УДК 004.04

С.Н. Баранов, В.В. Никифоров

ТРАНЗИТИВНОЕ НАСЛЕДОВАНИЕ ПРИОРИТЕТОВ В МНОГОЗАДАЧНЫХ ПРИЛОЖЕНИЯХ РЕАЛЬНОГО ВРЕМЕНИ

Баранов С.Н., Никифоров В.В. **Транзитивное наследование приоритетов в многозадачных приложениях реального времени.**

Аннотация. Рассматриваются методы контроля доступа задач к разделяемым ресурсам в программных приложениях для систем реального времени. Приводится детальное представление двух процедур наследования приоритетов задач: непосредственной и транзитивной. Сформулированы достаточные условия, при которых применение непосредственной процедуры предотвращает инверсию приоритетов. Предложена модификация транзитивной процедуры снимающая известные ограничения на структуру приложения, накладываемые ее традиционной реализацией. Эта модификация, кроме того, обеспечивает динамическое обнаружение некорректных ситуаций типа взаимного блокирования задач с возможностью запланированной реакции на такие ситуации.

Ключевые слова: системы реального времени, модели многозадачных приложений, выполнимость задач, протоколы доступа к разделяемым ресурсам.

Baranov S.N., Nikiforov V.V. **Transitive Priority Inheritance in Real-Time Multi-Task Applications.**

Abstract. Control procedures for accessing shared resources in multi-task real-time software applications are analyzed. Two approaches to preventing priority inversion – direct and transitive procedures of priority inheritance – are analyzed in detail. To illustrate the considered notions and statements, concrete examples of multi-task application configurations are provided along with their execution diagrams under particular scenarios of system events, which demonstrate insufficient response time of tasks with sufficient computing resources and even mutual task clinches. The nomenclature of attributes to be included into task and resource descriptors for task management with the two priority inheritance procedures is proposed. The sufficient conditions for the direct procedure to prevent priority inheritance are formulated. The procedure of transitive priority inheritance is demonstrated to be capable of detecting a mutual task clinch in the application.

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

1. Введение. Программные продукты, предоставляющие пользователю широкий спектр функциональных возможностей, обычно строятся в виде многозадачных приложений, состоящих из ряда задач τ_1, τ_2, …, τ_n, каждой из которых предоставляется доступ к различным активным (исполнительным) и пассивным (информационным) ресурсам как локальным – доступным лишь одной задаче, так и глобальным – попеременно доступным разным задачам. Такое построение программных продуктов типично для систем реального времени (СРВ) [1], встроенных систем, систем имитационного моделирования и систем, ориентированных на исполнение с использованием многопроцессорных компьютеров и многоядерных процессоров [2].

При создании многозадачных программных приложений обычно возникают две проблемы:

1) определение порядка предоставления задачам ресурса процессора (процессорного времени), гарантирующего своевременность их исполнения [3];

2) обеспечение целостности глобальных информационных ресурсов (т.е., предотвращение одновременного доступа к нему двух и более заданий) [4].

Первая проблема решается применением дисциплины планирования, обеспечивающей выполнимость задач – гарантированную своевременность их выполнения. Эффективные дисциплины планирования для однопроцессорных систем с классическими одноядерными процессорами предлагались с начала 1970-х годов – в частности RM (Rate-Monotonic) и EDF (Earliest Deadline First) дисциплины планирования [5]. В работе [6] было показано, что RM и EDF могут терять свою эффективность при использовании многопроцессорных систем. В 2000-е годы предлагались модификации этих дисциплин планирования, обеспечивающие приемлемую эффективность для систем на многоядерных процессорах [7]. В настоящее время как отечественными (например, [8]), так и зарубежными (например, [9], [10]) исследователями продолжается поиск эффективных дисциплин планирования. Специально для многоядерных процессоров разработаны высокоэффективные Pfair (справедливо-пропорциональные) дисциплины планирования с квантованием [11] и без квантования [12] интервалов выделяемого процессорного времени. Вторая проблема решается путем использования протоколов доступа к разделяемым информационным ресурсам [13].

2. Структура и параметры задач. Будем считать каждую задачу последовательной программой, замкнутой в себе по передачам управления. Задачи активизируются в порядке реакции на внешние события через какие-то промежутки времени, не меньшие некоторой величины – их периода. Таким образом, каждая задача τ_i характеризуется своим периодом T_i, весом C_i – объемом процессорного времени, необходимого для выполнения ее вычислительной работы, предельным сроком D_i для завершения ее исполнения, и фазой P_i, задающей момент первой активизации данной задачи. Очевидно, что для всех задач должно выполняться неравенство $C_i \le D_i$.

Очередная (k-ая) активизация задачи τ_i означает порождение задания ${}_k\tau_i$ (порождение очередного k-ого экземпляра задачи τ_i). Задание является активным в некоторый конкретный момент времени, если к этому моменту оно уже порождено, но еще не завершено. То есть, ка-

ждое задание является активным в рамках интервала его существования – от момента его порождения до момента его завершения. Порождение задания означает увеличение числа претендентов на ресурс процессора (процессорное время) и, следовательно, изменение условий его распределения. Поскольку изменение условий распределения системных ресурсов в общем случае является следствием какого-либо системного события, то порождение задания следует рассматривать как его разновидность. Другим примером системного события является завершение задания, приводящее к уменьшению числа претендентов на ресурс процессора.

Участок кода задачи τ_i, в рамках которого осуществляется доступ к какому-либо из глобальных информационных ресурсов g, называется критическим интервалом по доступу к этому ресурсу [14]. Для обеспечения целостности глобального информационного ресурса g_x используются механизмы контроля доступа к критическим интервалам, которые строятся на базе синхронизирующих элементов – мьютексов [16]. Для каждого разделяемого информационного ресурса g_x формируется мьютекс *m_x*, принимающий одно из двух значений: «открыт», если ресурс свободен, и «закрыт», если ресурс занят. Критический интервал по доступу к ресурсу g_x обрамляется специальными системными операторами над соответствующим мьютексом *m_x*. На входе в критический интервал выполняется системный оператор запроса ресурса g_x *lock(m_x),* переводящий мьютекс *m_x* из состояния «открыт» в состояние "закрыт", на выходе – оператор освобождения ресурса g_x *unlock(m_x),* переводящий этот мьютекс из состояния «закрыт» в состояние «открыт». Исполнение задания, запрашивающего занятый ресурс, приостанавливается до момента его освобождения. Различные критические интервалы по доступу к одному и тому же ресурсу g_x называются однотипными критическими интервалами.

Операторы *lock* предполагают системное событие, поскольку приводят либо к занятию глобального информационного ресурса, либо (если ресурс уже занят) к приостановке текущего задания с соответствующим уменьшением числа претендентов на ресурс процессора. Аналогично, операторы *unlock* также вызывают системное событие, поскольку приводят к освобождению глобального информационного ресурса и в случае возобновления ожидающего его задания – увеличение числа претендентов на процессорное время.

Структура задачи представляется в виде конечной последовательности сегментов, где каждый сегмент выполняет некоторое вычисление в течение некоторого отрезка процессорного времени (длина сегмента) и завершается одним из следующих системных событий:

«занять ресурс», «освободить ресурс», либо «завершить задачу». Вес задачи равен сумме длин ее сегментов, поскольку предполагается, что продолжительность системных событий, завершающих каждый сегмент, пренебрежительно мала.

Порядок предоставления задачам процессорного времени определяется используемой дисциплиной планирования. Любую дисциплину планирования можно выразить в виде способа наделения заданий целочисленными приоритетами, по которому активным задачам присваиваются определенные значения приоритетов, а ресурс процессора предоставляется наиболее приоритетной из активных задач. В большинстве реализаций многозадачных приложений приоритеты задач назначаются статически. Условимся индексировать задачи τ_1, τ_2, …, τ_n, составляющие приложение, в порядке снижения их приоритета: τ_1 – наиболее приоритетная задача, τ_n – наименее приоритетная, так что номер задачи задает ее приоритет.

Таким образом, в каждый момент времени своего существования активное задание ${}_k\tau_i$ может находиться в одном из трех состояний:

• использует ресурс процессора (является текущим заданием);

• ожидает освобождения процессора более приоритетным заданием;

• ожидает освобождения ресурса, занятого другим активным заданием.

В общем случае при исполнении многозадачных приложений, в которых целостность глобальных информационных ресурсов обеспечивается механизмами защиты одновременного попадания различных задач в однотипные критические интервалы, возможно возникновение таких некорректных ситуаций, как инверсия приоритетов и взаимное блокирование задач [16]. В первом случае менее приоритетное задание занимает ресурс процессора, тогда как более приоритетное задание ожидает освобождения информационного ресурса, занятого этим или другим менее приоритетным заданием; во втором – каждое из блокирующих заданий ожидает освобождения информационного ресурса, занятого другим заданием. Для предотвращения инверсии приоритетов предлагается дополнить дисциплину планирования (способ назначения приоритетов активным заданиям) функцией наследования приоритетов. Ниже описаны два варианта реализации этой функции – упрощенное непосредственное и более сложное транзитивное наследование приоритетов. Сформулированы условия, при которых использование непосредственного наследования приоритетов гарантирует предотвращение инверсии приоритетов. Представлен также способ такой модификации транзитивного наследования приоритетов, который позво-

ляет динамически обнаруживать возникновение ситуаций типа взаимного блокирования задач с возможностью запланированной реакции на такую ситуацию (выход из нее).

3. Инверсия приоритетов. Представляемые ниже особенности реализации функции наследования приоритетов иллюстрируются диаграммами исполнения многозадачного приложения из четырех задач τ_1, τ_2, τ_3 и τ_4, разделяющих два ресурса g_1 и g_2. Структура задач изображена на рис.1 в соответствии с предложенным в [17] подходом к представлению межзадачных интерфейсов средствами языка XML.

Задача τ_1

```
<task name="t_1" prio="1"
                 period="15">
  <segment length="1"
           interface="m_1"
           op_type="lock"/>
  <segment length="1"
           interface="m_1"
           op_type="unlock"/>
  <segment length="1"
           op_type="end"/>
</task>
```

Задача τ_2

```
<task name="task_1" prio="2"
                    period="35">
  <segment length="9"
           op_type="end"/>
</task>
```

Задача τ_3

```
<task name="t_3" prio="3"
                 period="25">
  <segment length="1"
           interface="m_1"
           op_type="lock"/>
  <segment length="2"
           interface="m_2"
           op_type="lock"/>
  <segment length="1"
           interface="m_2"
           op_type="unlock"/>
  <segment length="1"
           interface="m_1"
           op_type="unlock"/>
  <segment length="1"
           op_type="end"/>
</task>
```

Задача τ_4

```
<task name="t_1" prio="1">
                 period="45">
  <segment length="2"
           interface="m_2"
           op_type="lock"/>
  <segment length="4"
           interface="m_2"
           op_type="unlock"/>
  <segment length="1"
           op_type="end"/>
</task>
```

Рис. 1. Вариант приложения из четырех задач, разделяющих два ресурса

Имеющая самый высокий приоритет задача τ_1 использует ресурс g_1, доступ к которому контролируется мьютексом *m_1*. Код задачи τ_1 состоит из трех сегментов, требующих для своего исполнения по одной единице процессорного времени. Первый сегмент заканчивается операцией *lock(m_1)* запроса доступа к ресурсу g_1. Второй сегмент – критический интервал по доступу к g_1 – заканчивается операцией *unlock(m_1)* освобождения этого ресурса. Заключительный третий сегмент заканчивается операцией завершения задачи.

Код задачи τ_2 состоит из единственного сегмента, который заканчивается оператором завершения задачи и требует для своего исполнения 9 единиц процессорного времени. Значение 2 ее приоритета

означает, что задача τ_2 менее приоритетна, чем задача τ_1, но более приоритетна, чем τ_3 и τ_4.

Код задачи τ_3 содержит 5 сегментов. Первый заканчивается операцией *lock(m_1)* запроса доступа к ресурсу g_1. Критический интервал по доступу задачи τ_3 к ресурсу g_1 включает второй, третий и четвертый сегменты ее кода. Второй сегмент заканчивается запросом доступа к ресурсу g_2. В рамках третьего сегмента, который завершается освобождением ресурса g_2, задача τ_3 владеет как ресурсом g_1, так и ресурсом g_2. Четвертый сегмент завершает критический интервал по доступу к g_1, а пятый сегмент завершает исполнение задачи τ_3.

Код наименее приоритетной задачи τ_4 содержит 3 сегмента: начальный сегмент длиной 2; второй сегмент, являющийся критическим интервалом по доступу задачи τ_4 к ресурсу g_2 длиной 4, и заключительный сегмент длиной 1.

Периоды T_1, T_2, ${}_1T_3$ и T_4 активизации задач равны соответственно 15, 35, 25 и 45 единицам времени, а их фазы равны 5, 5, 3 и 0 единиц времени соответственно. Условимся считать, что для каждой из задач τ_i предельный срок выполнения равен ее периоду: $D_i = T_i$.

Состав действий, реализуемых при исполнении операций *lock* и *unlock*, определяется выбором протокола доступа к глобальным информационным ресурсам. Простейший протокол, обеспечивающий сохранение целостности каждого из ресурсов g_x, предписывает формирование для каждого ресурса g_x синхронизирующего элемента типа mutex с полями состояние («открыт» или «закрыт») и список заданий, ожидающих освобождения данного ресурса.

При инициализации мьютекса, соответствующего ресурсу g, его поле состояния получает значение «открыт» (т.е., ресурс свободен), а список заданий, ожидающих освобождения ресурса, делается пустым.

В реализации простейшего протокола выполнение операций *lock* и *unlock* сводится к действиям, приведенным в таблице 1. При непустом списке заданий, ожидающих освобождения ресурса, текущее задание, выполняющее операцию *unlock(m_x)*, может быть вытеснено более приоритетным новым владельцем ресурса g_x, стоявшим в начале этого списка.

На рисунке 2*а* приведена диаграмма исполнения приложения из четырех задач с параметрами, соответствующими рисунку 1, при использовании протокола доступа, представленного в таблице 1. Диаграмма соответствует конкретному сценарию системных событий (конкретному порядку активизации задач) при котором возникает инверсия приоритетов, приводящая к задержке исполнения высокоприоритетных заданий.

Таблица 1. Реализация простейшего протокола доступа к глобальным ресурсам

Операция	Условие	Изменения состояний задач и ресурсов
lock(m)	Мьютекс *m*, соответствующий ресурсу *g*, открыт	Мьютекс m переводится в состояние "закрыт". Задание, выполнившее операцию *lock(m)*, становится владельцем ресурса *g*, приступает к исполнению критического интервала по этому ресурсу.
	Мьютекс *m*, соответствующий ресурсу *g*, закрыт	Дескриптор задания, выполняющего операцию *lock(m)*, переносится из списка претендентов на ресурс процессора в список заданий, ожидающих освобождения ресурса *g*
unlock(m)	Список заданий, ожидающих освобождения ресурса *g*, пуст	Мьютекс m переводится в состояние «открыт». Задание выполнившее операцию *unlock(m)*, продолжает исполняться
	Список заданий, ожидающих освобождения ресурса *g*, не пуст	Задание, возглавляющее список ожидающих освобождения ресурса *g*, становится владельцем ресурса *g*, соответствующего мьютексу *m*; его дескриптор переносится из этого списка в список претендентов на ресурс процессора

В таблице 2 приведен комментарий к диаграмме на рисунке 2*а*: перечень действий, выполняемых в моменты возникновения каждого из системных событий. Для упрощения изложения вторая активизация задачи τ_1 (т.е. порождение задания ${}_2\tau_1$ в момент времени после t=20) не рассматривается.

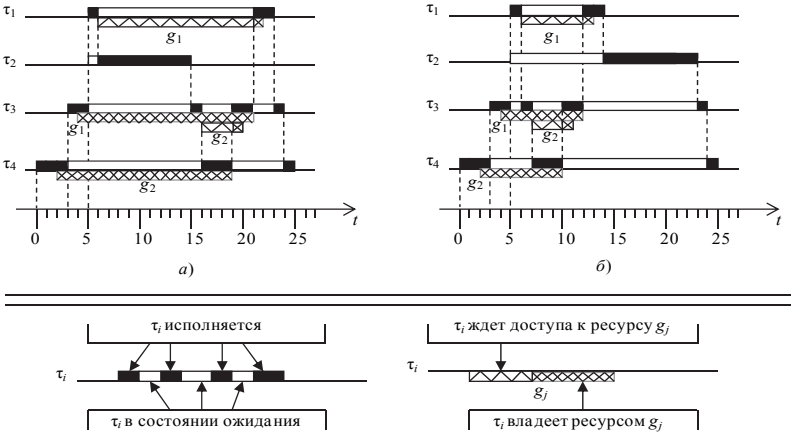

Рис. 2. Исполнение комплекса задач, использующих разделяемые ресурсы

Таблица 2. Сценарий системных событий, вызывающий инверсию приоритетов

Время	Событие	Изменения состояний заданий и ресурсов
t=0	Активизация задачи τ_4	Порождается и инициализируется задание ${}_1\tau_4$ типа τ_4, процессор переключается на его исполнение.
t=2	Запрос ресурса g_2 заданием ${}_1\tau_4$	Задание ${}_1\tau_4$ запрашивает доступ к свободному ресурсу g_2. Ресурс g_2 предоставляется в его распоряжение и задание ${}_1\tau_4$ приступает к исполнению своего критического интервала по доступу к нему.
t=3	Активизация задачи τ_3	Порождается задание ${}_1\tau_3$ типа τ_3. Приоритет τ_3 выше, чем у τ_4, вследствие чего задание ${}_1\tau_3$ инициализируется и вытесняет задание ${}_1\tau_4$ – процессор переключается на исполнение ${}_1\tau_3$.
t=4	Запрос ресурса g_1 заданием ${}_1\tau_3$	Задание ${}_1\tau_3$ запрашивает доступ к свободному ресурсу g_1. Ресурс g_1 ему предоставляется, и процессор продолжает исполнение ${}_1\tau_3$ уже в рамках критического интервала по доступу к g_1.
t=5	Активизация задач τ_1 и τ_2	Порождаются задания ${}_1\tau_1$ типа τ_1 и ${}_1\tau_2$ типа τ_2. Приоритет τ_1 выше, чем приоритет текущего задания ${}_1\tau_3$, – выполняется инициализация задания ${}_1\tau_1$, оно вытесняет ${}_1\tau_3$, процессор переключается на исполнение ${}_1\tau_1$.
t=6	Запрос ресурса g_1 заданием ${}_1\tau_1$	Задание ${}_1\tau_1$, запрашивает доступ к ресурсу g_1. Поскольку ресурс g_1 занят заданием ${}_1\tau_3$, задание ${}_1\tau_1$ переводится в состояние ожидания момента освобождения g_1. Освободившийся таким образом процессор переключается на исполнение наиболее приоритетного из заданий, готовых использовать ресурс процессора. Таким заданием является задание ${}_1\tau_2$ – оно инициализируется, и процессор приступает к его исполнению.
t=15	Завершение задания ${}_1\tau_2$	В результате завершения задания ${}_1\tau_2$ процессор освобождается и переключается на продолжение исполнения задания ${}_1\tau_3$, а менее приоритетное задание ${}_1\tau_4$ остается в состоянии ожидания момента предоставления ему процессора.
t=16	Запрос ресурса g_2 заданием ${}_1\tau_3$	Ресурс g_2 занят заданием ${}_1\tau_4$ – исполнение задания ${}_1\tau_3$ приостанавливается, процессор переключается на исполнение задания ${}_1\tau_4$.
t=19	Задание ${}_1\tau_4$ освобождает ресурс g_2	Завершается исполнение критического интервала задания ${}_1\tau_4$ по ресурсу g_2. Освободившийся ресурс g_2 предоставляется ожидающему его заданию ${}_1\tau_3$. Процессор переключается с задания ${}_1\tau_4$ на исполнение более приоритетного задания ${}_1\tau_3$.
t=20	Задание ${}_1\tau_3$ освобождает ресурс g_2	Завершается исполнение пересечения критических интервалов задания ${}_1\tau_3$ по ресурсам g_1 и g_2. Список заданий, претендующих на ресурс процессора, не изменяется. Процессор остается в распоряжении задания ${}_1\tau_3$.
t=21	Задание ${}_1\tau_3$ освобождает ресурс g_1	Освободившийся ресурс g_1 предоставляется ожидающему его более приоритетному заданию ${}_1\tau_1$. Процессор переключается на исполнение задания ${}_1\tau_1$.
t=23	Завершение задания ${}_1\tau_1$	Процессор переключается на исполнение завершающих участков кода задания ${}_1\tau_3$ и затем задания ${}_1\tau_4$.

Максимальная продолжительность исполнения заданий типа τ_i обозначается символом R_i и называется временем отклика задачи τ_i. Требование своевременности исполнения отдельной задачи τ_i, входящей в состав приложения, выражается неравенством $R_i \le D_i$.

По рисунку 2*а* видно: продолжительность интервала существования задания ${}_1\tau_1$, порождаемого в момент t=5 и заканчивающегося в момент t=23, равна 18, что превышает заданный предельный срок D_1=15. Следовательно, для приложения на рисунке 1 при использовании протокола согласно таблице 1 требование $R_i \le D_i$ не выполняется.

Предельная продолжительность исполнения задачи τ_1 нарушается, несмотря на то, что она является самой приоритетной из всех задач и для ее исполнения требуется лишь 3 единицы процессорного времени, тогда как остальные 15 единиц процессорного времени на интервале (5, 23) расходуются менее приоритетными заданиями ${}_1\tau_2$, ${}_1\tau_3$ и ${}_1\tau_4$. Из этих 15-ти единиц 6 расходуются блокирующими задачами τ_3 и τ_4. В течение 3-х единиц времени (15, 19–20) задание ${}_1\tau_3$ блокирует задание ${}_1\tau_1$ непосредственно, занимая ресурс g_1, требующийся заданию ${}_1\tau_1$. Еще три единицы времени (16-18) расходуются заданием ${}_1\tau_4$ для освобождения ресурса g_2, без доступа к которому задание ${}_1\tau_3$ не может завершить критический интервал по ресурсу g_1. Но в рамках используемого протокола (таблица 1) большая часть интервала (5, 23) расходуется на исполнение задания ${}_1\tau_2$, несмотря на то, что у него и приоритет ниже, чем у τ_1, и сама задача τ_2 не связана ни прямо, ни опосредованно с требуемым задачей τ_1 ресурсом. При добавлении в приложение других среднеприоритетных задач (задач со значениями приоритетов ниже, чем у задачи τ_1, но выше, чем у задач τ_3 и τ_4) их исполнение может еще более тормозить исполнение самой приоритетной задачи τ_1.

Для обозначения рассмотренного влияния среднеприоритетных задач на увеличение времени отклика высокоприоритетной задачи, разделяющей глобальные ресурсы с низкоприоритетными задачами, используется термин *инверсия приоритетов*. Предотвращение инверсии приоритетов достигается путем дополнения протокола доступа задач к глобальным ресурсам функцией наследования приоритетов.

4. Непосредственное наследование приоритетов. Отличительной особенностью протокола доступа к ресурсам с наследованием приоритетов является возможность изменения приоритета заданий в процессе их исполнения. При этом различаются приоритет инициализации (исходный приоритет) задания и его действующий (текущий) приоритет. Приоритет инициализации равен приоритету задачи, из которой порождается данное задание; он один и тот же для всех таких заданий. Действующий приоритет устанавливается равным приоритету инициализации задания в момент его порождения и в дальнейшем может изменяться, в частности, через механизм наследования приоритетов.

Принцип наследования приоритетов выражается следующим образом: если при выполнении операции *lock(m_x)* высокоприоритетным заданием ${}_m\tau_i$ ресурс g_x, соответствующий мьютексу *m_x*, занят менее приоритетным заданием ${}_n\tau_j$, то действующий приоритет задания ${}_n\tau_j$ временно, до освобождения им ресурса g_x, устанавливается равным действующему приоритету задания ${}_m\tau_i$.

Для реализации наследования приоритетов структуру для дескрипторов заданий следует дополнить полями для приоритета инициализации и действующего приоритета, а структуру мьютекса – полем для указателя на дескриптор задания, владеющего данным ресурсом в настоящий момент.

При дополнении протокола доступа к разделяемым ресурсам функцией наследования приоритетов процедуры, приведенные в таблице 1, должны быть дополнены действиями для предотвращения инверсии приоритетов. В таблице 3 приведен их перечень для непосредственного наследования приоритетов.

Таблица 3. Непосредственное наследование приоритетов

Операция	Условие	Действия, выполняемые в добавление к приведенным в таблице 1
lock(m)	Мьютекс *m*, соответствующий ресурсу *g*, открыт	Указатель на дескриптор текущего задания ${}_x\tau_i$ сохраняется в структуре мьютекса.
	Мьютекс *m*, соответствующий ресурсу *g*, закрыт	Действующий приоритет задания ${}_y\tau_j$, владеющего ресурсом *g*, повышается до значения действующего приоритета текущего задания
unlock(m)	Задание ${}_y\tau_j$, выполняющее операцию *unlock(m),* владеет и другими ресурсами с непустыми списками ожидающих заданий	В этих списках ищется задание ${}_x\tau_i$ с максимальным значением действующего приоритета. Приоритет задания ${}_y\tau_j$, освобождающего ресурс, делается равным высшему из двух приоритетов: – действующий приоритет ${}_x\tau_i$, – приоритет инициализации ${}_y\tau_j$. Указатель на дескриптор задания – нового владельца данного ресурса сохраняется в структуре мьютекса.
	Таких ресурсов нет	Действующий приоритет текущего задания делается равным его приоритету инициализации. Указатель на дескриптор нового владельца данного ресурса сохраняется в структуре мьютекса.

На рисунке 2*б* приведена диаграмма исполнения примера приложения (рисунок 1) при использовании протокола доступа (таблица 1), дополненного реализацией непосредственного наследования приоритетов (таблица 3). Диаграмма соответствует тому же сценарию

системных событий, что и в диаграмме рисунок 2*а*, но благодаря дополнительным действиям (таблица 3), исполнение задания ${}_1\tau_2$ выносится за рамки интервала существования задания ${}_1\tau_1$. В результате нарушение предельного срока исполнения задания ${}_1\tau_1$ не происходит.

Предлагаемые в классической работе [16] действия по выполнению операции *unlock* ориентированы на соблюдение следующего структурного ограничения: допускаются только вложенные критические интервалы (если два критических интервала пересекаются, то один из них должен быть вложен в другой). Такое требование вложенности критических интервалов, с одной стороны, позволяет существенно упростить реализацию процедуры *unlock*, но, с другой стороны, сужает возможности программистов, вынужденных учитывать приведенное ограничение. Модификация процедуры *unlock*, приведенная в таблице 3, пригодна для обслуживания приложений с любыми вариантами пересечений критических интервалов.

5. Отношения предшествования и транзитивное наследование приоритетов. В ходе отраженного на рисунке 2*б* исполнения приложения, соответствующего рисунку 1, при запросе в момент времени t=6 ресурса g_1 заданием ${}_1\tau_1$ обнаруживается, что требуемый ресурс занят заданием ${}_1\tau_3$. Так возникает отношение предшествования ${}_1\tau_1 \succ {}_1\tau_3$: прежде, чем сможет продолжиться исполнение задания ${}_1\tau_1$, задание ${}_1\tau_3$ должно завершить исполнение критического интервала по ресурсу g_1. В соответствии с таблицей 3, в момент t=6 действующий приоритет задания ${}_1\tau_3$ устанавливается равным действующему приоритету задания ${}_1\tau_1$. Далее, в момент времени t=7 аналогичным образом возникает отношение предшествования ${}_1\tau_3 \succ {}_1\tau_4$, действующий приоритет задания ${}_1\tau_4$ устанавливается равным действующему приоритету ${}_1\tau_3$. Поскольку к моменту t=7 еще сохраняется отношение ${}_1\tau_1 \succ {}_1\tau_3$, возникает цепочка ${}_1\tau_1 \succ {}_1\tau_3 \succ {}_1\tau_4$ отношений предшествования. При сценарии системных событий, отраженном на рисунке 2, по этой цепочке в соответствии с процедурой непосредственного наследования приоритетов действующий приоритет задания ${}_1\tau_1$ передается (транзитивно, через ${}_1\tau_3$) заданию ${}_1\tau_4$. Для предотвращения инверсии приоритетов процедура наследования должна при любом сценарии обеспечивать транзитивную передачу действующего приоритета через всю цепочку отношений предшествования [18].

Для приложения, представленного на рисунке 1, возможны такие сценарии системных событий, при которых непосредственное наследование приоритетов не обеспечивает передачу действующего приоритета заблокированного задания по всей цепочке отношений предшествования. Пример приведен на рисунке 3.

Сценарий системных событий, отражаемых диаграммами на рисунках 3*в* и 3*б*, отличается от сценария на рисунках 2*а* и 2*б* только тем, что задачи τ_1 и τ_2 активизируются в момент t=7 вместо t=5. В этих условиях непосредственное наследование приоритетов не предотвращает инверсии приоритетов (рисунок 3*а*).

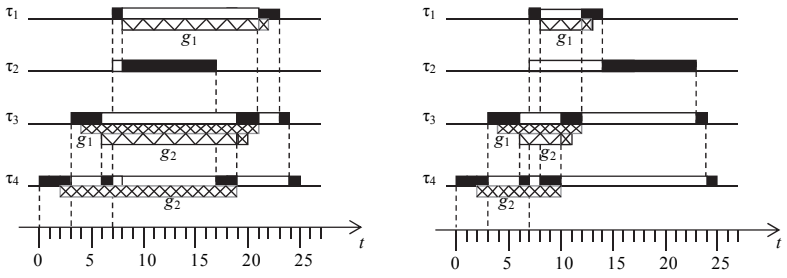

Рис. 3. Пример реализации процедур наследования приоритетов: а) непосредственное наследование приоритетов; б) транзитивное наследование приоритетов

К моменту порождения задания ${}_1\tau_1$ уже возникло (в момент t=6) отношение предшествования ${}_1\tau_3 \succ {}_1\tau_4$, и в соответствии с таблицей 3, действующий приоритет задания ${}_1\tau_4$ был повышен до приоритета инициализации ${}_1\tau_3$. Далее, в момент t=8 добавляется отношение ${}_1\tau_1 \succ {}_1\tau_3$ и возникает цепочка ${}_1\tau_1 \succ {}_1\tau_3 \succ {}_1\tau_4$. Однако непосредственное наследование приоритетов не обеспечивает транзитивную передачу действующего приоритета через всю эту цепочку от задания ${}_1\tau_1$ к заданию ${}_1\tau_4$. Действительно, в момент t=8 в соответствии со второй строкой таблицы 3 действующий приоритет задания ${}_1\tau_3$, владеющего ресурсом g_1, становится равным наивысшему приоритету – приоритету инициализации задания ${}_1\tau_1$. Но действующий приоритет текущего задания ${}_1\tau_2$ остается равным его приоритету инициализации. Поэтому процессор переключается на исполнение задания ${}_1\tau_2$, чей приоритет выше действующего приоритета задания ${}_1\tau_4$. По этой причине непосредственное наследование приоритетов, как показано на рисунке 3*а*, не предотвращает возникновения инверсии приоритетов – оно не обеспечивает передачу действующего приоритета ${}_1\tau_1$ через всю возникшую цепочку отношений предшествования.

6. Транзитивная процедура наследования приоритетов. Для предотвращения инверсии приоритетов при любом сценарии системных событий процедуру непосредственного наследования приоритетов необходимо дополнить действиями, обеспечивающими транзитивную передачу действующего приоритета блокируемого задания через всю

цепочку отношений предшествования. Ниже приведены дополнения в процедуру наследования приоритетов, обеспечивающие такую транзитивную передачу. Получающуюся в результате процедуру назовем транзитивной процедурой наследования приоритетов. Отметим, что в любой момент времени функционирования многозадачного приложения количество отношений предшествования ограничено числом активных заданий.

Для выполнения транзитивной процедуры следует дополнить дескриптор задания полем для номера ресурса, блокирующего исполнение этого задания.

При порождении задания в этом поле помещается значение, свидетельствующее о том, что задание не находится в состоянии ожидания глобального ресурса. В дальнейшем, при переходе задания в состояние ожидания перед входом в критический интервал, охраняемый мьютексом, в это поле заносится номер блокирующего ресурса.

Дополнения, переводящие процедуру непосредственного наследования приоритетов в транзитивную, касаются действий, выполняемых в рамках операции *lock(m_x)*, если мьютекс *m_x* закрыт.

При выполнении текущим заданием операции *lock(m_x)* вслед за действиями, приведенными в таблицах 1 и 3, в дескриптор этого задания помещается номер *m_x* блокирующего ресурса. После этого необходимо обработать всю цепочку начинающихся с этого задания отношений предшествования, чтобы повысить действующий приоритет каждого из них.

Обработка цепочки отношений предшествования состоит в циклическом выполнении приведенной ниже последовательности шагов (шаги 1-3). В их описании используется символ τ^*. В рамках первой итерации цикла символ τ^* означает задание, владеющее ресурсом и указатель на дескриптор которого сохранен в структуре мьютекса. В рамках каждой последующей итерации τ^* означает очередное задание из цепочки отношений предшествования.

Шаг 1. Перед входом в этот шаг действующий приоритет задания τ^* уже повышен. Проверяется поле дескриптора задания τ^* с номером ресурса, освобождения которого оно ожидает. Его нулевое значение говорит о том, что задание τ^* не блокировано и находится в списке заданий, претендующих на использование процессора. Отсюда следует, что обработка цепочки отношений предшествования завершена – в цепочке не осталось заданий, которым необходимо повысить действующий приоритет. Выполняется выход из цикла.

Шаг 2 (выполняется при ненулевом значении проверенного поля). Выявляется задание, требующее (транзитом) повышения своего

действующего приоритета. Таковым является задание, владеющее ресурсом с номером из проверенного поля. Символ τ* с этого момента означает новое звено цепочки отношений предшествования – найденное задание.

Шаг 3. Действующий приоритет задания τ* повышается до значения действующего приоритета текущего задания, выполняющего операцию *lock(m_x)*. Выполняется переход к шагу 1 – к очередной итерации цикла по обработке цепочки отношений предшествования.

Результат применения такой транзитивной процедуры наследования приоритетов представлен на рисунке 3*б*. Инверсия приоритетов предотвращена.

7. Влияние инверсии приоритетов на выполнимость задач. Ключевым требованием к приложению реального времени является обеспечение своевременности выполнения составляющих его задач. Задача τ_i называется выполнимой, если максимально возможная продолжительность ее исполнения (время отклика R_i) не превосходит заданного значения предельного срока D_i: $R_i \le D_i$. Приложение называется выполнимым, если гарантируется выполнимость каждой из составляющих его задач.

Возможность возникновения инверсии приоритетов в системе, не оснащенной механизмом наследования приоритетов, не обязательно приводит к нарушению выполнимости. Оценка выполнимости программных приложений СРВ осуществляется известными методами на этапе проектирования по заданной конфигурации приложения [14].

Для приложений, допускающих инверсию приоритетов, учет этого обстоятельства может быть включен в методы оценки выполнимости. Механизм наследования приоритетов становится излишним, если оценки R_i времени отклика задач, увеличенные вследствие инверсии приоритетов, остаются в рамках заданных предельных сроков D_i.

Кроме того, инверсия приоритетов возможна не во всяком многозадачном приложении с разделяемыми ресурсами. Для ее возникновения необходимо наличие среднеприоритетных задач между двумя разделяющими общий ресурс высокоприоритетной и низкоприоритетной задачами.

При работе приложений, не содержащих задач с пересекающимися критическими интервалами, нет необходимости включения механизмов транзитивного наследования приоритетов в управление заданиями. Наличие таких пересечений является необходимым условием возникновения цепочек отношений предшествования (в конфигурации приложения на рисунке 1 такое пересечение имеет место в задаче τ_3).

Вместе с тем, не во всех приложениях, содержащих задачи с пересекающимися критическими интервалами, возможно возникновение цепочек отношений предшествования. Так, если в приложении на рисунке 1 задаче τ_3 назначить низший приоритет, то цепочка отношений предшествования не возникнет (будет достаточно использовать процедуру непосредственного наследования приоритетов). Необходимость использования транзитивного наследования приоритетов может возникнуть только для приложений, допускающих цепное блокирование задач. Метод проверки наличия в приложении условий для возникновения цепного блокирования представлен в [18].

8. Динамическое обнаружение и выход из ситуаций взаимного блокирования заданий. На структуру программных приложений, использующих традиционный вариант процедуры наследования приоритетов [16], накладываются два ограничения. Одно из них – отмеченное в разделе 3 требование вложенности пересекающихся критических интервалов. Другое структурное ограничение – требование гарантированной живучести. В ходе работы приложений, отвечающих требованию гарантированной живучести, невозможно возникновение ситуаций типа взаимного блокирования заданий – ситуаций, отличающихся тем, что в ориентированном графе, отражающем текущий состав отношений предшествования ${}_i\tau_x \succ {}_j\tau_y$ исполняемых заданий, возникают замкнутые маршруты (контуры). Классический пример структуры приложения из пяти задач с пятью ресурсами, не отвечающий требованию живучести, приведен в работе [19].

В системах реального времени использование приложений, не отвечающих требованию живучести, недопустимо. Отсюда следует, что программные комплексы для систем реального времени, ориентированные на использование традиционного варианта транзитивной процедуры наследования приоритетов, должны статически (на этапе проектирования) проверяться на живучесть. Объем вычислений для такой проверки растет экспоненциально с ростом числа взаимосвязанных задач. Для приложений, содержащих десятки взаимосвязанных задач, выполнение такой проверки может оказаться нереальным. В этом случае обеспечение живучести может обеспечиваться динамически путем встраивания процедуры транзитивного наследования приоритетов в механизм обработки исключительных ситуаций.

В ходе исполнения программного приложения, не отвечающего требованию живучести, возможно возникновение состояния, в котором процедура транзитивного наследования приоритетов обнаружит, задание, выявляемое на шаге 2, совпадает с тем заданием, которое обратилось к операции *unlock* (цепочка отношений предшествования

замыкается, что соответствует возникновению взаимного блокирования заданий). Для приложений, не отвечающих требованию живучести, шаг 2 процедуры транзитивного наследования должен быть дополнен проверкой на возникновение такого совпадения. Если при этом выполнение операции *unlock* встроено в механизм обработки исключительных ситуаций (*unlock* выполняется в рамках оператора *try*), то информация о попытке замыкания отношений предшествования может быть передана из контекста подсистемы управления заданиями в контекст приложения – в *catch*-ветвь оператора *try*. Таким образом приложение информируется о попытке взаимного блокирования заданий. Дальнейшие поведение приложения зависит от действий, заложенных проектировщиком приложения в *catch*-ветви. В частности, текущая задача может обойти ситуацию взаимного блокирования путем досрочного освобождения одного из уже занятых ею ресурсов. Тем самым обеспечивается динамическое обнаружение и выход из ситуаций взаимного блокирования заданий.

9. Заключение. В многозадачных программных приложениях реального времени требование целостности разделяемых задачами глобальных информационных ресурсов обеспечивается защитой критических интервалов по доступу к разделяемым ресурсам специальными синхронизирующими элементами типа мьютексов. При исполнении таких приложений возникают отношения предшествования между критическими интервалами взаимосвязанных задач. Возникающая в результате этого инверсия приоритетов может привести к нарушению предельных сроков выполнения задач. Предотвращение инверсии приоритетов обеспечивается либо непосредственной, либо транзитивной процедурой наследования приоритетов. Применение упрощенной непосредственной процедуры гарантирует предотвращение инверсии приоритетов в случае, если в структуре задач отсутствуют пересечения критических интервалов. При наличии таких пересечений могут возникать цепочки отношений предшествования. В этом случае предотвращение инверсии приоритетов достигается применением транзитивной процедуры. Предложенная в настоящей статье модификация транзитивной процедуры со встраиванием операции освобождения ресурса в механизм обработки исключительных ситуаций обеспечивает обнаружение и выход из ситуаций типа взаимного блокирования заданий.

Литература

1. *Давиденко К.Я.* Технология программирования АСУТП. Проектирование систем реального времени, параллельных и распределенных приложений // М.: Энергоатомиздат, 1985. 183 с.
2. *Baker T.* Multiprocessors EDF and Deadline Monotonic Schedulability Analysis // Proceedings of 24 IEEE Real-Time Systems Symposium. 2003. pp. 120–129.

3. *Таненбаум Э*. Современные операционные системы // СПб.: Питер. 2010. 1037 с.
4. *Сорокин С.В.* Системы реального времени: операционные системы // Современные технологии автоматизации. 1997. №2. С. 22–31.
5. *Liu C., Layland J.* Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment // Journal of the ACM. 1973. vol. 20. no. 1. pp. 46–61.
6. *Dhall S.K.*, *Liu C.L.* On a Real-Time Scheduling Problem // Operating Research. 1978. vol. 26. no. 1. pp. 127–140.
7. *Никифоров В.В.* Выполнимость приложений реального времени на многоядерных процессорах // Труды СПИИРАН. 2009. Вып. 8. С. 255–284.
8. *Докучаев А.Н*. К оценке эффективности механизмов диспетчеризации мультипроцессорных систем реального времени с учетом влияния длительных блокировок // Программная инженерия. 2012. №9. С. 2–7.
9. *Anderson B.* Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38% // Proceedings of the7th International Conference on Principles of Distributed Systems. Egypt. 2008. pp. 73–88.
10. *Baker T.P., Cirinei M., Bertogna M*. EDZL scheduling analysis // Real-Time Systems 2008. vol. 40. no. 3. pp. 264–289.
11. *Baruah S.K*. Fairness in Periodic Real-Time Scheduling Algorithms // Proceedings of 16 IEEE Real-Time Symposium. 1995. pp. 200–209.
12. *Cho H., Ravindran B., Jensen D*. An Optimal Real-Time Scheduling Algorithm for Multiprocessors // Proceedings of the 27 IEEE Real-Time Symposium. 2006. pp. 101–110.
13. *Liu J.W.S*. Real-Time Systems // NJ: Prentice Hall. 2000. 590 p.
14. *Laplante P.A*. Real-Time Systems Design and Analysis // John Wiley & Sons, Inc. 2004. 530 p.
15. *Данилов М.В.* Методы планирования выполнения задач в системах реального времени // Программные продукты и системы. 2001. №4. С. 28–35.
16. *Sha L., Rajkumar R., Lehoczky J.P*. Priority Inheritance Protocols: An Approach to Real-Time Synchronization // IEEE Transactions on Computers. 1990. vol. 20. no. 9. pp.1175–1185.
17. *Никифоров В.В., Шкиртиль В.И.* Спецификация средствами языка XML системы интерфейсов в приложениях реального времени // Труды СПИИРАН. 2009. Вып 11. С.159–175.
18. *Никифоров В.В., Шкиртиль В.И.* Цепное блокирование взаимосвязанных задач в системах на многоядерных процессорах // Информационно-измерительные и управляющие системы. 2013. №9. С. 17–21.
19. *Dijkstra E.W*. Hierarchical ordering of sequential processes // Acta Informatica. 1971. vol. 1. no. 2. pp. 115–138.

References

1. Davidenko K.Ya. *Tekhnologiya programmirovaniya ASUTP. Proyektirovaniye sistem realnogo vremeni, parallelnyh i raspredelennyh prilozheniy* [Technology of programming with CAD/CAM. Design of real-time systems, parallel and distributed applications]. M. 1985. 183 p. (In Russ.).
2. Baker T. Multiprocessors EDF and Deadline Monotonic Schedulability Analysis. Proceedings of 24 IEEE Real-Time Systems Symposium. 2003. pp. 120–129.
3. Tannenbaum E. *Sovremennye operatsionnye sistemy* [Modern operating systems]. St.Petersburg. 2010. 1037 p. (In Russ.)
4. Sorokin S.V. [Real-time systems: operating systems]. *Sovremennye tekhnologii avtomatizatsii – Modern technologies of automation*. 1997. vol. 2. pp. 22–31. (In Russ.).
5. Liu C., Layland J. Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment. *Journal of the ACM*. 1973. vol. 20. no.1. pp. 46–61.

6. Dhall S.K., Liu C.L. On a Real-Time Scheduling Problem. *Operating Research*. 1978. vol. 26. no. 1. pp. 127–140.
7. Nikiforov V.V. [Feasibility of real-time applications on multi-core processors]. *Trudy SPIIRAN – SPIIRAS Proceedings*, issue 8. 2009. pp. 255–284. (In Russ.).
8. Dokuchayev A.N. [To esrimating the efficiency pf scheduling mechanisms of multi-processor real-time systems taking into account long-term clinches.]. *Programmnaya inzheneriya – Software engineering*. 2012. vol. 9. pp. 2–-7. (In Russ.).
9. Anderson B. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. Proceedings of the7th International Conference on Principles of Distributed Systems. Egypt. 2008. pp. 73–88.
10. Baker T.P., Cirinei M., Bertogna M. EDZL scheduling analysis. *Real-Time Systems*. 2008. vol. 40. no. 3. pp. 264–289.
11. Baruah S.K. Fairness in Periodic Real-Time Scheduling Algorithms. Proceedings of 16 IEEE Real-Time Symposium. 1995. pp. 200–209.
12. Cho H., Ravindran B., Jensen D. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. Proceedings of the 27 IEEE Real-Time Symposium. 2006. pp. 101–110.
13. Liu J.W.S. Real-Time Systems. NJ: Prentice Hall. 2000. 590 p.
14. Laplante P.A. Real-Time Systems Design and Analysis. John Wiley & Sons, Inc. 2004. 530 p.
15. Danilov M.V. [Scheduling methods od task execution in real-time systems]. *Programmnye produkty i sistemy – Software products and systems*. 2001. vol. 4. pp. 28–35. (In Russ.).
16. Sha L., Rajkumar R., Lehoczky J.P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*. 1990. vol. 20. no. 9. pp. 1175–1185.
17. Nikiforov V.V, Shkirtil V.I. [Specifying the interface system in real-time applications with XML]. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2009. vol. 11. pp.159–175. (In Russ.).
18. Nikiforov V.V, Shkirtil V.I. [Chained blocking of interrelated tasks in systems on multi-core processors]. *Informatsionno-izmeritelnye i upravlyushie sistemy – Informational-measuring and control systems*. 2013. vol. 9. pp. 17–21. (In Russ.).
19. Dijkstra E.W. Hierarchical ordering of sequential processes. *Acta Informatica*. 1971. vol. 1. no. 2. pp. 115–138.

Баранов Сергей Николаевич — д-р физ.-мат. наук, профессор, главный научный сотрудник лаборатории информационно-вычислительных систем и технологий программирования, Федеральное государственное бюджетное учреждение науки Санкт-Петербургского института информатики и автоматизации Российской академии наук (СПИИРАН), профессор, международная научная лаборатория Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики (ИТМО). Область научных интересов: технология программирования, формальные методы. Число научных публикаций — 100. snbaranov@googlemail.com; 14-я линия В.О., д. 39, Санкт-Петербург, 199178; р.т.: +7-812-328-0887.

Baranov Sergey Nikolaevich — Ph.D., Dr. Sci., professor, chief researcher of computing and information systems and programming technologies laboratory, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS), professor, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University). Research interests: software engineering, formal methods in software development. The number of publications — 100. snbaranov@googlemail.com; 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone: +7-812-328-0887.

Никифоров Виктор Викентьевич — д-р техн. наук, профессор, ведущий научный сотрудник лаборатории информационно-вычислительных систем и технологий программирования, Федеральное государственное бюджетное учреждение науки Санкт-Петербургского института информатики и автоматизации Российской академии наук (СПИИРАН). Область научных интересов: системы реального времени, встроенные системы, операционные системы. Число научных публикаций — 110. nik@iias.spb.su; 14-я линия В.О., д. 39, Санкт-Петербург, 199178; р.т.: +7(812)3280887.

Nikiforov Victor Vikentievich — Ph.D., Dr. Sci., professor, leading researcher of computing and information systems and programming technologies laboratory, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: real-time systems, embedded systems, operating systems. The number of publications — 110. nik@iias.spb.su; 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone: +7(812)3280887.

Поддержка исследований. Работа выполнена при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01), университет ИТМО.

Acknowledgements. This work was partially financially supported by the Government of the Russian Federation, Grant 074-U01.

РЕФЕРАТ

Баранов С.Н., Никифоров В.В. **Транзитивное наследование приоритетов в многозадачных приложениях реального времени.**

Программные продукты с широким спектром функциональных возможностей для пользователя обычно строятся в виде приложений, состоящих из ряда задач, каждой из которых предоставляется доступ к различным активным (исполнительным) и пассивным (информационным) ресурсам. При создании многозадачных программных приложений возникают две проблемы: 1) определение порядка предоставления задачам ресурса процессора (процессорного времени), гарантирующего своевременность их исполнения; 2) обеспечение целостности глобальных информационных ресурсов (т.е., предотвращение одновременного доступа к нему двух и более заданий). Если первая проблема решается применением дисциплины планирования, обеспечивающей корректность работы данного приложения, то вторая решается через использование специальных синхронизационных механизмов, предотвращающих одновременных доступ задач к общим ресурсам. Требование целостности разделяемых задачами информационных ресурсов обеспечивается защитой критических интервалов по доступу к ним специальными синхронизирующими механизмами типа мьютексов. При этом возникают отношения предшествования между критическими интервалами взаимосвязанных задач. Возникающая в результате этого инверсия приоритетов может привести к нарушению предельных сроков выполнения задач. Предотвращение инверсии приоритетов обеспечивается либо непосредственной, либо транзитивной процедурой наследования приоритетов. Процедура непосредственного наследования гарантирует предотвращение инверсии приоритетов в случае, если в структуре задач отсутствуют пересечения критических интервалов. При наличии таких пересечений могут возникать цепочки отношений предшествования. В этом случае предотвращение инверсии приоритетов достигается применением процедуры транзитивного наследования. В случае попадания подмножества задач в состояние взаимного блокирования транзитивная процедура позволяет обнаруживать этот факт. Предложенная в настоящей статье модификация транзитивной процедуры со встраиванием операции освобождения ресурса в механизм обработки исключительных ситуаций обеспечивает обнаружение ситуаций типа взаимного блокирования заданий с возможностью соответствующей реакции на них.

SUMMARY

Baranov S.N., Nikiforov V.V. **Transitive Priority Inheritance in Real-Time Multi-Task Applications.**

Software products with a wide range of functionality are usually built as multitask applications which consist of a number of tasks with access to various active (computational) and passive (informational) resources. Two problems often arise when developing such multi-task applications: 1) how to define the order of allocating the processor resource (processor time) to tasks which ensures their on-time execution; 2) how to preserve the integrity of global informational resources (i.e., how to exclude simultaneous access of two or more tasks to such a resource). While the first problem is resolved through a scheduling mode, which ensures the correctness of application execution, the second one is resolved via special synchronization mechanisms which prevent simultaneous task access to shared resources. The integrity of global resources shared among tasks is provided by protecting critical intervals of shared resources access with special synchronization mechanisms of the mutex type, which gives rise to the relation of precedence among critical intervals of interrelated tasks. The resulting priority inversion may cause a violation of task deadlines. Such priority inversion may be prevented by either a direct or a transitive procedure of priority inheritance. The direct procedure prevents priority inversion only if there are no intersections of critical intervals in the task structure. When such intersections do occur, chains of precedence relations may arise. In this case, priority inversion may be prevented with the proposed advanced transitive procedure. In addition, this procedure allows detecting an exception of mutual task clinch which a subset of tasks is trapped in during execution, with an option for a pre-planned reaction on such exceptions.