Автоматизация построения расписаний для периодических систем реального времени

Третьяков Андрей
andrromeda@ispras.ru

Аннотация. В статье рассматривается проблема построения расписаний для систем, имеющих жёсткие ограничения по времени работы. Это является одной из основных проблем при проектировании систем авионики. В работе проанализированы существующие алгоритмы, выделены их недостатки, и предложено собственное решение, удовлетворяющее необходимым требованиям. Кроме того, разработаны инструменты для визуализации, редактирования и валидации создаваемых расписаний.

1. Введение

Вычислительные системы в промышленности и технике используются для решения широкого спектра разнообразных задач. Чаще всего эти задачи связаны с важными производственными функциями, от которых зависит работоспособность системы в целом. Компьютерные системы обычно контролируют какое-нибудь устройство или физический процесс, собирают данные о нём, выполняют необходимые вычисления и отвечают на внешние физические события путём отправки соответствующих управляющих воздействий. Причём, поскольку процесс получения и обработки информации зачастую является непрерывным, система должна успевать реагировать на внешние события. То есть, задачи должны выполняться за такие промежутки времени, которые обеспечивают успешное решение всех поставленных перед системой задач. Такие системы, гарантирующие своевременную реакцию на внешние события, называются **системами реального времени** (СРВ). Существует несколько эквивалентных определений систем реального времени [7, 8, 9], в данной работе под этим термином будут подразумеваться системы, которые производят вычисления и решают свои задачи в заданные ограниченные промежутки времени. Если задача не успевает к требуемому моменту времени вычислить результат, последствия могут быть различной степени тяжести, вплоть до выведения системы из строя, что может привести к катастрофе. Поэтому системы реального времени проектируются особо тщательно, и задачи, которые они будут выполнять, просчитываются заранее, исходя из их временных свойств и заданных ограничений.

Если несколько событий происходят в близкие промежутки времени, системе необходимо распланировать выполнение своих вычислений так, чтобы ответ системы на каждое событие поступал в заданные временные рамки. Для этого обычно используется планировщик, который и определяет, как именно и в каком порядке должны решаться задачи. Во многих системах планировщик работает по заранее составленному **расписанию**, определяющему политику планирования.

Составление расписаний может оказаться очень большой проблемой, сложность которой зависит как от параметров системы и ограничений на целевые задачи, так и от количества задач. В работе показывается, как изменение одного, на первый взгляд несущественного, условия может превратить задачу построения расписания из задачи, для которой существуют классические полиномиальные алгоритмы, в NP-трудную задачу. При планировании задач для небольших систем составлять расписание можно любым способом, в том числе и вручную или с использованием каких-либо эвристических методов, – главное, чтобы оно в итоге оказалось корректным. Однако на практике для систем, насчитывающих сотни и тысячи задач, или для которых приходится часто переделывать расписания, требуется автоматизировать этот процесс.

2. Особенности целевой системы

Одной из сфер применения систем реального времени является авиастроение, причём особого внимания заслуживает бортовая система автоматического управления самолётом. Она выполняет ряд задач высокой степени важности по измерению микроклиматических условий окружающей среды (температура, давление), параметров полёта (высота, скорость) и по управлению некоторыми отдельными устройствами (например, подкрылками самолёта). Кроме того, она выполняет и другие функции (например, автомат тяги, автопилот, предупреждение экипажа, техническое обслуживание). Без сомнения, эти функции являются задачами «жёсткого» реального времени [1], так как если они не успеют отреагировать на внешние события в заданные промежутки времени, это может привести к нарушению работы бортовой системы и потенциально к крушению самолёта. Поэтому в качестве целевой системы реального времени выберем бортовую систему автоматического управления самолётом, которую и будем в дальнейшем рассматривать.

Прежде чем выяснить особенности выбранной системы, введём несколько необходимых определений. В зарубежной литературе срок, до которого задача должна выполниться, называется *deadline*, что может быть переведено как *критический срок обслуживания* или *предельный срок выполнения* [8]. Кроме этого, в большинстве систем задачи могут иметь также следующие свойства [1]:

- время запуска (или время готовности) – момент времени, в который задача приводится в состояние готовности и ожидает выполнения;
- минимальная (максимальная) задержка – минимальное (максимальное) количество времени, которое должно пройти между временем готовности и моментом времени фактического начала выполнения задачи;
- наибольшее время выполнения (или максимальное время отклика) – максимальное время, необходимое для завершения задачи;
- время выполнения (или длительность) – время, взятое без прерываний, необходимое для выполнения задачи;
- приоритет (или вес) – относительная важность задачи.

В связи с усложнением программного обеспечения в области авиастроения, в последнее время стало целесообразным возлагать на один процессорный модуль авиационные функции, выполняемые ранее несколькими модулями, при этом улучшив массогабаритные характеристики системы управления и снизив ее стоимость. Такая тенденция в авиационной технике привела к появлению концепции интегрированной модульной авионики – ИМА (Integrated Modular Avionics, IMA). Отсюда возникает проблема разделения ставших теперь общими ресурсов (процессорное время, память, каналы обмена) между различными приложениями. В авионике ключевую роль в решении данной проблемы играют операционные системы реального времени, работающие по стандарту ARINC 653 [10]. Поскольку от некоторых функций зависит безопасность полёта, для них ресурсы, в первую очередь, процессорное время, распределяются планировщиком между задачами особенно тщательно, по заранее определённой схеме. То есть, в этих системах, исходя из временных характеристик задач и других их свойств, предварительно составляется **расписание** вычислений системы, используя которое, планировщик распределяет во времени решаемые задачи так, что они гарантированно удовлетворяют всем временным ограничениям. При этом сам процесс составления расписания происходит за рамками системы реального времени, и потому является некритичным по времени. Кроме того, заранее становится известно, возможно ли в принципе составить расписание для данного набора задач, до того, как система реального времени будет введена в эксплуатацию.

Понятие «авиационной функции» близко к понятию «процесса» в обычной пользовательской операционной системе, принципы организации и реализации также схожи. Поэтому авиационные функции, периодически запускающиеся в бортовой системе, в данной работе будем называть процессами, а каждый конкретный запуск – итерацией процесса.

Итак, все процессы распределяются по нескольким процессорным модулям, на каждом модуле выполняется набор взаимодействующих процессов, для

которого и составляется расписание. Между процессами из различных модулей зависимости не учитываются. Поэтому можно считать, что мы имеем дело с однопроцессорной системой, так как для других процессоров будет составляться другое расписание аналогичным образом.

Планировщик задач в данной системе срабатывает через некоторые фиксированные промежутки времени (порядка 0,25 мс), называемые **тиками** процессора. Таким образом, тик – это минимальная распределяемая единица времени в расписании, и поэтому все числовые параметры процессов измеряются в тиках процессора.

Бортовая система самолёта работает с множеством авиационных функций, среди которых есть и периодические, и апериодические, и спорадические [1]. Апериодические процессы, важность которых не особо критична, выполняются на отдельном процессорном модуле, и в данной работе рассматриваться не будут. Для построения расписаний для спорадических процессов существуют эффективные классические алгоритмы, удовлетворяющие всем особенностям бортовой системы самолёта. Наибольший интерес представляют периодические процессы, размещаемые на одном процессоре.

Требование периодичности процессов следует рассмотреть подробнее, так как его можно определить по-разному. В некоторых работах [1, 2, 3] под периодом процесса понимается промежуток времени между двумя его последовательными моментами времени ***готовности***, в которые процесс переводится из состояния ожидания в состояние готовности и ожидает начала выполнения. То есть, процесс должен запуститься и завершиться в промежутке между очередными моментом времени готовности и предельным сроком выполнения (*deadline*). Причём момент фактического начала работы процесса может быть несколько позже времени готовности (см. рисунок 1).

Рисунок 1

На рисунке 1 в качестве примера приведён процесс с периодом 6 и предельным сроком выполнения, равным 5 единицам времени. Причём 3-я

итерация процесса запустилась не в 12, а в 13 момент времени из-за другого процесса (выделен серым цветом).

Из авиационной практики известно, например, что если через определённые промежутки времени не срабатывают функции управления элеронами и закрылками самолёта, то крыло может отвалиться. Поэтому ряд экспертов в области авиастроения выдвигают более жёсткие требования по периодичности процессов. А именно, процесс должен запускаться в строго периодичные моменты времени и отрабатывать хотя бы одну единицу времени. То есть, время фактического запуска должно совпадать со временем готовности. Иными словами, максимальная задержка должна быть сведена к нулю.

Таким образом, расписание, представленное на рисунке 1, не является корректными с точки зрения периодичности процесса, а на рисунке 2 – является:

Рисунок 2

Требование строгой периодичности компенсируется следующим условием: процессы могут быть прерваны в любой момент (кроме начального, то есть времени запуска), любое число раз, другими процессами. После завершения прерывающего процесса прерываемый должен доработать оставшееся его время. При этом накладные расходы процессора на переключение между процессорами не учитываются, они уже включены в длительности процессов.

При прерывании процесса увеличивается его максимальное время отклика, и существует вероятность, что один или несколько последних фрагментов нарушат предельный срок выполнения. Многие авиационные функции работают с неразделяемыми ресурсами, то есть такими ресурсами, которые доступны только одной итерации конкретного процесса. Поэтому случай, когда какой-либо фрагмент *k*-ой итерации процесса выполняется после того, как началась (*k*+1)-ая итерация, является недопустимым. Во избежание этого относительный предельный срок выполнения устанавливается равным периоду процесса. Это условие рассматриваемую систему как раз и делает системой реального времени.

Следует подчеркнуть, что длительность каждого процесса в каждом периоде также должна строго соответствовать заданному значению.

Расписание, в котором для всех процессов выполняются вышеперечисленные условия, будем называть *корректным* (или *допустимым*). Остаётся добавить, что хотя накладные расходы на переключение между процессами не учитываются в явном виде, само число переключений требуется сократить. Корректное расписание, в котором число переключений процессора минимальное для данных входных параметров, будем называть *оптимальным*.

Резюмируя всё вышесказанное, сформулируем требования к целевой системе:

- Система является системой ***«жёсткого»*** реального времени.
- Рассматривается один процессор, на котором запускается *n* процессов.
- Процессы могут прерываться в любой момент и любое число раз.
- Процессы должны завершаться до запуска своей следующей итерации.
- Для каждого процесса должны **строго** соблюдаться заданные входные параметры:
  - фиксированный период фактического запуска (начала работы);
  - суммарная длительность процесса в каждом периоде.
- Зависимости процессов и накладные расходы на переключение не учитываются; все параметры измеряются в целых числах.
- Количество переключений между процессами должно быть минимальным.

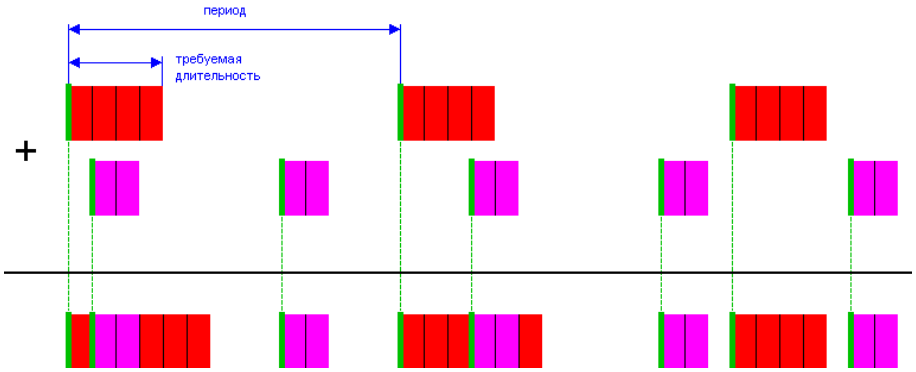

Рисунок 3

На рисунке 3 показаны в качестве примера два периодических процесса, как они запускались бы каждый на отдельном процессоре, и расписание их работы на одном общем процессоре. Каждый процесс запускается строго

периодически, даже если он в следующий момент времени будет прерван другим процессом.

3. Понятие расписания для периодической системы реального времени

Расписание для системы реального времени, на которую накладываются дополнительные требования, описанные в предыдущем разделе, построить весьма непросто, так как приходится учитывать множество ограничений. В различных авиационных компаниях поступают по-разному: в одних подобные расписания строят вручную, в других используют различные закрытые разработки. Однако на практике вручную строить расписания нерационально, так как число и характеристики процессов могут постоянно варьироваться, для больших значений параметров приходится делать много рутинной работы, причём очень легко можно допустить ошибку. Более того, для некоторых входных данных в принципе нельзя построить расписание, удовлетворяющее всем ограничениям. Такие же проблемы возникают при необходимости проверить, действительно ли построенное расписание удовлетворяет всем поставленным требованиям. Сложнее всего дело обстоит с поиском оптимального расписания, так как приходится гарантировать, что не может быть найдено расписания с меньшим количеством переключений процессора. Из всего вышесказанного следует, что оптимальным решением проблемы будет инструмент, позволяющий персоналу, ответственному за проектирование систем авионики (так называемым системным интеграторам), проектировать расписания в автоматическом или полуавтоматическом режиме.

Для анализа алгоритмов построения расписаний формализуем понятия периодической системы реального времени и расписания для неё. В общем случае, периодическую систему реального времени можно формально определить следующим образом. Периодическая система T, состоящая из n процессов, задаётся множеством своих процессов T_i, каждый из которых определяется четвёркой: $T_i = (e_i, d_i, p_i, s_i)$, $1 \le i \le n$, где:

- e_i – длительность i-ого процесса, то есть время работы, взятое без прерываний;
- d_i – относительный предельный срок выполнения i-ого процесса, то есть промежуток времени между очередными моментом времени готовности и предельным сроком выполнения;
- p_i – период i-ого процесса, то есть промежуток времени между двумя последовательными моментами времени готовности;
- s_i – начальное смещение i-ого процесса, то есть время начала первой итерации, первый момент времени готовности.

То есть, для k-го вызова i-ого процесса заданы следующие временные ограничения: должно существовать e_i моментов времени t, таких, что

$$s_i + (k-1)p_i \le t < s_i + (k-1)p_i + d_i \quad (1)$$

где k – натуральное число (натуральные числа, напомним, не включают 0) [3].

Для этих параметров процессов выполняются следующие очевидные соотношения:
$0 < e_i \le d_i \le p_i$, где $1 \le i \le n$. Для смещения чаще всего выполняется $s_i \le d_i - e_i$.

Для простоты все числовые параметры будем полагать целыми, поскольку, как показывается в работе [3], для периодических систем реального времени случай расписаний с действительными параметрами может быть сведён к случаю с дискретными параметрами.

Если задаются начальные смещения, то системы называются **полными**, в противном случае – **неполными**. То есть, для неполных систем процессы задаются тройкой:

$T_i = (e_i, d_i, p_i)$, где $1 \le i \le n$.

В рассматриваемой авиационной системе начальные смещения процессов не задаются, поэтому система является неполной. Одной из задач алгоритма составления расписания как раз и является определение начальных смещений.

Расписание для однопроцессорной системы, в свою очередь, можно формально определить как числовую функцию S от натурального переменного t:
S: $N \to \{0, 1, \ldots, n\}$, то есть, для каждого момента времени определяющая, какой из n процессов работает, или возвращающая 0, если ни один из процессов не запущен в данный момент. Таким образом, расписание S будем называть корректным, если для любого натурального k и $i \in \{1, \ldots, n\}$ существует e_i моментов времени t таких, что $S(t) = i$ и выполняется условие (1).

Собственно, проблему нахождения разбиения итераций процессов для исходного набора процессов и называют построением расписания (см. рис. 3).

Очевидно, что в общем случае корректное расписание для системы не единственно. При составлении расписаний обычно максимизируют или минимизируют некоторую целевую функцию, например, количество процессоров, выполняющих задачи, или размер расписания, или общее количество работ, или суммарное запаздывание, и так далее. В данном случае это число переключения процессора между процессами. Расписание, для которого целевая функция достигает своего экстремума, мы будем называть оптимальным. Однако, в зависимости от свойств системы, даже оптимальное расписание может быть не единственным. Или, даже если оно единственное,

получить его можно разными способами. Можно, например, составить его вручную, исходя из логических умозаключений. Или, если требования к системе не слишком жёсткие, попытаться сгенерировать расписание полуавтоматическим образом, а затем проверить, выполняются ли все требования системы. Во многих системах используются инструменты автоматического построения расписания, работающие по какому-либо алгоритму, подобранному для конкретной системы. На практике может применяться любой из описанных способов, и в каждом отдельном случае выбирается тот или иной подход.

4. Существующие решения

4.1. Циклические расписания

Для построения расписаний существуют различные алгоритмы.

Однако в большинстве печатных изданий рассматриваются либо непериодические системы, либо системы, не являющиеся системами реального времени. Например, в статьях [4, 5] авторов Peter Brucker и Thomas Kampmeyer даётся следующее определение: расписание для системы из n процессов называется периодическим с циклом α, если для каждого i-го процесса момент времени $t(i; k)$ начала k-ой его итерации определён формулой:

$t(i; k) = t(i; 0) + \alpha k$, где $1 \le i \le n$, k – целое.

То есть, периоды всех процессов одинаковы и равны α. Такую систему они называют *циклической*.

Тем не менее, периодическую систему (с разными периодами процессов), которая рассматривается в этой работе, можно свести к циклической (с одинаковыми периодами).

Но прежде мы определим, какова должна быть длина (размер) расписания, достаточная для функционирования системы, в которой бесконечно запускаются процессы с определённым периодом. Пусть мы имеем периодическую систему из n задач $T = \{1, \ldots, n\}$, с соответствующими периодами процессов $p_1, \ldots, p_n$ и их начальными смещениями $s_1, \ldots, s_n$. Тогда k-ый запуск процесса i происходит в момент времени $s_i + (k - 1)p_i$. Допустим, удалось построить расписание, в котором нет ни одной коллизии процессов. Возьмём длину этого расписания, равную наименьшему общему кратному всех периодов процессов, $L = \text{НОК}(p_1, \ldots, p_n)$, и назовём его первым циклом расписания. Пусть эти процессы за весь первый цикл расписания запускаются соответственно $k_1, \ldots, k_n$ раз, где $k_i = L / p_i$. Второй цикл расписания начинается непосредственно за первым. Тогда для каждого процесса его $(k_i + 1)$-ый запуск (т.е. уже во втором цикле расписания) будет в момент времени $s_i + ((k_i + 1) - 1)p_i = s_i + k_i p_i = s_i + L / p_i * p_i = s_i + L$. Таким образом, смещения каждого процесса относительно начала второго цикла расписания равно смещению относительно первого цикла. Далее процессы продолжают

запускаться с тем же периодом. То есть второй цикл расписания полностью идентичен первому. Промежуток времени между последним запуском i-го процесса в первом цикле и первым запуском во втором равен $(s_i + L) - (s_i + (k_i - 1)p_i) = (s_i + ((k_i + 1) - 1)p_i) - (s_i + (k_i - 1)p_i) = (s_i + k_i p_i) - (s_i + (k_i - 1)p_i) = k_i p_i - (k_i - 1)p_i = p_i$. Таким образом, мы доказали, что требуемая длина расписания может быть равна $\text{НОК}(p_1, \ldots, p_n)$. В [1, 2] показано, что меньше она быть не может. Рисунок 4 иллюстрирует цикличность расписания:

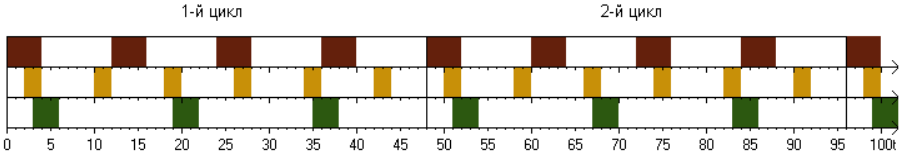

Рисунок 4

На рисунке 4 в качестве примера приведены 3 процесса с периодами 12, 8, 16 и начальными смещениями 0, 2, 3 соответственно. Длина расписания равна НОК(12, 8, 16)=48. Из рисунка видно, что второй цикл полностью повторяет первый, причём на их границе периодичность процессов не нарушается.

Сведём периодическую систему к циклической. Обозначим каждую итерацию каждого процесса – новым процессом. Новые процессы будут отличаться между собой только начальным смещением. Занумеруем все новые процессы за один цикл расписания сквозной нумерацией. Получим $k_1 k_2 \ldots k_n$ процессов, у каждого из которых уникальное начальное смещение. Во втором цикле расписания сделаем то же самое. Тогда получим, что каждый новый процесс (в старых обозначениях (i, j), где $1 \le j \le k_i$) будет запускаться периодически с периодом L. То есть, мы пришли к циклической системе.

Таким образом, построив один цикл расписания, мы строим расписание для всей шкалы по времени. Однако это не даёт ответа на вопрос, каким образом можно построить первый цикл расписания, то есть, как определить циклическую систему. А в работах [4, 5] она уже полагается заданной.

4.2. Алгоритмы Rate-Monotonic и Earliest Deadline First

Наиболее близкими алгоритмами к рассматриваемой сфере являются классические алгоритмы Rate-Monotonic (RM) и Earliest Deadline First (EDF) [1]. Оба они основываются на понятии *приоритета* процесса – целого положительного числа, определяющего относительную срочность; чем меньше число, тем выше считается приоритет процесса. Алгоритмы проходят временную шкалу от 0 до L и в каждый момент времени определяют, сколько процессов находятся в состоянии готовности. Если найден только один

процесс, то он работает согласно своей периодичности, либо завершается один из его фрагментов после того, как был завершён более приоритетный процесс. Если нет ни одного, то в расписании образуется свободное место (*idle time*). Если же несколько процессов находятся в состоянии готовности, выбирается тот из них, чей приоритет выше. Для полных систем алгоритмы учитывают начальные смещения процессов, а для неполных полагают их равными 0.

Различие между алгоритмами заключается, в основном, в том, что Rate-Monotonic подразумевает, что процессы имеют фиксированный приоритет (если приоритеты не задаются в условии, то алгоритм их задаёт сам: чем меньше относительный предельный срок выполнения процесса, тем его приоритет выше), а Earliest Deadline First – что динамический. То есть, приоритет разных фрагментов одной и той же итерации процесса может различаться в разные моменты времени. EDF полагает, что чем ближе предельный срок выполнения для данного процесса, тем его приоритет выше. Другими словами, на каждом шагу (т.е. в каждый момент времени), при анализе списка работающих в данный момент процессов, алгоритм RM выбирает тот из них, чей относительный *deadline* численно меньше, а EDF – тот, чей очередной абсолютный *deadline* ближе по времени.

И RM и EDF для периодических систем имеют алгоритмическую сложность $O(N^2)$ в худшем случае, где N – суммарное число итераций всех процессов за время одного цикла расписания [1], то есть, они являются полиномиальными.

Кроме того, алгоритм EDF является оптимальным в том смысле, что может нарушить предельный срок выполнения какого-либо процесса только в том случае, если никакой другой алгоритм построения расписания для системы реального времени не может выполнить задачу в срок [1].

И, наконец, алгоритмы RM и EDF довольно просты в реализации, различные их варианты и модификации описаны во многих работах [1, 2, 3, 11].

Однако они имеют очень существенный недостаток, который перечёркивает все их достоинства. Они не отвечают требованиям системы в том виде, в котором они поставлены в разделе 2, а именно: нарушается требование строго периодичного запуска процессов. То есть, в случае, когда в некоторый момент времени должен начать работать один процесс при запущенном втором процессе с более высоким приоритетом, предпочтение будет отдано второму. То есть, нередки ситуации, отражённые на рисунке 1, что является недопустимым.

Кроме того, RM и EDF вовсе не гарантируют, что число переключений процессора минимальное из всех возможных. Зачастую, задав начальные смещения, отличные от 0, можно получить лучше результат.

4.3. Приближённый алгоритм

В статье С. Зеленова [11] для решения этой задачи используется другой подход: предложен алгоритм, который строит расписание, близкое к оптимальному с точки зрения минимизации общего количества прерываний задач. Этот подход основан на выборе таких начальных смещений, для которых наиболее вероятно будет происходить непрерывное выполнение процессов. Такой алгоритм выполняется очень быстро, позволяет получить результат до 25 прерываний на 1000 тиков процессора, что во многих случаях вполне приемлемо. Однако часто можно получить и более оптимальный результат.

5. Система автоматического составления расписаний

5.1. Архитектура системы

В ИСП РАН разрабатывается система инструментов для автоматизации различных операций по составлению расписаний для системы реального времени, на которую накладываются дополнительные ограничения, изложенные в разделе 2. Параметры авиационных функций содержатся базе данных бортовой системы самолёта, копия которой локально сохранена на компьютере системного интегратора. Работая с ней с помощью этих инструментов, он создаёт расписания для каждого процессорного модуля, которое затем сохраняется в эту же базу данных и впоследствии используется в самом самолёте.

Разрабатываемая система получила название «Рабочее место системного интегратора ИМА» (*англ*. Modular Avionics System Integrator Workplace, MASIW) и основана на каркасе приложений (*framework*), предоставляемом популярной кроссплатформенной расширяемой средой разработки Eclipse. Схема работы с системой представлена на рисунке 5:

В рамках данной работы система работает с параметрами авиационных функций (процессов), сохранёнными в специальном проекте Eclipse (своего рода тоже локальной базе данных, только сильно упрощённой), которые загружаются из локальной базы данных бортовой системы самолёта. Эти параметры проверяются на согласованность и передаются на вход алгоритму (одному или нескольким), а на выходе получается расписание для этих процессов, которые сохраняются в этот же проект Eclipse. Позже система позволяет открыть полученные расписания и отобразить их в графическом или табличном виде, там же произвести исправления, если таковые окажутся необходимыми ответственному персоналу, и проверить на корректность модифицированную версию расписания. Затем окончательное расписание сохраняется в базу данных бортовой системы самолёта, но это уже выходит за рамки данной работы.

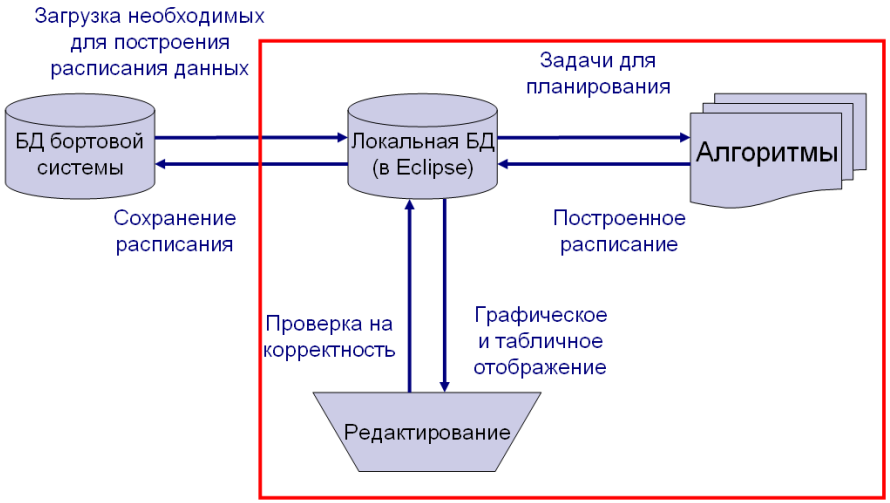

Рисунок 5

5.2. Задача построения расписаний

Дадим формальное определение рассматриваемой системы реального времени и корректного расписания для данного конкретного случая.

В разделе 3 было дано формальное определение более общей системы. Теперь учтём, что в данной системе относительный предельный срок выполнения равен периоду процесса (то есть $p_i = d_i$ для всех i, $1 \le i \le n$, где n – число процессов в системе), а также начальные смещения процессов не заданы (т.е. есть система неполная). Тогда система из n процессов задаётся как $T = \{T_1, \ldots, T_n\}$, где T_i $(1 \le i \le n)$ задаётся следующим образом: $T_i = (e_i, p_i)$, причём $0 < e_i \le p_i$ для всех $i \in \{1, \ldots, n\}$. Начальные смещения s_i алгоритм должен выбрать самостоятельно.

Формула (1) преобразуется к следующему виду:

$$s_i + (k-1)p_i \le t < s_i + kp_i \qquad (2)$$

где k – натуральное.

В разделе 4.1 было показано, что для периодической системы реального времени длина расписания равна $L = \text{НОК}(p_1, \ldots, p_n)$.

Расписание $S(t)$ как функция натурального переменного t ($S: N \to \{0, 1, \ldots, n\}$) называется корректным, если для любого натурального k и $i \in \{1, \ldots, n\}$ существует **ровно** e_i моментов времени t таких, что $S(t) = i$ и выполняется условие (2), причём $S(s_i + (k-1)p_i) = i$ (то есть, момент запуска каждой итерации **строго** периодичен).

5.2.1. Необходимые условия существования расписания

Как уже было сказано, не для любых наборов процессов в принципе возможно построить расписание.

1) Самое очевидное ограничение на входные данные следующее: для каждого процесса его период должен быть не меньше его длительности:

$$0 < e_i \le p_i \qquad (*)$$

для всех $i \in \{1, \ldots, n\}$.

Иначе предельные сроки выполнения заведомо нарушаются даже в случае одного процесса в системе.

2) Второе условие является своего рода обобщением первого, и смысл его заключается в том, чтобы суммарная длительность всех процессов с учётом числа их запусков была не больше длины расписания. То есть:

$$\sum_{i=1}^{n} \sum_{j=1}^{k_i} e_i \le L,$$

где k_i – число запусков i-го процесса за один цикл расписания. Упростим это выражение. Так как $L = \text{НОК}(p_1, \ldots, p_n)$, то $k_i = L / p_i = \text{НОК}(p_1, \ldots, p_n) / p_i$, а суммарная длительность i-го процесса за один цикл равна $e_i * \text{НОК}(p_1, \ldots, p_n) / p_i$. Таким образом, имеем:

$$\sum_{i=1}^{n} \frac{НОК(p_1,\ldots,p_n)}{p_i} e_i \le НОК(p_1,\ldots,p_n)$$

Разделив обе части неравенства на $\text{НОК}(p_1, \ldots, p_n)$, имеем:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \le 1 \qquad (**)$$

3) Последнее, самое неочевидное, условие вытекает из соотношения Безу для наибольшего общего делителя (НОД) двух чисел:

Для любых двух натуральных чисел a и b существуют такие целые числа k_1 и k_2, что наибольший общий делитель a и b представляется в виде:

$$\text{НОД}(a, b) = k_1 a + k_2 b. \qquad (3)$$

Очевидно, что если $a > 0$ и $b > 0$, то в выражении (3) один из коэффициентов отрицательный. Пусть для определённости это будет k_2, тогда изменив знак перед ним, будем считать оба коэффициента положительными. Теперь возьмём в качестве a и b взаимно простые числа p_1 и p_2, являющихся периодами двух процессов, получим:

$$k_1p_1 - k_2p_2 = 1.$$

Умножим это соотношение на $(s_2 - s_1)$, где s_1 и s_2 – начальные смещения процессов:

$$(s_2 - s_1)k_1p_1 - (s_2 - s_1)k_2p_2 = (s_2 - s_1).$$

Обозначим $k'_1 = k_1(s_2 - s_1)$, $k'_2 = k_2(s_2 - s_1)$ и перенесём слагаемые:

$$k'_1p_1 - k'_2p_2 = (s_2 - s_1);$$
$$s_1 + k'_1p_1 = s_2 + k'_2p_2.$$

В обеих частях равенства стоит формула момента запуска процесса. Мы получили, что если периоды p_1 и p_2 являются взаимно простыми числами, то какие бы начальные смещения s_1 и s_2 мы ни выбрали, найдутся такие коэффициенты k'_1 и k'_2, что соответствующие итерации этих процессов будут начинаться в один и тот же момент, что недопустимо по причине единственности процессора и строгой периодичности процессов. Поэтому 3-е условие формулируется следующим образом:

Для каждой пары процессов периоды не должны являться взаимно простыми числами. (***)

Здесь следует отметить, что требуется отсутствие именно **попарной** взаимной простоты периодов. Если во всей системе нашлось хотя бы 2 процесса с взаимно простыми периодами, составить расписание для системы уже будет невозможно. Однако в **совокупности** периоды процессов могут быть взаимно простыми, то есть допустимо, если НОД(p_1, …, p_n) = 1. Пример успешно построенного расписания для периодов 6, 10 и 15 приведён на рисунке 6:

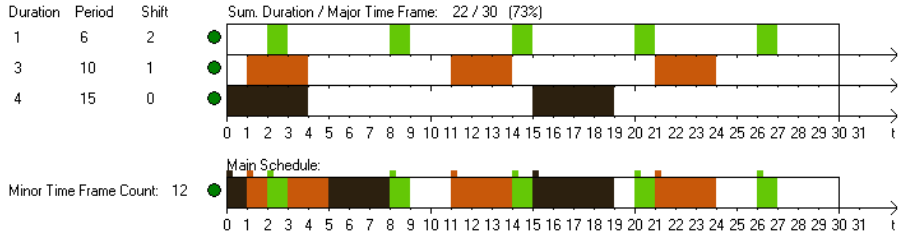

Рисунок 6

На рисунке 6 в верхней части показано взаимное расположение процессов, как они бы запускались на отдельных процессорах. В нижней части – получившееся результирующее расписание, соответствующее данной расстановке процессов.

Если хотя бы одно из трёх перечисленных условий нарушается, система выдаёт соответствующее диагностическое сообщение и расписание не строит.

Тем не менее, эти условия не являются достаточными. На рисунке 7 приведён пример, удовлетворяющий всем трём условиям, однако, какие бы расстановки процессов мы ни брали, расписание построить не удастся:

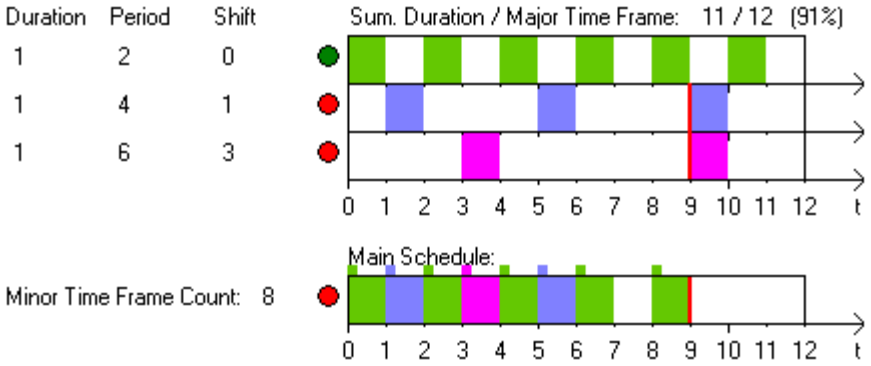

Рисунок 7

Однако формулировка и доказательство условий, отсекающих этот и подобные случаи, слишком сложны, чтобы приводить их здесь, поэтому в данной работе рассматриваться не будут. Если они всё же имеют место, алгоритм не может построить расписание, и в этом случае система выдаёт соответствующее сообщение.

5.2.2. Сложность задачи построения расписаний

В этом разделе мы покажем, что задача нахождения оптимального расписания, удовлетворяющего поставленным требованиям, как и многие другие задачи комбинаторной оптимизации, является NP-трудной.

Прежде всего, заметим, что для оптимального расписания число переключений процессора лежит между числом всех итераций всех процессов за один цикл (то есть, когда в данном расписании нет ни одной коллизии процессов) и числом, равным суммарной длительности всех итераций всех процессов (когда процессор переключается с одной задачи на другую каждый момент времени). Причём нижняя оценка для некоторых систем достигается.

В ряде работ [2, 3] исследуется задачи допустимости периодических систем реального времени, то есть возможности построить для них расписание. В этих работах рассматриваются системы в общем виде, в котором они были определены в разделе 3 данной работы. Авторы [2] доказывают теорему, что задача определения, является ли данная система недопустимой, принадлежит классу NP-трудных задач. К этой задаче они сводят NP-трудную задачу о «K одновременных конгруэнций». Однако в этих работах, как и во многих других, под периодом процесса понимается промежуток времени между его двумя последовательными моментами времени готовности, причём фактическое начало очередной итерации процесса может быть позже. То есть, такой подход ближе к спорадическим системам.

С другой стороны, в работе [6] рассматриваются системы реального времени, в которых задачи не могут прерываться. Авторы показывают, что задача определения того, возможно ли для системы с задачами без прерываний построить расписание, удовлетворяющее всем предельным срокам выполнения, – является NP-полной (а значит, и NP-трудной).

Теперь допустим, что в нашей рассматриваемой системе параметры длительностей и периодов подобраны так, что существует расписание, при котором не возникает коллизий процессов. Очевидно, оно будет оптимальным, так как при этом достигается нижняя оценка числа переключений процессора. Предположим, что рассматриваемый точный алгоритм строит это расписание за полиномиальное время. Но тогда получается, что за полиномиальное время была решена NP-полная задача, рассматриваемая в работе [6], чего не может быть в предположении $P \neq NP$. Если же такого расписания без прерываний для данной конфигурации системы не существует, то алгоритм строит какое-либо другое оптимальное расписание, в котором есть хотя бы одна коллизия процессов. Сравнив число переключений процессора с суммарным числом итераций процессов и обнаружив, что первое больше, алгоритм решает NP-трудную задачу из работы [6] с ответом «нет».

Таким образом, известная NP-трудная задача о построении расписания в системе без прерываний была сведена к данной задаче, что доказывает, что последняя также является NP-трудной.

5.2.3. Алгоритм построения расписаний

Поскольку задача построения расписаний для данной системы является NP-трудной, алгоритм не может найти оптимальное расписание за полиномиальное время. По сути, он является переборным и имеет экспоненциальную сложность.

Параметром перебора в алгоритме для данной неполной системы являются конфигурации полных систем, то есть векторы начальных смещений процессов $\{s_1, \ldots, s_n\}$, иначе говоря, всевозможные расстановки процессов относительно начала расписания. Для каждой такой расстановки алгоритм строит ровно одно корректное расписание и считает количество переключений между процессами. Из всех вариантов расстановок алгоритм запоминает и выбирает ту, в которой число переключений получилось меньше. Если данное число оказалось равным числу всех итераций всех процессов за один цикл расписания, то это значит, что найдено гарантированно оптимальное расписание, и алгоритм завершает свою работу досрочно. Если на каком-либо шаге при построении очередного варианта число переключений превысило минимальное число из уже обработанных вариантов, то алгоритм переходит к следующему шагу, не достроив неудачный вариант.

Составная часть рассматриваемого алгоритма – алгоритм построения отдельного варианта расписания – по сути, является эвристической, однако построенное расписание оказывается почти всегда оптимальным для данной расстановки или близким к оптимальному. Эта часть имеет две стратегии. Первая стратегия очень похожа на алгоритм EDF, с той лишь разницей, что она учитывает те моменты времени, в которых должен запуститься какой-либо процесс, и назначает на этот момент времени именно его. Однако в следующий момент времени может произойти переключение на другой процесс, чей относительный предельный срок выполнения ближе. Поэтому для такой стратегии нередки ситуации, когда последний фрагмент одного процесса лишний раз нарушает непрерывность другого. Следовательно, более предпочтительной является вторая стратегия, при которой запущенный процесс продолжает работать до своего завершения, либо до того, как его прервёт какой-нибудь другой процесс. Однако в этом случае алгоритм может нарушить *deadline* какого-либо процесса. Поэтому действия итогового алгоритма следующие: сначала он пытается применить вторую стратегию, а если нарушаются предельные сроки, то он возвращается к началу неудавшейся итерации процесса, применяет первую стратегию и проводит затем эвристическую оптимизацию. С применением только первой стратегии возможно построить расписание, кроме случаев, упомянутых в разделе 5.2.1.

Таким образом, на отдельном шаге может возникнуть потенциально неоптимальный вариант расписания (для данной расстановки), однако на следующем – оптимальный для данной расстановки. И если в новом варианте число переключений будет меньше, старый будет заменён. В конце концов, лучший из построенных вариантов почти всегда (в 97% случаев, проверявшихся на конфигурациях, в которых параметры выбирались случайным образом в небольших диапазонах) оказывается действительно оптимальным расписанием.

Без ограничения общности можно считать, что в момент времени 0 запускается один из процессов, иначе можно просто сместить рамки цикла расписания не необходимое число единиц времени. Поэтому далее будем полагать, что в момент времени 0 зафиксирован запуск какого-то одного процесса (например, с наибольшим периодом).

Рассмотрим случай, когда к моменту времени L, равному размеру цикла расписания, осталось нераспределённым несколько единиц времени некоторых процессов. Это не обязательно означает, что построить расписание не удалось. Момент L – это предельный срок выполнения только одного процесса, который начинается в момент времени 0, а у остальных *deadline* позже. Поэтому в этом случае алгоритм строит ещё и второй цикл расписания, учитывая процессы, не завершённые в первом цикле. Тогда если в конце второго цикла остаются незавершёнными ровно те же самые процессы, что и в конце первого (то, что это именно так, доказывается, например, в [2], хоть и без условия строгой периодичности запуска), расписание считается корректным и в качестве результата берётся второй цикл. На рисунке 8 приведён пример корректного расписания, при котором последние итерации жёлтого и зелёного процессов работают на границе первого и второго цикла расписания. Фрагменты, которые остались нераспределёнными в первом цикле, размещены во втором в моменты времени 4 и 5-6.

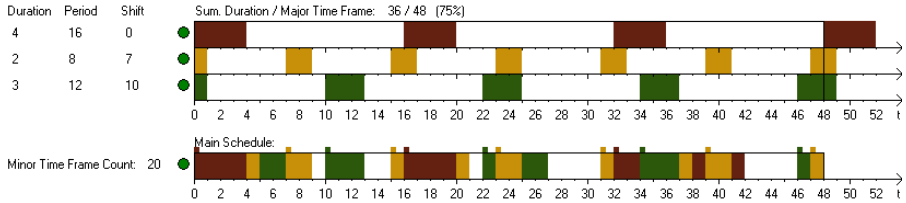

Рисунок 8

Перебор возможных вариантов расписания осуществляется рекурсивным образом. Процесс n проходит последовательно все свои допустимые значения s_n от 0 до $p_n - 1$, затем увеличивается значение начального смещения s_{n-1} процесса $(n-1)$, и для n-ного процесса смещение снова принимает все свои

допустимые значения. Когда s_{n-1} проходит все свои значения, смещается процесс $(n-2)$, и так далее, до 1-го процесса.

Однако не любой вектор начальных смещений $\{s_1, \ldots, s_n\}$ является допустимым. Если для каких-либо двух процессов разность их начальных смещений окажется кратной наибольшему общему делителю их периодов, то в некоторый момент времени совпадут запуски итераций этих процессов. Это условие также вытекает из соотношения Безу. Докажем его. Возьмём два процесса с периодами p_1 и p_2 и начальными смещениями s_1 и s_2. Пусть $НОД(p_1, p_2) = d > 1$, тогда периоды представимы в виде $p_1 = p'_1 d$, $p_2 = p'_2 d$, где p'_1 и p'_2 – взаимно простые числа. Пусть также существует натуральное s' такое, что: $s_2 - s_1 = s'd$. Так как p'_1 и p'_2 – взаимно простые, то по соотношению Безу найдутся коэффициенты k'_1 и k'_2 такие, что:

$k'_1 p'_1 - k'_2 p'_2 = 1$

Умножим это равенство на s' и обозначим $k_1 = k'_1 s'$, $k_2 = k'_2 s'$ и получим:

$s'k'_1 p'_1 - s'k'_2 p'_2 = s'$

$k_1 p'_1 - k_2 p'_2 = s'$

Затем умножим получившееся равенство ещё на d, учтём равенства $p_1 = p'_1 d$, $p_2 = p'_2 d$, $s_2 - s_1 = s'd$ и перенесём слагаемые из одной части в другую:

$k_1 p'_1 d - k_2 p'_2 d = s'd$

$k_1 p_1 - k_2 p_2 = s_2 - s_1$

$s_1 + k_1 p_1 = s_2 + k_2 p_2$

Мы получили, что если

$s_2 - s_1 \equiv НОД(p_1, p_2)$, (4)

то найдутся такие k_1 и k_2, что соответствующие итерации процессов будут запускаться одновременно, что недопустимо. Условие доказано.

Алгоритм при каждом смещении i-го процесса проверяет, что ни для какого другого процесса не выполняется условие (4), и только тогда строит расписание, иначе сразу переходит к следующему шагу. На рисунке 9 в качестве примера представлены два последовательных шага алгоритма:

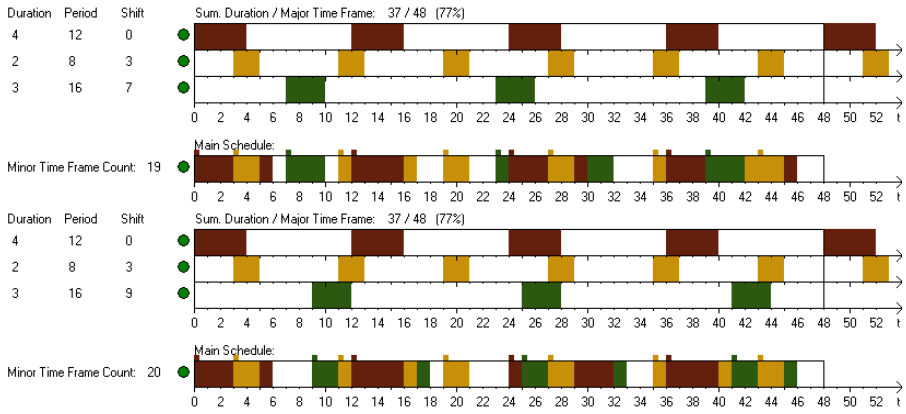

Рисунок 9

Между ними один шаг был пропущен, и алгоритм на него не тратит времени. Остаётся добавить, что существенно сокращают перебор случаи, когда пропускается сразу несколько шагов (например, когда *i*-й процесс пропускает какое-либо своё недопустимое смещение, весь перебор всевозможных смещений для процессов $j > i$ также пропускается).

Таким образом, точный алгоритм, выполняющий все требования, по своей сути является переборным, однако этот перебор можно во многом сократить. Тем не менее, для больших данных он малоприменим. Поэтому был предпринят **другой подход**: алгоритм в процессе перебора находит и отображает графически в качестве промежуточных результатов всё менее и менее фрагментированные расписания. И когда найдено расписание с достаточно малым числом переключений процессов, системный интегратор его может остановить и использовать промежуточный результат в качестве окончательного. При этом какое-либо расписание (неоптимальное) появляется почти сразу.

5.3. Графическое и табличное представления расписаний

Разработанная система отображает построенные окончательные (или промежуточные) расписания в графическом виде. Расписания могут быть представлены в двух видах: с детализацией (с разбивкой) по процессам или в одну строчку. Соответствующие примеры расписания приведены на рисунках 9 и 10.

Кроме описанного алгоритма, в системе могут работать и другие, решающие несколько модифицированные задачи составления расписаний. Например, для случая, когда строгая периодичность запуска процессов не так важна, могут применяться классические алгоритмы RM и EDF. Результаты их работы система также позволяет отобразить графически. На рисунке 10 представлены результаты работы трёх алгоритмов на одном примере. Красные штрихи

обозначают моменты запусков периодических процессов. Отсюда видно, что главный (точный) алгоритм выдерживает периодичность процессов, а классические алгоритмы её нарушают.

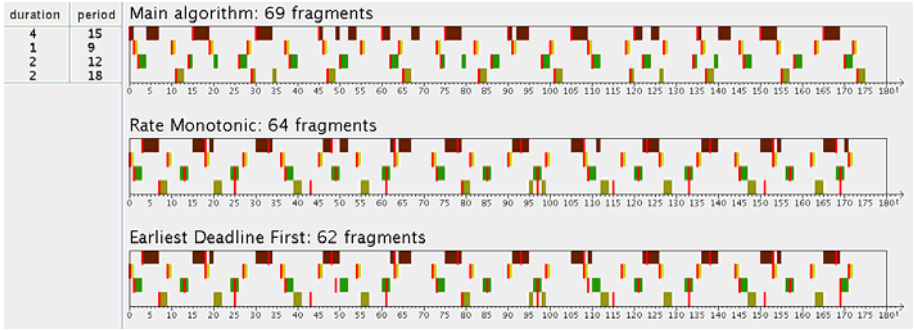

Рисунок 10

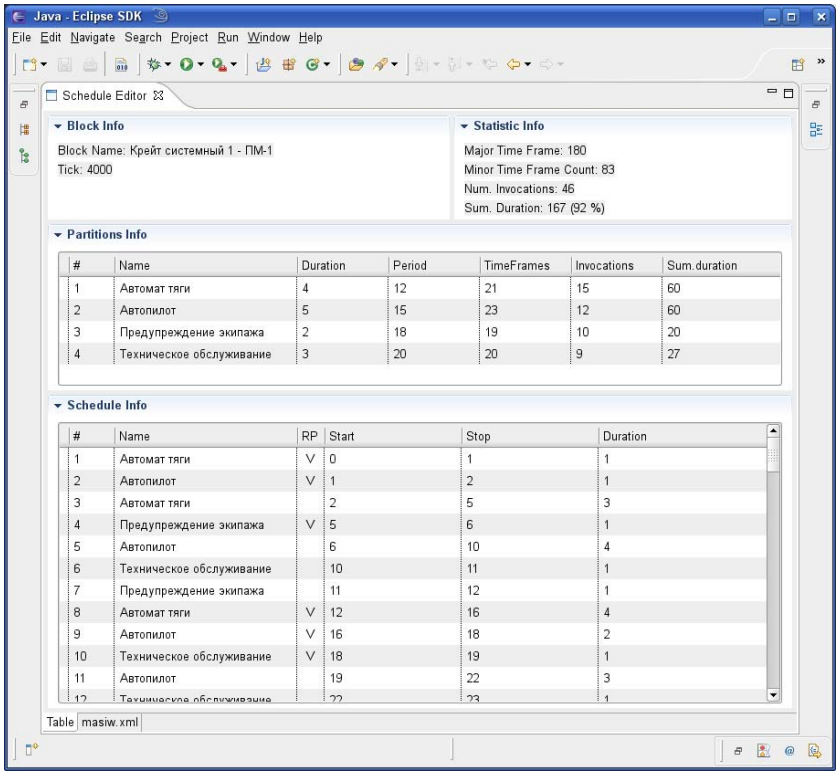

Рисунок 11

Разработанная система позволяет представить расписание и в табличном виде, в рабочем окружении интегрированной среды разработки Eclipse. На рисунке 11 приведён пример одной из таблиц, отображающих построенное расписание.

В верхней части таблицы приведена информация о процессоре, для которого было составлено расписание (его имя, размер тика), и краткая статистика (длина цикла расписания, число переключений процессора, число всех итераций всех процессов, суммарная длительность всех итераций всех процессов за один цикл и «плотность заполнения» расписания, выраженная в процентном соотношении). В середине таблицы перечисляются параметры процессов (партиций), такие как длительность и период и некоторая статистика по каждому процессу (число фрагментов, число итераций и суммарная длительность за один цикл расписания). Само расписание, занимающее всю оставшуюся часть таблицы, выражено не в виде функции S: $\{0, \ldots, L-1\} \to \{0, 1, \ldots, n\}$, а более компактно – в виде набора фрагментов (*англ.* Minor Time Frame) непрерывного выполнения конкретной задачи. Для каждого такого фрагмента указывается имя соответствующего ему процесса, содержит ли он момент времени периодического запуска процесса (флаг RP), а также начало, конец и длительность фрагмента.

Разработанная система позволяет изменять параметры процессов (длительность, период) путём занесения в соответствующую ячейку таблицы нового значения, а затем вызвать алгоритм построения расписания, чтобы получить его новую версию.

Кроме того, поскольку системный интегратор может остановить процесс поиска оптимального расписания, ему требуется возможность довести промежуточный результат до оптимального путём редактирования расписания. Для этого система позволяет изменить длительности существующих фрагментов, либо вообще удалить их, а также предоставляет меню, вызывающее мастер создания новых фрагментов.

5.4. Проверка корректности расписания

В случае если изменения нарушают соответствие расписания параметрам процессов, последние выделяются и для них отмечаются те параметры, которые оказались некорректными. Причём в самом расписании отмечаются те моменты времени, которые вызвали проблемы.

На рисунке 12 приведён пример, в котором после редактирования нарушилась периодичность запуска жёлтого процесса из-за того, что была увеличена длительность коричневого (добавлен момент времени 43 для коричневого процесса). Соответственно, для коричневого была помечена некорректная длительность, а для жёлтого – период.

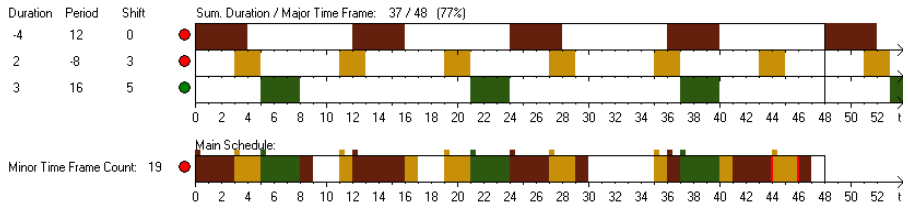

Рисунок 12

Алгоритм проверки сначала проходит по расписанию и определяет число процессов. Затем для каждого процесса вычисляется промежуток времени между первой и второй итерацией и длительность первой итерации и запоминает эти значения. Для всего оставшегося расписания он сравнивает длительности и периоды текущего процесса с сохранёнными значениями. Сумма начального смещения и промежуток между началом последней итерации и концом цикла расписания – также сравнивается с периодом. Случай, когда процесс работает на границе двух циклов расписания, тоже учитывается. Если на каком-либо этапе возникает несогласованность данных, алгоритм отмечает соответствующий процесс и его фрагмент в расписании.

Если все параметры оказались согласованными, то можно говорить о том, что алгоритм по имеющемуся расписанию восстановил список процессов и их исходные параметры.

Алгоритм не может определить, какой из нескольких фрагментов процесса привёл к нарушению согласованности параметров, поэтому он отмечает тот фрагмент, на котором расхождение стало очевидным.

Конечно, априори нельзя определить, каким алгоритмом (точным, RM или EDF) было построено расписание. Поэтому при проверке на корректность результатов работы классических алгоритмов всё расписание окажется помечено красными штрихами, а для большинства процессов будут отмечены (или даже неправильно определены) периоды. Длительности, тем не менее, будут указаны верно.

6. Заключение

В данной работе, при решении задачи составления расписаний для целевой системы, были получены следующие результаты:

- Сформулированы необходимые условия существования расписания для системы, удовлетворяющей заданным ограничениям.
- Доказана NP-трудность решаемой задачи построения расписаний.
- Разработан алгоритм, решающий задачу построения расписаний точно и находящий оптимальное расписание или

близкое к оптимальному. Использован метод ограниченного перебора.

- Разработана система получения промежуточных результатов работы алгоритма и управления перебором.
- Разработанная система отображает построенные расписания в графическом и табличном виде.
- Предоставлена возможность редактировать построенные расписания и проверять корректность изменений.
- Реализованы классические алгоритмы RM и EDF в качестве альтернативы для тех случаев, когда строгая периодичность запусков процессов не так важна.

В работе было показано, что для разных, несколько похожих задач построения расписаний существуют совершенно различные алгоритмы. Особенно хорошо это видно на примере контраста разработанного алгоритма и классических алгоритмов Rate Monotonic и Earliest Deadline First. Добавление всего лишь одного требования строгой периодичности запуска процессов делает из полиномиально разрешимой задачи NP-трудную.

Поэтому был предпринят другой подход: для целевой системы реального времени была разработана система инструментов, позволяющих системным интеграторам автоматизировать отдельные операции составления расписаний. Для систем с небольшим числом процессов и с их небольшими периодами алгоритм находит оптимальное расписание (т.е. решает задачу точно). А для больших систем, когда алгоритм работает очень долго, разработанный механизм время от времени выводит промежуточные корректные, но не оптимальные результаты в графическом виде. И в тот момент, когда ответственному персоналу покажется, что полученное расписание уже приемлемо или его легко довести до оптимального, он может остановить работу алгоритма, сохранить промежуточный результат, вручную довести его до оптимального, а затем проверить на корректность. Для тех случаев, когда скорость построения расписаний важнее требования строгой периодичности запуска процессов, система инструментов позволяет применять классические полиномиальные алгоритмы.

Литература

[1] Mohammadi A., G. Akl S.G. Scheduling Algorithms for Real-Time Systems // School of Computing, Queen's University . 2005. Technical Report N 2005–499.

[2] Leung J.Y.-T., Merrill M.L. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks // Information Processing Letters. 1980. Vol. 11. N 3. P.115–118.

[3] Baruah S.K., Rosier L.E., Howell R.R. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor // Real Time Systems. 1990. Vol. 2. P. 301–324.

[4] Brucker P., Kampmeyer T. Tabu search algorithms for cyclic Machine scheduling problems // Journal of Scheduling. 2005. N 8. P. 303–322.

[5] Brucker P., Kampmeyer T. A general model for cyclic machine scheduling problems // Discrete Applied Mathematics. 2008. Vol. 156. N 13. P. 2561–2572.

[6] Georges L., Muhlethaler P., Rivierre N. A Few Results on Non-Preemptive Real time Scheduling // Rapport de recherché. 200. N 3926.

[7] Сорокин С. Системы Реального Времени // СТА. 1997. №2. С. 22–29.

[8] Горошко Е. Операционные системы реального времени [PDF] (http://www.qnxclub.net/files/articles/rtos/rtos.html).

[9] Timmerman M., Beneden B.V., Uhres L. RTOS Evaluation Kick Off! // Real-Time Magazine. 1998. N3. P. 6–10.

[10] Операционные системы реального времени для авионики: обзор [HTML] (http://rnd.cnews.ru/reviews/index_science.shtml?2008/05/05/299461_1).

[11] Зеленов С.В. Планирование строго периодических задач в системах реального времени // Труды Института системного программирования РАН. 2011. Т. 20. С. 113–122.

Automation of scheduling for periodic real-time systems

A.V. Tretyakov.
andrromeda@ispras.ru
ISP RAS, Moscow, Russia

Abstract. In the paper, we consider the scheduling problem for strictly periodic real-time systems. Strict periodicity means that all release points of all instances of any task must produce an arithmetic progression. Classical scheduling algorithms, such as Earliest Deadline First (EDF) and Rate-Monotonic Scheduling (RMS), are not applicable under strict periodicity constraint. The main problem is to search release points for all tasks. In fact, this search is NP-hard problem.

First, we study some necessary schedulability conditions for both classical and strictly periodic schedulability problem. Next, we suggest scheduling algorithm that consists of two main parts: heuristic algorithm of release points search, and scheduling algorithm for given release points. The algorithm of release points search is based on some ideas in number theory that allows iterate not all possible release points but only such points that with a high probability yields correct schedule. The scheduling algorithm for given release points is based on ideas of EDF algorithm.

Finally, we present developed tool set that provides automated scheduling of given tasks, allows visualization of generated schedule, provides GUI to edit schedule, and supports validation of built schedules. This tool set is a part of workspace for design of modern avionics systems developed in ISP RAS.

Keywords: scheduling, real-time systems, hard periodic tasks, deadlines, NP-hard problem, Integrated Modular Avionics

References

[1]. Mohammadi A., G. Akl S.G. Scheduling Algorithms for Real-Time Systems. School of Computing, Queen's University . 2005. Technical Report N 2005–499.
[2]. Leung J.Y.-T., Merrill M.L. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. Information Processing Letters. 1980. Vol. 11. N 3. P.115–118.
[3]. Baruah S.K., Rosier L.E., Howell R.R. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. Real Time Systems. 1990. Vol. 2. P. 301–324.
[4]. Brucker P., Kampmeyer T. Tabu search algorithms for cyclic Machine scheduling problems. Journal of Scheduling. 2005. N 8. P. 303–322.
[5]. Brucker P., Kampmeyer T. A general model for cyclic machine scheduling problems. Discrete Applied Mathematics. 2008. Vol. 156. N 13. P. 2561–2572.
[6]. Georges L., Muhlethaler P., Rivierre N. A Few Results on Non-Preemptive Real time Scheduling. Rapport de recherché. 200. N 3926.
[7]. Sorokin S. Sistemy Real'nogo Vremeni [Real Time Systems]. Sovremennye tekhnologii avtomatizatsii [Contemporary Technologies in Automation]. 1997. No2. P. 22–29 (in Russian).

[8]. Goroshko E. Operatsionnye sistemy real'nogo vremeni [Real Time Operating Systems]. Khar'kov, 2003 (in Russian).
[9]. Timmerman M., Beneden B.V., Uhres L. RTOS Evaluation Kick Off! Real-Time Magazine. 1998. N3. P. 6–10.
[10]. Operatsionnye sistemy real'nogo vremeni dlya avioniki: obzor [Real Time Operating Systems for Avionics: Review]. R&D.CNews.ru, 2008 (in Russian). http://rnd.cnews.ru/reviews/index_science.shtml?2008/05/05/299461_1
[11]. Zelenov S.V. Planirovanie strogo periodicheskikh zadach v sistemakh real'nogo vremeni [Scheduling of Strictly Periodic Tasks in Real-Time Systems]. Trudy ISP RAN [Proceedings of ISP RAS], 2011, vol. 20, pp. 113–122 (in Russian).