. То есть каждое задание является активным в рамках интервала его существования — от момента порождения $t_{arr}\left(\tau_j^{(k)}\right)$ до момента завершения $t_{end}\left(\tau_j^{(k)}\right)$. На интервале существования независимые задачи могут находиться в одном из двух *системных состояний*: 1) задание, владеющее исполнительным ресурсом, находится в состоянии «*текущее*»; 2) задание, ожидающее предоставления ему исполнительного ресурса, находится в состоянии «*готовность*» (готовность использовать процессорное время).

Ход исполнения задания может быть приостановлен ввиду переключения процессора на исполнение вновь порожденного более приоритетного задания (вытеснение текущего задания). Через некоторый интервал времени исполнение вытесненного задания возобновляется для продолжения недовыполненных вычислений. Таким образом, возникает *интерференция* — взаимное влияние активных заданий на сроки их исполнения. Несмотря на такую интерференцию, в СРВ должна гарантироваться своевременность выполнения заданий: продолжительность $r_i^{(j)} = t_{\text{end}}\left(\tau_j^{(k)}\right) - t_{\text{arr}}\left(\tau_j^{(k)}\right)$ интервала существования любого экземпляра задачи τ_i не должна превышать максимально допустимого значения D_i. Используя понятие «*время отклика*» $R_i = \max\left\{r_i^{(j)} \middle| \; j = 1, 2, \ldots\right\}$ задачи τ_i требование выполнимости задачи (гарантированной своевременности выполнения любого ее экземпляра) выражается неравенством $D_i \geq R_i$.

Для независимых задач время отклика равно сумме $R_i = C_i + I_i$ двух составляющих: 1) фактора веса C_i задачи τ_i (максимальной продолжительности пребывания в состоянии «текущее») и 2) фактора интерференции L_i (максимальной продолжительности пребывания τ_i в состоянии «готовность»). Вклад в I_i каждой задачи τ_j (более приоритетной, чем τ_i) равен $\lceil R_i / T_j \rceil$, где $\lceil x \rceil$ — ближайшее сверху к x целое число, T_j — минимальный интервал времени между порождениями заданий типа τ_i. Таким образом, значение фактора интерференции равно сумме $I_i = \sum_{j<i} C_j \lceil R_i / T_j \rceil$. Подставляя эту сумму в выражение для

R_i, получаем уравнение, решение которого методом последовательных приближений дает длительность времени отклика для задачи τ_i [2–4].

3. Структурные модели. Для обеспечения целостности информационных ресурсов используются механизмы защиты доступа к каждому сегменту кода задач, в котором осуществляется доступ к тому или иному разделяемому информационному ресурсу (критическому интервалу кода задачи по доступу к ресурсу). Контроль доступа к критическим интервалам строится на базе использования синхронизирующих элементов, называемых мьютексами: для каждого разделяемого информационного ресурса формируется мьютекс, принимающий значение 1, если ресурс свободен, и значение 0, если ресурс занят. Критический интервал по доступу к разделяемому ресурсу g обрамляется специальными системными операторами над соответствующим мьютексом `m`: на входе в критический интервал выполняется системный оператор `lock m` (запрос ресурса g), на выходе — оператор `unlock m` (освобождение ресурса g). Исполнение задания, запрашивающего занятый ресурс, приостанавливается до момента освобождения требуемого ресурса. Таким образом, при исполнении задач, разделяющих информационные ресурсы, в добавление к системным состояниям «текущее» и «готовность» возникает состояние «ожидание».

Для статической оценки корректности построения программных приложений с состояниями «ожидание», использование параметрических моделей может оказаться недостаточным. Как показано ниже, в случае пересечения критических интервалов использование параметрических моделей может привести к недооценке времени отклика отдельных задач и, следовательно, к ошибочным выводам относительно гарантий своевременности выполнения этих задач.

Представление структурных особенностей многозадачных программных систем может быть выполнено, например, путем использования формальных моделей, построенных средствами языка XML [5, 6].

Представление структуры программных приложений средствами языка XML. Различаются два рода объектов, составляющих программное приложение СРВ: 1) задачи, реализующие требуемые алгоритмы функционирования приложения; и 2) интерфейсные объекты, связывающие задачи в единое программное приложение. Примером разновидности интерфейсных объектов являются мьютексы, обеспечивающие защиту критических интервалов по доступу к информационным ресурсам. Доступ задач к интерфейсным объектам осуществляется

через интерфейсные операции (для мьютексов — операции захвата/освобождения ресурсов).

При практической реализации различных СРВ используется большое разнообразие типов интерфейсных объектов (глобальных структур данных, очередей сообщений, сигналов, семафоров и т.п.). В рамках настоящей работы представлена одна разновидность интерфейсных объектов — мьютексы.

Для представления мьютексов в XML-модели строятся элементы, соответствующие следующему формату:

```
<!ELEMENT mutex>
<!ATTLIST mutex name ID #REQUIERED>
```

Условимся ограничивать рассмотрение такими задачами, в которых интерфейсные операции не встречаются внутри операторов ветвлений и циклов. Тогда интерфейсные операции разбивают программный код задачи на последовательность сегментов. Первая интерфейсная операция в теле задачи завершает первый сегмент задачи, последующие операторы до второй интерфейсной операции включительно составляют второй сегмент задачи и т.д. В качестве параметра интерфейсной операции выступает указатель мьютекса. Конечный сегмент программного кода задачи замыкается локальной операцией.

XML-элемент, представляющий сегмент задачи, имеет либо один, либо три атрибута:

```
<!ELEMENT segment>
<!ATTLIST segment length #PCDATA #REQUIERED
 interface IDREF
 op_type (”get”|”put”) >
```

Все три атрибута имеют элементы, представляющие не конечные сегменты; элемент, представляющий конечный сегмент кода задачи имеет единственный атрибут `length` (атрибуты `interface` и `op_type` отсутствуют).

Для представления задач используются XML-элементы следующего вида:

```
<!ELEMENT task (segment+)>
<!ATTLIST isr name ID #REQUIERED
 priority #PCDATA #REQUIERED
 period #PCDATA #REQUIERED
 deadline #PCDATA #REQUIERED >.
```

Программное приложение состоит из одной или более задач и множества (возможно, пустого) мьютексов.

```
<!ELEMENT application (mutex*, task+)>
<!ATTLIST application protocol (”ПНП”|”ППП”)>
```

Единственный атрибут приложения — `protocol`, указывает тип используемого протокола доступа к разделяемым ресурсам.

В таких обозначениях структура двухзадачного приложения с двумя разделяемыми информационными ресурсами представляется следующим образом:

```
<application>
 <mutex name="m1"/>
 <mutex name="m2"/>
 <task name="t1" priority="1" period="20"
 deadline="20">
 <segment length="1" interface="m1" op_type="get"/>
 <segment length="3" interface="m2" op_type="get"/>
 <segment length="2" interface="m1" op_type="put"/>
 <segment length="3" interface="m2" op_type="put"/>
 <segment length="1" />
 </task>
 <task name="t2" priority="2" period="32"
 deadline="32">
 <segment length="1" interface="m1" op_type="get"/>
 <segment length="4" interface="m2" op_type="get"/>
 <segment length="2" interface="m2" op_type="put"/>
 <segment length="4" interface="m1" op_type="put"/>
 <segment length="1" />
 </task>
</application>.
```

Приведенные форматы описания структуры многозадачных приложений на языке XML могут дополняться средствами представления других типов системных объектов и отношений. В работе [5] рассмотренные форматы дополняются элементами, ориентированными на представление аппаратно-активизируемых задач (обработчиков прерываний), программно-активизируемых задач (процессов, потоков), семафоров, очередей сообщений, а также внешних по отношению к программной системе объектов — измерительных подсистем и исполнительных устройств.

Маршрутные сети. В ряду средств представления структуры программных комплексов и их компонент всегда были популярны графические средства — блок-схемы, IDEF- и UML-диаграммы [7]. К средствам такого рода относятся маршрутные сети [8]. На рис. 1 изображены элементы одного из вариантов маршрутных сетей.

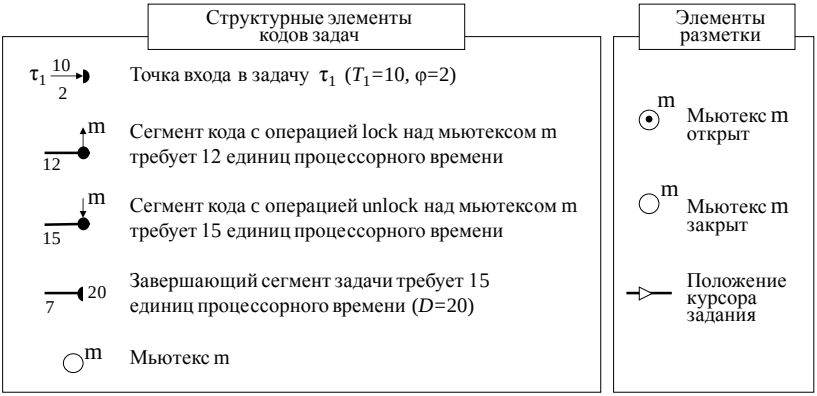

Рис 1. Элементы маршрутных сетей.

В левой части рис. 1 приведены структурные элементы для представления статических свойств графических моделей задач (аналогично позициям, переходам и дугам сетей Петри), в правой части — элементы для представления текущего состояния хода исполнения задачи (аналогично меткам в позициях сетей Петри).

Приведенный на рис. 1 набор структурных элементов маршрутных сетей полностью соответствует представленному выше набору XML-элементов. Маршрутная сеть на рис. 2 в точности соответствует структуре двухзадачного приложения, представленного выше в виде XML-текста.

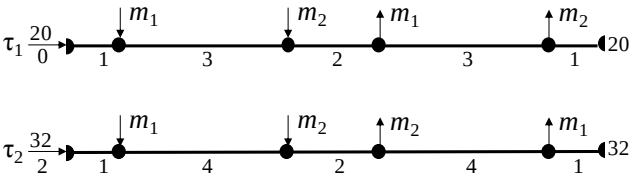

Рис 2. Изображение структуры приложения в виде маршрутной сети

Элемент «точка входа в задачу» снабжается именем (τ_1, τ_2), значением периода $(T_1 = 20,\ \ T_2 = 32)$ и фазой первой активизации ($\varphi_1 = 0$, $\varphi_2 = 2$) задачи. Задачи индексируются в порядке убывания приоритетов. Заключительный элемент снабжается указанием предельного срока выполнения задачи $(D_1 = 20,\ \ D_2 = 32)$. Каждый из сегментов снаб-

жается указанием максимального числа единиц процессорного времени, требуемого для его выполнения.

Часть информации, представляемой элементами маршрутных сетей, изображенными на рис. 1, может оказаться избыточной. Например, при рассмотрении вопросов логической корректности взаимосвязей задач не требуется информация о параметрах, связанных с частотой активизации задач, с продолжительностью исполнения составляющих сегментов кода и задач в целом.

Представление структуры многозадачных программных приложений в виде текстов на языке XML может непосредственно использоваться программами автоматического анализа. Преобразование этих текстов к графическому виду типа маршрутных сетей обеспечивает возможность визуального анализа структурных особенностей системы.

4. Граф состояний и переходов. Рассмотрим подход к анализу логической корректности системы, представляемой маршрутной сетью, на примере известной модели «Пять обедающих философов» (рис. 3) [9, 10].

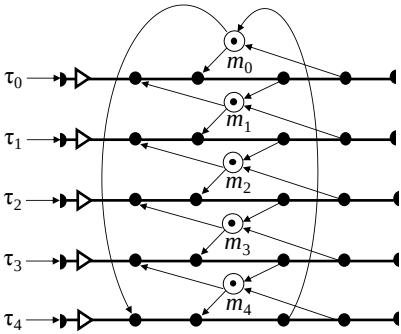

Рис 3. Структура модели «Пять философов».

Заметим, что на рис. 3, в отличие от рис. 2, отсутствуют числовые атрибуты маршрутной сети, зато присутствуют элементы разметки, фиксирующие текущее положение курсоров каждого из пяти действующих заданий и текущее состояние каждого из пяти синхронизирующих элементов-мьютексов. Каждая из пяти задач содержит по пять сегментов, разделенных операторами над мьютексами.

Задача τ_i на рис. 3 может находиться в одном из шести состояний, которые мы будем обозначать следующим образом:

$St_i(\tau_i) = 0$ — задача не активна (курсор на соответствующем маршруте отсутствует),

$St_i(\tau_i) = 1$ — задача активизирована, исполняется первый из сегментов ее кода (курсор размещается в первом сегменте соответствующего маршрута),

. . .

$St_i(\tau_i) = 5$ — исполняется завершающий сегмент кода задачи (курсор размещается в последнем сегменте соответствующего маршрута).

Поскольку текущие состояния мьютексов однозначно определяются текущим положением курсоров, состояние всей системы однозначно определяется вектором ($St_i(\tau_0)$, $St_i(\tau_1)$, $St_i(\tau_2)$, $St_i(\tau_3)$, $St_i(\tau_4)$).

В рассматриваемом примере текущее состояние каждой из задач представляется однозначным числом, поэтому конкретное состояние всей системы можно задавать последовательностью из пяти цифр. Состояние системы, соответствующее рис. 3, изображается кодом 11111 — курсор каждой из задач находится в первом сегменте. Система находится в таком состоянии до тех пор, пока одной из задач не будет выполнен очередной системный оператор.

Из состояния 11111 система на рис. 3 может перейти в одно из пяти состояний: в состояние 21111 (если задачей τ_0 выполнен оператор `lock m0`), . . . , в состояние 11112 (если задачей τ_4 выполнен оператор `lock m4`).

Если курсор задачи τ_i находится в сегменте, завершающимся операцией `lock` над некоторым мьютексом m_k, который занят задачей τ_j, то τ_i не сможет покинуть текущий сегмент до тех пор, пока τ_j не освободит m_k. То есть возможность продолжения исполнения τ_i зависит от действий τ_j (τ_i ожидает действий τ_j). Условимся представлять такое отношение зависимости между τ_i и τ_j обозначением $j-i$. Так, в состоянии 12211 системы на рис. 3 имеет место отношение 3–2., в состоянии 12221 — парой отношений 3–2–1. В состоянии 22222 система попадает в тупик (все пять задач связаны в кольцо взаимных ожиданий 4–3–2–1–0–4).

Универсальным способом проверки системы на возможность возникновения тупиков и «клинчей» является построение графа состояний и переходов. На рис. 4 приведен фрагмент такого графа, содержащий пути перехода системы из исходного состояния системы (см. рис. 3) в тупиковое состояние 22222.

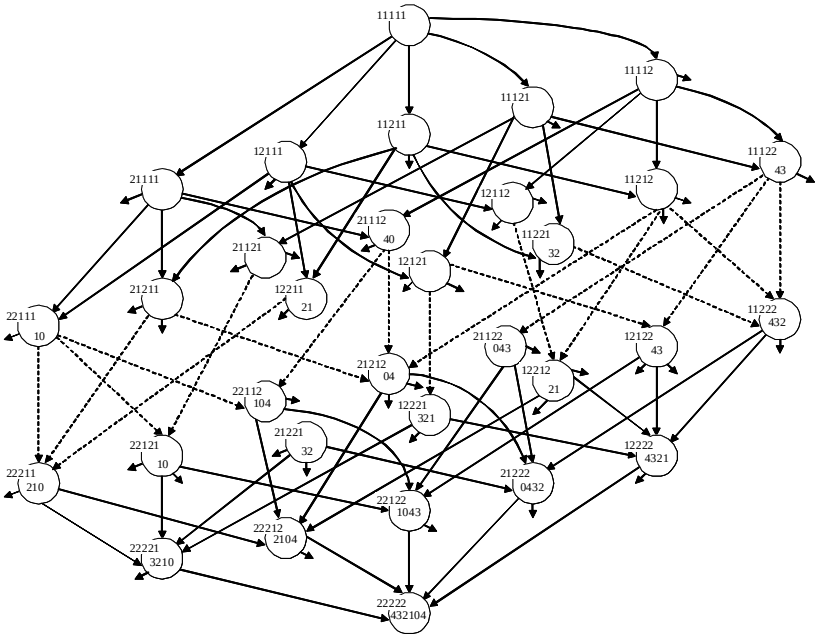

Рис 4. Фрагмент графа состояний и переходов для модели «Пять философов».

Пятизначные коды вершин графа указывают соответствующие им состояния системы. Каждая дуга, исходящая из вершины, соответствует срабатыванию системного оператора в одной из задач (в результате срабатывания оператора выполняется переход в следующее состояние).

Дополнительные коды, размещенные в некоторых вершинах, указывают на имеющиеся в этом состоянии отношения ожидания между задачами (использована возможность опустить дефисы между номерами задач, поскольку в данной модели задачи именуются однозначными числами).

Эта дополнительная информация не требуется при поиске вершин, соответствующих тупиковым состояниям — такие вершины отличаются отсутствием исходящих дуг. Но она необходима при поиске вершин, соответствующих частичному взаимному блокированию, при котором кольцом взаимных ожиданий связаны не все действующие задания, а только часть из них. Пример системы, способной попасть в состояние частичного взаимного блокирования, приведен на рис. 5.

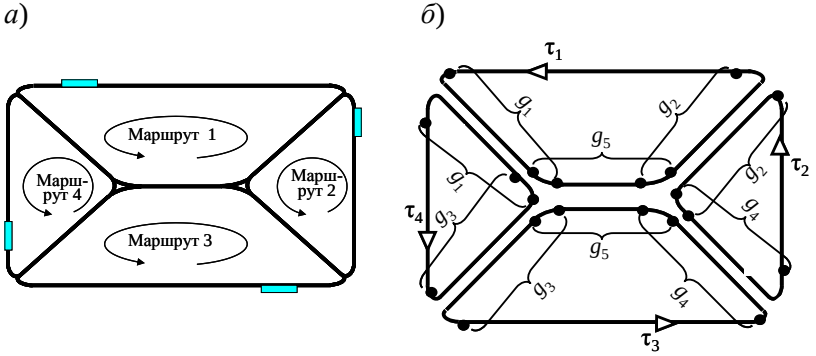

Рис 5. Система с четырьмя маршрутами:
а) постановка проблемы, *б*) вариант решения проблемы.

На рис. 5*а* изображена предложенная П.С. Краснощековым модель системы путей, по которым движутся четыре тележки. Длина каждой из тележек превышает длину разъездов на концах однопутных перегонов. Ставится задача такого размещения синхронизирующих элементов на этих путях, которое обеспечило бы безаварийное движение тележек.

На рис. 5*б* приведен вариант расстановки операторов захвата/освобождения ресурса. Этот вариант на первый взгляд кажется естественным. Анализ, однако, показывает, что это при исполнении приложения со структурой операторов, изображенной на рис. 5*б*, могут возникать кольца взаимных ожиданий.

Разметка маршрутной сети на рис. 6 соответствует варианту достижимого состояния 4225 системы с четырьмя маршрутами. В этом состоянии три задания (задания типов τ_1, τ_2, τ_3) связаны в кольцо взаимных ожиданий: τ_1 удерживает ресурс g_5 и ждет доступа к ресурсу g_2, τ_2 удерживает ресурс g_2 и ждет доступа к ресурсу g_4, τ_3 удерживает ресурс g_4 и ждет доступа к ресурсу g_5. В формате кодов рис. 4 этому кольцу взаимных ожиданий соответствует дополнительный код 3123. Задания типа τ_4 могут продолжать беспрепятственно завершаться, порождаться и исполняться произвольное число раз, соответствующие коды состояний принимают значения $422x$, $0 \le x \le 5$.

Анализ показывает, что для системы рис.5*б* возможно возникновение еще одного кольца взаимных ожиданий с кодами состояний 2*x*42 (дополнительный код 4134).

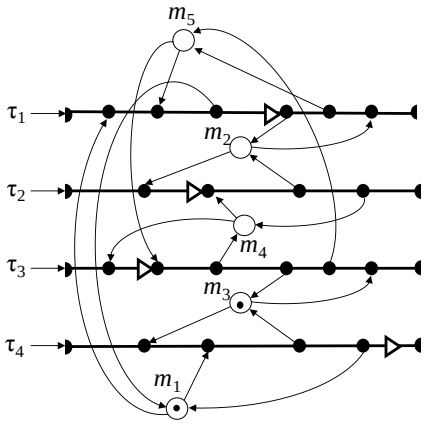

Рис 6. Кольцо взаимных ожиданий.

Чтобы обнаружить возможность частичного взаимного блокирования, необходимо для каждой вершины графа состояний и переходов отслеживать отношения ожидания между задачами.

5. Объемы памяти для построения графа состояний и переходов. При автоматическом построении графа состояний и переходов каждая вершина графа задается структурой, содержащей следующее:

— массив исходящих дуг,

— массив отношений зависимости задач,

— данные о текущем состоянии процесса построения графа.

При использовании 32- разрядной адресации компонент графа состояний общий объем массива исходящих дуг составляет $4n$ байт (n — число задач в моделируемой системе). Если число задач моделируемой системы не превышает 236, то под массив отношений зависимости задач достаточно выделить n байт. Для представления текущего состояния процесса построения графа может оказаться достаточным выделение одного байта. Отсюда получаем оценку объема памяти для представления отдельной вершины графа состояний и переходов: $1+5n$ байт.

Общий объем памяти, который следует выделить для представления графа состояний, составляет $(1+5n)(1+k_a)^n$ байт, где k_a — среднее геометрическое числа состояний имеющихся в моделируемой системе задач $k_a = \left(\prod_{1\le i\le n}(1+\max(St(\tau_i)))\right)^{1/n}$. Для системы на рис. 3 $k_a = 6$, для системы на рис. 6 $k_a = 6{,}93$. В большинстве практических систем k_a не

превышают этих значений. Граф состояний и переходов системы с $k_a = 7$, может уместиться в объеме машинной памяти 4 Гбайт, если число задач в системе отвечает неравенству $(1+5n)8^n < 2^{32}$. Из этого неравенства следует, что если число задач в системе равно 22 при $k_a = 7$, то ее граф состояний и переходов умещается в памяти 4 Гбайт, но при большем числе задач для размещения графа состояний и переходов 4 Гбайт памяти оказывается уже недостаточно. Система из 30 задач потребует памяти в несколько терабайт, для 40 задач потребуются уже петабайты памяти.

Таким образом, для систем с большим количеством задач анализ логической корректности путем построения графа состояний и переходов практически невозможен. По этой причине в настоящее время программные приложения СРВ строят без статического анализа систем на возможность возникновения тупиков и клинчей — вместо этого используются такие специализированные механизмы планирования исполнения задач, которые позволяют гарантированно обходить подобные некорректные состояния исполняемой программной системы. К таким механизмам относится протокол пороговых приоритетов (ППП).

6. Фактор блокирования при использовании ППП. Действие этого механизма опирается на статическое присвоение каждому ресурсу (мьютексу) специального параметра планирования, называемого пороговым приоритетом ресурса. Пороговый приоритет ресурса (мьютекса) равен высшему уровню статического приоритета задачи, которая может занять ресурс. Использование этого параметра при реализации ППП в системах с одиночным исполнительным ресурсом (одним классическим одноядерным процессором) состоит в том, что выполнение над мьютексом операции «захват» сопровождается повышением (на период владения ресурсом) приоритета текущего задания до уровня порогового приоритета занимаемого ресурса.

В системах с разделением доступа к информационным ресурсам выражение, определяющее значение R_i для времени отклика задачи, должно быть дополнено фактором блокирования B_i, отражающим увеличение длины интервала существования заданий типа τ_i за счет возможного пребывания в состоянии «ожидание» $R_i = C_i + B_i + I_i$.

Выбор ППП в качестве механизма, предотвращающего возникновение колец взаимных ожиданий, влияет на размер фактора блокирования B_i.

Работы, представляющие подходы к оценке значений фактора блокирования для систем с одиночным исполнительным ресурсом с использованием ППП, опираются на тезис о невозможности составного блокирования. Причем этот тезис формулируется в следующем виде [7]. «Задание может быть блокировано на протяжении исполнения не более чем одного критического интервала, что дает основание ограничивать модели задач, используемые для оценки фактора блокирования и представлением для каждой задачи перечня требуемых ей информационных ресурсов с указанием максимальной длины критического интервала по каждому ресурсу».

Принятие этого тезиса приводит к выводу, что значение фактора блокирования каждой из задач τ_i равно максимальной из длин критических интервалов, способных блокировать τ_i. Внутренняя структура кодов задач, взаимное расположение критических интервалов при этом не рассматриваются. Оценка фактора блокирования выполняется в рамках параметрической модели. Оказывается, однако, что тезис о невозможности составного блокирования в представленной формулировке может оказаться неверным, если в кодах задач имеются пересекающиеся критические интервалы (связки критических интервалов).

Рассмотрим маршрутную сеть, изображающую структуру и параметры программного приложения СРВ, состоящего из пяти задач τ_0, τ_1, τ_2, τ_3, τ_4. Для каждой из задач указаны длины всех сегментов. Символ $m_{i\oplus 1}$ означает сумму по модулю, равную 5. Структура кодов всех задач совпадает, различаются только длины сегментов и имена мьютексов, над которыми выполняются системные операции.

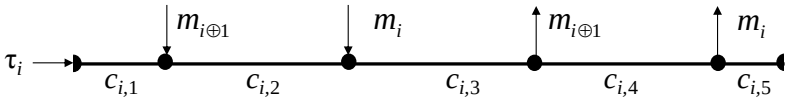

Рис 8. Маршрут с указанием длин сегментов.

Топология сети рис. 8 повторяет топологию сети рис. 3, но в представлении атрибутов сети имеются различия. На рис 3 нет указаний о длинах сегментов задач (при анализе системы на возможность возникновения тупиков и «клинчей» эта информация не нужна). На рис. 8 нет разметки, отражающей текущее состояние системы (при оценке значений фактора блокирования состояния системы принимается во внимание критический сценарий системных событий, а не конкретная разметка). Информация о периодах задач на рис. 8 тоже опущена — она

существенна для оценки фактора интерференции, для оценки фактора блокирования она не требуется.

Задачи	Ресурсы				
	g_0	g_1	g_2	g_3	g_4
τ_0	$C_0(g_0)=c_{0,3}+c_{0,4}$	$C_0(g_1)=c_{0,2}+c_{0,3}$	0	0	0
τ_1	0	$C_1(g_1)=c_{1,3}+c_{1,4}$	$C_1(g_2)=c_{1,2}+c_{1,3}$	0	0
τ_2	0	0	$C_2(g_2)=c_{2,3}+c_{2,4}$	$C_2(g_3)=c_{2,2}+c_{2,3}$	0
τ_3	0	0	0	$C_3(g_3)=c_{3,3}+c_{3,4}$	$C_3(g_4)=c_{3,2}+c_{3,3}$
τ_4	$C_4(g_0)=c_{4,2}+c_{4,3}$	0	0	0	$C_4(g_4)=c_{4,3}+c_{4,4}$

Рис 9. Параметры доступа к ресурсам.

На рис. 9 приведена параметрическая модель доступа задач к разделяемым ресурсам при использовании ППП. В рамках параметрической модели в каждой клетке таблицы доступа к разделяемым ресурсам указывается максимальная продолжительность критического интервала доступа конкретной задачи к конкретному ресурсу. Для оценки значения B_i фактора блокирования задачи τ_i предлагается отобрать все те значения $C_i(g_k)$, которые соответствуют критическим интервалам, способным блокировать τ_i.

В рамках использования параметрической модели для оценки фактора блокирования B_0 отбираются параметры $C_i(g_k)$, находящиеся на пересечении столбцов 0–1 со строками 1–4. В подтаблице, образуемой таким пересечением, находятся длины критических интервалов $C_1(g_1)$ и $C_4(g_0)$. Максимальное из отобранных значений дает размер фактора блокирования B_0. Для оценки значения B_3 в рамках процедуры использования параметрической модели отбираются параметры $C_i(g_k)$, находящиеся на пересечении столбцов 0–4 со строкой 4. В подтаблице, образуемой таким пересечением, находятся значения длин критических интервалов $C_4(g_0)$ и $C_4(g_4)$. В рамках процедуры использования параметрической модели оценкой фактора блокирования B_3 является величина $\max\{C_4(g_0), C_4(g_4)\}$. Оказывается, что в случае системы рис. 8 такая процедура приводит к недооценке значения фактора блокирования B_3.

Это обстоятельство становится очевидным при рассмотрении структурной модели системы. Действительно, при захвате ресурса g_0 задачей τ_4 максимальный приоритетный порог занятого ресурса становится равным 1. Если непосредственно вслед за этим задача τ_3 запросит ресурс g_4, то ее исполнение будет блокировано (ее приоритет ниже максимального значения приоритетного порога занятого ресурса).

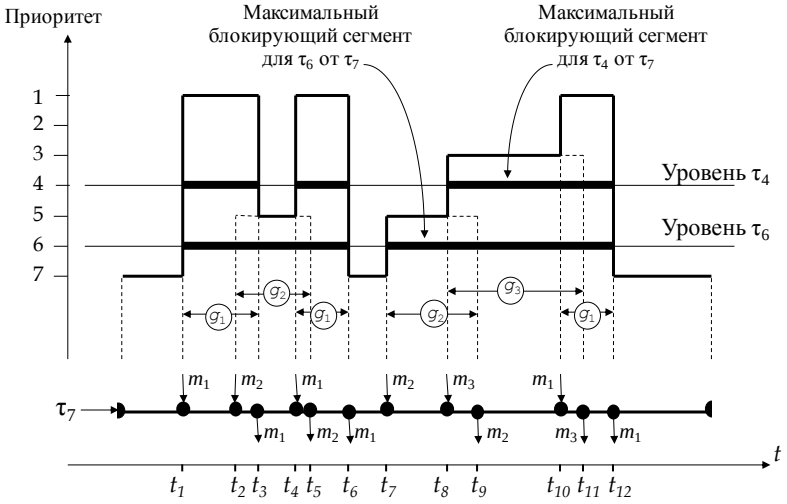

Рис. 10. Пример профиля задачи.

В момент освобождения ресурса g_0 задачей τ_4 ею уже занят ресурс g_4, требующийся задаче τ_3, поэтому задача τ_3 остается блокированной и за рамками исполнения задачей τ_4 критического интервала по доступу к g_0. Интервал блокирования задачи τ_3 задачей τ_4 заканчивается только после того, как τ_4 освободит не только g_0, но и g_4. То есть значение фактора блокирования B_3 равно не максимальной из длин $C_4(g_0)$ и $C_4(g_4)$, а общей длине связки критических интервалов по доступу τ_4 к ресурсам g_0 и g_4.

7. Профили задач. Общий подход к корректной оценке фактора блокирования для задач со сложной внутренней структурой может быть сформулирован за счет использования понятия «профиля зада-

чи». Проще всего понятие профиля может быть определено для систем, использующих протокол ППП в системах с одиночным исполнительным ресурсом: под профилем задачи понимается график изменения приоритета при ее исполнении с монопольным использованием процессора [5].

Определенное так понятие профиля задачи иллюстрирует рис. 10: представлен профиль задачи τ_7, имеющей критические интервалы по доступу к разделяемым ресурсам g_1 (интервалы $[t_1, t_3]$, $[t_4, t_6]$, $[t_{10}, t_{12}]$), g_2 (интервалы $[t_2, t_5]$, $[t_7, t_9]$) и g_3 (интервал $[t_8, t_{11}]$) со значениями пороговых приоритетов $p(g_1) = 1$, $p(g_2) = 5$, $p(g_3) = 3$. Каждый критический интервал задачи τ_7 изображен на рис. 10 в виде интервального прямоугольника: левое и правое ребро интервального прямоугольника соответствуют моментам начала и конца критического интервала, высота прямоугольника определяется пороговым приоритетом соответствующего ресурса. Шести критическим интервалам задачи τ_7 соответствуют шесть прямоугольников; для сцепленных критических интервалов соответствующие прямоугольники пересекаются. Профиль задачи строится как огибающая всех интервальных многоугольников (показан на рис. 10 сплошной ломаной линией).

Профиль задачи τ_i содержит всю информацию о том, как τ_i может блокировать исполнение более приоритетных задач. Для того чтобы определить возможное влияние τ_i на блокирование более приоритетной задачи τ_j, нужно провести горизонтальную линию на уровне приоритета τ_j и выделить пересечения этой горизонтальной линии с профилем задачи τ_i.

Рассмотрим пересечения изображенной на рис. 10 горизонтали приоритетного уровня 6 с профилем задачи τ_7. На данном приоритетном уровне выделяются два сегмента горизонтальной линии, целиком расположенных ниже профиля задачи τ_7 (эти сегменты выделены на рис. 10 жирными отрезками). Сегмент $[t_7, t_{12}]$ длиннее сегмента $[t_1, t_6]$, откуда следует: максимально возможный вклад задачи τ_7 в блокирование задачи с приоритетом 6 равен длине сегмента $[t_7, t_{12}]$. Аналогичное рассмотрение пересечений горизонтали на приоритетном уровне 4 показывает, что максимально возможный вклад задачи τ_7 в блокирование задачи с приоритетом 4 равен длине сегмента $[t_8, t_{12}]$.

8. Заключение. Представленный графический формализм «маршрутные сети» позволяет строить наглядные представления структуры межзадачных связей в многозадачных программных приложениях для СРВ. Формализм «маршрутные сети», ориентированный на эффективное визуальное представление структуры системы, полностью согласован с разработанным авторами форматом представления структуры межзадачных связей средствами языка XML — форматом, пригодным для непосредственного автоматического анализа свойств конструируемых систем. Средства разметки маршрутных сетей, аналогичные разметке сетей Петри, позволяют представлять не только статические структурные особенности многозадачных программных приложений, но и конкретные состояния системы, возникающие в ходе исполнения приложения, в частности — состояния с кольцами взаимных ожиданий.

Универсальным подходом к проверке системы на возможность возникновения колец взаимных ожиданий (тупиков и «клинчей») является построение (на основе структур типа маршрутных сетей) графа состояний и переходов системы. При использовании 32-разрядных компьютеров с оперативной памятью в несколько Гбайт этот подход применим, если программное приложение содержит до 20 задач. В случае существенно большего числа задач возникает необходимость поиска специализированных подходов.

В СРВ разделением доступа к информационным ресурсам при проверке гарантий своевременности выполнения отдельных задач и проверке выполнимости программного приложения в целом необходимы оценки значений фактора блокирования высокоприоритетных задач низкоприоритетными задачами, занимающими разделяемые ресурсы. Для нахождения таких оценок в системах со сцепленными критическими интервалами необходимо учитывать особенности внутренней структуры блокирующих задач. С этой целью можно использовать моделирование задач маршрутными сетями с построением профилей блокирующих задач и анализ тех сечений построенных профилей, которые соответствуют блокируемым задачам.

Литература

1. *Давиденко К.Я.* Технология программирования АСУТП. Проектирование систем реального времени, параллельных и распределенных приложений. М.: Энергоатомиздат. 1985. 183 с.
2. *Данилов М.В.* Методы планирования выполнения задач в системах реального времени // Программные продукты и системы. 2001. № 4. С. 28–35.
3. *Гома Х.* UML. Проектирование систем реального времени, параллельных и распределенных приложений. М.: ДМК Пресс, 2002. 739 с.

4. *Liu J.W.S*. Real-Time Systems. N.J.: Prentice Hall, 2000. 590 p.
5. *Никифоров В.В*., *Шкиртиль В.И*. Спецификация средствами языка XML системы интерфейсов в приложениях реального времени // Тр. СПИИРАН. 2009. Вып 11. С. 159–175.
6. *Н. Питц-Моултис, Ч. Кирк.* XML. СПб.: БХВ-Петербург, 2001. 736 с
7. *Г.Буч, А.Якобсон, Дж. Рамбо.* UML. СПб.: Питер, 2006. 736 с.
8. *Никифоров В. В*.. Представление структуры сигнальных связей алгоритмических процессов в гибких производственных системах // Интегрированные производственные комплексы / Под общ. ред. В.М. Пономарева. – Л.: Машиностроение, 1987. С. 42–46.
9. *Котов В.Е.* Сети Петри. М.: Наука, 1984. 215 с.
10. *Питерсон Дж*. Теория сетей Петри и моделирование систем. М.: Мир, 1984. 263 с.

Никифоров Виктор Викентьевич — д-р техн. наук, проф.; в. н. с. лаборатории технологий и систем программирования Санкт-Петербургского института информатики и автоматизации РАН. Область научных интересов: программирование систем реального времени, встроенных систем, операционные системы. Число научных публикаций — 90. nik@iias.spb.su; СПИИРАН, 14-я линия В.О., д. 39, Санкт-Петербург, 199178, РФ; р.т. +7(848)328-0887.

Nikiforov Victor Vikentievitch — Dr.Sci. (Tech.), Prof.; Senior researcher, Laboratory for Software Technology and Systems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: methods for real-time software development, embedded systems, operating systems. The number of publications — 90. nik@iias.spb.su; SPIIRAS, 39, 14th Line V.O., St. Petersburg, 199178, Russia; office phone +7(848)328-0887.

Шкиртиль Вячеслав Иванович — канд. техн. наук, доцент; заведующий лабораторией технологий и систем программирования Санкт-Петербургского института информатики и автоматизации РАН. Область научных интересов: Программное обеспечение систем реального времени. Число научных публикаций — 60. jvatlas@mail.rcom.ru; СПИИРАН, 14-я линия В.О., д. 39, Санкт-Петербург, 199178, РФ; р.т. +7(848)328-0887.

Shkirtil Viacheslav Ivanovitch — PhD (Tech.), associate professor; Dr.Sci., Head of the Laboratory for Software Technology and Systems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: software development for real-time systems. The number of publications — 60. nik@iias.spb.su; SPIIRAS, 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone +7(848)328-0887.

РЕФЕРАТ

Никифоров В.В., Шкиртиль В.И. **Маршрутные сети – графический формализм представления структуры программных приложений реального времени.**

Данный формализм ориентирован на проверку логической и динамической корректности межзадачных интерфейсов. Формы маршрутных сетей позволяют получить наглядные представления о структуре межзадачных связей в многозадачных программных приложениях для СРВ. Этот формализм, ориентированный на эффективное визуальное представление структуры системы, полностью согласован с разработанным авторами форматом представления структуры межзадачных связей средствами языка XML — форматом, пригодным для непосредственного автоматического анализа свойств конструируемых систем. Средства разметки маршрутных сетей, аналогичные разметке сетей Петри, позволяют наряду со статическими структурными особенностями многозадачных программных приложений представлять конкретные состояния системы, возникающие в ходе исполнения приложения, в частности, состояния с кольцами взаимных ожиданий.

Универсальным подходом к проверке системы на возможность возникновения колец взаимных ожиданий (тупиков и «клинчей») является построение (на основе структур типа маршрутных сетей) графа состояний и переходов системы. При использовании 32-разрядных компьютеров с оперативной памятью в несколько гигабайт этот подход применим, если программное приложение содержит до 20 задач. В случае существенно большего числа задач возможно либо применение специализированных подходов к анализу логической корректности приложений, либо использование таких динамических механизмов обхода колец взаимных ожиданий, как различные варианты протоколов пороговых приоритетов.

В СРВ с разделением доступа к информационным ресурсам при проверке гарантий своевременности выполнения отдельных задач и проверке выполнимости программного приложения в целом необходимы оценки значений фактора блокирования высокоприоритетных задач низкоприоритетными задачами, занимающими разделяемые ресурсы. Применение методов оценки фактора блокирования на базе параметрических моделей, не учитывающих особенности внутренней структуры задач, может привести к недооценке значений времени отклика задач. Для нахождения точных оценок фактора блокирования в системах со сцепленными критическими интервалами необходимо учитывать особенности внутренней структуры блокирующих задач. С этой целью можно использовать моделирование задач маршрутными сетями с построением профилей блокирующих задач и анализ тех сечений построенных профилей, которые соответствую блокируемым задачам.

SUMMARY

Nikiforov V.V., Shkirtil V.I. **Itinarary nets – the graphical formalizm for presentation of real-time sofware application strucrures.**

Itinerary network — graphical formalism for presentation structures of software applications. The formalism is focused on check logical and dynamic correctness of inter-task links in multitask software applications. The forms of itinerary nets allow to build a visual representation of inter-task links structures in software applications for real-time systems (RTS). This formalism, focused on efficient visual presentation of RTS structure, is conformed to the forms of textual formalism on the base of Extensible Markup Language (XML) that was previously suggested by authors (this XML formalism is suitable for direct automatic analysis of the developed system properties). The itinerary net markup tools are similar to the markup tools of Petri nets: they permit to represent not only static structural characteristics of multitasking software applications, but also separate system states that occur during system execution — particularly the states with rings of mutual waiting.

Universal approach to checking the possibility of rings of mutual expectations (deadlocks and clinches) is formation of oriented graph — the graph of system states and transitions (such graph may be formed on the base of corresponding itinerary net). If you are using 32-bit computers with several GB of RAM this approach is suitable if the software application contains up to 20 tasks. In the case of significantly more tasks you may either use specialized approaches to evaluating logical correctness of application or use of dynamic mechanisms that ensure avoidance of the states with rings of mutual waiting (such property is provided by variants of priority ceiling protocols).

Task response time estimation or feasibility analysis of the whole software application for RTS with mutual access to information resources requires take into account the values of blocking factor for high priority tasks by low priority tasks that occupy mutual resources. The methods for blocking factors estimation which are based on parametric models with ignoring of task internal structure can lead to underestimation of task response time. To find the exact evaluation of blocking factor for systems with concatenated critical intervals it is necessary to take into account the peculiarities of the structure of blocking tasks. To do this, you can use modeling of developed system by itinerary networks with building of blocking task profiles and with analysis of those cuttings of such profiles, which correspond to blocked tasks.