Планирование строго периодических задач в системах реального времени

С.В. Зеленов
zelenov@ispras.ru

Аннотация. Одним из важнейших аспектов функционирования систем реального времени является планирование задач. Классические алгоритмы планирования периодических задач работают лишь в случае, когда время запуска каждой задачи может варьироваться внутри разных периодов ее выполнения. Однако, в настоящее время имеется потребность в составлении таких расписаний, в которых время между соседними запусками одной периодической задачи было бы фиксировано и равнялось бы длине периода. Такое дополнительное требование не позволяет использовать в планировщике классические алгоритмы планирования. В работе предлагается алгоритм построения расписаний, близких к оптимальным в смысле минимизации общего количество прерываний задач. Оптимизация производится с учетом того, чтобы за приемлемое время строилось как можно более оптимальное расписание.

Ключевые слова: система реального времени; периодическая задача; планирование; авионика; непересекающиеся арифметические прогрессии.

1. Введение

Задачи реального времени составляют одну из сложнейших и крайне важных областей применения вычислительной техники. Как правило, они связаны с контролем и управлением процессами, являющимися неотъемлемой частью современной жизни. Управление прокатными станами, роботами, движение на автомагистралях, контроль за состоянием окружающей среды, управление атомными и космическими станциями, авиационными системами, GRID-системы и многое другое – область задач реального времени. Эти задачи предъявляют такие требования к аппаратному и программному обеспечению, как надежность, высокая пропускная способность передающей среды в распределенных системах, своевременная реакция на внешние события и т.д. Для выполнения этих требований и создаются системы реального времени [1].

Одним из важнейших аспектов работы систем реального времени является своевременное обеспечение процессорными ресурсами всех выполняющихся в системе задач. Этим аспектом занимается специальная подсистема планирования задач. Проблема планирования как зависимых, так и

независимых задач в однопроцессорной или многопроцессорной системах реального времени достаточно хорошо изучена. Многочисленные результаты, полученные в этой области, приведены, например, в работах [2], [3], [4], [5], [6], [7], [8].

Наиболее частым случаем задач реального времени являются *периодические* задачи, каждая из которых должна быть запущена и полностью выполнить свою работу на каждом соответствующем этой задаче периоде времени. Примерами периодических задач являются разнообразные датчики, функции управления оборудованием, и пр. В реальных ситуациях, как правило, редко удается спланировать работу периодических задач так, чтобы каждая задача на каждом своем периоде работала непрерывно. Обычно постановка проблемы планирования предполагает, что любая задача в любой момент может быть прервана для выполнения более приоритетной задачи. В теории планирования предполагается, что прерывание задачи и передача управления другой задаче является бесплатной операцией. Однако, очевидно, в реальности такая операция требует некоторых ресурсов системы.

В классической постановке проблемы планирования задач время запуска периодической задачи не привязывается к определенному моменту времени внутри периода, но может варьироваться. Однако, в настоящее время в ряде случаев, например, при разработке систем управления авионикой, возникает необходимость обеспечить, чтобы запуски каждой периодической задачи выполнялись через строго определенные моменты времени (такие задачи мы будем называть *строго периодическими*). Это дополнительное очень сильное требование не позволяет напрямую использовать классические алгоритмы составления расписаний, такие как *Rate Monotonic* (RM), в котором преимущество отдается задачам с самыми короткими периодами выполнения, и *Earliest Deadline First* (EDF), в котором предпочтение отдается задачам с наиболее ранним предельным временем завершения выполнения [9].

В настоящей работе предлагается алгоритм построения расписания для строго периодических задач в однопроцессорной системе, так чтобы минимизировалось общее количество прерываний задач. Поскольку, вообще говоря, подобная проблема минимизации предполагает переборное решение, основные усилия в настоящей работе были направлены на то, чтобы алгоритм как можно быстрее находил достаточно хорошее приближенное решение.

2. Формальная постановка задачи

Пусть имеется один процессор, на котором требуется выполнять несколько независимых задач. Каждой задаче для своего завершения требуется некоторое процессорное время. При этом процессор может иногда прерывать выполнение любой задачи T, начать выполнять другую задачу T' с более высоким приоритетом, а через некоторое время продолжить выполнение задачи T. Процессорное время разбито на равные отрезки – *тики*, так что на каждом тике может выполняться только одна задача, т.е. запуск, завершение,

прерывание и возобновление выполнения (после прерывания) любой задачи возможно только в начале тика. Переключение с выполнения одной задачи на другую считается мгновенным.

Будем считать, что точки, являющиеся началами тиков, последовательно перенумерованы неотрицательными целыми числами начиная с нуля. Везде далее под словом «*точка*» будем подразумевать эти перенумерованные точки, а также будем отождествлять точки с их номерами. Будем говорить, что некоторая точка *x принадлежит* задаче *T*, если *x* является началом тика, на котором выполняется задача *T*.

Задача $T(d, p)$, требующая для своего выполнения d тиков, называется *строго периодической*, если ее запуск периодически повторяется, причем между соседними точками запуска проходит ровно p тиков, и выполнение задачи на каждом периоде должно завершиться до следующей точки запуска. Величина p называется *периодом*, d – *длительностью* задачи T. Всегда верно соотношение $p \geq d \geq 1$.

Для данной строго периодической задачи $T(d, p)$ точка r называется *начальной точкой запуска*, если $p > r \geq 0$, и задача T запускается только в точках $r + kp$ для всех целых $k \geq 0$. В этих обозначениях условия строгой периодичности задачи T формально записываются так:

```
задача T запускается в каждой точке r + kp,
на каждом участке [ r + kp, r + kp + (p – 1) ]
включительно имеется ровно d точек, принадлежащих T.
```

Обратим внимание на то, что завершение выполнения строго периодической задачи на очередном периоде *не является* прерыванием этой задачи. Действительно, в начале следующего периода будет произведен *новый запуск* этой задачи (но *не* возобновление выполнения после прерывания).

В рамках веденных здесь определений формальная постановка основной задачи формулируется следующим образом.

Пусть заданы n строго периодических задач { $T_i(d_i, p_i)$ }, выполняющихся на одном процессоре. Требуется найти для них начальные точки { r_i } и распределить время выполнения задач между собой так, чтобы минимизировать общее количество прерываний задач.

Как видно из приведенной формулировки, задача распадается на две подзадачи:

```
найти начальные точки { ri }, так чтобы для любых двух
задач никакие их точки запуска не совпадали;
распределить время выполнения задач между собой, т.е.
построить расписание.
```


3. Предварительные сведения

Наибольший общий делитель (НОД) n целых чисел $a_1, \ldots, a_n$ обозначается $(a_1, \ldots, a_n)$. Два целых числа a и b называются *сравнимыми по модулю* целого числа c (записывается: $a \equiv b \pmod{c}$), если разность $(a – b)$ делится на c.

Из теории чисел известна следующая теорема (см., например, [10]):

Теорема 1. Пусть даны целые числа a, b и c, и дано сравнение с одним неизвестным x:

$$ax \equiv b \pmod{c}.$$

Тогда если b не делится на (a, c), то сравнение не имеет решений, иначе – сравнение имеет (a, c) решений по модулю c, причем все эти решения сравнимы друг с другом по модулю $c / (a, c)$.

Из Теоремы 1 вытекает следующая теорема:

Теорема 2. Пусть даны целые числа t, c, c'. Тогда для всевозможных целых l решения сравнения

$$x \equiv t + cl \pmod{c'} \qquad (1)$$

удовлетворяют соотношению

$$x – t \equiv 0 \pmod{(c, c')}. \qquad (2)$$

Доказательство. По Теореме 1, для любого l сравнение (1) имеет единственное решение. Решения сравнения (1) для разных l и l' совпадают, если $c(l – l') \equiv 0 \pmod{c'}$, т.е. (по Теореме 1) если l и l' сравнимы друг с другом по модулю $c' / (c, c')$. Отсюда следует, что существует $c' / (c, c')$ различных решений сравнения (1) для разных l. Следовательно, поскольку для любого l разность $x – t = cl$ делится на (c, c'), и число c' делится на (c, c'), то все $c' / (c, c')$ различных решений сравнения (1) для разных l удовлетворяют соотношению (2). ►

Если даны n периодических задач, то расписание для них достаточно построить на участке длины *наименьшего общего кратного* (НОК) их периодов. Планировщик в этом случае будет выполнять задачи, циклически проходя по построенному расписанию.

Необходимым условием возможности построения расписания на однопроцессорной системе для n периодических задач с длительностями d_i и периодами p_i является условие на суммарную загруженность процессора:

$$\sum (d_i / p_i) \leq 1$$

Для алгоритма EDF это условие является и достаточным [9]. Однако, в случае строго периодических задач это условие уже не является достаточным.

Пример. Пусть даны три строго периодические задачи: $T_1(1, 2)$, $T_2(1, 4)$, $T_3(1, 6)$. Тогда суммарная загруженность процессора строго меньше 1, а НОК периодов этих задач равен 12. Без ограничения общности будем считать, что задача T_1 стартует в точке 0. Тогда она будет запускаться во всех четных точках. В какой бы из оставшихся свободных точек 1 или 3 ни стартовала задача T_2, она будет запускаться каждый четвертый тик, а значит свободными для запуска задачи T_3 останутся точки, образующие арифметическую прогрессию с периодом 4. Однако на такой прогрессии невозможно уместить прогрессию с периодом 6. Таким образом, задачу T_3 спланировать не удастся.

4. Алгоритм перебора потенциальных начальных точек

Сопоставим каждой задаче $T_i(d_i, p_i)$ множество R_i, состоящее из p_i точек, последовательно перенумерованных числами от 0 до $(p_i - 1)$. Каждое множество R_i содержит все потенциальные начальные точки для задачи T_i. Однако, в ходе выбора начальных точек для некоторых из задач, множества потенциальных начальных точек для остальных задач будут постепенно сокращаться. Действительно, если для некоторой задачи T_k выбрана начальная точка r_k, то для любого $m \neq k$ нужно удалить из отрезка R_m все такие точки r, что

$$r - r_k \equiv 0 \pmod{(p_k, p_m)}, \qquad (3)$$

поскольку если в качестве начальной точки для T_m будет выбрана точка со свойством (3), то тогда в некоторый момент, по Теореме 2, точки запуска задач T_k и T_m совпадут.

Замечание. Иногда для некоторых задач может и не произойти реального сокращения множеств потенциальных начальных точек в силу того, что все точки, которые должны быть удалены согласно приведенному выше правилу, уже были удалены в результате выбора начальных точек для других задач. Поясним это на следующем примере. Пусть даны три задачи: $T_1(1, 4)$, $T_2(1, 8)$, $T_3(1, 10)$. Пусть для первой задачи выбрана начальная точка $r_1 = 0$. Тогда из множества R_2 будут удалены точки 0 и 4 (поскольку (4, 8) = 4), а из множества R_3 – точки 0, 2, 4, 6, 8 (поскольку (4, 10) = 2). Тогда если для второй задачи выбрать начальную точку $r_2 = 2$, то из множества R_3 нужно будет удалить точки 0, 2, 4, 6, 8 (поскольку (8, 10) = 2), однако все эти точки уже были удалены из этого множества.

Замечание. В том же примере из предыдущего замечания, если для второй задачи выбрать начальную точку $r_2 = 1$, то реальное сокращение множества R_3 произойдет, а именно нужно будет удалить точки 1, 3, 5, 7, 9.

Будем считать, что выбор начальных точек происходит последовательно по порядку, начиная с r_1 и заканчивая r_n. Из сделанных выше замечаний следует,

что для любого m состав множества R_m зависит как от периодов задач, так и от выбора всех предыдущих начальных точек $r_1, \ldots, r_{m-1}$.

Установим на кортежах начальных точек $\{ r_i \}$ лексикографический порядок: будем говорить, что кортеж $(r_1, \ldots, r_n)$ предшествует кортежу $(r'_1, \ldots, r'_n)$, если для некоторого m выполнены соотношения:

$$r_m < r'_m;$$

$$r_k = r'_k \text{ для всех } k < m.$$

Алгоритм перебора потенциальных начальных точек в первом приближении состоит в переборе кортежей $\{ r_i \}$ в лексикографическом порядке; при этом для любого k перебор r_k осуществляется по актуальному множеству R_k (т.е. полученному в соответствии с выбором начальных точек $r_1, \ldots, r_{k-1}$).

В реальных приложениях количество задач n может достигать значения 20, 30, и даже 40. При этом реальные периоды задач могут достигать значения в несколько тысяч тиков. На подобных входных данных приведенный алгоритм будет работать неприемлемо долго. Однако, на деле нам не требуется перебирать все варианты кортежей $\{ r_i \}$. Требуется как можно быстрее найти такой вариант кортежа начальных точек, при котором количество прерываний задач будет как можно меньше.

Исходя из этого, мы модифицируем алгоритм так, чтобы в первую очередь перебирались те варианты кортежей $\{ r_i \}$, на которых наиболее вероятно будет происходить непрерывное выполнение задач. Более точно, мы сконцентрируемся на обеспечении как можно более длительного непрерывного выполнения первых тиков после запуска каждой задачи. Для этого мы воспользуемся двумя следующими приемами.

Первый прием состоит в следующем. Заметим, что при выборе очередной начальной точки r_k мы фактически резервируем один начальный тик выполнения задачи T_k. Однако можно таким же образом резервировать для этой задачи s_k начальных тиков, где $d_k \geq s_k \geq 1$. При этом для перебора следующих начальных точек r_m (при $m > k$) придется из R_m удалять не только точки, соответствующие (в смысле соотношения (3)) точке r_k, но все точки, соответствующие всем точкам из участка $[r_k, r_k + s - 1]$.

Очевидно, наилучшим вариантом будет такое резервирование, когда для всех задач $s_k = d_k$, поскольку при этом все задачи будут выполняться без прерываний. Однако такое возможно далеко не всегда. Действительно, после такого резервирования, проведенного лишь для нескольких первых задач, может оказаться, что для какой-то из оставшихся задач множество потенциальных начальных точек стало пустым.

Заметим, что чем меньше период задачи, тем чаще она будет запускаться, а, следовательно, приоритетным является зарезервировать $s_k = d_k$ для задач с наименьшими периодами. Исходя из этого, поскольку задачи у нас

отсортированы по возрастанию их периодов, а резервирование осуществляется по порядку номеров задач от 1 до n, будем резервировать начальные тики для задач по остаточному принципу: для текущей задачи T_k будем резервировать s_k как можно больше, чтобы для остальных задач T_m (при $m > k$) можно было зарезервировать хотя бы по одному тику (будем говорить, что в этом случае для задачи T_k резервирование *корректно*).

Алгоритм поиска наибольшего корректного резервирования s_k основан на методе половинного деления отрезка [1, d_k], так что на каждом шаге деления левая граница отрезка позволяет корректное резервирования, а правая – нет.

Второй прием для обеспечения как можно более длительного непрерывного выполнения первых тиков после запуска каждой задачи состоит в следующем. Заметим, что после осуществления резервирования для первых k-1 задач и удаления всех соответствующих точек из R_k, множество R_k представляет собой участок целых чисел от 0 до p_k – 1, в котором некоторые точки «выколоты», а оставшиеся точки образуют несколько непрерывных участков. Для того, чтобы обеспечить наибольшее резервирование s_k, следует в первую очередь перебирать левые концы этих оставшихся непрерывных участков из R_k в порядке убывания длины соответствующего непрерывного участка.

5. Алгоритм построения расписания для данного набора точек старта

Алгоритм построения расписания является модификацией алгоритма EDF. Модификация направлена на то, чтобы обеспечить как можно более длительное непрерывное выполнение первых тиков после запуска каждой задачи.

В алгоритме EDF задачам динамически назначаются приоритеты, так что наибольший приоритет имеет та (еще не завершенная) задача, для которой ближе всего очередная следующая точка запуска.

Наша модификация этого алгоритма состоит в следующем. Рассмотрим две последовательные точки x и y запусков задач ($x < y$). Пусть в точке x запущена задача T, для которой зарезервировано s начальных тиков. Построим по алгоритму EDF предварительное расписание на участке [$x + s$, $y - 1$]. После этого, если на этом участке имеются точки, принадлежащие задаче T, то перенесем их в начало этого участка. В результате все точки, принадлежащие задаче T на участке [x, $y - 1$], будут следовать непрерывно в начале этого участка. Таким же образом будем поступать для всех последовательных точек запусков задач.

6. Результаты экспериментов

Предложенный в настоящей работе алгоритм предполагается использовать в реальных условиях, когда количество задач может достигать значения 20, 30 и

даже 40. При этом расчетная априорная загрузка процессора составляет порядка 50–70%.

Предложенный алгоритм испытывался на множествах задач (от 16 до 44 задач) с псевдослучайными параметрами, удовлетворяющими следующим свойствам:

Период каждой задачи является делителем числа 29·3·53 = 192000 (с тем, чтобы, из соображений потребления ресурсов алгоритмом, НОК периодов всех задач не превышал бы 192000);

Длительность для каждой задачи равна (с учетом округлений) либо ¼, либо ½, либо ¾ от периода этой задачи, деленного на количество задач (так что максимальная загрузка процессора не будет превышать 75%).

Эксперименты проводились на ПК с процессором Intel с тактовой частотой 2,4 GHz и ОЗУ объемом 1 GB. Для каждого множества задач осуществлялся запуск предложенного алгоритма для поиска наилучшего (в смысле общего количества прерываний задач) расписания в течение первых 10 секунд работы алгоритма. Результаты экспериментов приведены в таблице (под *лучшим решением* здесь понимается лучшее, найденное за 10 с).

Число задач	НОК периодов задач	Общая загрузка	Время нахождения первого решения (мс)	Время нахождения лучшего решения (мс)	Количество прерываний на участке НОК	Среднее количество прерываний (на 1000 тиков)
16	48000	56.43%	18	451	455	9.48
17	96000	51.90%	34	5045	1807	18.82
18	192000	52.81%	125	8559	2047	10.66
19	96000	57.93%	66	1204	1409	14.68
20	48000	56.05%	26	6412	941	19.60
21	192000	53.98%	105	7650	2425	12.63
22	96000	53.69%	89	3635	1397	14.55
23	96000	59.19%	37	1573	1941	20.22
24	192000	54.58%	103	4758	1859	9.68
25	48000	52.87%	68	6635	738	15.37

26	96000	54.74%	75	3325	1363	14.20
27	192000	55.47%	206	6437	2160	11.25
28	96000	61.22%	66	235	2559	26.66
29	192000	59.37%	238	3480	2819	14.68
30	96000	64.09%	106	9782	2560	26.67
31	192000	51.29%	189	2818	2301	11.98
32	96000	56.58%	96	3730	1878	19.56
33	192000	56.19%	240	6497	2108	10.98
34	96000	64.68%	130	3706	2308	24.04
35	192000	56.28%	290	9993	2978	15.51
36	192000	65.39%	193	2848	4652	24.23
37	96000	58.30%	129	1134	2109	21.97
38	192000	60.03%	302	1698	4348	22.65
39	96000	66.39%	155	9234	2628	27.37
40	192000	61.80%	328	2961	4717	24.57
41	192000	58.73%	229	4857	4531	23.60
42	96000	57.96%	181	2527	1678	17.48
43	192000	69.38%	385	8249	6032	31.42
44	192000	58.58%	781	5073	4320	22.50

Табл. 1. Результаты экспериментальных испытаний алгоритма.

Как видно из результатов испытаний, предложенный алгоритм позволяет в условиях с реальными общими характеристиками достаточно быстро строить расписания со средним количеством прерываний, как правило, не превышающим величины 25 прерываний на 1000 тиков, что является вполне приемлемым результатом.

7. Заключение

В работе предложен алгоритм построения расписания, близкого к оптимальному, для строго периодических задач на однопроцессорной системе, где в качестве оптимизационного критерия рассматривается требования минимизации общего количества прерываний задач.

Эксперименты показывают, что алгоритм позволяет за приемлемое для человека время (порядка 10 секунд) получать достаточно оптимальное расписание. При количестве задач 20–40, когда периоды задач могут достигать значения в несколько тысяч тиков, алгоритм позволяет получать расписания со средним количеством прерываний не более 25 прерываний на 1000 тиков.

В настоящее время алгоритм активно используется в реальных проектах по разработке систем управления авионикой.

Литература

[1] А.С.Косачев, И.Б.Бурдонов, В.Н.Пономаренко. Операционные системы реального времени. // Препринт Института системного программирования РАН, 2006, № 14. http://citforum.ru/operating_systems/rtos/

[2] Liu J.W.S. Real-Time Systems. // Prentice Hall, Englewood Cliffs, NJ, 2000. 600 p.

[3] Cottet F., Kaiser J., Mammeri Z. Scheduling in Real-Time Systems. // John Wiley & Sons Ltd. 2002. 282 p.

[4] N.N. Kuzjurin. Multiprocessor scheduling and expanders. Information Process. Letters, 1994, v. 51, № 6, 315–319.

[5] Н. Н. Кузюрин. Многопроцессорные расписания и комбинаторные конфигурации. Дискретная математика, 1995, т. 7, № 2, 77–87.

[6] S. Zhuk, A. Tchernykh, N. Kuzjurin, A. Pospelov, A. Shokurov, A. Avetisyan, S. Gaissaryan, D. Grushin. Comparison of Scheduling Heuristics for Grid Resource Broker. Proc. of the Third International Conference on Parallel Computing Systems (PCS2004). IEEE Computer Society Press, 2004, 388–392.

[7] A. Tchernykh, J. M. Ramirez, A. Avetisyan, N. Kuzjurin, D. Grushin, S. Zhuk. Two Level Job-Scheduling Strategies for a Computational Grid. Proc. of the Second Grid Resource Management Workshop. LNCS 3911, 2006, 774–781.

[8] A. Tchernykh, U. Schwiegelsohn, R. Yahyapour, N. Kuzjurin. On-line Hierarchical Job Scheduling in Grids with admissible allocation. J. of Scheduling. 2010, v. 13, № 5, 545–552.

[9] Liu C. and Layland J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. // Journal of ACM , 20(1): 46–61, 1973.

[10] И.М.Виноградов. Основы теории чисел. // М.-Л.: Гостехиздат, 1952. 180 стр.