

ОБЗОР СОВРЕМЕННЫХ ПРИМЕНЯЕМЫХ МЕТОДОВ И СРЕДСТВ  
ВЕРИФИКАЦИИ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Россия, г. Пенза, Пензенский государственный технологический университет

*The article provides a theoretical overview devoted to the description of modern methods and means of verification of real-time systems. The existing algorithms of event planning in real-time systems, formal ways of describing requirements and ways of their verification are described. The properties of real-time systems are described, namely: periodic, aperiodic, sporadic tasks; multiprocessor / single-processor systems; tasks with fixed/dynamic priority; dependent/ independent tasks;*

*A brief overview of the Peceprio Tracealyzer, Segger SistemView and Rapita Systems Rapitime event monitoring software is shown. The programs mentioned in the article help to increase the quality and reliability of real-time systems, allow you to visualize ongoing processes and events in a user-friendly way. However, despite their advantages, the use of such tools mainly helps to detect and eliminate existing errors in the system at the design stage. It is worth noting that critical systems require constant monitoring of events occurring in them and verification of compliance with system requirements, which is possible only with the use of run-time verification.*

**Введение.** В системах реального времени, где основным критерием эффективности является предоставление определенного времени отклика, планирование имеет особое значение. Каждая система реального времени должна оперативно реагировать на сигналы контролируемого объекта в течение заданного периода времени [1, 2]. В связи с высокой надежностью и предсказуемостью, требуемых от таких систем, проверка и валидация – фундаментальные действия, которые требуется выполнять в соответствии со спецификацией системы. Статическая проверка является одним из таких средств. Тем не менее, такая проверка испытывает практические ограничения, такие как неразрешимость свойств базовой формальной модели или «взрыв состояний» при отслеживании. Более того, дополнительные функциональные свойства, такие как время, в которое происходит событие, доступны только во время выполнения. Проблеме планирования и верификации систем реального времени сегодня уделяется много внимания [3, 4], но, в основном, проводимые исследования сосредоточены на статической проверке систем во время проектирования. Необходимость использования run-time верификации достаточно полно рассматривается в работе [5], где описаны недостатки статических подходов планирования, разработан фреймворк (программная платформа), позволяющий генерировать мониторы для наблюдения за системой в режиме работы и проводить проверку с помощью конечных автоматов, создаваемых мониторами процессов.

Основная задача работы в том, что нужно провести обзор существующих средств наблюдения за событиями и обнаружения ошибок. На сегодня существует множество программных средств наблюдения за событиями в таких системах. Примерами являются Персеприо Tracealyzer, Segger SystemView, а также продукты компании Rapita Systems. Данные средства позволяют облегчить контроль, увеличить качество и надежность систем реального времени, но внедрение средств наблюдения за событиями не может не влиять на скорость и эффективность выполнения системы. Все эти инструменты имеют возможность журналирования событий в реальном времени и

последующего анализа оператором. Некоторые из них имеют возможности run-time анализа, но только при подключении дополнительных инструментальных средств, например J-link для SeggerView.

**Алгоритмы планирования задач.** Для определенного набора заданий основная задача планирования состоит в определении порядка выполнения заданий, удовлетворяющего все имеющиеся ограничения. Как правило, задание характеризуется временем выполнения, временем готовности, сроками и требованиями к ресурсам. Выполнение задания может быть прервано или нет. Над набором заданий существует отношение приоритета, которое ограничивает порядок выполнения. В частности, выполнение задания не может начаться до завершения выполнения всех его предшественников (в соответствии с отношением приоритета). Система, в которой должны выполняться задания, характеризуется объемом имеющихся ресурсов. При планировании в системах реального времени необходимо учитывать следующие факторы: Удовлетворение временных ограничений системы; Предотвращение одновременного доступа к общим ресурсам и устройствам. В целом, задача планирования состоит в том, чтобы определить график выполнения заданий так, чтобы все они были завершены до общего срока, за данного системой реального времени. Соответствующий подход планирования должен быть разработан на основе свойств системы и задач, возникающих в ней. Системы реального времени обладают следующими свойствами:

**1. Периодические, аperiodические, спорадические задания.** Периодические задания – задания реального времени, которые появляются регулярно с определенным периодом. Обычно периодические задания имеют ограничения, указывающие, что их экземпляры должны выполняться один раз за период  $P$ . Аperiodические задания появляются нерегулярно с некоторым заранее неизвестным периодом и частотой. Ограничение времени обычно описывается крайним сроком (deadline)  $D$  выполнения задания. Спорадические задания появляются нерегулярно, но имеют заранее известную частоту. Аperiodические задания имеют крайний срок, к которому они должны начаться или закончиться, или они могут иметь ограничение как на время начала, так и на время окончания. Большая часть заданий, возникающих в системах реального времени, носит периодический характер.

**2. Многопроцессорные / однопроцессорные системы.** Количество доступных процессоров является одним из основных факторов при планировании заданий в системах реального времени. В многопроцессорных системах алгоритмы планирования должны предотвращать одновременный доступ к общим ресурсам и устройствам.

**3. Задания с фиксированным / динамическим приоритетом.** При таком планировании каждому заданию присваивается приоритет. Определение приоритетов может быть выполнено статически до старта системы, или динамически, во время работы.

**4. Зависимые / независимые задания.** При планировании систем реального времени, содержащих такие задания, должен быть составлен строгий порядок начала и окончания выполнения.

Основными целями планирования в реальном времени являются выполнение заданий в рамках определенных временных ограничений и предотвращении одновременного доступа к общим ресурсам и устройствам. Алгоритмы планирования, используемые или предлагаемые для использования в режиме реального времени, варьируются от относительно простых до чрезвычайно сложных. Множество алгоритмов планирования делится на два основных подмножества: off-line алгоритмы планирования и on-line алгоритмы планирования.

**Off-line алгоритмы** генерируют информацию о планировании до начала работы

системы. Сгенерированная информация используется системой во время выполнения. В системах, использующих off-line планирование, существует, как правило, если не всегда, необходимый порядок исполнения процессов. Это может быть осуществлено с помощью отношений приоритета, которые применяется во время такого планирования. Предотвращение одновременного доступа к общим ресурсам и устройствам – еще одна функция, которую должен обеспечивать вытесняющий off-line алгоритм с фиксированными приоритетами. Это может быть достигнуто путем определения, какая часть процесса не может быть вытеснена другим заданием, а затем определение ограничений и исключений, и их принудительное применение в off-line режиме. Еще одним параметром, который желательно принимать во внимание при составлении off-line расписаний, является снижение стоимости контекстных переключений, вызванных вытеснением. Это можно осуществить, выбирая алгоритмы, которые не приводят к большому количеству вытеснений. Желательно также увеличить шансы на то, что будет найдено осуществимое расписание. Если входные данные заданной off-line алгоритма совпадают с входными данными всей системы, тогда математические off-line алгоритмы с большей вероятностью найдут возможное расписание.

Off-line алгоритмы успешно применяются, когда все характеристики системы известны априори и изменяются нечасто. Таким алгоритмам требуется полная характеристика системы и такие параметры, как время выполнения, крайние сроки и время готовности. Для off-line алгоритмов требуется большое количество автономного времени обработки, чтобы составить окончательное расписание, поэтому эти алгоритмы достаточно негибкие. Любое изменение системных процессов требует запуска задачи планирования с самого начала. Несмотря на то, что строгий off-line планировщик не предусматривает обработку аperiodических заданий, можно перевести аperiodический процесс в периодический, что позволяет планировать аperiodические процессы с использованием off-line планирования. Основным такого планирования является значительное сокращение ресурсов времени выполнения планирования, включая время обработки.

**On-line алгоритмы** генерируют информацию о планировании во время работы системы. Такие планировщики не предполагают каких-либо знаний о характеристиках процессов, которые еще не были обработаны. Эти алгоритмы требуют большого количества времени обработки во время выполнения. Одной из серьезных проблем, которые могут возникать с приоритетными on-line алгоритмами, являются приоритетные инверсии. Такое происходит, когда задание с более низким приоритетом использует ресурс, требуемый для задания с более высоким приоритетом, и это приводит к блокировке задания с более высоким приоритетом заданием с низким приоритетом. Основное преимущество on-line планирования заключается в том, что нет необходимости заранее знать характеристики заданий, и то, что они, как правило, гибки и легко адаптируются к изменениям среды. Однако исходное предположение о том, что система не знает характеристик процесса для еще не выполненных заданий, серьезно ограничивает возможности выполнения ограничений в сроках и распределении ресурсов. Если планировщик не обладает такими знаниями, невозможно гарантировать, что системные ограничения по времени будут выполнены. Несмотря на недостатки on-line планирования, этот метод используется для планирования многих систем реального времени, так как он работает достаточно надежно в большинстве случаев и является гибким. Алгоритмы on-line планирования можно разделить на алгоритмы на основе статического приоритета и алгоритмы на основе динамического приоритета.

**Методы верификации временных ограничений.** Верификация системы реального времени является непростым процессом по многим причинам [6]. Система создает и работает с несколькими параллельными задачами пользователя, гарантирует

выполнение этих задач в течение фиксированных сроков с использованием некоторого механизма приоритета, гарантирует невмешательство этих задач в работу друг друга, и доступ к общим ресурсам памяти, позволяет задачам передавать информацию с помощью очередей, синхронизирует их с помощью мьютексов или семафоров и взаимодействует напрямую с оборудованием через драйверы устройств, счетчики и механизмы прерываний. Для некоторого набора процессов с известными периодами, сроками и временем работы, основной проблемой является нахождения расписания, позволяющего выполнить все задания в определенные сроки. Это называется разрешимость- существует ли такое расписание с требуемыми параметрами? Другой задачей является проблема вычислимости - учитывая график, необходимо выяснить, является ли он выполнимым. В on-line системах расписания должны создаваться во время работы системы, поэтому важно убедиться, что вычислительные усилия ограничены (в частности, проблема должна быть проблемой класса  $P$ ). Вычислительно сложная проверка расписания может помешать выполнению критических процессов и привести к сбою системы. Чтобы этого избежать, необходимо найти методы, которые могут быстро и с минимальной вычислительной мощностью проверить, является ли расписание выполнимым и возможно ли построение расписания для некоторого набора заданий.

Самый простой случай – все задания независимы друг от друга и периодичны. В данном случае целесообразно использовать вытесняющий планировщик, который присваивает приоритеты на основе периода процессов. Процессы, которые происходят с наибольшей частотой, имеют более высокий приоритет, который присваивается заранее. Это пример on-line планировщика с фиксированным приоритетом. При верификации такой системы используется скоростно-монотонный алгоритм (Rate-monotonic scheduling, RM). На первом этапе алгоритм выясняет, является ли набор процессов планируемым. Для этого рассчитывается загрузка системы

$$U = \sum_{i=1}^n \frac{C_i}{P_i}, \quad (1)$$

Это по существу является количественной мерой использования системы всеми заданиями. Очевидно, что для контекстных переключений также требуется время, если используется вытесняющий планировщик. Суммарная загрузка должна быть меньше, чем ограничение расписания

$$U < n(2^{1/n} - 1), \quad (2)$$

В случае большого количество задач, это значения приближается к 69%. Если загрузка системы больше, чем 69%, скоростно-монотонный алгоритм не сможет составить корректное расписание. Но и обратное не всегда верно: если загрузка системы менее 69%, расписание может быть составлено не во всех случаях. Это означает, что ограничение необходимо, но недостаточно. Достаточное и необходимое ограничение выражается в объеме рабочей нагрузки. Набор процессов  $\tau_1 \dots \tau_n$  упорядочен по приоритету (зависящему от их периода  $T_i$ ). Рабочая нагрузка рассчитывается как сумма отдельных рабочих нагрузок

$$W_i(t) = \sum_{i=1}^n C_i \frac{t}{T_i}, \quad (3)$$

Если выражение  $\min_{0 < t \leq T_i} \frac{W_i(t)}{t} \leq 1$  справедливо для всех процессов, то для всех них может быть составлено корректное расписание. Аналогичные ограничения могут быть сформулированы для динамических on-line систем так же, как и для off-line систем. В таких системах процесс с самым близким крайним сроком имеет наивысший приоритет. Загрузка системы может быть рассчитана так же, как в предыдущем случае, с той разницей, что она не ограничена 70%. Напротив, динамические on-line системы с

планированием на основе приоритетов могут использовать процессор на 100%, если

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq 1, \quad (4)$$

В таком случае это условие является одновременно необходимым и достаточным [7].

Оба представленных алгоритма используют вытесняющее планирование. Это означает, что процессы могут быть приостановлены и возобновлены без задержки контекстных переключений. При проверке системы необходимо учитывать несколько моментов. Системы реального времени обладают характеристиками, которые могут влиять на описанные выше ограничения. К ним относятся резерв времени, переходные перегрузки, изменения в периодических и аperiodических процессах.

Очевидно, что процессы с наименьшим резервом времени имеют наивысший приоритет. В таком случае, может быть использован вытесняющий off-line планировщик, а загрузка системы может достигать 100%. Однако процессы с одинаковым резервом времени могут приостанавливать друг друга, что приводит к увеличению времени контекстных переключений. Возможная ошибка, которая возникает из-за этого называется «замусоривание», и это показывает, что контекстные переключения занимают процессор большую часть времени [8].

Другой проблемой является переходная перегрузка. Хотя периодические процессы и имеют фиксированный период в отношении их крайних сроков, время выполнения изменяется вероятностно. Необходимо учитывать наибольшее время выполнения при планировании периодических процессов, если допускаются только жесткие сроки. В большинстве процессов допустимы мягкие сроки. Системы, допускающие только жесткие предельные сроки, должны гарантировать достаточно низкую загрузку системы, чтобы все процессы завершались вовремя, что не оптимально с точки зрения использования процессора. Оценка времени выполнения может быть затруднена, и система может потерпеть неудачу, если некоторые сроки пропущены. RM-алгоритм не решает эту проблему, а это значит, что должны применяться другие подходы. Чтобы обойти эту проблему при использовании RM-планировщика, периоды можно искусственно увеличить, чтобы некоторые процессы получили более высокий приоритет. Если для низкочастотного процесса существует жесткий срок, может быть полезно разделить процесс на несколько частей (вытесняющим способом) и выполнить их с большим периодом. В таком планировщике это изменит приоритет процесса на высокий и будет гарантировать, что он выполнится в жесткий срок. Это можно сделать, специально разделив задачу на более мелкие или позволив системе самостоятельно решить, какие процессы нужно обрабатывать в первую очередь.

В системах реального времени появляются не только периодические, но и аperiodические задания. Было доказано, что срокo-монотонный планировщик (Deadline-monotonic scheduler, DM) может построить оптимальное расписание, в системах, где появляются оба типа заданий. Это требует минимального деления между аperiodическими заданиями, но такое деление не всегда возможно. Также DM-планировщик может при определенных обстоятельствах обрабатывать и спорадические задания, однако это может быть непредсказуемым при наличии временных перегрузок [9].

Вместо этого, RM-планировщик может быть адаптирован для работы с аperiodическими заданиями. Может быть создано искусственное периодическое задание, заменяющее аperiodические. Очевидная проблема этого подхода заключается в том, что аperiodические задания могут быть обработаны только в том случае, если периодическое задание запущено. Это означает, что аperiodическое событие может произойти, но не обработаться, так как процессор не готов. В некоторых случаях процессор может быть готов, но не существует аperiodического задания, которое

эффективно загружает систему. Чтобы избежать этих проблем, были изобретены алгоритмы сохранения пропускной способности. Наиболее важными являются обмен приоритетами, отложенный сервер и спорадический сервер [10, 11]. Описанные алгоритмы позволяют проводить статическую проверку системы до ее запуска с помощью алгоритмов планирования задач, более подробно классификация методов верификации представлена на рисунке 1.

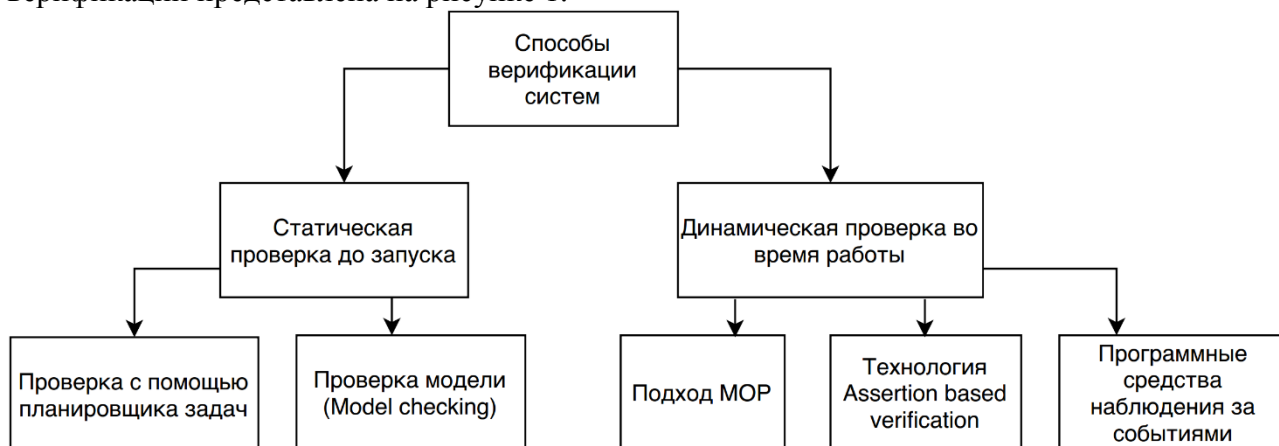


Рисунок 1 – Способы верификации СРВ

### **Известные средства встроенной верификации временных ограничений.**

**Подход Monitor oriented programming.** Мониторинг-ориентированное программирование, сокращенно МОР, представляет собой основу разработки и анализа программного обеспечения, направленную на сокращение разрыва между формальной спецификацией и реализацией [12]. В МОР run-time мониторинг является фундаментальным принципом построения надежного программного обеспечения: мониторы автоматически генерируются из заданных свойств и интегрируются в исходную систему для проверки ее поведения во время выполнения. Если спецификация нарушена, будут инициированы заранее определенные действия, которые могут быть любыми, начиная от записи логов до запуска восстановления системы. Можно рассматривать МОР по крайней мере с трех точек зрения: как дисциплину, позволяющую повысить безопасность, надежность и устойчивость системы, контролируя и сравнивая ее требования с реализацией во время выполнения; как расширение языков программирования некоторой логикой (можно добавлять логические инструкции в любое место программы, ссылаясь на прошлое или будущее состояния); и как простой формальный метод проектирования систем [13]. Подход МОР являются применимым для систем реального времени, написанных на языке программирования Java, но так как большое количество таких систем написаны на С или С++, зачастую требуется более универсальное средство наблюдения за событиями. В следующей главе описываются примеры таких средств.

**Программные средства.** Сегодня существует большое множество программных средств наблюдения за событиями. Примерами могут являться Percepio Tracealyzer [14], Segger SystemView [15], а также продукты компании Rapita Systems [16]. Данные средства помогают увеличить качество и надежность систем реального времени, позволяют визуализировать происходящие процессы и события удобным для пользователя образом. Несмотря на преимущества данных программных средств, их использование в основном помогает обнаружить и устранить имеющиеся ошибки в системе на этапе проектирования. Критически-важные системы требуют постоянного наблюдения за происходящими в них событиями, и проверки соблюдения системных требований, что возможно только при применении run- time верификации.

**Проверка требований в режиме run-time.** Run-time верификация предполагает

проверку системы на соответствие спецификации во время работы. Обычно это происходит с помощью добавления фрагментов кода, называемых мониторами, в готовое приложение. Мониторы тщательно изучают поведение системы и проверяют, удовлетворяет ли она соответствующим спецификациям. Проверка требований и наблюдение за системой в режиме run-time позволяет существенно повысить уровень надежности систем реального времени, далее описываются варианты расположения встроенного средства верификации, язык спецификации требований, алгоритм проверки требований на корректность и алгоритм верификации системы во время работы.

**Выводы.** В работе приведен обзор программных средств наблюдения за событиями Reserpio Tracealyzer, Segger SystemView и Rapita Systems Rapitime. Эти средства помогают увеличить качество и надежность систем реального времени, позволяют визуализировать происходящие процессы и события удобным для пользователя образом. Несмотря на их преимущества, использование таких средств в основном помогает обнаружить и устранить имеющиеся ошибки в системе на этапе проектирования. Критически важные системы требуют постоянного наблюдения за происходящими в них событиями и проверки соблюдения системных требований, что возможно только при применении run-time верификации.

**Исследование выполнено за счет гранта Российского научного фонда № 21-71-00110, <https://rscf.ru/project/21-71-00110/>.**

1. Жданов, А. А. Операционные системы реального времени. [Электронный ресурс] / Жданов, А. А. // PCWeek. – 1999. – Режим доступа: <http://www.rtsoft-training.ru/about/news/services/operating-systems-real-time/> (Дата обращения: 05.11.2021).
2. Голубев, А. С. Системы реального времени: конспект лекций. / Голубев, А. С. – Владимир: Издательство Владимирского государственного университета, 2010. – 127 с.
3. RTX Real-Time Operating System. [Электронный ресурс] // Режим доступа: <http://www.keil.com/pack/doc/CMSIS/RTOS/html/rtxImplementation.html> (Дата обращения: 28.10.2021).
4.  $\mu$ C/OS-II TM Real-Time Operating System [Электронный ресурс] // Режим доступа: [http://davidhoglund.typepad.com/files/ucosii\\_datasheet.pdf](http://davidhoglund.typepad.com/files/ucosii_datasheet.pdf) (Дата обращения: 24.10.2021).
5. Kochanthara, S. REVERT: Runtime Verification for Real-Time Systems: дис. маг. / Kochanthara, S. – Delhi: Indraprastha Institute of Information Technology, 2016. – 68 с.
6. Карпов, Ю. Model Checking. Верификация параллельных и распределенных программных систем / Карпов, Ю. – БХВ-Петербург, 2010. – 560 с.
7. EIT Digital. Development of Real-Time Systems. [Электронный ресурс] // Режим доступа: <https://ru.coursera.org/learn/real-time-systems> (Дата обращения: 25.10.2021).
8. Pandit, S. Survey of Real Time Scheduling Algorithms / Pandit, S., Shedge, R. // Mumbai: IOSR Journal of Computer Engineering (IOSR-JCE), 2013.– pp. 44-51.
9. Audsley, N. C., Hard real-time scheduling: The deadline-monotonic approach / Audsley N. C., Burns A., Richardson M. F., Wellings A. J. // York: Department of Computer Science, University of York, 1991. – pp. 127-132.
10. Barr, M., Introduction to Rate Monotonic Scheduling. [Электронный ресурс] / Barr, M., Stewart, D. B. // Режим доступа: <https://www.embedded.com/electronics-blogs/beginner-s-corner/4023927/Introduction-to-Rate-Monotonic-Scheduling> (Дата обращения: 06.11.2021).
11. Bertossi, A. A. Rate-monotonic scheduling for hard-real-time systems / Bertossi, A. A., Fusiello, A // European Journal of Operational Research. – Т. 96. - №. 3. – pp. 429 – 443.

12. Monitoring-Oriented Programming. [Электронный ресурс] // Режим доступа: [http://fsl.cs.illinois.edu/index.php/Monitoring-Oriented\\_Programming](http://fsl.cs.illinois.edu/index.php/Monitoring-Oriented_Programming) (Дата обращения: 03.11.2021).

13. Meredith, P., Jin, D., Griffith, D., Chen, F. и Ros, G. An Overview of the MOP Runtime Verification Framework / Meredith, P., Jin, D., Griffith, D., Chen, F., Ros, G. // International Journal on Software Tools for Technology Transfer. – Т. 14. - №. 3. – pp. 249 – 289.

14. Stop Guessing – Trace Visualization for RTOS Firmware Debugging. [Электронный ресурс] // Режим доступа: <https://percepio.com/stop-guessing.pdf> (Дата обращения: 10.11.2021).

15. SEGGER SystemView User Guide. [Электронный ресурс] // Режим доступа: <https://www.segger.com/downloads/free-utilities/UM08027> (Дата обращения: 04.11.2021).

16. Measurement based timing and WCET analysis with RapiTime. [Электронный ресурс] // Режим доступа: [https://www.rapitasystems.com/system/files/downloads/mc-pb-101\\_rapitime\\_v2.pdf](https://www.rapitasystems.com/system/files/downloads/mc-pb-101_rapitime_v2.pdf) (Дата обращения: 11.10.2021).