5. *Неусыпин К. А., Фам Суан Фанг*. Алгоритмические методы повышения точности навигационных систем ЛА. Ханой: Мир, 2009. 126 с.

6. Управление и наведение беспилотных маневренных летательных аппаратов на основе современных информационных технологий / Под ред. *М. Н. Красильщикова и Г. Г. Серебрякова*. М.: Физматлит, 2003. 280 с.

7. *Ковальчук И. А., Кошеля И. А.* Алгоритм вычисления нижней границы ковариаций ошибок оценивания при нелинейной фильтрации // Радиоэлектроника. 1985. Т. 28, № 7. С. 82—84.

Сведения об авторах

Павел Валерьевич Васильев — канд. техн. наук, доцент; Военно-космическая академия им. А. Ф. Можайского, Санкт-Петербург; E-mail: vasp1971@mail.ru

Алла Вячеславовна Мелешко — канд. техн. наук; ОАО «НПП „Радар ММС“», Санкт-Петербург; ведущий специалист; E-mail: allaluna@list.ru

Вячеслав Викторович Пятков — д-р техн. наук, профессор; ОАО „НИИ телевидения“, Санкт-Петербург; начальник научно-технического комплекса; E-mail: pyatkov@niitv.ru

Рекомендована
НИИ телевидения

Поступила в редакцию
24.04.14 г.

УДК 681.3

В. В. Никифоров

ПРОТОКОЛ ПРЕДОТВРАЩЕНИЯ ВЗАИМНОГО БЛОКИРОВАНИЯ ЗАДАЧ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Разработан протокол доступа прикладных задач к глобальным информационным ресурсам в системах реального времени. Протокол позволяет применять дисциплины планирования с переменными приоритетами задач, что обеспечивает существенное повышение эффективности использования процессорного времени в системах с многоядерными процессорами.

Ключевые слова: *многозадачные системы, системы на многоядерных процессорах, системы реального времени, взаимосвязанные задачи, протоколы доступа к ресурсам.*

Введение. Программные приложения для систем реального времени (СРВ) строятся в виде фиксированного набора задач τ_1, τ_2, ..., τ_n. Очередная (*j*-я) активизация задачи τ_i означает порождение ее очередного (*j*-го) экземпляра — *задания* $\tau_i^{(j)}$. Порядок предоставления задачам процессорного времени определяется применяемой дисциплиной планирования. Для СРВ важно выбрать дисциплину планирования, гарантирующую своевременное выполнение задач при эффективном использовании ресурсов. Проверка гарантий своевременности выполнения прикладных задач τ_1, τ_2, ..., τ_n осуществляется с учетом максимального объема C_i процессорного времени, требуемого для однократного исполнения задачи τ_i и *периода* T_i (минимально допустимого интервала времени между двумя активизациями задачи τ_i) [1, 2].

При решении прикладных задач, совместно использующих глобальные (разделяемые) информационные ресурсы, требуются механизмы, обеспечивающие: *а*) целостность ресурсов, *б*) предотвращение взаимного блокирования задач, ожидающих доступа к разделяемым информационным ресурсам.

Целостность информационных ресурсов обеспечивается, например, синхронизирующими механизмами типа мьютексов (*mutex*), а с целью предотвращения взаимного блокирования используются стандартные *протоколы доступа* к ресурсам [2, 3], ориентированные на применение дисциплин планирования со статическими приоритетами задач.

В настоящей статье предлагается протокол доступа, обеспечивающий предотвращение взаимного блокирования в условиях применения дисциплин планирования с динамически модифицируемыми приоритетами: разработчик может выбирать дисциплину планирования, позволяющую достичь требуемой эффективности использования ресурса процессора.

Дисциплины планирования. Простейшие способы проверки гарантий своевременности выполнения набора прикладных задач τ_1, τ_2, ... , τ_n опираются на определение значения *нагрузки* $u_i = C_i/T_i$ — доли процессорного времени, требуемого для исполнения задачи τ_i. Для оценки выполнимости программного приложения существенно значение *интегральной нагрузки* $U = \sum_{i=1}^{n} u_i$. Очевидно, что в случае однопроцессорной системы с классическим одноядерным процессором необходимо, чтобы значение интегральной нагрузки U не превышало единицы. В случае многоядерного процессора с m ядрами необходимо, чтобы выдерживалось неравенство $U \le m$.

К дисциплинам планирования с наиболее жесткими ограничениями на модификацию приоритетов относятся дисциплины со статическими приоритетами задач (в ходе работы системы приоритеты задач, определенные при ее разработке, остаются неизменными). Смягчение ограничений на модификацию приоритетов задач позволяет выбирать дисциплины планирования, обеспечивающие более высокую эффективность использования процессорного времени.

На рис. 1 представлены три класса дисциплин планирования. Класс S_0 соответствует дисциплинам со статическими приоритетами задач, к нему относится широко применяемая в СРВ RM-дисциплина: приоритеты задач τ_1, τ_2, ..., τ_n снижаются с увеличением значений T_i. Для системы на одноядерном процессоре эффективность (т.е. гарантированная выполнимость задач τ_1, τ_2, ..., τ_n) дисциплины RM выражается равенством $U \le \ln 2 \cong 0,69$.

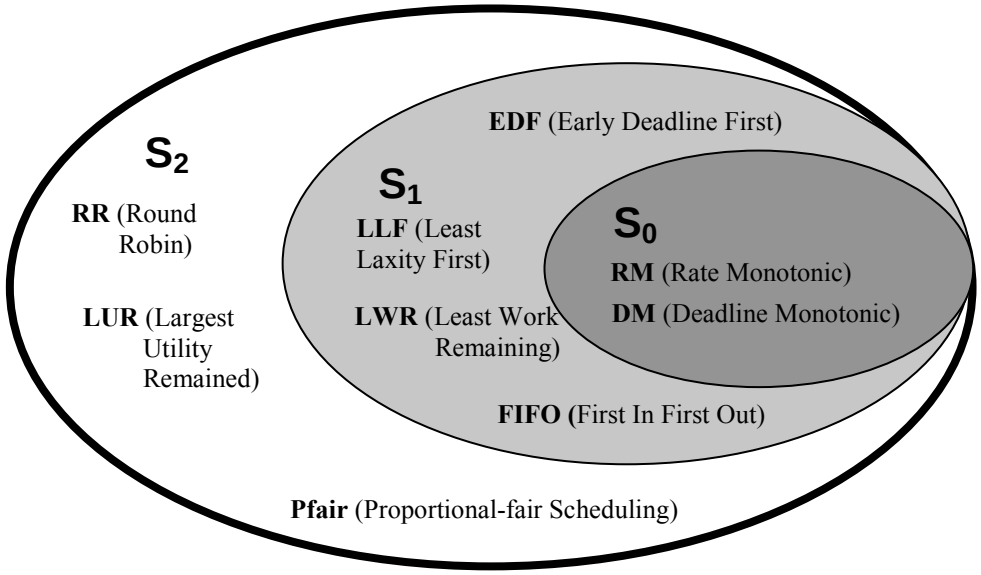

Рис. 1

Для дисциплин планирования класса S_1 приоритеты однотипных заданий $\tau_i^{(j)}$ и $\tau_i^{(k)}$ ($j \ne k$) могут различаться, т.е. экземпляры одной задачи могут исполняться с разными приоритетами. Ограничение на модификацию приоритетов для дисциплин планирования класса

S_1 состоит в том, что в рамках всего интервала существования конкретного задания $\tau_i^{(j)}$ его приоритет остается неизменным. Благодаря смягчению (относительно класса S_0) ограничений на модификацию приоритетов применение дисциплин класса S_1 позволяет повысить эффективность использования процессора. Так, дисциплина планирования EDF (приоритеты заданий снижаются в порядке размещения на оси времени предельно допустимых моментов их завершения) обеспечивает максимально возможную эффективность использования процессорного времени в однопроцессорных СРВ с классическим одноядерным процессором. При применении дисциплины EDF в таких СРВ выполнение неравенства $U \le 1$ является не только необходимым, но и достаточным условием своевременности выполнения задач.

При построении СРВ на многоядерных процессорах применение RM и EDF неэффективно [4], для таких СРВ разработаны модифицированные варианты дисциплин планирования. Дисциплина RM_US класса S_0 обеспечивает выполнимость всех задач при $U \le \frac{m^2}{3m-2}$: ее эффективность изменяется от 1/2 до 1/3 в зависимости от числа ядер процессора.

Классу S_2 соответствует полное снятие ограничений на модификацию приоритетов заданий — приоритеты активных заданий могут изменяться в ходе их исполнения. Такое снятие ограничений открывает возможность повышения эффективности использования процессора в СРВ на многоядерных процессорах. Дисциплина Pfair обеспечивает выполнимость всех задач приложения при $U \le m$, что означает стопроцентную эффективность использования ресурса процессора [5, 6].

Протоколы доступа к разделяемым ресурсам. Участок кода задачи, в рамках которого реализуется доступ к глобальному ресурсу g, называют *критическим интервалом* по ресурсу g. К *однотипным* относятся различные критические интервалы, в рамках которых реализуется доступ к одному глобальному ресурсу. Обеспечить целостность разделяемых ресурсов позволяют механизмы, предотвращающие одновременное исполнение однотипных критических интервалов различными заданиями. С этой целью для каждого глобального ресурса g_i в программном коде формируется синхронизирующий элемент типа мьютекса `mut_i`. Каждый участок кода программы, реализующий доступ к g_i (каждый из *критических интервалов* по доступу к ресурсу g_i), ограничивается операторами над мьютексом `mut_i`. Критический интервал начинает оператор `lock(mut_i)` — закрыть мьютекс `mut_i` и заканчивает `unlock(mut_i)` — открыть мьютекс `mut_i`. Если в момент обращения задания к оператору `lock(mut)` мьютекс находится в состоянии „закрыт“, то исполнение приостанавливается до тех пор, пока владеющее ресурсом задание не освободит его выполнением операции `unlock(mut)`. Операторы `lock/unlock` разбивают код задачи на сегменты.

На рис. 2 в соответствии с предложенным в работе [7] подходом к представлению межзадачных интерфейсов средствами языка XML приведена структура взаимосвязей двух задач. Задачи τ_1 и τ_2 разделяют ресурсы g_1 и g_2, критические интервалы защищены мьютексами `mut_1` и `mut_2`.

Каждая из задач (см. рис. 2) содержит два критических интервала: интервал задачи τ_1 по ресурсу g_1 содержит 2-й и 3-й сегменты ее кода, по ресурсу g_2 — ее 3-й и 4-й сегменты. Критический интервал задачи τ_2 по ресурсу g_1 состоит из 3-го сегмента ее кода, по ресурсу g_2 — 2, 3, 4-го. Существенно, что обе задачи содержат пересекающиеся критические интервалы:

— в τ_1 критические интервалы сцеплены — 3-й сегмент является, с одной стороны, завершающим сегментом критического интервала по g_1, а с другой — головным сегментом по g_2;

— в τ_2 критический интервал по g_1 вложен в интервал по g_2.

Задача τ_1

```
<thread name="task_1" prio="1">
    <segment length="1"
             interface="mutex_1"
             op_type="get"/>
    <segment length="3"
             interface="mutex_2"
             op_type="get"/>
    <segment length="1"
             interface="mutex_1"
             op_type="put"/>
    <segment length="4"
             interface="mutex_2"
             op_type="put"/>
    <segment length="1"
             op_type="end"/>
</thread>
```

Задача τ_2

```
<thread name="task_2" prio="2">
    <segment length="1"
             interface="mutex_2"
             op_type="get"/>
    <segment length="4"
             interface="mutex_1"
             op_type="get"/>
    <segment length="2"
             interface="mutex_1"
             op_type="put"/>
    <segment length="4"
             interface="mutex_2"
             op_type="put"/>
    <segment length="1"
             op_type="end"/>
</thread>
```

Рис. 2

При наличии в программном приложении задач с пересекающимися критическими интервалами возможно взаимное блокирование активных заданий. Такая ситуация может возникнуть при исполнении приложения, содержащего задачи τ_1 и τ_2 со структурой кода, представленной на рис. 2.

Рассмотрим следующий сценарий. Пусть задача τ_2 активизируется в момент времени $t = 0$ и исполняется до $t = 3$, когда активизируется более приоритетная задача τ_1 и процессор переключается на ее исполнение. На интервале (3, 7) исполняются 1-й и 2-й сегменты τ_1. В момент $t = 7$ в результате обращения к операции `lock(mut_2)` исполнение τ_1 приостанавливается, поскольку запрашиваемый ресурс g_2 занят задачей τ_2. Процессор переключается на исполнение τ_2 — с момента $t = 7$ до $t = 10$ завершается исполнение 2-го сегмента τ_2. В момент $t = 10$ при обращении к операции `lock(mut_1)` происходит приостановка исполнения τ_2, поскольку запрашиваемый ресурс g_1 занят задачей τ_1. Таким образом, задания τ_1 и τ_2 попадают в состояние взаимного блокирования: τ_1 не может продолжаться, пока τ_2 не освободит ресурс g_2, τ_2 не может освободить g_2, пока не получит доступа к занятому ресурсу g_1.

Стандартный подход к предотвращению взаимного блокирования задач опирается на снабжение синхронизирующих механизмов типа мьютексов дополнительными условиями и/или действиями (проверяемыми и/или осуществляемыми) при выполнении операций над мьютексами. Состав этих дополнительных условий/действий называют протоколами доступа к разделяемым информационным ресурсам.

Стандартные протоколы доступа предполагают применение дисциплин планирования класса S_0 [2, 8]. Поэтому при построении систем, содержащих пересекающиеся критические интервалы, разработчики ориентируются на использование протоколов доступа, исключающих применение эффективных дисциплин планирования классов S_1 или S_2. Однако наличие пересекающиеся критических интервалов не означает взаимного блокирования задач. В работе [9] предложен метод проверки реальной возможности возникновения взаимного блокирования, основанный на анализе структурных особенностей специального многодольного графа — графа связок критических интервалов. На основе метода разработчик может проверить, действительно ли конфигурация взаимосвязей задач требует использования стандартных

протоколов доступа к разделяемым ресурсам. Если такая проверка дает отрицательный результат (т.е. несмотря на наличие пересекающихся критических интервалов возникновение взаимного блокирования задач невозможно), то при реализации рассматриваемой многозадачной системы можно применять высокоэффективные дисциплины планирования классов S_1 или S_2.

Граф связок. Критические интервалы задачи τ по основному (головному) g и дополнительному g^* ресурсам назовем связанными (образующими *связку* $L =< \tau, g, g^* >$), если они пересекаются, т.е. содержат общие сегменты кода задачи. Каждая связка L состоит из трех участков, на начальном (*головном*) участке задача τ имеет доступ к *головному ресурсу* g связки. На центральном участке τ имеет доступ и к g и и g^*. На завершающем один из ресурсов уже освобожден задачей τ. Центральный участок образуется пересечением связанных критических интервалов. В структуре приложения на рис. 2 имеются две связки: $L_1 =< \tau_1, g_1, g_2 >$ и $L_2 =< \tau_2, g_2, g_1 >$.

Необходимым условием возникновения взаимных ожиданий является наличие в программном приложении таких пар связок, для которых имеет место следующее отношение зависимости. Связка $L_x =< \tau_i, g_a, g_b >$ зависима от $L_y =< \tau_j, g_c, g_d >$, если τ_i и τ_j — разные задачи и $g_b \equiv g_c$, то есть 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 L_x. Имеющиеся в системе зависимости связок могут быть представлены в виде многодольного ориентированного *графа связок* [9]. Каждая связка критических интервалов представляется отдельной вершиной, если L_x зависит от L_y, то вершина L_x графа связок соединяется с вершиной L_y дугой, ведущей из L_x в L_y. Граф связок для программного приложения (см. рис. 2) представлен на рис. 3.

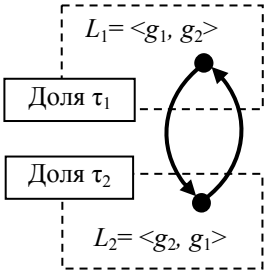

Рис. 3

Для проверки возможности возникновения взаимного блокирования задач следует в графе связок выделить множество междольных контуров $W = \{w_1, w_2, ...,\}$. Контур w_k является междольным, если в нем нет двух вершин, принадлежащих одной и той же доле. Выражение $L \in w_k$ отражает факт принадлежности связки L контуру w_k. Условимся обозначать символом $|w_k|$ число связок, принадлежащих контуру w_k. В общем случае связка L может:

— принадлежать одному, двум, либо большему числу контуров w_k из W;

— не принадлежать ни одному из контуров w_k.

В работе [9] показано, что взаимное блокирование задач возможно в том, и только в том случае, если множество W не пустое. Тот факт, что для приложения со структурой, приведенной на рис. 2, взаимное блокирование возможно, отражается наличием междольного контура в графе связок на рис. 3.

Протокол предотвращения взаимного блокирования. Условимся считать, что в ходе исполнения задачи τ связка $L =< \tau, g, g^* >$ находится в активном состоянии (является *активной*), если исполняется ее головной участок. Две связки, принадлежащие задаче τ, могут одновременно пребывать в активном состоянии только в том случае, если пересекаются их головные участки. Следовательно, при отсутствии подобных пересечений в ходе исполнения τ в каждый момент времени активной может быть не более одной из принадлежащих ей связок.

Отмеченное обстоятельство позволяет для приложений, свободных от пересечений головных участков связок, использовать протокол предотвращения взаимного блокирования (ППВБ), который

— гарантирует логическую корректность системы, содержащей контуры межзадачных зависимостей;

— может быть использован в условиях применения любых дисциплин планирования, в частности, с динамически модифицируемыми приоритетами задач.

Реализация ППВБ опирается на учет текущего числа активных связок в каждом из контуров $w_k \in W$, учет обеспечивается введением соответствующих счетчиков:

— значение счетчика $q(w_k)$ увеличивается на единицу в тот момент времени, когда одна из связок, принадлежащих w_k, переходит в активное состояние;

— значение счетчика $q(w_k)$ уменьшается на единицу в тот момент времени, когда одна из связок, принадлежащих w_k, перестает быть активной.

В рамках использования стандартных протоколов, предотвращающих взаимное блокирование задач, проверка/модификация значений мьютексов сопрягается с дополнительными проверками/действиями, выполняемыми при входе/выходе для каждого критического интервала по ресурсам. В рамках использования ППВБ дополнительные проверки/действия выполняются только при входе в пересекающиеся критические интервалы и, более того, только при входе в интервалы, являющиеся элементами одной из связок $L =< \tau, g, g^* >$. Дополнительные проверки/действия для ППВБ состоят в следующем.

1. При запросе заданием $\tau_i^{(j)}$ головного ресурса связки $L =< \tau, g, g^* >$ проверяется выполнение неравенств $q(w_k)+1 < |w_k|$ для каждого из контуров w_k, содержащих эту связку. Если хотя бы одно из неравенств не выполняется, то исполнение $\tau_i^{(j)}$ приостанавливается до тех пор, пока не будет обеспечено выполнение всех проверяемых неравенств.

2. При входе задания $\tau_i^{(j)}$ в головной участок связки $L =< \tau, g, g^* >$ значение счетчиков $q(w_k)$ каждого из контуров $L \in w_k$ увеличивается на единицу.

3. При входе задания $\tau_i^{(j)}$ в дополнительный участок связки $L =< \tau, g, g^* >$ значение счетчиков $q(w_k)$ каждого из контуров $L \in w_k$ уменьшается на единицу.

Основное преимущество ППВБ состоит в том, что выполняемые в его рамках дополнительные проверки/действия не связаны с приоритетами задач и он может использоваться не только с дисциплинами планирования класса S_0, но и с более эффективными дисциплинами планирования классов S_1 или S_2.

Заключение. Предложенный протокол предотвращения взаимного блокирования задач в системах реального времени обеспечивает эффективную реализацию программных приложений, содержащих задачи с пересекающимися критическими интервалами по доступу к разделяемым информационным ресурсам. Возможность использования стандартных протоколов доступа к разделяемым ресурсам ограничена требованием применения дисциплин планирования со статическими приоритетами задач, на представленный протокол это ограничение не распространяется. Это расширяет для разработчиков СРВ возможности широкого выбора дисциплины планирования. При построении СРВ на многоядерных процессорах такая возможность может обеспечить повышение эффективности использования процессорного времени в два-три раза.

Работа выполнена при государственной финансовой поддержке ведущих университетов РФ (субсидия 074-U01).

СПИСОК ЛИТЕРАТУРЫ

1. *Зыль С.* Проектирование, разработка и анализ программного обеспечения систем реального времени. СПб: БХВ-Петербург, 2010. 336 с.

2. *Liu J. W. S.* Real-Time Systems. NJ: Prentice Hall, 2000. 590 p.

3. *Никифоров В. В., Шкиртиль В. И.* Составное блокирование взаимосвязанных задач в системах на многоядерных процессорах // Изв. вузов. Приборостроение. 2012. Т. 55, № 1. С. 25—31.

4. *Dhall S. K., Liu C. L.* On a Real-Time Scheduling Problem // Operating Research. 1978. Vol. 26, N 1. P. 127—140.

5. *Никифоров В. В.* Выполнимость приложений реального времени на многоядерных процессорах // Тр. СПИИРАН. СПб: Наука, 2009. Вып. 8. С. 255—284.

6. *Baker T.* Multiprocessors EDF and Deadline Monotonic Schedulability Analysis // Proc. of the 24th IEEE Real-Time Systems Symposium. 2003. P. 120—129.

7. *Никифоров В. В., Шкиртиль В. И.* Спецификация средствами языка XML системы интерфейсов в приложениях реального времени // Тр. СПИИРАН. СПб: Наука, 2009. Вып. 11. С. 159—175.

8. *Никифоров В. В., Шкиртиль В. И.* Маршрутные сети — графический формализм представления структуры программных приложений реального времени // Тр. СПИИРАН. СПб: Наука, 2010. Вып. 14. С. 7—28.

9. *Никифоров В. В., Павлов В. А.* Операционные системы реального времени для встроенных программных комплексов // Программные продукты и системы. 1999. № 4. С. 24—30.

Сведения об авторе

Виктор Викентьевич Никифоров — д-р техн. наук, профессор; Санкт-Петербургский институт информатики и автоматизации РАН, лаборатория технологий и систем программирования; E-mail: nik@iias.spb.su

Рекомендована СПИИРАН

Поступила в редакцию
16.04.14 г.