ЭФФЕКТИВНОЕ ИСПОЛЬЗОВАНИЕ ПАМЯТИ В СИСТЕМАХ, СОДЕРЖАЩИХ ПРОСТЫЕ И СОСТАВНЫЕ ЗАДАЧИ РЕАЛЬНОГО ВРЕМЕНИ

М. В. Данилов

Санкт-Петербургский институт информатики и автоматизации РАН
199178, Санкт-Петербург, 14-я линия ВО, д. 39
<mvd@dipaul.ru>

УДК 681.3.06
М. В. Данилов. **Эффективное использование памяти в системах, содержащих простые и составные задачи реального времени** // Труды СПИИРАН. Вып. 2, т. 2. — СПб.: Наука, 2005.
Аннотация. *Рассказывается об алгоритмах назначения приоритетов в системах реального времени, содержащих простые и составные задачи. Предлагаемые алгоритмы позволяют повысить эффективность использования оперативной памяти.* — Библ. 7 назв.

UDC 681.3.06
M. V. Danilov. **Effective memory use in systems inclusive simple and complex real-time tasks** // SPIIRAS Proceedings. Issue 2, vol. 2. — SPb.: Nauka, 2005.
Abstract. *Algorithms of priorities assignment for real-time systems inclusive simple and complex tasks are discussed. The algorithms provide RAM use efficiency increase.* — Bibl. 7 items.

Введение

Разработчики встроенных систем реального времени уделяют большое внимание вопросам повышения эффективности использования аппаратных ресурсов. Наибольший интерес вызывает проблема минимизации требований программных систем к оперативной памяти, так как оперативное запоминающее устройство является одним из наиболее дорогостоящих. В многозадачных системах реального времени использование общего стека для всех простых задач позволяет существенно сократить требования системы к оперативной памяти. Однако желаемый эффект достигается только в случае, если параметры планирования задачам были назначены соответствующим образом. В настоящей статье рассматриваются предлагаемые автором методы назначения задачам параметров планирования. Эти методы позволяют минимизировать размер общего стека задач.

1. Общий стек

Согласно общепризнанному определению, системами реального времени (СРВ) называются системы, для которых специфицированы ограничения на время выполнения решаемых задач [1]. В данной статье рассматриваются только жесткие СРВ, т.е. такие системы, для которых нарушение установленных сроков выполнения задач недопустимо.

Системное программное обеспечение для СРВ должно быть ориентировано на обеспечение своевременного выполнения прикладных задач. Выполнение временных ограничений многозадачных приложений обеспечивается за счет применения операционных систем реального времени (ОСРВ). В ОСРВ используются механизмы оптимального в терминах СРВ вытесняющего планирования, основанного на приоритетах. Требование вытесняемости задач обя-

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

зывает ОС откладывать выполнение текущей задачи, если ресурс процессора требует задача с большим приоритетом. Задачи с равными приоритетами обслуживаются в порядке поступления.

Существует ряд моделей задачи реального времени [2]. Задача может быть многоразовой или одноразовой. Тело многоразовой задачи, код, определяющий ее функциональность, содержится в цикле, часто бесконечном. Внутри цикла, перед телом задачи, ставится системный вызов ожидания события, по которому задача должна активироваться. Модель многоразовой задачи является традиционной. Ее существенным недостатком является невысокая точность моментов активации задач, так как синхронизация осуществляется на прикладном уровне. В отличие от многоразовых задач, одноразовая задача порождается при каждом возникновении инициирующего события. После выполнения тела задачи, она завершается. Причем тело задачи выполняется только один раз. То обстоятельство, что принятие решения о порождении задачи производится на системном уровне, делает поведение системы более предсказуемым. Модель одноразовой задачи появилась относительно недавно. Ее активно используют во встроенных СРВ.

Одноразовые задачи в свою очередь могут быть составными или простыми. Составные задачи могут выполнять синхронизирующие системные вызовы, в результате которых выполнение задачи может быть приостановлено. Таким образом, отличительной особенностью составной задачи является возможность нахождения в состоянии *ожидающая*. Простые задачи не могут находиться в состоянии ожидания. Отсутствие этого состояния не означает, что простые задачи не могут разделять ресурсы, обмениваться сообщениями и т.п. Существуют специальные техники, позволяющие простым задачам синхронизироваться друг с другом. Учет межзадачного взаимодействия не внесет ничего существенного в наше рассмотрение. Поэтому, в дальнейшем будем считать, что приложение представляет собой множество независимых задач.

Переход к модели простой задачи связан с внесением дополнительных ограничений на структуру приложения реального времени. Однако это неудобство окупается существенным повышением предсказуемости поведения системы. Кроме того, ОСРВ, работающие с моделью простой задачи, могут использовать для всех существующих в системе задач один общий стек. Прикладные задачи ведут себя как обработчики прерываний, данные и контекст вытесненной низкоприоритетной задачи хранятся в стеке под данными высокоприоритетной задачи. В случае использования модели составной задачи, каждой задаче необходимо выделять собственный стек. Максимальное число задач, данные которых располагаются в общем стеке, определяется не числом задач в системе, а количеством приоритетных уровней задач, так как задачи с равными приоритетами не вытесняют друг друга. Последние исследования показали, что даже в системах с большим числом задач достаточно лишь относительно небольшого числа приоритетных уровней [3]. Таким образом, использование общего стека для простых задач позволяет многократно сократить требования системы к оперативной памяти.

Прежде чем приступить к изложению сути методов минимизации размера общего стека задач рассмотрим формальную модель многозадачного приложения реального времени.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

2. Формальная модель

Приложение реального времени представляет собой множество независимых задач $T = \{\tau_1, \tau_2, ..., \tau_n\}$. Поведение каждой задачи τ_i во времени характеризуется следующими неизменными параметрами: T_i — период задачи, C_i — время выполнения задачи в худшем случае и D_i — относительный срок выполнения задачи. Причем величина относительного срока выполнения каждой задачи не превышает ее периода ($T_i \geq T_i, \forall i$).

Каждой задаче τ_i ставится в соответствие фиксированный приоритет (p_i). Для рассматриваемого класса систем, оптимальным является монотонный по срокам алгоритм планирования (*deadline-monotonic scheduling — DMS*, МСАП) [4]. Согласно этому алгоритму, все задачи упорядочиваются по значению относительного срока выполнения, задаче с наименьшим относительным сроком назначается наибольший приоритет. В динамике задачи обслуживаются вытесняющим планировщиком.

Оригинальный МСАП работает только с уникальными приоритетами. Однако для того, чтобы организация общего стека обеспечила сокращение требований к оперативной памяти необходимо существование нескольких задач с одинаковыми приоритетами. Смягчим требование уникальности приоритетов. Будем говорить, что назначение приоритетов не противоречит явно МСАП, если не существует такой пары задач τ_i и τ_j, для которой одновременно выполнялись бы два следующих условия: $p_i < p_j$ и $D_i < D_j$. В противном случае будем говорить, что назначение приоритетов явно противоречит МСАП. В динамике задачи с равными приоритетами обслуживаются в порядке поступления (*FIFO*).

В жестких СРВ ограничения на время выполнения задач системы четко сформулированы и нарушение установленных сроков получения результатов вычислений может привести к выходу системы из строя. Для проверки соблюдения сроков задач производится математический анализ выполнимости, успешное завершение которого дает гарантию того, что при любой возможной нагрузке выполнение всех задач будет завершено в срок. Рассмотрим метод анализа выполнимости, соответствующий исследуемому классу систем [5].

Суть анализа выполнимости заключается в определении максимального значения времени отклика задач (R) — общей продолжительности выполнения задач с учетом времени их вытеснения более приоритетными задачами. Выполнение задачи τ_i всегда будет завершаться своевременно, если максимальное значение времени отклика не превышает относительного срока выполнения ($R_i \leq D_i$).

Итак, время отклика задачи τ_i складывается из суммарного времени выполнения задач, имеющих приоритет, равный приоритету задачи τ_i, и времени ее вытеснения:

$$R_i = \sum_{j \in ep(i)} C_j + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j, \qquad (1)$$

где $hp(i)$ — множество более приоритетных, чем τ_i задач, $ep(i)$ — множество задач, имеющих приоритет, равный приоритету задачи τ_i, а $\lceil \ \rceil$ — оператор ок-

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ругления в большую сторону.

Приложение реального времени признается выполнимым, если для всех задач максимальное значение времени отклика не превышает относительного срока выполнения.

Проблема минимизации размера стека заключается в назначении задачам таких приоритетов, с которыми, с одной стороны, приложение было бы выполнимым, с другой стороны, размер общего стека был бы минимальным. Таким образом, существуют две цели. Причем первая цель является более важной, минимизация размера общего стека задач должна производиться в рамках пространства допустимых распределений приоритетов, т.е. распределений приоритетов, обеспечивающих выполнимость приложения реального времени.

3. Минимизация числа приоритетных уровней

В [3] для минимизации размера общего стека задач предлагается минимизировать общее число приоритетных уровней, занимаемых прикладными задачами. Авторами работы не приводится описание алгоритма оптимизации. Соответствующие методы реализованы в коммерческом наборе инструментальных средств, и, очевидно, используемые технологии являются ноу-хау. Поэтому автором статьи был разработан свой алгоритм.

Перед рассмотрением сути предлагаемого алгоритма обратим внимание на ряд особенностей планирования задач, приоритеты которых не уникальны.

Лемма 1: Пусть есть две задачи τ_i и τ_j. Приоритеты задач τ_i и τ_j равны ($p_i = p_j$). Тогда время отклика задачи τ_i и время отклика задачи τ_j равны ($R_i = R_j$).

Доказательство: Справедливость этого утверждения очевидна. Все задачи с равными приоритетами имеют одинаковую формулу для времени отклика.

Лемма 2: Пусть есть две задачи τ_i и τ_j. Задача τ_i имеет больший приоритет, чем задача τ_j ($p_i > p_j$), и относительный срок выполнения задачи τ_i не меньше времени отклика задачи τ_j ($D_i \geq R_j$). Тогда, если задаче τ_i назначить приоритет задачи τ_j, то время отклика задачи τ_i будет равно времени отклика задачи τ_j при начальном распределении приоритетов.

Доказательство: Обозначим через R_i' и R_j' время отклика задач τ_i и τ_j после того, как задаче τ_i был назначен приоритет задачи τ_j. Сначала покажем, что изменение приоритета задачи τ_i не приводит к изменению времени отклика задачи τ_j ($R_j' = R_j$). Для этого преобразуем выражение для времени отклика задачи τ_j при начальном распределении приоритетов (R_j):

$$R_j = \sum_{k \in ep(j)} C_k + \sum_{k \in hp(j)} \left\lceil \frac{R_j}{T_k} \right\rceil C_k$$

Разобьем вторую сумму, выделив задачу τ_i из множества более приоритетных, чем τ_j задач:

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

$$R_j = \sum_{k \in ep(j)} C_k + \sum_{k \in (hp(j)-i)} \left\lceil \frac{R_j}{T_k} \right\rceil C_k + \left\lceil \frac{R_j}{T_i} \right\rceil C_i$$

Рассмотрим последнюю составляющую суммы. Напомним, что, во-первых, $D_i \geq R_j$ (по исходным условиям леммы), во-вторых, $T_i \geq D_i$ (всегда для исследуемого класса СРВ). Следовательно, $T_i \geq R_j$. Тогда результатом округления отношения R_j и T_i в большую сторону всегда будет единица. Поэтому:

$$R_j = \sum_{k \in ep(j)} C_k + \sum_{k \in (hp(j)-i)} \left\lceil \frac{R_j}{T_k} \right\rceil C_k + C_i$$

Объединив C_i и первую сумму, получаем выражение:

$$R_j = \sum_{k \in (ep(j) \cup i)} C_k + \sum_{k \in (hp(j)-i)} \left\lceil \frac{R_j}{T_k} \right\rceil C_k$$

Итак, в результате преобразований мы получили такое выражение для времени отклика задачи τ_j при начальном распределении приоритетов, как если бы задача τ_i имела приоритет, равный ее приоритету (с точностью до обозначений). Следовательно, $R_j' = R_j$. Из леммы 1 следует, что $R_i' = R_j'$. Следовательно, $R_i' = R_j$. Что и требовалось доказать.

Лемма 3: Пусть есть две задачи τ_i и τ_j. Задача τ_i имеет больший приоритет, чем задача τ_j ($p_i > p_j$), относительный срок выполнения задачи τ_i не меньше времени отклика задачи τ_j ($D_i \geq R_j$) и приложение выполнимо. Тогда, если задаче τ_i назначить приоритет задачи τ_j, то приложение останется выполнимым.

Доказательство: Посмотрим, как повлияет изменение приоритета задачи τ_i на время отклика прикладных задач.

Время отклика задач, имеющих приоритет больший, чем приоритет задачи τ_i не изменится, так как время отклика высокоприоритетных задач не зависит от количества и распределения по приоритетным уровням низкоприоритетных задач (см. формулу 1).

Время отклика задач с приоритетами, лежащими в интервале $(p_j, p_i]$, уменьшится, так как в результате понижения приоритета задачи τ_i оно утрачивает одну из составляющих.

Время отклика задачи τ_i увеличивается, но, вследствие справедливости леммы 2, не превышает ее относительного срока выполнения.

Время отклика задач, имеющих приоритет равный, приоритету задачи τ_j не изменится. Справедливость этого утверждения для задачи τ_j была показана при доказательстве леммы 2. Тогда, для остальных задач это следует из леммы 1.

Время отклика задач, имеющих приоритет меньший, чем приоритет задачи τ_j не изменится, так как время отклика низкоприоритетных задач зависит только от состава множества высокоприоритетных задач (см. формулу 1).

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Подведем итог. Вследствие изменения приоритета задачи τ_i время отклика увеличивается только у одной задачи (у нее же). Но ее новое время отклика не превышает относительного срока выполнения. Следовательно, приложение остается выполнимым.

Лемма 4: Пусть есть две задачи τ_i и τ_j. Задача τ_i имеет больший приоритет, чем задача τ_j ($p_i > p_j$), и относительный срок выполнения задачи τ_i меньше времени отклика задачи τ_j ($D_i < R_j$). Тогда, если задаче τ_i назначить приоритет задачи τ_j, то время отклика задачи τ_i будет превышать ее относительный срок выполнения (задача будет невыполнимой).

Доказательство: Будем использовать обозначения, предложенные при доказательстве леммы 2. Допустим, что R_i' может не превышать D_i. Но $D_i < R_j$, а из леммы 1 следует, что $R_i' = R_j'$. Следовательно, для того, чтобы наше допущение было справедливым необходимо, чтобы выполнялось условие $R_j' < R_j$. Покажем, что в этом случае неизбежно выполняется условие $R_j' > T_i$. Доказательство построим от обратного. Пусть может выполняться условие $R_j' \le T_i$. Преобразуем выражение для времени отклика задачи τ_j при условии, что $p_i = p_j$ (R_j'):

$$R_j' = \sum_{k \in ep(j)} C_k + \sum_{k \in hp(j)} \left\lceil \frac{R_j'}{T_k} \right\rceil C_k$$

$$R_j' = \sum_{k \in (ep(j)-i)} C_k + \sum_{k \in hp(j)} \left\lceil \frac{R_j'}{T_k} \right\rceil C_k + C_i$$

При условии, что наше последние допущение справедливо, можем написать:

$$R_j' = \sum_{k \in (ep(j)-i)} C_k + \sum_{k \in hp(j)} \left\lceil \frac{R_j'}{T_k} \right\rceil C_k + \left\lceil \frac{R_j'}{T_i} \right\rceil C_i$$

$$R_j' = \sum_{k \in (ep(j)-i)} C_k + \sum_{k \in (hp(j) \cup i)} \left\lceil \frac{R_j'}{T_k} \right\rceil C_k$$

Итак, в результате преобразований мы получили такое выражение, как если бы задача τ_i имела оригинальный приоритет (с точностью до обозначений). Следовательно, $R_j' = R_j$. Но нами рассматривается случай, когда $R_j' < R_j$. Таким образом, мы имеем противоречие. Следовательно, последнее допущение неверно, и $R_j' > T_i$. В то же время $T_i \ge D_i$ (всегда для исследуемого класса СРВ). Следовательно, $R_j' > D_i$, а из равенства R_j' и R_i' получаем неравенство $R_i' > D_i$. Что и требовалось доказать.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Рассмотренные леммы позволяют сформулировать теорему, на которой основываются предлагаемые алгоритмы минимизации числа приоритетных уровней.

Теорема: Пусть есть две задачи τ_i и τ_j. Задача τ_i имеет больший приоритет, чем задача τ_j ($p_i > p_j$), и приложение выполнимо. Если задаче τ_i назначить приоритет задачи τ_j, то приложение останется выполнимым тогда и только тогда, когда относительный срок выполнения задачи τ_i не меньше времени отклика задачи τ_j при начальном распределении приоритетов ($D_i \geq R_j$).

Доказательство: Необходимость и достаточность условия сохранения свойства выполнимости ($D_i \geq R_j$) следуют из леммы 4 и леммы 3, соответственно.

3.1. Алгоритм минимизации общего числа приоритетных уровней

На вход алгоритма поступает выполнимое приложение. Приоритеты задачам назначены в соответствии с МСАП. Для всех задач рассчитано время отклика. Для простоты изложения алгоритма переопределим порядок следования задач. Упорядочим их по значению приоритета. Первой задачей будет наименее приоритетная, последней — наиболее приоритетная. Ниже приведен псевдокод алгоритма минимизации общего числа приоритетных уровней. Синтаксис псевдокода приближен к синтаксису *ANSI-C* [6].

```
int j= 1;
while (j<n)
{    int i= j+1;
     while ((i≤n)&&(Di≥Rj))
     {    pi= pj;
          Ri= Rj;              //Не обязательно
          i++;
     }
     j= i;
}
```

Алгоритм работает следующим образом. Берется первая (наименее приоритетная) задача. Всем следующим (более приоритетным) задачам, относительный срок выполнения которых больше либо равен времени отклика первой задачи, назначается ее приоритет. Далее для первой задачи из числа неперемещенных на более низкий приоритетный уровень, если такие имеются, повторяется все, выполненное для самой первой задачи. И т.д.

Важной особенностью представленного алгоритма является то, что результатом его работы является выполнимое приложение. Действительно, любое изменение приоритета задачи производится только в случае выполнения условий рассмотренной теоремы и, следовательно, не приводит к нарушению выполнимости. Очевидно, что совокупность таких изменений также не может вывести приложение за рамки пространства допустимых распределений приоритетов. Кроме того, по завершении работы алгоритма нет необходимости выполнять перерасчет времени отклика прикладных задач. Для коррекции времени отклика задач достаточно включить в алгоритм строку, помеченную комментарием «Не обязательно». Корректность такого присвоения обеспечивается выполнением условий леммы 2.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

В результате работы алгоритма между приоритетными уровнями, занятыми задачами, могут образоваться пустые приоритетные уровни. В случае, если это нежелательно, алгоритм следует дополнить инструкциями, обеспечивающими непрерывность ряда занятых приоритетных уровней.

Покажем, что предлагаемый алгоритм распределяет прикладные задачи по минимальному числу приоритетных уровней. Рассмотрим множество задач, неперемещенных в ходе работы алгоритма на другой приоритетный уровень, задач, приоритет которых был назначен более приоритетным задачам (если такие были). Число приоритетных уровней, полученное в ходе работы алгоритма, равно числу элементов этого множества. Это число минимально. Действительно, согласно сформулированной теореме неперемещенные задачи не могут разделять приоритетные уровни. Этому препятствуют их временные характеристики и прочие задачи, располагающиеся между ними при их упорядочивании по МСАП. Последние могут быть перемещены на другой приоритетный уровень так, чтобы они не препятствовали разделению приоритетных уровней неперемещенными задачами. Однако, полученное таким образом распределение приоритетов явно противоречит оптимальному МСАП и, следовательно, по определению не может обеспечить уменьшение общего числа приоритетных уровней.

Рассмотрим на примере, как работает предлагаемый алгоритм. В таблице 1 представлены параметры приложения, состоящего из пяти задач. Приоритеты распределены в соответствии с МСАП. Расчет времени отклика показывает, что приложение выполнимо.

Таблица 1. Пример работы алгоритма минимизации общего числа приоритетных уровней

№	*C*	*T*	*D*	*p*	*R*
1	10	120	120	1	95
2	30	100	100	2	85
3	20	90	90	3	45
4	15	70	70	4	25
5	10	50	50	5	10

На первом этапе алгоритма определяется состав задач, выполнение которых возможно на низшем приоритетном уровне. Для этого берется первая задача, задача с наименьшим приоритетом, ее время отклика сравнивается с относительными сроками выполнения более приоритетных задач. Только у второй задачи относительный срок выполнения не меньше времени отклика первой задачи. Поэтому в создаваемом распределении наименьший приоритет будут иметь две задачи: первая и вторая. Далее осуществляется переход ко второму этапу. Берется первая задача из числа неперемещенных на предыдущем этапе, третья задача, ее время отклика сравнивается с относительными сроками выполнения более приоритетных задач. Обе высокоприоритетные задачи, четвертая и пятая, имеют подходящие сроки. Следовательно, на третьем приоритетном уровне будут располагаться третья, четвертая и пятая задачи. Больше задач не осталось. Следовательно, работа алгоритма завершена. Итак, для успешной работы приложения реального времени достаточно двух приоритетных уровней. Низкий приоритет будут иметь первая и вторая задачи, высокий — третья, четвертая и пятая.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

В [3] рассматривается набор инструментальных средств для операционной системы, разработанной в соответствии с *OSEK* [7], стандартом интерфейсов системного программного обеспечения встраиваемой в автомобили электроники. Согласно этому стандарту в рамках одной системы могут существовать и простые и составные задачи. В этом случае, предложенный в [3] метод уменьшения размера общего стека за счет минимизации общего числа приоритетных уровней не совсем корректен. Действительно, размер общего стека задач скорее определяется числом приоритетных уровней, на которых располагаются простые задачи, чем общим числом приоритетных уровней. Следовательно, минимизировать следует только число приоритетных уровней, занятых простыми задачами. Между тем, алгоритм минимизации общего числа приоритетных уровней не обеспечивает минимизации числа приоритетных уровней, на которых располагаются простые задачи. Для демонстрации справедливости последнего утверждения рассмотрим пример. В таблице 2 представлены параметры приложения, состоящего из четырех задач. Прикладные задачи, отмеченные звездочкой (вторая и третья), являются простыми.

Таблица 2. Пример неадекватности алгоритма минимизации общего числа приоритетных уровней

№	C	T	D	$p^{МСАП}$	p^1	p^2
1	30	100	100	1	1	1
2*	30	100	90	2	1	2
3*	20	100	60	3	2	2
4	10	100	30	4	2	3

В примере приведены три варианта распределения приоритетов: в соответствии с МСАП ($p^{МСАП}$), с минимальным числом приоритетных уровней (p^1) и дополнительное допустимое распределение с большим числом приоритетных уровней (p^2). В случае распределения p^1 простые задачи занимают два приоритетных уровня. Распределение p^2 обеспечивает разделение простыми задачами одного приоритета. Таким образом, несмотря на то, что распределение p^2 не обеспечивает минимальное число приоритетных уровней, для него число приоритетных уровней, занятых простыми задачами, в два раза меньше, чем в случае распределения p^1, обеспечивающего минимальное число приоритетных уровней.

Следовательно, простой минимизации общего числа приоритетных уровней недостаточно для минимизации числа приоритетных уровней, на которых располагаются простые задачи. Недостатком алгоритма минимизации общего числа приоритетных уровней является то, что простые задачи могут быть определены на соседние приоритетные уровни потому, что они не могут выполняться своевременно на одном уровне с низкоприоритетной составной задачей. Так в рассмотренном примере в ходе работы алгоритма простым задачам были определены разные приоритеты из-за невозможности разделения ими одного приоритетного уровня с составной задачей. Следовательно, необходимо разработать такой алгоритм, который бы располагал простые задачи на разных приоритетных уровнях только в случае, если они не могут выполняться своевременно на одном уровне.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

3.2. Алгоритм минимизации числа приоритетных уровней, занятых простыми задачами

Рассмотрим один из возможных вариантов такого алгоритма.

```
int j= 1;
while (j<n)
{    if (is_simple(j))
     {    int i= j+1;
          while ((i≤n)&&(Di≥Rj))
          {    pi= pj;
               Ri= Rj;         //Не обязательно
               i++;
          }
          j= i;
     } else
          j++;
}
```

К входным данным представленного алгоритма предъявляются те же требования, что и к входным данным для алгоритма минимизации общего числа приоритетных уровней. Отличие этого алгоритма от предыдущего заключается в том, что в данном случае в начале (в основе) каждого приоритетного уровня, на котором располагаются простые задачи, может находиться только неперемещенная простая задача (неперемещенные составные задачи имеют уникальные приоритеты). Благодаря этому обстоятельству настоящий алгоритм лишен указанного недостатка алгоритма минимизации общего числа приоритетных уровней, простые задачи не могут быть разнесены на разные приоритетные уровни на основании неудавшейся проверки на возможность своевременного выполнения на одном уровне с низкоприоритетной составной задачей.

Покажем, что предлагаемый алгоритм распределяет простые задачи по минимальному числу приоритетных уровней. Рассмотрим множество неперемещенных задач, образующих приоритетные уровни, занятые простыми задачами. Как уже было сказано ранее, эти неперемещенные задачи также являются простыми. Задачи, входящие в это множество, могут разделять приоритетные уровни только в случае распределения приоритетов, явно противоречащего МСАП. Следовательно, число приоритетных уровней, занятых простыми задачами, определяемое числом элементов этого множества, минимально (см. доказательство корректности алгоритма минимизации общего числа приоритетных уровней).

Рассмотрим на примере, как работает предлагаемый алгоритм. В таблице 3 представлены параметры приложения, состоящего из пяти задач. Прикладные задачи, отмеченные звездочкой (первая, третья и пятая), являются простыми. Приоритеты распределены в соответствии с МСАП. Расчет времени отклика показывает, что приложение выполнимо.

Таблица 3. Пример работы алгоритма минимизации числа приоритетных уровней, занятых простыми задачами

№	*C*	*T*	*D*	*p*	*R*
1*	30	200	120	1	100
2	25	200	110	2	70
3*	20	200	100	3	50
4	15	200	50	4	25
5*	10	200	30	5	10

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

На первом этапе алгоритма определяется состав задач, выполнение которых возможно на низшем приоритетном уровне. Для этого берется первая задача, задача с наименьшим приоритетом, и проверяется ее тип. Первая задача является простой. Следовательно, на ее основе можно создавать разделяемый приоритетный уровень. Поэтому ее время отклика сравнивается с относительными сроками выполнения более приоритетных задач. Только у второй и третьей задачи относительные сроки выполнения не меньше времени отклика первой задачи. Поэтому в создаваемом распределении наименьший приоритет будут иметь три задачи: первая, вторая и третья. Далее переходим ко второму этапу. Берется первая задача из числа неперемещенных на предыдущем этапе, четвертая задача. Четвертая задача является составной. Поэтому осуществляется переход к следующей неперемещенной задаче. Пятая задача — простая, но последняя. Следовательно, работа алгоритма завершена. Итак, для успешной работы приложения реального времени для простых задач достаточно двух приоритетных уровней. Общее число приоритетных уровней равно трем.

Заключение

Одним из методов повышения эффективности использования оперативной памяти во встроенных СРВ является уменьшение числа приоритетных уровней прикладных задач. В настоящей статье предложен алгоритм минимизации общего числа приоритетных уровней. Показана неадекватность этого алгоритма для целей минимизации размера общего стека задач в смешанных системах. Предложенный алгоритм минимизации числа приоритетных уровней, на которых располагаются простые задачи, является более эффективным методом сокращения требований программных комплексов к оперативной памяти.

Использование предложенных алгоритмов также позволяет увеличить производительность системы. Так как задачи, разделяющие один приоритетный уровень, могут, в частности, разделять ресурсы без дополнительных расходов процессорного времени на синхронизацию.

Литература

[1] *В. В. Никифоров, В. А. Павлов.* Операционные системы реального времени для встроенных программных комплексов. Тверь, Программные продукты и системы, 1999, №4 — С. 24–30.

[2] *В. В. Никифоров.* Разработка программных средств для встроенных систем. С-Пб, С-ПбГЭТУ, 2000.

[3] Stack Optimization. <http://www.livedevices.com>.

[4] *J. Y. T. Leung, J. Whitehead.* On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Perf. Eval. (Netherlands), 1982-2 — pp. 237–250.

[5] *K. Tindell, H. Hansson.* Real Time Systems by Fixed Priority Scheduling. DoCS, Uppsala University, 1997.

[6] *H. Schildt.* The Annotated ANSI C Standard. Osborne McGraw-Hill, Berkeley, 1990.

[7] OSEK/VDX. Operating System. Version 2.0 revision 1. <http://www.osek-vdx.org>.