ПЛАНИРОВАНИЕ ЗАДАНИЙ В СИНХРОННЫХ СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

В. В. Никифоров

Санкт-Петербургский институт информатики и автоматизации РАН
199178, Санкт-Петербург, 14-я линия В.О., д.39
nik@iias.spb.su

УДК 681.3
В. В. Никифоров. **Планирование заданий в синхронных системах реального времени** // Труды СПИИРАН. Вып. 1, т. 1. — СПб: СПИИРАН, 2002.
Аннотация. *Предлагается подход к назначению параметров периодичности выполнения задач в синхронных системах реального времени. Приводятся формы представления и процедуры выполнения графиков активизации задач для систем с двоичным и двоично-десятичным ранжированием задач по периодичности их выполнения.* — Библ. 10 назв.

UDC 681.3
V. V. Nikiforov. **Task scheduling in time-triggered real-time systems** // SPIIRAS Proceeding. Issue 1, v. 1. — SPb: SPIIRAS, 2002.
Abstract. *Approach is suggested for setting of the task execution periods in time-triggered real-time systems. Schedule presentation forms and schedule execution procedures are adduced for the systems with binary ranged and binary-decimal ranged task periods.* — Bibl. 10 items

Системами реального времени (СРВ) называют такие программные системы, к которым наряду с требованиями логической корректности результатов вычислений предъявляются требования своевременности получения этих результатов [1]. Наличие требований ко времени выполнения программных компонент является ключевой особенностью СРВ. В зависимости от характера требований ко времени выполнения программных компонент различают мягкие и жесткие СРВ. Для мягких СРВ нарушение установленных сроков выполнения задач является нежелательным, но допустимым отклонением в работе системы. При этом, естественно, снижается качество функционирования системы, что, однако, не приводит к недопустимым последствиям. Для жестких СРВ нарушение установленных сроков выполнения возложенных на систему функций недопустимо, соответствующие операционные системы должны гарантированно обеспечивать своевременное выполнение программных компонент.

Значительная часть жестких СРВ относятся ко встроенным системам, для которых характерна организация автономного электропитания от миниатюрных аккумуляторов. В этих условиях экономия энергопотребления является одним из ключевых обстоятельств, определяющих организацию системы. Необходимость экономии энергопотребления приводит к требованию минимизации объема используемой памяти, поскольку элементы оперативной памяти относятся к числу наиболее энергоемких узлов системы. Соответственно и реализация сервисов операционных систем для СРВ этого типа должна быть ориентирована на минимизацию использования памяти.

В связи с асинхронным характером протекания множества разнородных внешних процессов, с которыми взаимодействует СРВ, возникает проблема предсказуемости поведения программных систем реального времени. Активизация тех или иных программных компонент является реакцией СРВ на соответствующие внешние события. В то же время, последовательность внешних событий не является заранее предопределенной. Это приводит к произвольно-

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

му порядку активизации программных компонент. Вследствие этого в ходе работы программной системы возможно возникновение перегрузок, т.е. ситуаций, в которых потребность в динамически распределяемых ресурсах (процессорное время, память) превышает их наличие. В таких случаях система не в состоянии своевременно выполнить все текущие задания. Для жестких СРВ такие ситуации недопустимы. Организация жестких СРВ должна гарантировать отсутствие перегрузок в работе системы [2].

1. Событийные и синхронные СРВ

Большинство современных СРВ обеспечивают оперативное переключение между программными компонентами в зависимости от наступления некоторых внешних или внутренних событий. Такие системы называются событийными (*event-triggered*) СРВ. Основным достоинством событийных СРВ является высокая гибкость в реагировании на изменения условий функционирования системы. Однако, событийные СРВ не обеспечивают высокой степени предсказуемости поведения системы и ее надежности, т.к. на этапе ее отладки и тестирования практически невозможно смоделировать все возможные комбинации событий. Под предсказуемым поведением СРВ понимается такое ее поведение, когда возможность нарушения временных ограничений может быть выявлена и устранена на этапе ее создания, либо оперативно устранена за счет введения в состав приложения таких задач, которые обеспечивали бы адекватную реакцию на угрозу сбоев в работе системы. В системах с предсказуемым поведением характер результатов и длительность действий могут быть предсказаны до того, как эти действия начнут исполняться.

Помимо событийных СРВ выделяют класс систем, в которых программные компоненты активизируются не по наступлении тех или иных событий, а в рассчитанные заранее моменты времени. Такие системы обеспечивают высокую предсказуемость поведения, надежность и помехозащищенность за счет реализации эффективных механизмов заблаговременного выявления возможных сбоев и отказов, благодаря жесткой привязке выполняемых действий к временной шкале. Такие СРВ называются синхронными (*time-triggered*). Противопоставление событийных и синхронных систем было введено Германом Копецом, предложившим достигать предсказуемости поведения СРВ за счет заблаговременной (статической) привязки ко времени таких системных операций как запуск процессов выполнения задач и коммуникационных операций [3].

Реализация синхронных систем связана с заблаговременным построением графиков выполнения системных операций. При построении жестких СРВ, отвечающих требованиям высокой надежности функционирования, эти графики формируются статически, в ходе разработки системы на инструментальной машине. Программа, строящая статический график, называется статическим планировщиком. Готовый статический график размещается в памяти целевой машины в качестве системной структуры данных. Интерпретация графика в ходе работы встроенного программного комплекса на целевой машине осуществляется на базе отслеживания хода системного времени. Отслеживание системного времени является одной из важнейших функций синхронной системы. Интерпретация структур, представляющих статический график на целевой машине выполняется модулем, называемым динамическим планировщиком [4].

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Задачи и задания. В дальнейшем изложении используется модель множества задач, близкая к представленной в работе [5].

Программное приложение представляет собой множество задач $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Каждая задача τ_i определяется в коде программного приложения статическим дескриптором, который содержит в первую очередь указатель точки входа в тело программы, представляющей алгоритм выполнения задачи. Кроме того, дескриптор задачи содержит информацию о тех или иных свойствах и параметрах, характеризующих особенности алгоритма выполнения задачи.

В ходе функционирования СРВ каждая из задач $\{\tau_1, \tau_2, \ldots, \tau_n\}$ выполняется многократно. В тот момент времени, когда впервые возникает необходимость выполнения задачи τ_i осуществляется активизация этой задачи: системой порождается задание (job) $\tau_{i,1}$ — динамический объект, обеспечивающий отслеживание хода выполнения соответствующего задаче алгоритма, хранение локальных данных, фиксацию фаз взаимодействия с другими объектами системы. При каждой последующей активизации задачи τ_i порождается новый объект $\tau_{i,j}$ — очередное задание на выполнение той же задачи. Отношение между задачей τ_i (статический объект) и множеством $\{\tau_{i,1}, \tau_{i,2}, \ldots\}$ соответствующих ей заданий аналогично отношению между типом переменной и конкретными переменными этого типа, аналогично отношению между классом объектов и конкретными объектами, являющимися представителями, экземплярами этого класса. Если под выполнением задачи τ_i понимается некая абстракция, объединяющая все возможные варианты последовательностей действий, соответствующих алгоритму задачи τ_i, то под выполнением задания понимается конкретная реализация одного из этих вариантов. Определение задачи фиксирует конфигурацию пространства для выполнения заданий. Выполнению каждого задания соответствует конкретная траектория движения в этом пространстве.

Ниже рассматриваются методы планирования периодически выполняемых заданий в синхронных СРВ. Рассматриваемые методы используются на том этапе разработки СРВ, когда соответствие между выполняемыми задачами и выполняющими их аппаратными подсистемами уже установлено и необходимо составить график выполнения каждым из аппаратных ресурсов возложенных на него заданий. При этом имеются в виду как универсальные или специализированные процессоры, так и коммуникационные каналы. В последнем случае место задачи занимает тип μ_i сообщения, определяющий его формат, назначение, круг его генераторов, адресатов и т.д.; передача конкретного сообщения означает выполнение задания $\mu_{i,j}$ по реализации коммуникационного взаимодействия типа μ_i. В [3] рассматриваются распределенные синхронные СРВ с синхронно функционирующими каналами передачи сообщений, предлагаются методы синхронизации системных часов в различных узлах системы.

Характеристики эффективности реализации системных операций. Активизация задач и пересылка сообщений являются примерами операций выполняемых системными компонентами СРВ, которые обеспечивают реализацию внутренних механизмов функционирования СРВ. Системные компоненты образуют операционную систему или ядро СРВ. Эффективность реализации системных операций характеризуется в первую очередь двумя параметрами — величиной системных издержек и значением реактивности. Величина системных издержек определяется общей продолжительностью выполнения системных операций. Реактивность системы характеризует ее способность быстро

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

реагировать на внешние события. Если все системные операции выполняются с максимальным приоритетом процессора, то реактивность представляется величиной **max**{C_S}. Здесь C_S — продолжительность выполнения системной операции ***S***; снижению значения этой величины соответствует повышение реактивности системы. Таким образом, уменьшение времени выполнения системных операций означает повышение эффективности функционирования системы как в отношении накладных расходов, так и в отношении реактивности.

Для синхронных СРВ характеристика реактивности является важной характеристикой эффективности реализации системных операций ввиду того, что наряду с задачами, активизация которых регламентируется статическим графиком, среди прикладных задач обычно присутствует небольшое число задач, активизируемых внешними событиями.

2. Предпосылки формирования графиков выполнения задач

В случае реализации статического подхода к построению СРВ статические графики активизации задач представляют собой одну из важнейших системных структур. Выбор способов представления статических графиков, методов генерации этих графиков по информации о множестве решаемых задач, эффективных методов интерпретации графиков стоят в ряду важнейших вопросов построения синхронной СРВ. Одна из проблем связана с необходимостью выделения памяти для размещения графика активизации задач. Неудачный выбор формы представления графика может привести к чрезмерно большим объемам памяти для его размещения. Кроме того, как отмечалось выше, существует проблема эффективности интерпретации графика, снижения величины системных издержек и повышения реактивности системы.

Периоды выполнения задач. Как отмечалось выше, при разработке синхронной программной системы для каждого из активных системных ресурсов (каждого процессора, каждого канала передачи сообщений) выполняется (статическое) планирование порядка использования этого ресурса периодически активируемыми задачами. Статическое планирование представляет собой преобразование множества описаний обслуживаемых процессором периодических задач в статический график, то есть, в статическую структуру системного уровня, задающую точные моменты времени активизации этих задач.

Для разработки процедуры статического планирования необходимо определить:

- состав входных данных об обслуживаемых периодических задачах и способ представления этих данных;
- формат получаемого в результате планирования статического графика;
- алгоритм преобразования входных данных в график активизации задач.

Построенный статический график должен быть проверен на выполнимость, то есть, производительность процессора должна обеспечивать своевременное выполнение каждого из запланированных в графике заданий.

Период выполнения ансамбля задач. В синхронной системе каждый процессор (и каждый канал передачи данных) функционирует циклически. Статический график активизации задач определяет содержание отдельного цикла

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

функционирования процессора. По завершении каждого цикла начинается новый цикл — процессор приступает к очередной итерации выполнения статического графика.

В качестве входных данных для составления статического графика используются по крайней мере четыре параметра, характеризующие существенные для планирования свойства периодических задач:

- p_i — период выполнения задачи τ_i,
- c_i — требуемая для выполнения задачи продолжительность работы процессора,
- ϕ_i — фаза активизации задачи (момент времени $t_{\text{nom}, i, j} = \phi_i + jp_i$ устанавливается в качестве номинального значения момента активизации задачи $\tau_{i,j}$),
- D_i — относительное значение предельного срока выполнения для задачи τ_i (момент времени $t_{\text{deadline}, i, j} = t_{\text{nom}, i, j} + D_i$ устанавливается в качестве предельного срока выполнения задания $\tau_{i,j}$).

Во многих синхронных системах для всех задач фаза активизации равна нулю ($t_{\text{nom}, i, j} = jp_i$) и предельный срок выполнения каждого задания определяется периодом задачи ($t_{\text{deadline}, i, j} = t_{\text{nom}, i, j} + p_i$). Последнее равенство означает, что выполнение задания $\tau_{i,j+1}$ может начаться только после завершения предыдущего $\tau_{i,j}$.

Представляет интерес распределение периодов выполнения задач и передачи сообщений в реальных прикладных системах. Авторы [6] приводят значения периодов генерации различных сообщений во встроенной программной системе для прототипа электромобиля. Перечень этих сообщений с ранжированием по частоте их передачи приведен в табл. 1.

Таблица 1. Ранги сообщений во встроенной программной системе для прототипа электромобиля

Ранг сообщений	Идентификаторы сообщений	Период (мс)
1	accelerator_position, brake_pressure, clutch_pressure_control, torque_command, torque_measure, processed_motor_speed	5,0
2	high_contacrtor_control, low_contacrtor_control	10,0
3	brake_switch	20,0
4	Hi&Lo_contactor_open/close, key_switch_run, key_switch start, accelerator_switch, emergency_brake, shift_lever,spead_control, 12V_power_vechile_control, 12V_power_vechile_inverter, brake_mode, SOC_reset, intrlock, DC/DC_convertor_current_control,12V_power_relay, brake_solenoid, backup_alarm, warning_lights,key_switch, main_contactor close, FWD/REV, FWD/REV_ack, idle, inhibit, shift_in_progress, inverter_temperature_status, shutdown, status/malfunction,	50,0

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

	main_contactor_acknowledge	
5	truction_battery_voltage, traction_batary_current, auxiliary_battery_voltage, tran-saxle_lubrication_pressure, vehicle_speed	100,0
6	traction_battery_temp, traction_battery_temp_max, traction_battery_ground_fault, mo-tor_trans_over_temperature, DC/DC_convertor	1000,0

Данные, приведенные в табл. 1, могут в первом приближении рассматриваться и как пример распределения периодов прикладных задач во встроенных программных системах, поскольку приведенные сообщения генерируются или обрабатываются прикладными задачами.

Другой пример (табл. 2, [7]) содержит распределение периодов прикладных задач в некоторой телеметрической системе жесткого реального времени.

Таблица 2. Ранги задач в телеметрической системе жесткого реального времени

Ранг задач	Идентификаторы задач	Период (мс)
1	Clock, Read_Bus_Ip	10,0
2	onemsg_here, Real_Time_Clock	50,0
3	tm_here, Telemetry_Responce	62,5
4	twomsg_here, z1_here, Process_IRES_Data	100,0
5	tc_here, Telecommands	187,0
6	fourmsg_here, Command_Actuators, Normal_Mode, Re-quest_DSS_Data, Request_Whell_Speeds	200,0
7	Calibrate_gyro, Process_DSS_Data	1000,0
8	Time0_Update	3600,0

Представленные в табл. 1 и табл. 2 данные иллюстрируют широту диапазона периодов активизации задач различных типов в синхронных системах реального времени. Однако, как бы ни различались периоды выполнения заданий, можно отметить периодичность в выполнении всего ансамбля задач, составляющих синхронную систему. Общий период выполнения всего ансамбля задач в синхронной системе равен наименьшему общему кратному периодов составляющих ансамбль задач. В случае системы для электромобиля (табл. 1) продолжительность общего периода выполнения ансамбля задач составляет одну секунду и равна продолжительности максимального периода активизации задач. В случае телеметрической системы (табл. 2) продолжительность общего периода выполнения ансамбля задач превышает 13 часов, то есть интервал времени, вмещающий выполнение миллионов отдельных заданий.

3. Ранжирование задач по значениям периода их выполнения

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Анализ практики построения синхронных систем реального времени и, в частности, анализ данных из табл. 1 и табл. 2 приводит к выводу, что задание входных данных для построения графика активизации задач (в частности, задание периодов выполнения задач) осуществляется не только на основе объективных обстоятельств. Во многих случаях объективные предпосылки оставляют какую-то меру неопределенности и в рамках этой неопределенности разработчик приложения выбирает конкретные значения параметров задач с соответствующей мерой произвола.

Двоично-десятичное ранжирование задач. Обратим внимание на то, что в табл. 1 имеется две разновидности значений периодов задач: период части задач представляется числом 10^n, период остальных задач отличается от 10^n ровно вдвое. Отсюда естественно предположить, что при назначении периода задачи разработчик действует в два этапа:

- пытается подобрать подходящее значение периода из ряда 10, 10^2, 10^3, . . .;
- если для какой-то из задач ранжирование по степеням 10 (десятичное ранжирование) представляется слишком грубым, то выбирается значение вдвое большее или вдвое меньшее, чем 10^n.

Для телеметрической системы (табл. 2) значения периодов части задач (ранги 3 и 5) диктуются жестко, поскольку строго привязаны к предопределенным значениям периодов работы внешнего оборудования.

Таким образом, если период задачи не является предопределенным внешними обстоятельствами, то его значение выбирают из множества, определяемого выражением

$$p = 2^n 10^m T_0, \qquad (1)$$

где $n \in \{ -1, 0, 1 \}$, $m \in \{ 1, 2, \ldots \}$, величина T_0 определяет дискретность задания периодов выполнения задач (например, одна миллисекунда). Ранжирование задач, соответствующее (1), уместно называть двоично-десятичным ранжированием.

Двоичное ранжирование задач. Как отмечено выше, выбор длины периодов выполнения задач во многих случаях выполняется разработчиком в определенной степени произвольно. Возникает противоречие: с одной стороны, период выполнения задачи задается с высокой точностью (например, с точностью порядка миллисекунды), но, с другой стороны, объективные обстоятельства не требуют такой высокой точности поддержания задаваемого значения периода. Можно указать два подхода к разрешению этого противоречия. Первый подход состоит в ограничении множества допустимых значений периодов выполнения задач — например, в соответствии с формулой (1). Альтернативный подход состоит в том, что разработчику предлагается наряду с номинальным значением $p_{\mathrm{nom},i}$ периода выполнения задачи τ_i указывать значение допуска δ_i, регламентирующего допустимые отклонения действительного значения периода выполнения задачи τ_i от величины $p_{\mathrm{nom},i}$. В этом случае действительное значение $p_{\mathrm{act},i}$ периода выполнения задачи τ_i может устанавливаться в рамках интервала $(p_{\mathrm{min},i}, p_{\mathrm{max},i}]$:

$$p_{\mathrm{min},i} = (p_{\mathrm{nom},i} - \delta_i) < p_{\mathrm{act},i} \leq (p_{\mathrm{nom},i} + \delta_i) = p_{\mathrm{max},i}.$$

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Одно из преимуществ, получаемых при реализации возможности варьировать значения $p_{act,i}$ состоит в том, что во многих случаях результат такого варьирования приводит к длине общего периода выполнения ансамбля задач существенно меньшей, чем наименьшее общее кратное величин $p_{nom,i}$.

Определение 1. Если для задачи τ_i определен интервал $(p_{min,i}, p_{max,i}]$ допустимых значений периода ее выполнения, то величину P назовем пригодной для τ_i в качестве длины общего периода выполнения ансамбля задач, если существует значение $p_i \in (p_{min,i}, p_{max,i}]$ такое, что P делится нацело на p_i.

Существует алгоритм, позволяющий для множества задач с определенными допустимыми интервалами $(p_{min,i}, p_{max,i}]$ найти такой набор значений $p_{act,i}$, который обеспечивает минимизацию длины общего периода выполнения ансамбля задач. Алгоритм опирается на следующее утверждение.

Утверждение 1. Если P удовлетворяет следующему условию

$$m_i p_{min,i} < P, \qquad \text{где} \qquad m_i = \lceil p_{min,i} / (p_{max,i} - p_{min,i}) \rceil$$

то величина P является пригодной для τ_i в качестве длины общего периода выполнения ансамбля задач.

Очевидно, найдется такое $n_i \geq m_i$ что выполняются неравенства

$$n_i p_{min,i} < P \tag{2}$$
$$P \leq (n_i + 1) p_{min,i}.$$

Заметим, что при выполнении неравенства (2) достаточным условием пригодности P для τ_i в качестве длины общего периода выполнения ансамбля задач является выполнение неравенства $P \leq n_i p_{max,i}$. Следовательно, справедливость утверждения 1 будет очевидна, если удастся показать, что

$$(n_i + 1) p_{min,i} \leq n_i p_{max,i}. \tag{3}$$

Для доказательства справедливости неравенства (3) достаточно убедиться, что следующие преобразования выражения $n_i p_{max,i}$ дают последовательность невозрастающих значений:

$$n_i p_{max,i} = n_i p_{min,i} (1 + (p_{max,i} - p_{min,i}) / p_{min,i}) =$$
$$= p_{min,i} (n_i + n_i (p_{max,i} - p_{min,i}) / p_{min,i}) \geq$$
$$\geq p_{min,i} (n_i + m_i (p_{max,i} - p_{min,i}) / p_{min,i}) =$$
$$= p_{min,i} (n_i + \lceil p_{min,i} / (p_{max,i} - p_{min,i}) \rceil (p_{max,i} - p_{min,i}) / p_{min,i}) \geq$$
$$\geq (n_i + 1) p_{min,i}$$

Таким образом, неравенство (3) выполняется, утверждение 1 доказано. Из этого следует, что значение

$$P = \mathbf{max}\{m_i p_{min,i}\}, \tag{4}$$

которое является подходящим для каждой из задач, может быть определено в качестве общего периода функционирования всей системы.

Следует отметить, что при выборе величины допуска δ_i на значение периода p_i выполнения каждой из задач может возникнуть прежняя проблема дефицита объективных предпосылок, а это означает, что вновь возникает необходимость принятия произвольных решений. Если формулировка предпосылок для выбора значения δ_i затруднительна, то может оказаться предпочтительным выбор значений p_i из предопределенного множества, например, задаваемого выражением (1). Другой подход состоит в использовании предопределенного множества значений для δ_i. Как будет показано ниже, существует предопреде-

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ленное значение δ_i, которое объединяет эти два подхода. Это значение $\delta_i = p_{nom,i}/3$ определяет следующие рамки для $p_{act,i}$:

$$p_{act,i} \in ((1-1/3)p_{nom,i}, (1-1/3)p_{nom,i}]. \quad (5)$$

Утверждение 2. Если для всех задач τ_i с номинальными периодами активизации $p_{nom,i}$ значение допуска δ_i на отклонение действительного периода активизации $p_{act,i}$ от $p_{nom,i}$ составляет одну треть $p_{nom,i}$, то есть, $p_{act,i}$ выбирается из диапазона (5), то значение $P = \mathbf{max}\{p_{min,\ i}\}$ может быть определено в качестве общего периода функционирования всей системы.

Действительно, если $\delta_i = p_{nom,i}/3$, то $m_i = 1$. Подставляя $m_i = 1$ в (4) убеждаемся в справедливости утверждения 2.

Утверждение 3. Если для всех задач τ_i синхронной системы $\delta_i = 1/3p_{nom,i}$ и продолжительность общего периода функционирования всей системы $P \geq \mathbf{max}\{p_{min,\ i}\}$ то для каждой задачи τ_i найдется целое число k_i, удовлетворяющее условию:

$$2^{-k_i} P \in (p_{min,i},\ p_{max,i}].$$

Предположим, что утверждение 3 не верно. Это означало бы, что для некоторой задачи τ_i существует целое число k, для которого

$$2^{-k-1}P \leq p_{min,i} \quad (6)$$

и в то же время

$$p_{max,i} < 2^{-k}P. \quad (7)$$

Однако, поскольку $\delta_i = p_{nom,i}/3$, имеет место равенство

$$2p_{min,i} = p_{max,I} \quad (8)$$

Подставляя (8) в (6) получаем $2^{-k}P \leq p_{max,i}$, что противоречит (7). Следовательно, предположение не верно, утверждение 3 доказано. Ясно также, что существует единственное значение k_i, отвечающее условиям $p_{min,i} < 2^{-k_i} P \leq p_{max,i}$.

Пусть P — длина общего периода выполнения ансамбля задач, $\delta_i = p_{nom,i}/3$, τ_I такая задача, для которой $p_{nom,I} = \mathbf{min}\{ p_{nom,j} \}$, и целое число n отвечает условиям

$$p_{min,I} < 2^{-n}P \leq p_{max,I.}$$

При выполнении этих условий будем называть интервал времени продолжительностью $T_0 = 2^{-n}P$ базовым интервалом для синхронной системы. Тогда для каждой задачи τ_i существует единственное целое число k, отвечающее условиям $p_{min,i} < 2^k T_0 \leq p_{max,i}$, и величина $2^k T_0$ может быть назначена в качестве действительного значения периода активизации задачи τ_i :

$$p_{act,i} = 2^k T_{0\ I}. \quad (9)$$

В итоге все множество задач синхронной системы подразделяется на ранги: $k^{ый}$ ранг содержит все те и только те задачи, для которых период активизации равен $2^k T_0$. Таким образом, в случае $\delta_i = p_{nom,i}/3$ отмеченные два подхода (предопределенное множество значений для периодов p_i или для допусков δ_i) объединяются. В самом деле, для любого набора значений $p_{nom,I}$ действительные значения периодов задач выбираются из множества (9). Назовем множество задач синхронной системы реального времени ранжированным по степеням двойки (двоично ранжированным), если периоды задач выбираются в согласно (9).

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Двоичное ранжирование задач часто используется при разработке синхронных систем реального времени безотносительно к понятию допуска на отклонение действительной длины периода от номинального значения. Разработчики считают ранжирование длины периодов по степеням двойки естественным и удобным приемом. Пример такого ранжирования приведен в табл. 3.

Таблица 3. Пример ансамбля задач с двоичным ранжированием периодов выполнения

Ранг задач	Идентификаторы задач	Период (мс)
0	`mthgt_kernel`	10
1	`Auto200`	20
2	`Auto100`	40
5	`Navigation10H, Gauge_Panel, Guidence, Auto10, Captain, GLTPC,`	320
7	`debug_device`	1280
8	`Earth_Model, Burst_Point`	2560

Этот пример взят из описания космической навигационной системы [8].

4. Формы задания графика системных действий

Статический график представляет собой структуру данных, которая задает моменты времени выполнения таких системных действий как активизация, приостановка, принудительное завершение периодических задач, прием или передача периодических сообщений. Статический график должен в явной или неявной форме представлять все события такого рода в интервале, охватывающем общий период функционирования системы. При конструировании синхронной системы необходимо обеспечить приведение системы в некоторое базовой состояние к началу каждого периода ее функционирования. Как правило, в качестве базового определяется такое состояние, в котором завершено выполнение всех активированных ранее задач, и все каналы передачи сообщений свободны [3]. В завершение составления статического графика необходимо убедиться в его выполнимости.

Явная табличная форма графика системных действий. Явная табличная форма представления статического графика формируется статически (т.е. на этапе разработки синхронной системы) по данным, извлекаемым из статических дескрипторов периодических задач. Табличная форма статического графика содержит по отдельному элементу на каждое системное действие в рамках одного периода функционирования системы. Такая форма эффективно интерпретируется обработчиком `TimeInterrupt` прерываний от системного таймера. При каждом выполнении обработчика `TimeInterrupt` системный таймер должен настраиваться на срабатывание в момент, соответствующий следующему элементу *`NextJob`* табличной формы. Продолжительность $c_{\text{TimeIntrpt},Nextjob}$ выполнения обработчика `TimeInterrupt` соответствует определяется выражением

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

$$c_{\texttt{TimeIntrpt},\,Nextjob} = c_0 + c_4 + (c_1 + c_3)k + \sum_{i=\text{Nextjob}}^{\text{Nextjob}+k} c_{2,I},$$

где c_0 — время, требуемое для активизации обработчика прерываний, c_1 — время для формирования динамического дескриптора вновь порожденного задания, c_2 — время для включения порожденного задания в структуры динамического планировщика, c_3 — время для подготовки к интерпретации следующего элемента табличной формы, c_4 — время для заказа очередного прерывания от системного таймера, k — число порождаемых заданий. Значение c_2 может зависеть от текущего состава выполняемых заданий. Так, в системах, использующих список готовности (ready list [9]), продолжительность $c_{2,i}$ операции включения очередного задания в этот список составляет $c_{2,i} = c_{2a} + c_{2b}m_i$, где c_{2a} и c_{2b} — константы, m_I — число более приоритетных заданий уже включенных в список готовности. В таком случае максимальная продолжительность выполнения обработчика прерываний от системного таймера может составить

$$\max(c_{\texttt{TimeIntrpt},\,Nextjob}) = c_0 + c_4 + (c_1 + c_3 + c_{2a})N + c_{2b}N(N-1)/2,$$

где N - общее количество задач в системе.

В ряду преимуществ представления графика системных действий в явной табличной форме следует отметить простоту и наглядность. Кроме того, эта форма составляется статически, т.е. в ходе работы системы не требуется тратить время на ее формирование. Табличная форма графика активизации задач применяется, например, в операционной системе *KURT* [10]. Однако, использование этой формы может оказаться проблематичным или даже невозможным в случае, если общий период выполнения ансамбля задач вмещает слишком большое количество системных действий. В частности, явная табличная форма не пригодна в случае системы задач, представленной в табл. 2, для которой, как отмечалось выше, в рамках одного периода работы системы выполняются миллионы системных действий. В таких случаях может оказаться целесообразным использование динамически формируемых графиков.

Динамическое формирование графика системных действий. Один из эффективных по затратам памяти подходов к представлению графиков системных действий связан с использованием дельта-списков [9]. Дельта-список представляет собой цепной список, строящийся из элементов следующего вида:

```
struct JobArrive {     // Элемент графика порождения заданий
     Task *TaskID;     // Дескриптор класса системных действий (задачи)
     Time TaskPeriod; // Период класса системных действий (задачи)
     Time Delta;       // Относительный срок порождения следующего зада-
ния
     JobArrive *Next; // Следующий элемент цепного списка
};
```

Величина атрибута `Delta` для $i^{\text{го}}$ элемента дельта-списка указывает длину интервала времени между системными действиями, соответствующими $i^{\text{-му}}$ и $(i+1)^{\text{-му}}$ элементам. В случае использования дельта-списка перечень операций, составляющих алгоритм обработчика `TimeInterrupt` прерываний от системного таймера пополняется действиями по модификации динамического графика:

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

а) формирование нового элемента дельта-списка для активизации по прошествии интервала времени `TaskPeriod` еще одного экземпляра системных действий данного класса (например, активизации периодической задачи),

б) поиск места в дельта-списке для включения в него вновь сформированного элемента, с попутным вычислением требуемого значения атрибута `Delta`,

в) включение в дельта-список вновь сформированного элемента,

г) удаление головного элемента дельта-списка.

Использование дельта-списка позволяет существенно сократить объем памяти, отводимой под график системных действий. Так, в случае ансамбля задач из табл. 2 вместо миллионов элементов табличной формы представления графика системных действий оказывается достаточным использовать 21 элемент дельта-списка. Правда, за это приходится платить дополнением алгоритма обработчика прерываний от системного таймера действиями а) – г) с соответствующим увеличением продолжительности его выполнения и, следовательно, возможным снижением реактивности системы.

Оба рассмотренные подхода к построению графика системных действий (статически генерируемая явная табличная форма и динамически модифицируемый дельта-список) универсальны в том смысле, что их можно использовать в системах с произвольными комбинациями периодов их выполнения. Если же значения периодов выбираются из каким-либо образом регулярно организованного множества (например, в случае задач с двоичным ранжированием), то возможно использование неявных форм представления графиков системных действий, которые более эффективны в отношении использования памяти, чем явная табличная форма, но не приводят к снижению реактивности, характерному для подхода с использованием дельта-списка. Ниже предлагаются эффективные способы неявного задания графиков выполнения заданий для подобных частных случаев. Изложение ведется на примере периодической активизации задач, однако, все предлагаемые способы могут быть применены не только для графиков активизации задач, но и для представления графиков любых других периодически выполняемых системных действий, если значения периодов их выполнения отвечают определенным ограничениям.

Системы с двоичным ранжированием задач. Если значения периодов всех задач синхронной системы выбираются из множества, задаваемого формулой (9), то системный таймер настраивается на генерацию прерываний с периодом T_0 и в качестве объекта, неявно задающего график активизации задач, может быть использован линейный массив `RankSet` описаний рангов задач. Роль элементов массива `RankSet` играют экземпляры структур следующего вида:

```
struct TaskRank {      // Описатель двоичного ранга задачи
     Task *TaskID;     // Указатель дескриптора задачи
     int Rank;         // Код, определяющий ранг задачи
};
```

Элементы массива `RankSet` должны быть упорядочены по возрастанию длины периодов активизации задач. Приведенный ниже алгоритм обработчика прерываний от системного таймера соответствует следующей кодировке рангов за-

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

дач: если период p_i задачи τ_i равен $2^k T_0$, то ранг задачи τ_i кодируется числом $(2^k - 1)$. В таблице 4 приведены двоичные коды рангов для задач с периодами T_0, $2T_0$, 2^2T_0, 2^3T_0.

Таблица 4. Коды рангов для двоичного ранжирования задач

Длина периода	T_0	$2T_0$	$4T_0$	$8T_0$	...
Код ранга	00000000	00000001	00000011	00000111	...

При такой кодировке рангов задач приведенная ниже процедура *A* обеспечивает эффективную реализацию выбора задач, подлежащих активизации на каждом такте системного таймера. Глобальная переменная `Tact` инициализируется нулевым значением и увеличивается на единицу при каждом прерывании от системного таймера.

Процедура A. Реализация графика активизации задач при двоичном ранжировании.

Шаг 1. Организация цикла обработки массива описаний рангов задач; инициализация переменной цикла `i = 0` , переход к шагу *2*.

Шаг 2. Если результат поразрядной конъюнкции `Tact & RankSet[i].Rank` не равен нулю, то выполнение процедуры завершается, иначе активизируется задача `RankSet[i].TaskId` , переход к шагу *3*.

Шаг 3. `i = i + 1`, если массив RankSet исчерпан, то выполнение процедуры завершается, иначе — переход к шагу *2*.

Число элементов массива `RankSet` равно числу периодических задач, а не числу заданий, порождаемых на протяжении общего цикла функционирования системы. Вместе с тем, выполнение процедуры *A* не приводит к увеличению продолжительности работы обработчика прерываний от системного таймера в сравнении с интерпретацией явной табличной формы графика системных действий. Следовательно, в этом случае экономия памяти достигается без увеличения накладных расходов или снижения реактивности системы.

Системы с гармоническими периодами задач. Изложенный подход к реализации системы с двоичным ранжированием задач может быть обобщен на более широкое множество систем, например, на системы с гармоническими периодами задач.

Определение 2. Множество задач $\{\tau\}$ составляет систему задач с гармоническими периодами, если для любых двух задач τ_i и τ_k из этого множества (пусть для определенности $p_i \geq p_k$) p_i делится на p_k без остатка.

Задачи, составляющие такое множество, упорядочиваются по возрастанию периодов:

$$p_1 \leq p_2 \leq p_3 \leq \ldots \leq p_n$$

Заметим, что отношение p_i / p_1 является целым для любого p_I, а p_1 играет роль базового интервала T_0, определяющего частоту прерываний от системного таймера.

Определение 3. Отношение p_i / p_1 называется абсолютным показателем ранга для задачи τ_i; отношение p_i / p_{i-1} называется относительным показателем ранга для задачи τ_i; относительный показатель ранга для задачи τ_1 равен единице.

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Для неявного представления графика активизации задач с гармоническими периодами, может быть использован линейный массив `HarmRankSet` с элементами `HarmTankRank`:

```
struct HarmTaskRank{ // Описатель ранга для задач с гармонич. периодами
  Task  *TaskID;    // Указатель дескриптора задачи
  Int   RelRank;    // Относительный ранг задачи
  int   CurPhase;   // Счетчик головного элемента в сегменте массива
};
```

Элементы массива расположены в порядке возрастания показателей абсолютного ранга задач. Таким образом массив разделяется на сегменты, каждый сегмент содержит элементы, соответствующие задачам с совпадающими показателями абсолютного ранга.

Структуры `TankRank` и `HarmTaskRank` различаются тем, что последняя имеет дополнительный атрибут `CurPhase`. Перед началом работы системы значения атрибутов `CurPhase` во всех элементах массива `HarmTankRank` обнуляются. В ходе выполнения процедуры *B*, вызываемой при возникновении прерываний от системного таймера, атрибут `CurPhase` модифицируется как минимум в одном из элементов массива `HarmTankRank`.

Процедура B. Реализация графика активизации задач с гармоническими периодами.

Шаг 1. Инициализация переменной цикла $i = 0$, переход к шагу *2*.

Шаг 2. Если значение относительного ранга `RelRank` в текущем элементе массива `HarmRankSet` равно единице, то выполняется активизация задачи `HarmRankSet[i].TaskId` и переход к шагу *4*, иначе — переход к шагу *3.*

Шаг 3. Значение атрибута `CurPhase` в текущем элементе массива `HarmRankSet` увеличивается на единицу; если оно при этом становится равным значению относительного ранга `RelRank`, то `CurPhase` обнуляется, выполняется активизация задачи `HarmRankSet[i].TaskId` и переход к шагу *4*, иначе выполнение процедуры завершается.

Шаг 4. $i = i + 1$, если массив HarmRankSet исчерпан, то выполнение процедуры завершается, иначе — переход к шагу *2*.

Таким образом реализация системы задач с гармоническими периодами обеспечивает почти те же показатели эффективности по затратам памяти, накладным расходам и реактивности, что и в случае системы с двоичным ранжированием задач.

Системы с двоично-десятичным ранжированием задач. Рассмотрение условия (1) показывает, что системы с двоично-десятичным ранжированием задач выходят за рамки систем с гармоническими периодами задач. Но рассмотренный выше метод представления и выполнения графиков активизации задач может быть адаптирован и для этого класса синхронных СРВ. Это достигается за счет использования следующей структуры описателя ранга задачи:

```
struct BDTaskRank{   // Описатель двоично-десятичного ранга задачи
  Task  *TaskID;    // Указатель дескриптора задачи
  int   MaxPhase;   // Граничное значение для счетчика CurPhase
  int   CurPhase;   // Счетчик
  int   SkipFlg;    // Флаг пропуска сегмента массива
```


Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

```
};
```

Элементы этого типа опять же сводятся в массив `BDRankSet` в порядке возрастания значений периода p_i соответствующей задачи. Элементы с одинаковыми значениями периода образуют сегмент массива. Если p_{k+1} — значение периодов задач $(k+1)^{ro}$ сегмента делится нацело на p_k (значение периодов задач k^{ro} сегмента), то флаг `SkipFlg` головного элемента k^{ro} сегмента устанавливается в ноль, в поле `MaxPhase` головного элемента $(k+1)^{ro}$ сегмента устанавливается результат деления p_{k+1} на p_k. Если p_{k+1} не делится нацело на p_k, то флаг `SkipFlg` головного элемента k^{ro} сегмента устанавливается в единицу, в поле `MaxPhase` головного элемента $(k+1)^{ro}$ сегмента устанавливается результат деления p_{k+1} на p_{k-1}. Для каждого неголовного элемента массива в поле `MaxPhase` устанавливается единица.

Процедура *C* активизации задач с двоично-десятичным ранжированием по месту в системе, структуре и функциям аналогична процедуре *B*.

Процедура C. Реализация графика активизации задач с двоично-десятичным ранжированием.

Шаг 1. Инициализация переменной цикла `i = 0`, переход к шагу *2*.

Шаг 2. Если значение `MaxPhase` в текущем элементе массива `BDRankSet` равно единице, то выполняется активизация задачи `BDRankSet[i].TaskId` и переход к шагу *5*, иначе — переход к шагу *3*.

Шаг 3. Значение атрибута `CurPhase` в текущем элементе массива `BDRankSet` увеличивается на единицу; если оно при этом становится равным значению `RelRank`, то `CurPhase` обнуляется, выполняется активизация задачи `HarmRankSet[i].TaskId` и переход к шагу *4*, иначе — переход к шагу *4*.

Шаг 4. Если значение `SkipFlg` в текущем элементе массива `BDRankSet` равно нулю, то выполнение процедуры завершается, иначе переменная цикла `i` увеличивается настолько, чтобы `BDRankSet[i]` стал последним элементом текущего сегмента, переход к шагу *5*.

Шаг 5. `i = i + 1`, если массив HarmRankSet исчерпан, то выполнение процедуры завершается, иначе — переход к шагу *2*.

Таким образом предложенный метод представления и выполнения графиков активизации задач обобщается на случай систем с двоично-десятичным ранжированием задач без существенного ухудшения показателей эффективности по затратам памяти, накладным расходам и реактивности.

Заключение

Универсальные методы формирования и выполнения графиков порождения заданий в синхронных СРВ обеспечивают либо эффективное выполнение системных действий (явная табличная форма представления графика), либо экономное использование памяти (динамические дельта-списки). Предложенный в настоящей статье подход к построению и выполнению статических графиков в синхронных СРВ с двоичным ранжированием задач, а также и обобщения этого подхода эффективны как по памяти, так и по быстродействию. Предложенный подход применим и для систем с множественной активизацией задач ($D_i > p_i$). Актуальна доработка предложенного подхода для его применения в системах с произвольной фазой порождения заданий ($\phi_i > 0$): как показано в [4], такие

Труды СПИИРАН. 2002. Вып. 1. Т. 1. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2002. Issue 1. V. 1. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

системы (например, системы с ритмическими графиками активизации задач) могут обеспечить повышение реактивности синхронной СРВ.

Литература

[1] *Stankovic J. A.* Real-Time Computing. — Amherst, MA: University of Massachusetts, 1992. — 19 p.

[2] *Никифоров В. В., Павлов В. А.* Операционные системы реального времени для встроенных программных приложений // Программные продукты и системы — 1999, №4. — с.24–30.

[3] *Kopetz H.* Real-Time Systems. Design principles for Distributed Embedded Applications. — Boston: Kluwer Academic Publisher, 1997. — 319 p.

[4] *Никифоров В. В., Перевозчиков М. В.* Ранжирование периодов задач в системах реального времени // Программные продукты и системы — 1999, №4. — с. 16–20.

[5] *Manabe Y., Aoyagi S.* A Feasibility Decision Algorithm for Rate Monotonic and Deadline Monotonic Scheduling // Real-Time Systems — March 1998, v.14, №.2. — pp. 171–182.

[6] *Tindell K., Burns A.* Guaranteed Message Latencies on Control Area Network (CAN). Proceedings 1st International CAN Conference. — Mainz, Germany, September 1994.

[7] *Bailey C.M., Burns A., Wellings A.J., Forsyth C.H.* A Performance Analysis of a Hard Real-time System. — York: University of York, Department of Computer Science, report YCS224,1994. — 29 p

[8] McConnell D., Lewis B., Cray L. Reengineering a Single Thread Embedded Missile Application onto a Parallel Processing Platform Using MetaH // Real-Time Systems — Jan 1998, v. 14, №1. — pp. 7–20

[9] *Comer D*. Operating System Design. The XINU Approach. — NJ: Prentice Hall, Inc., 1984. — 320 p.

[10] *Hill R., Srinivasan B., Pather S., Niehaus D.* Temporal Resolution and Real-Time Extensions to Linux. — Kansas: University of Kansas, 1998. — 32 p.