КОМПЬЮТЕРНАЯ МАТЕМАТИКА

Инструментальные средства информационных технологий

УДК 629.735.05

С.В. ЗИНЧЕНКО, В.П. ЗИНЧЕНКО, Ф.Н. ГОРИН

ПРОГРАММНАЯ ПОДДЕРЖКА И ПЛАНИРОВАНИЕ ЗАДАЧ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Рассмотрены особенности систем реального времени, функционирования систем реального времени, операционных систем реального времени и разработки приложений для них; стандарты реального времени, характеристики и анализ особенностей реального времени; проблемы расширений в системах реального времени для Linux и Windows; планирования задач в системах реального времени, условия гарантированного завершения задач, параметры задач; алгоритмы статического и динамического планирования; критерии/теоремы планируемости; примеры.

Введение. Проблемы систем реального времени (СРВ) затрагивают фундаментальные и прикладные вопросы их развития как важное и актуальное направление современных цифровых технологий [1, 2].

Главная задача СРВ – это своевременная реакция на события на объекте, которые регистрируются датчиками, передаются в модули ввода-вывода и генерируют запросы на прерывание. Время запуска прикладной программы (ПП) обработки этого события зависит от аппаратуры, свойств операционной системы (ОС) и архитектуры микропроцессора (МП) [3]. Известно, что время реакции СРВ на события зависит от области применения и варьируется от $1.0 \cdot 10^{-6}$ до $1.0 \cdot 10^{4}$ с. Общепринятого метода определения данного параметра не существует, но его можно предсказать/вычислить.

Проектирование СРВ состоит в выборе/создании средств для работы в условиях жесткого реального времени (РВ), что гарантируют завершения выполнения каждой критической задачи для всех возможных сценариев ее работы.

Цель работы – анализ особенностей выполнения задач и функций СРВ, особенности систем реального времени (ОСРВ) и разработки ПП для СРВ; стандарты СРВ; характеристики и анализ ОСРВ; проблемы создания расширений РВ для Linux и Win-dows; планирование задач в СРВ; выполнение ПП; временные характеристики и параметры задач; проблемы построения расписаний.

ОСРВ и операционные системы общего назначения (ОСОН) ориентированы на оптимальное распределение ресурсов компьютера между пользователями и задачами.

В ОСРВ – это второстепенная задача, так как главная задача – успеть среагировать на события, происходящие на объекте. ОСРВ всегда связано с аппаратурой, объектом, событиями на объекте и ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям ОСРВ от ОСОН в структуре, функциях ядра, построении ввода-вывода. Часто внешний интерфейс ОСРВ похож на ОСОН, но ОСРВ устроена совершенно иначе, и ее применение всегда конкретно. ОСОН – это готовый набор приложений, а ОСРВ – инструмент для создания конкретной СРВ. Характерные функции ОСРВ: многозадачность с вытеснением (preemptable); организация приоритетов для потоков; поддержка предсказуемых механизмов синхронизации; обеспечение механизмов наследования приоритетов; поведение известно и предсказуемо (задержки прерываний, переключения задач, драйверов и т. п.); известно максимальное время отклика. ОСРВ – это ОС которая: имеет средства для организации работы в условиях жесткого РВ; фиксирует сбой в работе, при нарушении временных ограничений; обеспечивает требуемый сервис в заданный промежуток времени [4]; гарантирует как логическую правильность работы ПП, так и время получения результата [2, 3].

Обработка событий в СРВ. Время реакции СРВ на события зависит от области применения: математическое моделирование – *мкс*; радиолокация – *мс*; складской учет – *с*; торговые операции – *мин*; управление производством – *мин*; химические реакции – *ч*. Следует отметить, что каждая система имеет свои аппаратные средства; события сообщаются посредством запросов на прерывание (IRQ). Ключевой параметр – это время реакции СРВ на прерывание (interrupt latency).

События на объекте регистрируются датчиками, данные с которых передаются в модули ввода-вывода/интерфейсы СРВ. Последние, получив и преобразовав информацию, генерируют запрос IRQ в управляющем компьютере, что и является сигналом о произошедшем событии на объекте. Получив такой сигнал, СРВ должна запустить ПП обработки данного события.

Интервал времени на объекте от события до выполнения первой инструкции в ПП обработки этого события – это время реакции СРВ на событие.

Время выполнения действий на объекте от события до генерации IRQ не зависит от ОСРВ, а в целом определяется аппаратурой, интервал времени от возникновения запроса IRQ до выполнения первой инструкции обработчика определяется свойствами ОС и архитектурой МП. Это время всегда оценивается для СРВ в предположении, что МП загружен, возможны другие прерывания и действия, блокирующие IRQ.

Оценка времен реакции на прерывание ОСРВ (pSOS + ≤4; VRTX ≤5; LunxOS ≤ 7; VxWorks, PDOS ≤4.5 в *мкс*) выполняется на основе данных тестирования целевой СРВ известной архитектуры с известными средствами измерения и точными значениями измеряемых промежутков времени. Оценка времени обработки параллельных событий в ОСРВ зависит от времени переключения контекста (pSOS + ≈ 122; VRTX ≈ 145; LunxOS ≈ 165; VxWorks ≈ 152; PDOS ≈ 90 в *мкс*).

Отметим, что ОСРВ СРВ выполняется на персональных, промышленных компьютерах (ПрК), контроллерах и встраиваемых системах. При выборе МП, определяющим является обеспечение требуемой производительности при наименьшей тактовой частоте. ПрК обычно не имеют монитора и клавиатуры, а ОСРВ должна работать в бездисковом исполнении; поддерживать как можно больше МП и видов оборудования; иметь инструментарий для создания и отладки ПП [5].

Стандарты СРВ. Различия в спецификациях ОСРВ и разнообразие МП привели к стандартизации для упрощения переноса ПП из одной системы в другую. Однако, задача обеспечения максимальной скорости работы и компактности СРВ стоит выше задачи стандартизации. Поэтому, с одной стороны, среди СРВ преобладают системы с уникальным устройством, а с другой – многие стандарты носят весьма общий характер. Важность стандартов состоит в том, что они являются аксиоматической базой, задающей определения рассматриваемых объектов и понятий.

Известный стандарт POSIX (Portable operating system interface for computer environments, IEEE 1003.1) изначально разработан для Unix-систем. Спецификации POSIX (>30 стандартов) определяют механизмы взаимодействия ПП и ОС. Для ОСРВ важны 7 из них (1003.1a-d/1j /21 /2h). Некоторые компании, например, TRON (the RTOS Nucleus, Япония) предложили в качестве стандарта СРВ свои спецификации. Военная и аэрокосмическая отрасли предъявляют жесткие требования к целевым СРВ, что изложено в стандартах DO-178B, ARINC-653 (США) и ED-12B (Европа). Распространен стандарт OSEK/VDX (OSEK), первоначально созданный для систем автомобильной индустрии. Известны и другие стандарты [1].

Стандарт POSIX был расширен особенностями режима РВ, и в настоящее время представляет семейство стандартов IEEE Std 1003.n, а именно: 1003.1a/OS Definition (базовые интерфейсы); 1003.1b /Realtime extensions (расширения РВ с такими алгоритмами планирования: Sched_fifo – до завершения в режиме fifo; Sched_rr /Round robin – процессам выделяются кванты времени; Sched_other – произвольный не переносимый); 003.1c/Threads (многопоточная обработка); 1003.1d (дополнительное расширение РВ; 1003.21 (распределенные СРВ); 1003.2h (сервисы) [4].

Стандарт DO-178A/B/C создан RTCA (Radio technical commission for aeronautics) для бортовых авиационных систем. Предусмотрены такие уровни отказов: A – катастрофический, B – опасный, C – существенный, D – несущественный, E – не влияющий. Каждый уровень имеет набор требований к ПП, гарантирующих работоспособность СРВ при возникновении отказов данного уровня.

Стандарт ARINC-653 (Avionics application software standard interface) определяет универсальный программный интерфейс APEX (Application/ Executive) между ОС авиационного компьютера и ПП. Требования к интерфейсу ПП и сервисами ОС определены так, чтобы разрешить ПП контролировать диспетчеризацию, связь и состояние внутренних обрабатываемых элементов. В новой редакции введена архитектура изолированных (partitioning) виртуальных машин [6].

Стандарт OSEK/VDX (OSEK – автомобильный консорциум и университет в Карлсруэ (Германия), VDX (Vehicle distributed eXecutive) – компании PSA и Renault (Франция). Первоначально OSEK/VDX был стандартом открытой архитектуры ОС и API для автомобильных СРВ, но со временем распространился на другие отрасли промышленности. OSEK/VDX включает три стандарта: ОС/OS; коммуникаций (COM); сетевого менеджера (NM). Определен реализационный язык (OIL). Эффективность OSEK состоит в интеграции компонентов [7].

В стандарте SCEPTRE (Standartisation du coeur des executifs des produits temps reel europeens) изложены спецификации для промышленных ПП; даны определения, методы и подходы, используемые в СРВ; определены понятия: адекватности, безопасности, стоимости, производительности, переносимости, адаптивности, модульности [8]. Сервис ОС разделен на такие группы: коммуникация, синхронизация процессов, контроль и планирование задач, управление памятью, временем, прерываниями и оборудованием ввода-вывода, высокоуровневый интерфейс ввода-вывода и управления периферийными устройствами, управление файлами и транзакциями, обработка ошибок и исключений. Основные группы функций – это выделение памяти и адресация объектов, создание и удаление объектов, доступ к другим компьютерам [9].

Задачи разделены на два класса: прямые/непосредственные (обеспечивают интерфейс между ПП и внешним окружением, например, ввод-вывод, обработка событий таймером; активизируются сигналом, приходящим с управляемого объекта, посредством IRQ); косвенные/отложенные (активизируются другими задачами; оперируют данными, полученными от прямых/других косвенных задач). Определены 10 возможных состояний задач и такие виды межзадачного взаимодействия: обмен сигналами/событиями; коммуникации (очереди); исключения; семафоры; клиент-сервер.

TCSEC (Trusted computer system evaluation criteria) – стандарт критериев оценки компьютерных систем Министерства обороны США, известен как «Orange book», где определены такие уровни защиты: А1 – верифицированная разработка; В3 – домены безопасности; В2 – структурированная защита; В1 – мандатный контроль доступа; С2 – дискреционный контроль доступа; С1 – избирательная защита; D – минимальная защита. На основании предложенных рядом стран аналогичных критериев создан международный стандарт «Общие критерии оценки безопасности информационных технологий» (Common criteria for IT security evaluation, ISO/IEC 15408).

«Common criteria …» определяют требования обеспечения безопасности в виде 7 оценочных уровней (Evaluation assurance levels – Eal): Eal7 (наивысший) – формальная оценка верификации модели объекта; Eal6 – полуформально верифицированный и протестированный; Eal5 – полуформально спроектированный и протестированный; Eal4 – методически спроектированный, протестированный и пересмотренный; Eal3 – методически протестированный и проверенный (более полный чем Eal2); Eal2 – структурно протестированный; Eal1 – функционально протестированный.

В соответствии с «Common Criteria …», например, ОС оцениваются на соответствие функциональным критериям и критериям доверия / профилям защиты. Введены определения профилей защиты в отношении ОС, брандмауэров, смарт-карт и т. п., которые соответствуют определенным требованиям безопасности. В соответствии с оценочными уровнями сертификат на соответствие более высокому уровню означает более высокую степень уверенности в том, что система защиты работает правильно и эффективно. Уровни 5 – 7 применяются для тестирования ПП, созданных с применением технологий безопасности.

Большинство оценок безопасности ПП сосредоточены на уровне Eal4 и ниже, что свидетельствует об ограниченном применении формальных методов. С точки зрения программиста, «Common criteria …» можно рассматривать как набор библиотек, с помощью которых пишутся задания по безопасности, типовые профили защиты и т. п.

Анализ свойств ОСРВ. Анализ выполнен для таких свойств ОСРВ:

• соответствует интерфейс POSIX для работы в режиме РВ, т. е.: обладают вытесняемым ядром; имеет вытесняемый динамический алгоритм планирования, основанный на статических приоритетах задач; возможна синхронизация потоков; имеет свой собственный API;

• является модульной и масштабируемой; размещается ядро в постоянно запоминающем устройстве (ПЗУ); возможно добавить средства ввода-вывода, управления файлами, сетевого взаимодействия (например, TCP/IP) и т. п.;

• быстродействие/эффективность достигается за счет: микроядра; низких расходов на работу планировщика, переключение контекстов, операций блокировки/деблокировки мьютексов и т. п.; быстрой реакции на прерывание (несколько *мкс*); оптимизации критических секций в ядре (участки кода, работающих без вытеснения);

• поддерживает двухэтапную обработку IRQ: сначала работает короткая ПП обслуживания прерывания ISR (interrupt service routine), которая ставит в очередь на выполнение более длинную ПП обработки прерывания IHR (interrupt handling routing); первая – выполняет минимальный объем действий, вторая – более емкие действия, например, передачу данных на внешнее устройство. Подобные алгоритмы есть в ядрах ОСОН. Например, в Linux – это upper/bottom half (верхняя/нижняя половина), где «bottom» соответствует понятию IHR, «upper» – ISR. В ОС W's есть понятие отложенного вызова процедур (DPC). ПП, вызываемая с помощью DPC, аналогична IHR;

• обладает/нет гибкими схемами планирования: уровней приоритетов (≥ 32, согласно POSIX); для потоков с одинаковым уровнем приоритета поддерживает алгоритмы планирования fifo и rr (round robin); поддерживает/нет алгоритм EDF (сложнее чем статический планировщик);

• разрешение таймеров и счетчиков (номинальное – *нс*, реальное – *мкс*);

• содержит механизмы подкачки и защиты памяти; задачи и ядро исполняются в едином адресном пространстве;

• соответствует схеме разработки ПП Host-Target /Self-Hosted.

Self-Hosted – это системы разработки для ПрК, в которых программист разрабатывает ПП, работая в ОСРВ. Это означает, что ОСРВ поддерживает файловую систему, средства ввода-вывода, пользовательский интерфейс, компиляторы, отладчик, редакторы и т. п. Достоинство таких систем – простой и наглядный инструментарий создания и запуска ПП, работающих на том же компьютере, что и пользователь. Недостаток в том, что для ПрК в эксплуатации часто не требуется пользовательский интерфейс и появляется возможность запуска компилятора. Следовательно, большинство из вышеописанных возможностей ОСРВ не используются и бесполезно занимают ресурсы ПрК.

Host-Target – это системы, в которых ОС и/или компьютер, на котором разрабатываются ПП (host), и ОС и/или компьютер на котором работает ПП (target), различны. Связь между компьютерами осуществляется с помощью COM-порта, Ethernet, compact PCI и т. п. В качестве host обычно используется компьютер под управлением Unix/W's, в качестве target – ПрК/встраиваемый компьютер под управлением ОСРВ. Есть системы, в которых на одном компьютере работает ОСОН и ОСРВ. Преимущество таких систем – использование всех ресурсов ОСОН (графический интерфейс, файловая система, быстрый МП и большой объем памяти) для создания ПП и уменьшение размеров ОСРВ за счет включения только нужных компонент. Недостаток – относительная сложность программных компонентов: кросс-компилятора, удаленная загрузчика и отладчика и т. д.

Отметим, что, с одной стороны, учитывая рост мощности ПрК можно использовать self-hosted на большинстве из них. С другой – распространение встраиваемых систем расширяет сферу применения host/target систем, где стоимость ПП – определяющий фактор.

Анализ ОСРВ представлен в формат: название, основные характеристики: тип; архитектура; стандарт; свойства как ОСРВ; ОС разработки (host); МП (target); линии связи host-target; минимальный размер; средства синхронизации и взаимодействия; средства разработки. Рассмотрены такие ОСРВ: CHORUS; LynxOS; OS-9; pSOS; RTC; VRTX; VxWorks; QNX и др. [10].

Например, LynxOS (Lynx RTS): self-hosted; микроядро; POSIX 1003; многозадачность – POSIX 1003; многопроцессорность; уровней приоритетов – 255; планирование – fifo, round robin, Quantum; preemptive ядро; нет; Intel 80x86, Motorola 68xxx, SPARC, PowerPC; полной/усеченной, только ядро – 256/ 124/ 33 Кб; POSIX 1003 (семафоры, mutex, condvar); комплект разработчика (компилятор C/C++, отладчик, анализатор); X W's/Motif для Lyux; Total View (многопроцессный отладчик; ядро – 28 Кб, обеспечивает диспетчеризацию прерываний, планирование задач и их синхронизацию; допускает добавления kernel plug-ins, и эту ОСРВ можно использовать как ОСОН типа Unix для разработки ПП; поддерживает механизмы защиты памяти и подкачки.

Специализированные ОСРВ проектируются под конкретную модель МП /задачу. Преимущества такие: высокая производительность, оптимальный состав оборудования, наибольшая компактность. Недостатки – длинное время разработки, высокая стоимость, непереносимость. Примерами таких систем являются ОСРВ компаний Sony, Sagem и системы, разработанные под конкретную задачу, например, управление железными дорогами TGV (Франция).

Проблемы адаптации ОСОН к требованиям РВ. Адаптация Linux к требованиям РВ выполняется по следующим направлениям:

• соответствие POSIX 1003.: 1c – работа с задачами (thread); 1b – расширение РВ: реализует алгоритмы управления памятью и планирования задач; не реализует алгоритмы работы с таймерами, сигналами, семафорами и очередями сообщений;

• поддержка специального оборудования, например, шины VME; моста VMEPCI, выполнения Linux из ПЗУ, повышения разрешения таймера системы;

• реализация механизма preemption в ядре, что является сложным для Linux, так как все Unix системы, надолго запрещают IRQ при входе в ядро, которое практически невытесняемое (nonpreemptive) [11].

Существуют два направления реализации premption для ядра Linux:

– переписать ядро, что связанно с необходимостью выполнить большой объем работы по переписыванию ядра, чему не способствует быстрое его изменение без учета интересов РВ;

– реализовать механизм preemption как микроядро, отвечающего за диспетчеризацию прерываний и задач, где ядро Linux работало бы как задача с низким приоритетом, само ядро незначительно изменяется для предотвращения блокирования им аппаратных прерываний, например, RT-Linux (без W's ≤ 2,7 Мб).

Адаптация W's к задачам РВ означала попытку применения программной технологии для создания их приложений. Огромный набор ПП под W's, мощный программный интерфейс Win32, наличие специалистов – это необходимость получить для СРВ все перечисленные возможности.

Например, W'nt создавалась как сетевая ОС, но в нее были заложены такие элементы РВ: двухуровневая система обработки прерываний (ISR и DPC), классы РВ (процессы с приоритетами 16 – 32 планируются в соответствии с правилами РВ). Однако W'nt не пригодна для построения жестких СРВ (непредсказуема – время исполнения системных вызовов и реакции на IRQ зависит от ее загрузки; объемна; нет защиты от зависаний, др.), и даже в мягких СРВ ее можно использовать только при выполнении ряда ограничений.

Разработка расширений РВ имеет два направления.

1. Использование ядер классических ОСРВ как дополнения к ядру W's для реализации функций РВ. Такое решение предложили фирмы «LP Eleknroniks» и «Radisys», где параллельно с W'nt работает ОС VxWorks и InTime соответственно. Идея использования двух ОС на одном компьютере состоит в том, что работу с объектом выполняет ПП РВ, передавая затем результаты W's для обработки, передачи в компьютерную сеть, архивирования и т. п.

2. Исключение задержек и зависаний в W's с помощью расширений РВ. Например, расширение фирмы «VenturCom» RTX модифицирует уровень аппаратных абстракций W'nt (HAL – hardware abstraction layer, программный слой взаимодействия драйвера с аппаратурой). Измененный HAL и дополнительные функции (RTAPI) обеспечивают стабильность и надежность системы, путем отслеживания вариантов W's, зависаний ПП и блокировки IRQ. RTX позволяет, используя функции RTAPI, создавать ПП управления устройствами, портами ввода-вывода и физической памятью. Можно создавать встроенные конфигурации W's (без дисков, клавиатуры и монитора).

Планирование задач. СРВ жесткого РВ должна гарантировать завершение выполнения каждой критической задачи для всех возможных сценариев ее работы. Такие гарантии можно обеспечить на основании: исчерпывающего тестирования всех возможных сценариев поведения управляемого объекта и управляющих ПП; построения статического расписания; выбора математически обоснованного динамического алгоритма планирования (АП). Последний должен обеспечивать планирование зависимых задач (включают: критические секции; ограничения на порядок исполнения), которые с практической точки зрения более важны, чем независимые задачи.

Отметим, что в случае использования простых МП не целесообразно организовывать мультипрограммное выполнение большого количества независимых задач на одном компьютере в силу сложности ПП. Одновременно выполняемые зависимые задачи должны обмениваться информацией и получать доступ к общим данным. Поэтому установление некоторой последовательности выполнения задач или взаимного исключения – это актуальная проблема для СРВ [12].

Оптимальное планирование работы зависимых задач сложно, и требует больших вычислительных ресурсов. Решается эта проблема путем разделения задачи планирования на две части так, что одна часть выполнятся заранее, до запуска СРВ, а вторая – во время ее работы. Например, анализ системы задач с взаимными исключениями заключается в выявлении запрещенных областей времени, при которых нельзя назначать выполнение задач, содержащих критические секции; введении ограничений на поведения задач. В таком случае планирование приближается к статическому [13, 14].

Параметры задач. Работа задач характеризуется такими моментами времени (t): r (release time) – возникновение необходимости в передаче управления задаче (момент события); d (absolute deadline/абсолютный крайний срок) – в срок завершения задачей очередной работы; s (start time) – начало исполнения задачи на МП; c (completion time) – окончание работы задачи, обработав событие; и такими временными параметрами: $D = d - r$ (relative deadline) – относительный крайний срок; $e = c - s$ (execution time) – время исполнения задачи при выполнении очередной работы; $R = c - r$ (response time) – время отклика. На рис. 1 показана диаграмма работы задачи с такими параметрами: $r = 2, d = 11, s = 5, c = 9, D = 11 - 2 = 9, e = 9 - 5 = 4, R = 9 - 2 = 7.$

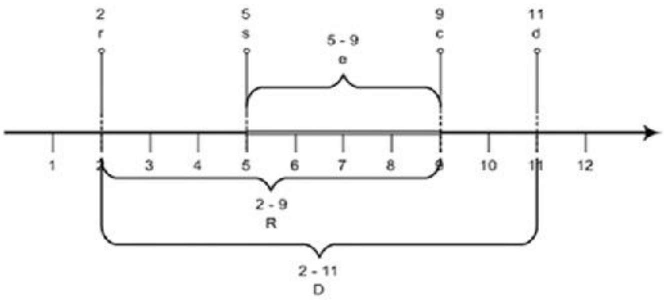

РИС. 1. Параметры T_j

Время перехода задачи в состояние готовности определяется природой управляемого объекта. Крайние сроки (d, D) определяет разработчик СРВ, исходя из свойств управляемого объекта. Время выполнения задач зависит от архитектуры МП, его тактовой частотой и конкретной реализации алгоритма. Для его определения подсчитывается необходимое для выполнения конкретной задачи число тактов МП или выполняется непосредственное измерения значения e. Характер возникновения событий на управляемом объекте определяется типом задач: периодические, спорадические (непериодические с жестким крайним сроком), апериодические (непериодические с мягким крайним сроком).

Пусть $\{T_j\}$ – система задач; $r_{jk} - t$ – необходимость выполнения T_j; $T_j[\varphi_j, p_j, e_j, D_j]$ – периодическая задача, r_{j1} – фаза задачи φ_j; допустимы такие соотношения между p_j и $D_j : p_j = D_j$; $p_j > D_j$; $p_j < D_j$ (одновременно могут $\exists : \{\overline{T}_j\}$ несколько $varT_j$). Тогда, коэффициент использования времени МП периодической T_j будет вычисляться так: $u_j = \dfrac{e_j}{p_j}$, а система периодических $\{T_j\} - U = \sum u_j = \dfrac{\sum e_j}{p_j}$ [15].

Статическое планирование. Обозначим H – расписание, которое является корректным $corH$, если соблюдены $allD_j$. Отметим, что $\{T_j\}$ называется планируемой некоторым АП, если он всегда дает $corH$ для $\{T_j\}$. АП оптимальный, если для $\forall\{T_j\}\exists H$, и оно $corH$.

Рассмотрим систему независимых периодических $\{T_j\}$, где могут находиться апериодические и спорадические задачи как некоторые случаи.

Расписание для периодических задач представляет собой таблицу $H\{t_j = \phi[T_j]\}$, в которой указано, в какой t какую T_j запускать на исполнение. Такое H составляется на промежутке гиперпериода $p = lcm_j\{p_j\}$. Планировщик периодически повторяет последовательность запуска задач согласно $H\{t_j = \phi[T_j]\}$. Например, для такой системы $\{T_j\}_{j=1}^{4} : T_1[*,4,*,1] : T_2[*,5,*,1.8] :$ $: T_a[*,20,*,2]$ $P = lcm\{4,5,20,20\} = 20$. На его протяжении T_1 должна получить управление пять раз, T_2 – 4, а T_a и T_4 – по одному разу. При этом должны быть соблюдены $allD_j$. Статическое H для этой системы приведено в таблице, где l – «задача простоя» (Idle task).

ТАБЛИЦА. Пример статического расписания

№	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
t_j	0	1	2	3.8	4	5	6	8	9.8	10.8	12	13.8	14.8	16	17	18	19.8
$\overline{t_j}$	0.2	1	1	1.8	0.2	1	1	2	1.8	1	1.2	1.8	1	1.2	1	1	1.8
}	*T*1	*T*3	*T*2	*I*	*T*1	*I*	*T*4	*T*2	*T*1	*I*	*T*2	*T*1	*I*	*T*1	*I*	*T*2	*I*

Планировщики, обычно, работают по прерываниям от таймера, а задачи представляют собой ПП, которые вызываются в нужные t обработчиком прерываний. Так как таймер удобно программировать на генерацию IRQ не через какой-то промежуток Δt от начала отсчета времени, а на прерывание через какой-то Δt от текущего t, то t_j в таблице необходимо представлять как промежутки относительного время от одного прерывания к другому, $t_j = t_{j+1} - t_j$.

Преимущества таких алгоритмов: простота, вызванная отсутствием понятий «процесс» /«поток»; передача управления задаче – это вызов ПП; результаты тестирований и проверок весьма надежны. Благодаря этим качествам, такие алгоритмы применяются в СРВ высокой надежности.

Недостатки: негибкость (изменение числа задач, их параметров и т. п. требует остановки СРВ, пересчета $H\{t_j = \phi[T_j]\}$ и перекомпиляции ПП); планировщик «отвязан» от внешнего мира, так как работает по прерываниям от таймера; сложно учитывать спорадические задачи; возможный большой $\dim H\{t_j = \phi[T_j]\}$.

Динамическое планирование с динамическими приоритетами. Рассмотрим два типа планировщиков с динамическими приоритетами:

• EDF (earliest deadline first) – наивысший приоритет имеет $T_i : D_i = \min[D_j]$ для задач с/без вытеснением/я;

• LLF (least laxity first) – наивысший приоритет имеет $T_i : L(t)_i = \min[e(rem)_j]$.

Известно, что (теорема) алгоритм EDF/LLF оптимален для $\forall\{T_j\}$ (периодических с любым соотношением между p_j и D_j, спорадических), если $allT_j$: независимы; возможно вытеснение; вытеснение не требует временных затрат.

Доказательство основывается на том, что если для $\forall\{T_j\}\exists corH,$ которое всегда можно привести к виду $\bar{H}$ при котором $cor\overline{H},$ но $t_j = \phi[T_j]$ будет соответствовать алгоритму EDF / LLF.

Под резервом/запасом времени (laxity) понимается разность между временем, оставшимся до крайнего срока, и временем, которое нужно проработать задаче, т. е. $L(t) = (d - t) - e^{(rem)}.$ На рис. 2 показана работа $T_j,$ которая стала готовой в $t = 2$ и имеет крайний срок в $t = 12.$ Задача работает с вытеснениями. Пока T_j выполняет свою работу $L(t) = \text{const},$ так как уменьшается время до D_j и время, необходимое еще для работы на одну и ту же величину. Если T_j простаивает по причине вытеснения ее более приоритетными $T_j,$ то D_j приближается, а время, нужное для работы, остается постоянным, поэтому в такие промежутки $L(t)$ уменьшаются.

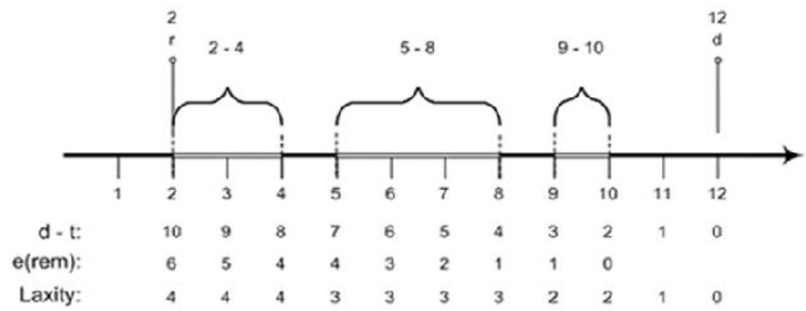

РИС. 2. Резерв времени (laxity)

Алгоритм не оптимален (теорема). Доказать это утверждение можно, приведя такую $[(\{T\})]j\},$ для которой $\exists corH,$ а АП дает $\neg corH.$

Пусть имеем систему из трех задач с такими параметрами: $T_j[r, e, d]:$ T_1[0, 3, 10]; T_2[2, 6, 14]; T_3[4, 4, 12], и таким, например, $corH$ (рис. 3).

Алгоритм даст следующее H (отметим, что моменты принятия решения планировщиком – это $t,$ когда появляется T_j в состоянии готовности, либо какая-то T_j завершает свою очередную работу) (рис. 4).

В $t = 3$ готова только $T_2,$ поэтому она выбирается для исполнения и не вытесняется до тех пор, пока полностью не сделает свою работу несмотря на то, что в $T = 4$ в состояние готовности переходит $T_a,$ у которой крайний срок наступает раньше, чем у $T_2.$ Как видим, из-за этого T_a пропускает свой крайний срок. Таким образом, алгоритм не является оптимальным.

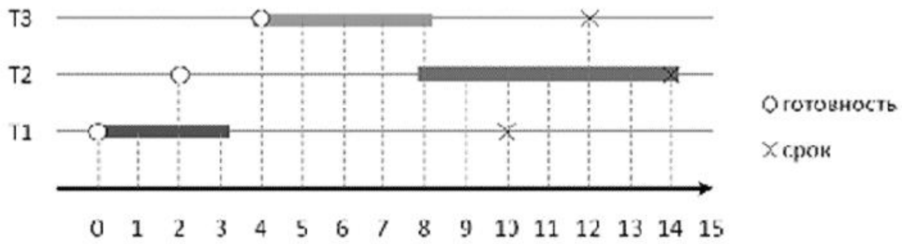

РИС. 3. *corH* для $\{T_j\}_{j=1}^{a}$

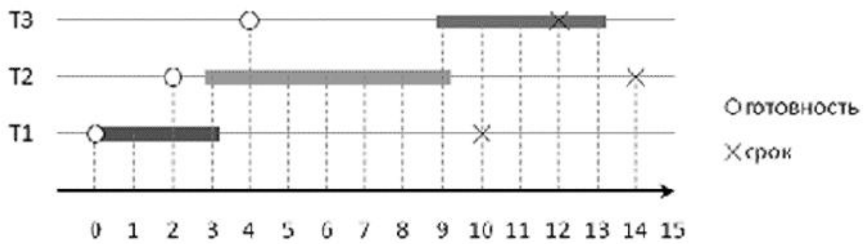

РИС. 4. Для $\{T_j\}_{j=1}^{a}$

Для сравнения приведем H, которое составил бы с возможностью вытеснения $[(\{T)]j\}$. Как видно из рис. 5 $allT_j$ завершили свои работы до D_j.. В $t=2$ стала готовой T_2, но она не вытесняет T_1, так как у T_2 крайний срок наступает позже, чем у T_1. T_2 получает управление только в $t=3$. Но появившаяся в $t=4$. T_a вытесняет T_2, так как у T_a крайний срок наступает раньше, чем у T_2. В $t=8$, когда T_a завершит свою работу, управление вновь получает T_2.

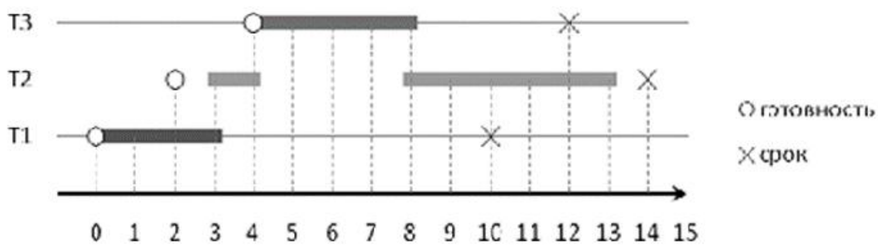

РИС. 5. Для $\{T_j\}_{j=1}^{a}$

Однако следует отметить, что поскольку доказательство проведено в предположении, так как вытеснение задач (например, переключение контекстов) не требует времени. Поэтому при анализе АП нужно учитывать накладные расходы на планирование, например, включать их в e.

Существует теорема для $\{T_j\}$ о возможности составления $H\{t_j = \phi[T_j]\}$. Для того чтобы система $\{T_j\}$ независимых вытесняемых периодических задач (таких, что для $\forall_j : D_j >>= p_j$), была планируема EDF-планировщиком, необходимо и достаточно, чтобы $U \le 1$.

Если же относительные $D_j \le p_j$, то данное условие является необходимым, но не достаточным. Например, система из двух задач $T_j(p,e,D): T_1(2;1;1,9)$; $T_2(2;1;1,9)$ заведомо не может быть корректно распланирована никаким алгоритмом, хотя для нее $U = 1$. В таких случаях вместо «коэффициент использования времени МП» используется понятие «плотность δ задачи», определяемую как: $\delta = \dfrac{e}{\min(D,p)}$. Суммарная плотность $\{T_j\}$ определяется как сумма плотностей $allT_j$ системы: $\Delta = \sum \delta_j = \dfrac{\sum e_j}{\min(p_j, D_j)}$. Для систем такого рода имеет место несколько иное утверждение относительно возможности составления $corH$ (теорема). Для того чтобы $\{T_j\}$ независимых вытесняемых периодических задач (таких, что для $\forall j : D_j < p_j$), была планируема EDF-планировщиком, достаточно, чтобы $\Delta < 1$. Отметим, что это условие не является необходимым. Это значит, что если оно выполняется, то для СРВ можно составить $corH$; если же это условие не выполняется, то $\exists corH$. Например, для $T_1(2;0,6;1)$ и $T_2(5;2,3)$ условие $\Delta \le 1$ (Δ = 0,6/1 + 2,3/5 = 1,06 > 1) не выполняется, но $\exists corH$.

Динамическое планирование со статическими приоритетами. Существуют два способа назначения приоритетов:

- RMS (rate monotonic scheduling) – наивысший приоритет имеет $T_i : D_i = \min[p_j]$ или чем чаще /rate T_i переходит в состояние готовности;
- DMS (deadline monotonic scheduling) – наивысший приоритет имеет $T_i : D_i = \min[D_j]$.

На рис. 6 показан пример H_{RMS} для синхронной системы $\{T_j\}_{j=1}^{2}$ с такими параметрами: $T_1 1(3;0,5); T_2(4;1); T_2(6;2)$. В соответствии с вышеприведенным правилом самый высокий приоритет имеет T_1, а самый низкий – T_2. В $t = 0$ имеется три готовых задачи, МП получает T_1. В $t = 0,5$ готовы две задачи (T_2 и T_3), управление получает T_2. В $t = 3$ снова становится готовой T_1, поэтому T_2 вытесняется. В $t = 3,5$ управление вновь получает T_a как единственно готовая. В $t = 4,0$ готова только T_2, она и получает управление. В $t = 6$ готовы две задачи (T_1 и T_a), управление отдается T_1. Далее работает T_a, но в $t = 8$ она вытеснится T_2. В $t = 9$ опять готова T_1, она отрабатывает и, в итоге, в $t = 9,5$ получает управление T_a. Затем все повторяется сначала при условии, что число задач в системе неизменимое.

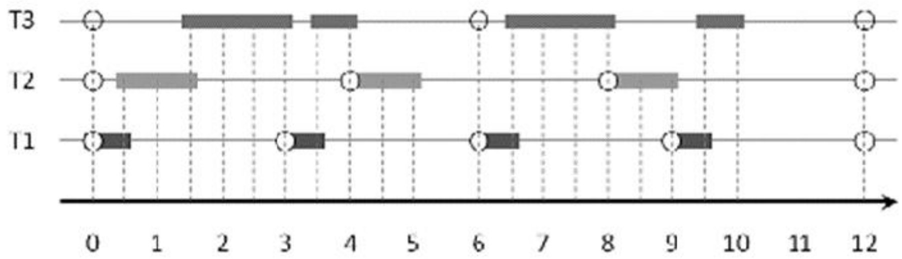

РИС. 6. H_{RMS} для $\{T_j\}_{j=1}^{2}$

Рассмотрим пример H_{DMS} (рис. 7) для следующей системы задач: $T_1(3;0,5); T_2(4;1;2); T_a(6;2)$. Отличие этой системы от предыдущей только в том, что у T_2 относительный крайний срок в два раза меньший периода. Согласно правилу назначения приоритетов, самый высокий приоритет будет иметь T_2, а самый низкий – T_a.

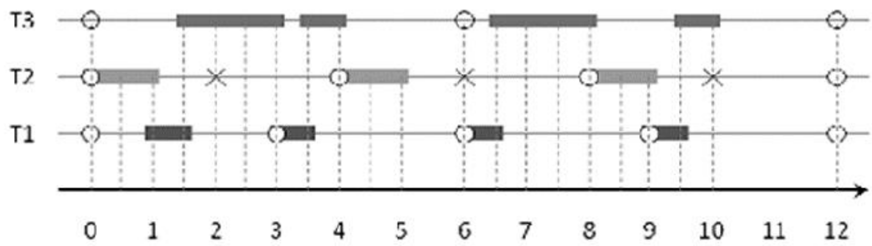

РИС. 7. H_{DMS} для $\{T_j\}_{j=1}^{a}$

Для рассмотренных правил назначения приоритетов существует следующее утверждение (теорема): алгоритмы DMS и RMS не являются оптимальными. Система $\{T_j\}$ периодических задач называется СРВ с простой периодичностью, если для $\forall T_j$ и T_k при $p_i < p_k$ имеет место равенство $p_k \bmod p_i = 0$. Иными словами, периоды $allT_j$ попарно делятся. Например, $\{T_1(2,1), T_2(4,1), T_a(8,11)\}$ – это система с простой периодичностью, а система $\{T_1(2,1), T_2(5,11), T_a(10,1)\}$ таковой не является ().

Для систем $\{T_j\}$ с простой периодичностью справедливо утверждение (теорема). Система $\{T_j\}$ независимых вытесняемых задач с простой периодичностью с $p_j < \mathrm{D}_j$ планируется на одном МП, тогда и только тогда, когда $U < 1$. Для систем с произвольными соотношениями между p_j задач справедливо следующее утверждение (теорема/критерий планируемости алгоритмом RMS).

Система $\{T_j\}_n$ независимых вытесняемых периодических задач с $p_j = D_j$ планируется на одном МП, если $U \le n\left(\frac{21}{n} - 1\right)$. Это условие достаточное, но не необходимое.

Выводы. Рассмотрены особенности СРВ, ОСРВ и разработки ПП для них; стандарты СРВ; характеристики и анализ ОСРВ; проблемы расширений РВ для Linux и Windows; особенности планирования задач в СРВ, условия гарантированного завершения задач, параметры задач; алгоритмы статического и динамического планирования; критерии/теоремы планируемости; примеры.

Результаты исследований использованы при решении задач автоматизации экспериментальных исследований [16, 17] и разработки ПП для таких СРВ.

С.В. Зінченко, В.П. Зінченко, Ф.М. Горін

ПРОГРАМНА ПІДТРИМКА І ПЛАНУВАННЯ ЗАВДАНЬ У СИСТЕМАХ РЕАЛЬНОГО ЧАСУ

Розглянуто особливості систем реального часу (СРЧ), функціонування СРЧ, операціонних систем РЧ (ОСРЧ) і розробки додатків для них; стандарти СРЧ; характеристики й аналіз ОСРЧ; проблеми розширень РЧ для Linux і Windows; особливості планування завдань у СРЧ, умови гарантованого виконання завдань, параметри завдань; алгоритми статичного і динамічного планування; критерії/теореми існування планування завдань; приклади.

S.V. Zinchenko, V.P. Zinchenko, F.N. Gorin

PROGRAM SUPPORT AND TASK PLANNING IN REAL-TIME SYSTEMS

The features of real-time systems (RTS), the operation of the RTS, the operating systems of the RT (RTOS) and the development of applications for them; RTS standards; characteristics and analysis of the RTOS; problems of extensions of the RT for Linux and Windows; planning tasks in the RTS, conditions for guaranteed completion of tasks, task parameters; algorithms for static and dynamic planning; criteria/theorems of planning; examples are considered.

1. *Зинченко С.В., Зинченко В.П.* Особенности систем реального времени. XI Междунар. конф. «Гиротехнологии, навигация, управление движением и конструирование авиационно-космической техники». Сб. докл. в 4-х частях. К.: «КПИ им. Игоря Сикорского», 13 – 14 апреля 2017. Ч. III. С. 48 – 57.
2. *Amitava Gupta, Anil Kumar.* Chandra Real-Time and Distributed Real-Time Systems: Theory and Applications. CRC Press. 2016. P. 167.
3. *Ernst-Rudiger Olderog, Henning Dierks* Real-Time Systems: Formal Specification and Automatic Verification. Cambridge University Press. 2008. P. 344.
4. *POSIX*.1-2008 is simultaneously IEEE Std 1003.1™-2008 and The Open Group Technical Standard Base Specifications, Issue 7. - http://pubs.opengroup.org/onlinepubs/9699919799/
5. *Khan M.A., Saeed S., Darwish A. et al.* Embedded and Real Time System Development: A Software Engineering Perspective, Springer. 2014. P. 305.
6. *Sławomir Samolej*. ARINC Specification 653 Based Real-Time Software Engineering. http:// www.scarlettproject.eu/publications/technical/ABS653RTSE_SS1-2011.pdf

7. *Joshua* Billert Automotive Operating Systems: OSEK/VDX. https://es.cs.uni-kl.de /publications /data/ Bill14.pdf
8. *SYSTEMES* TEMPS REELS - http://perso.numericable.fr/ pdeiberale69/patrick.deiber/pub/ os-tr.pdf
9. *Зинченко С.В.* Исследование характеристик знание ориентированных интеллектуальных систем мягкого реального времени. *Комп'ютерні засоби, мережі та системи*. 2012. № 11. С. 13 – 22.
10. *Египко В.М., Зинченко В.П.* Метод и инструментальные средства обработки данных эксперимента в реальном времени. Киев: 1995. 24 с. (Препринт / АН Украины. Ин-т кибернетики им. В.М. Глушкова; 95-20).
11. *Стивенс У.* UNIX: взаимодействие процессов. СПб.: Питер, 2003. 576 с.
12. *Зинченко С.В., Зинченко В.П.* К вопросу планирования задач в системах реального времени. XI Междунар. конф. «Гидротехнологии, навигация, управление движением и конструирование авиационно-космической техники». Сб. докл. Ч. 3. Секц. Информ. технол. К.: КПИ им. Игоря Сикорского, 13 – 14 апреля 2017. С. 40 – 47.
13. *Конвей Р.В., Максвелл В.Л., Миллер Л.В.* Теория расписаний. М.: Наука, 1975. 360 с.
14. *Теория* расписаний и вычислительные машины / Под ред. Э.Г. Коффмана. М.: Наука, 1984. 334 с.
15. *Организация* вычислительных процессов: конспект лекций / сост. Б.М. Степанов. – Улан-Удэ: Изд-во ВСГТУ, 2001. 41 с.
16. *Зинченко В.П., Зинченко С.В.* Алгоритмы и базовые программные модули для управления технологическими модулями Prometheus. УСиМ, 2007. № 5. С. 52 – 60.
17. *Зинченко В.П.* Исследование и реализация алгоритма адаптивного управления экспериментом. *Проблемы управления и информатики.* 2001. № 3. С. 58 – 69.

Получено 20.03.2017

Об авторах:

Зинченко Сергей Валерьевич,
научный сотрудник
Института кибернетики имени В.М. Глушкова НАН Украины,
E-mail: zinchenkosv@gmail.com

Зинченко Валерий Петрович,
кандидат технических наук, доцент,
доцент НТУ Украины «КПИ» им. Игоря Сикорского,
E-mail: zinchenko.vp@gmail.comи

Горин Феликс Николаевич,
кандидат технических наук, заведующий лабораторией
Института кибернетики имени В.М. Глушкова НАН Украины.
E-mail: gorin@mail.ru