SAE TECHNICAL PAPER SERIES

960623

ERCOS: Операционная система для автомобильной электроники

S. Poledna, Th. Mocken, and J. Schiemann
Robert Bosch GmbH

Th. Beck
ETAS GmbH & Co. KG

перевод выполнен locon
locon@mail.ru

Reprinted from: Design Innovations in Engine Management and Driveline Controls (SP-1153)

The Engineering Society For Advancing Mobility Land Sea Air and Space®
INTERNATIONAL

International Congress & Exposition
Detroit, Michigan
February 26-29,1996

400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A. Tel: (412)776-4841 Fax:(412)776-5760

ISSN 0148-7191

Printed in USA **96-0049**

960623

ERCOS: Операционная система для автомобильной электроники

S. Poledna, Th. Mocken, and J. Schiemann
Robert Bosch GmbH

Th. Beck
ETAS GmbH & Co. KG

Анонс

В статье описывается концепция операционной системы ERCOS (встраиваемая операционная система управления в реальном времени), разработанной специально для удовлетворения функциональных и эксплуатационных потребностей автомобильной электроники.

Все более функциональные требования для современных электронных блоков управления приводят к проблемам значительной сложности в области разработки программного обеспечения Известно, что операционные системы реального времени предоставляют мощные средства для обработки сложных функций в условиях реального недостатка времени. Однако опыт прошлых лет показал, что эффективность и гибкость операционных систем очень часто недостаточна для применения в автомобильной электронике.

Чтобы преодолеть эти недостатки операционная система ERCOS была целенаправленно разработана с учетом требований автомобильной промышленности. Добиться этого удалось путем добавления к операционной системы реального времени, мощнейшего off-line инструментария. Данный инструментарий поддерживает структуру многократного использования и модульности программного обеспечения, в приложениях реального времени, в силу объектно-ориентированной модели и строгого разделение системно-независимого функционала и конфигурации времени выполнения. Кроме того, этот инструментарий позволяет оптимизировать обращения к операционной системе для достижения более высокой производительности.

В статье представлено, как в ERCOS реализована основная концепция операционных систем реального времени, т. е. планирование и взаимодействие процессов, взаимоисключающий доступ к ресурсам, управление синхронизацией процессов и отказоустойчивость системы.

Фирма Bosch использует операционную систему ERCOS в качестве стандартной платформы для своей продукции. Операционная система совместима со спецификацией [OSE9S] OSEK. Проектирование системы было поручено независимой фирме по разработке программного обеспечения “ETAS GmbH & Co. KG”.

1. Введение

За последнее десятилетие функциональность и сложность автомобильной электроники претерпела резкий подъем. Примером этому могут служить электронные системы управления ДВС, системы АБС, системы управления АКПП и другие. Это развитие, поддерживаемое постоянно растущей вычислительной мощностью микроконтроллеров, шло наряду с быстрым ростом размера и сложности программного обеспечения. В прошлом, достижимый уровень функциональности автомобильной электроники был в основном определен схемными решениями, а также производительностью микроконтроллеров. В настоящее время сдерживающий фактор смещается все больше и больше от производительности оборудования к процессу разработки программного обеспечения. Особенно в случаи проектирования надежных (отказоустойчивых) систем управления фактором, ограничивающим достижение нужного уровня функциональности, является программное обеспечение.

Хорошо известно, что разработка сложных программно-аппаратных систем требует поддержки операционной системой. Это понимание, взятое из опыта работы с большими компьютерными системами, также относится и к автомобильной электронике. Есть, однако, специальные требования для операционных систем в сфере автомобильной электроники. Во-первых, операционная система должна поддерживать требования “жесткого” реального времени, которые диктуются средой работы системы, например, синхронизация момента впрыска с положением коленвала двигателя. Во-вторых, так как стоимость имеет первостепенное значение, должна быть достигнута очень высокая эффективность операционной среды. Особенная потребность в высокой эффективности идет в разрез с производительностью операционных систем реального времени общего назначения. Это стало причиной того, почему коммерческие операционные системы, например, [Rea86, ISI93] не добились признания в этой области применения.

Поэтому операционная система ERCOS специально была разработана для удовлетворения потребностей автомобильной электроники.

Фирма Bosch использует ERCOS в качестве

стандартной операционной системы для автомобильной электроники.

Далее в этой статье … Во второй главе будут определены цели и задачи программного обеспечения для автомобильной электроники. В третьей главе обсуждаются вопросы, касающиеся структуры объектно-ориентированного программного обеспечения для требований “жесткого” реального времени в многозадачных средах, а также дается введение в объектную модель ERCOS. Главы с четвертой по восьмую представляет основные функции системы и ее понятия. Такие как: стратегия планирования, взаимодействие процессов, взаимоисключающий доступ к важным ресурсам, функции таймера, особенности отказоустойчивости и обработка исключений. Наряду с этим представлены возможные альтернативные решения, и обоснование решений реализованных в ERCOS. Наконец, девятая глава завершает эту статью.

2. Цели и задачи

Разработка ERCOS находилась под влиянием ряда конкретных целей и задач. Была поставлена цель, создать единую платформу для автомобильной электроники фирмы Bosch. Дальнейшие требования к операционной системе были сосредоточены на качестве программного обеспечения и процессе разработки программного обеспечения. Т.е. состав целей и задач в первую очередь был ориентирован не на функциональные возможности операционной системы, а на разработку программного обеспечения для автомобильной электроники и на сам процесс разработки. Обзор этих целей и задач сосредоточен в следующем:

Повторное использование:
Необходимо для многократного использования программного обеспечения в различных системах управления и в различных проектах. Что в свою очередь требует, чтобы функциональная реализация была независима от видов синхронизации, а также глобальных свойств системы, таких как приоритеты или стратегия планирования.

Модульность:
Программное обеспечение должно быть структурировано, согласно концепции объектно-ориентированного программирования, для поддержки модульности в процессе разработке и тестирования.

Эффективность:
Ресурсы микроконтроллера, такие как ОЗУ, ПЗУ, процессор, а также периферийные устройства, такие как таймеры, порты ввода/вывода и АЦП должны использоваться весьма эффективно. В автомобильной промышленности, где весьма чувствительно сказывается стоимость реализации, эффективность имеет первостепенное значение.

Расширяемость:
Должна быть простая возможность изменения и дополнения к существующему программному обеспечению. Эти изменения не должны пересекать границы интерфейсов и влиять на другие функции, кроме случая внесения изменений в интерфейс. Стоит отметить, что это требование может быть выполнено только в области значений, а не в области синхронизации.

Поддержка режима реального времени:
Программное обеспечение должно поддерживать работу в реальном времени. Поэтому необходимо, чтобы система гарантированно реагировала на запросы в течение определенного периода ожидания. Существует очень широкий диапазон периода ожидания, от 100 миллисекунд до всего несколько микросекунд.

Указанные цели и задачи направлены на управление неуклонным ростом сложности программного обеспечения, на сокращение сроков разработки и улучшения качества программного обеспечения. Опыт также показывает, что для решения этих задач должны быть применены четко выверенные методы разработки программного обеспечения и специализированный инструментарий поддержки разработки. Ключевым моментом в разработке современного программного обеспечения, в свете поставленных целей и задач, следует считать ООП, кроме поддержки режима реального времени и производительности системы. Этот момент следует пристальнее рассматривать в процессе разработки программного обеспечения в области автомобильной электроники. Однако есть и часть проблем в использовании ООП при разработке многозадачной операционной системы реального времени. Эти вопросы обсуждаются в следующих разделах.

3. Объектная модель ERCOS

В течение последнего десятилетия стиль объектно-ориентированного программирования [Мey88] нашел широкое признание в области создания приложений работающих не в режиме реального времени. Основными целями ООП является улучшение гибкости, надежности и повторного использования программного обеспечения. Хотя эти цели также важны и в приложениях реального времени, классические языки ООП, такие как C++ [Str91] недостаточны для создания многозадачных сред “жесткого” реального времени. Помимо таких функций, как динамическое связывание и множественное наследование, основной проблемой ООП многозадачных сред реального времени является реализация объекта. ООП языки не поддерживают параллельное программирование, и их наивное применение приводит к значительным накладным расходам в реализации. Таким образом, необходима

подходящая адаптация объектно-ориентированной концепции программирования и целенаправленная поддержка механизмов операционной системы, чтобы использовать преимущества ООП в рамках систем “жесткого” реального времени, когда производительность является критической величиной.

Объект - некий абстрактный объект, который имеет внутреннее состояние (невидимое извне), с четко определенным интерфейсом и характерной функциональностью. В контексте автомобильной электроники, таким объектом может быть, например, топливная форсунка или таблицы (карты) топливоподачи. Важный аспект объекта - граница в его реализации, которая заключена между объектом и его интерфейсом. Как правило, интерфейс объекта состоит из методов (функций) и атрибутов (переменных). Взаимодействующий с переменными объект, однако, не может быть использован для многозадачных систем реального времени, так как в этом случае может быть нарушена целостность данных.

3.1. Вопрос целостности данных

Системы реального времени обычно поддерживают вытесняющее планирование, чтобы гарантировать короткие периоды ожидания реакции системы. Это может привести к случаям, где выполнение некоторого низкоприоритетного процесса будет вытеснено более высокоприоритетным процессом. В предположении, что вытесненный процесс читал один из атрибутов объекта, а вытесняющий процесс записывал данные в тот же атрибут объекта, может возникнуть несоответствие, если вытеснение происходит между двумя последовательными операциями чтения, в низкоприоритетном процессе, см. Рис 1.

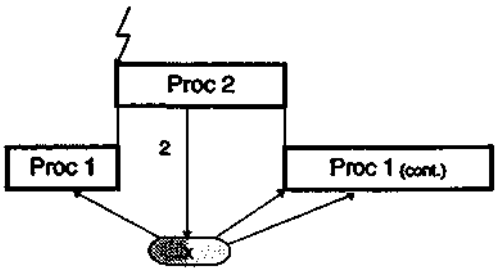

Рис. 1: Противоречивость данных объекта

После вытеснения процесса *Proc 2*, процесс *Proc 1* считывает атрибут *x* объекта, второй и третий раз, и получает другой результат (x = 2), чем до вытеснения (x = -1). Этот пример показывает, как данные, к которым обращается процесс *Proc 1*, становятся недостоверными. Как следствие, это может привести к ошибке в работе процесса *Proc 1*. Например, пусть *Proc 1*, реализует функцию, которая проверяет атрибут *x* и имеет две различных ветки выполнения, где *x* используется снова. Если *Proc 1* не будет вытесняться, то реализация будет работать правильно. Но если синхронизация процессов немного изменится или если процесс *Proc 1* многократно используется в приложении, то может легко случиться, что процесс *Proc 1* будет вытеснен, и больше не будет функционировать правильно. Рассмотрим случай, где процесс *Proc 1* вычисляет абсолютное значение *x*:

```
if (x<0)
   {y= -x;}
else
   {y=x;}
```

Так как атрибут *x* может измениться между операцией сравнения и операцией присваивания, выполнение процедуры *Proc 1* может привести к ошибке.

Таким образом, правильность выполнения алгоритма зависит от синхронизации процессов и последовательности приоритетных прерываний в определенной системе, как показано в примере. Однако все это находится в противоречии с основными целями многократного использования программного обеспечения, его модульности и расширяемости. Это также находится в противоречии с целями ООП, где объекты должны инкапсулироваться, и их правильное функционирование не должно зависеть от среды функционирования. Чтобы избежать программных сбоев, которые зависят от синхронизации процессов и системной конфигурации, должна быть гарантирована целостность данных.

Целостность данных:
В течении, между началом и завершением, процесса P_1 нужно гарантировать, что все данные, к которым может обращаться P_1, могут изменить свое значение, если и только если они изменены самим P_1.

3.2. Объектная модель ERCOS

Для гарантирования целостности данных, ERCOS предоставляет механизм обмена сообщениями (глава 5), вместо переменных. Этот механизм разделяет области памяти данных различных процессов, и обеспечивает функции обмена информацией между процессами. Интерфейсы объектов ERCOS, поэтому состоят из функций (методы) и сообщения (вместо атрибутов, читай переменных).

Основными классами объектов, предоставляемых ERCOS, являются процессы, функции, сообщения и ресурсы. Из основных классов только процессы, являются активными. Процессы изменяют свое состояние автономно, потому что они активизируются исключительно операционной системой. Процессы предоставляют методы для инициализации и активации. Функция - пассивный объект, который можно вызвать.

Сообщение - основной объект для коммуникации между процессами. Сообщения предоставляют методы передачи и получения информации. К ресурсам можно обращаться только исключительно.

Система ERCOS предоставляет объектам-ресурсам методы захвата и освобождения ресурса. На рис. 2 показан пример основных объектов ERCOS и их связей. Объекты, которые рассматривают здесь, являются основными объектами на самом низком уровне иерархии. Система ERCOS предоставляет возможность создавать составные объекты, с хорошо определенными интерфейсами, из основных объектов.

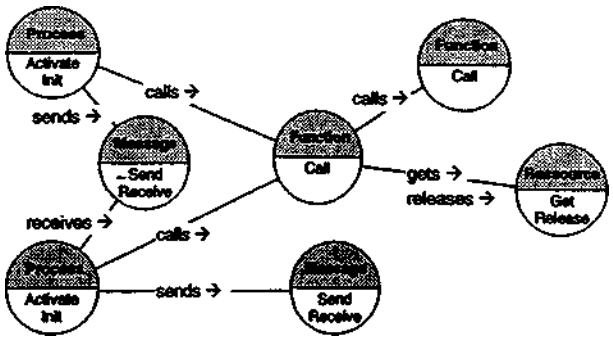

Рис 2: Основные объекты ERCOS и их связи

Эти составные объекты называют *подсистемами.* Каждая подсистема определяет, какие объекты составляют интерфейс, а какие скрыты внутри. На рис. 3 показан пример подсистемы, которая предоставляет интерфейс из функции и сообщения. А также подсистема имеет скрытые (недоступные извне) два процесса, две функции и один ресурс, для реализации своей функциональности.

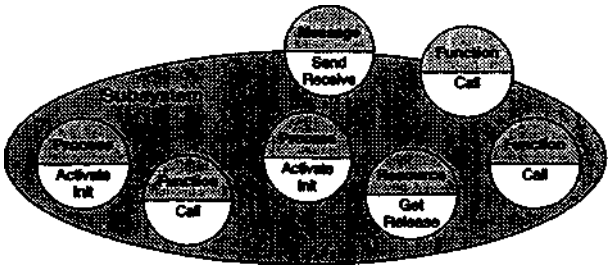

Рис. 3: Составной объект (подсистема)

В пределах подсистемы нет никакого ограничения в последовательности действий. Подсистемы могут содержать произвольное число процессов, которые могут быть выполнены параллельно, посредством режима многозадачности. Это дополнительно дает возможность параллельного обращения к интерфейсам подсистемы.

Объектная модель ERCOS хорошо удовлетворяет требования для многозадачных систем реального времени. Поддерживает ООП структуру, многократное использование, модульность и расширяемость. Объектная модель ERCOS поддерживается рядом всесторонних инструментальных средств, которые позволяют легко разрабатывать ООП приложения, и в тоже время обеспечивают контроль целостности данных и методику оптимизации управления объектами во время выполнения.

4. Планирование

Планирование - одна из основных функций операционной система реального времени. Планировщик должен решить, какой процесс должен быть начат из ряда готовых к выполнению процессов [CSR87]. Эта решающая стратегия, называемая алгоритмом планирования, очень важна, так как оказывает существенное влияние на возможности и эффективность системы в режиме реального времени. Чтобы достигнуть строгих требований эффективности и производительности в реальном времени, ERCOS использует комбинацию статического и динамического планирования вместе со смешанной вытесняющей и кооперативной многозадачностью.

4.1. Статическое и динамическое планирование

Основное различие стратегии планирования - статическое оно, или динамическое [SSN95]. При статическом планировании у планировщика есть конечные знания обо всех процессах и их ограничениях. Обычно к ограничениям относят - время вычисления, окончание работы задачи, приоритетность и взаимное исключение. Так как все ограничения процесса известны перед запуском системы, то возможно определить порядок выполнения процессов еще в off-line. Если осуществлять такое off-line планирование, то достаточно вовремя запускать процессы во время выполнения в предопределенных точках с предопределенным порядком. С другой стороны, у динамического планировщика есть только знание о готовых к выполнению процессах, но он ничего не знает о времени активации этих процессов. Так как новые процессы могут стать готовыми спонтанно, планировщик должен сам решить во время выполнения, какой процесс должен быть выбран среди готовых процессов.

Преимущество динамического планирования по отношению к статическому планированию - гибкость в реакции на внешние события. Особенно сказывается уменьшение эффективности статического планирования с уменьшением периода ожидания [Pol95b]. К недостаткам динамического планирования можно отнести более высокие требования к вычислительным ресурсам и памяти при управлении процессами. Поддерживая как статическое, так и динамическое планирование, ERCOS предоставляет реализацию объединенной стратегии, оптимальной к требованиям приложения в отношении времени реакции системы и требований к памяти.

4.2. План-список

Объектно-ориентированная структура операционной системы и современные методы разработки программного обеспечения нацелены на результат многократного использования программного кода, разбитого на большое число мелких процессов. Анализ и опыт показывает, что между многими из этих процессов, существуют простые последовательные связи предшествования. Система ERCOS использует этот факт в своих интересах и реализует задачи как *план-список*, который представляет эти связи предшествования.

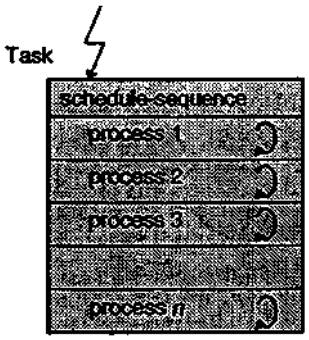

Рис. 4: План-список

План-список может быть определен как результат статического планирования. Список содержит последовательность процессов, которые будут выполнены ядром в определенном порядке и с заданным уровнем приоритета, после возникновения определенного события активации. Динамическое планирование задач (или мультизадачность) касается план-списка только в целом, а не индивидуальных процессов из списка. На рис. 4 изображено принципиальное понятие задачи, определенной как план-список. В пределах план-списка планировщик не должен принимать решений о планировании, так как порядок выполнения процессов задачи определен статически. Это снижает требования к вычислительным ресурсам во время выполнения. А также, значительно снижаются требования к наличию памяти. Этот метод планирования, как правило, сокращает количество объектов, которыми нужно управлять во время выполнения в 10-20 раз.

4.3. Кооперативное и вытесняющее планирование

Есть две возможных стратегии по переключению выполняющейся задачи на ожидающую, готовую к выполнению, задачу с более высоким приоритетом. Одна из стратегий, называемая *кооперативным* планированием, переключает выполнение в предопределенных точках в программном коде. Эти предопределенные точки - границы между процессами в пределах задачи. В случае необходимости, также возможно осуществить момент переключения прямым обращением к операционной системе.
Так как планировщик вынужден ожидать, пока выполняющийся процесс не будет готов к переключению (т.е. как бы “сотрудничает” с приложением), то такой планировщик является *кооперативным,* см. рис. 5.

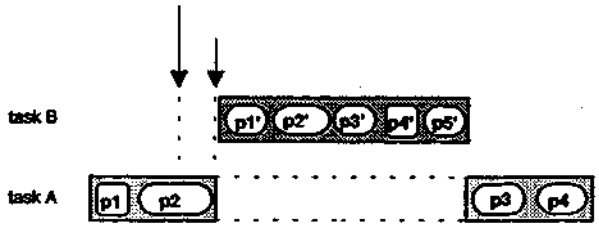

Рис. 5:Работа кооперативного планировщика

При другой стратегии, называемой *вытесняющим* планированием, выполнение может быть переключено в пределах процесса на границе машинной инструкции (предполагается, что прерывания не запрещены). Т.е. планировщик в состоянии приостановить, в текущий момент, выполняющийся процесс в пределах задачи и начать выполнение задачи с более высоким приоритетом, см. рис. 6.

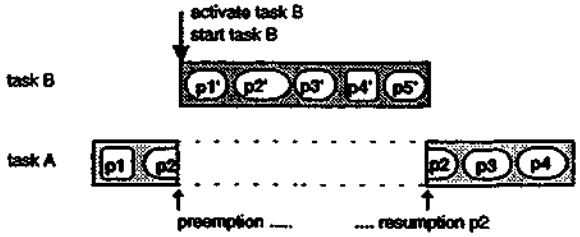

Рис. 6:Работа вытесняющего планировщика

Преимуществом кооперативного планирования является эффективное использование ресурсов. В соответствии с проектом гарантируется, что доступ к ресурсам, таким как стек, регистры или сообщения будет осуществляться исключительно. Вдобавок, нет никакой необходимости сохранять контекст процесса, переключаясь между процессами, все процессы могут выполнить с тем же самым банком регистров и с одним стеком. Неудобством кооперативного планирования является относительно медленная реакция на событие, которая зависит от самого худшего (большого) времени выполнения из всех процессов. Для внешних событий (прерываний) и для периодических действий с управляемым джиттером, необходимо иметь короткое время ответа. Вытесняющее планирование может соответствовать требованию малого времени реакции на событие. Неудобством вытесняющего планирования является требование больших объемов памяти, так как необходимо сохранять

контекст вытесненных процессов и гарантировать целостность данных (см. глава 3.1).

В связи с этим система ERCOS предоставляет комбинированный кооперативно-вытесняющий планировщик, что позволяет выбирать самую эффективную комбинацию для определенного приложения. Как правило, только у небольшого количества приложений есть требования малого времени реакции на событие и вытесняющего планирования, в то время как основная масса приложений может работать под кооперативным планированием. Таким образом, можно минимизировать необходимое количество памяти, продолжая гарантировать приложению, специфические требования соответствия режиму реального времени. Комбинированный планировщик реализован в соответствии с иерархической концепцией планирования, где кооперативный планировщик подчинен вытесняющему планировщику. Кооперативный планировщик работает как отдельная задача на самом низком приоритетном уровне вытесняющего планировщика. Оба планировщика используют фиксированные, установленные ранее приоритеты, как описано в [LL73, KRP+93]. Способность планирования может быть проанализирована для вытесняющей и кооперативной части согласно [Bak91] и [JSM91] соответственно. Принципиальная схема иерархического комбинированного планировщика показана на рис. 7.

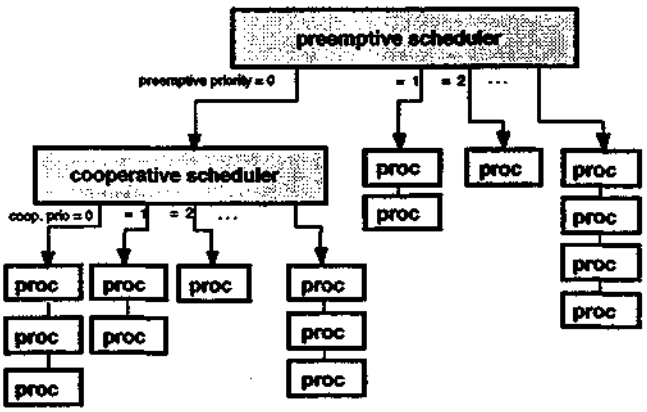

Рис. 7: Комбинированный кооперативно-вытесняющий планировщик

Вытесняющий планировщик обеспечивает дополнительную гибкость в обработке внешних событий. Это позволяет назначить источник прерывания на приоритетный вытесняющий уровень. Нет никакого различия, как были активизированы задачи, аппаратно (прерывания) или программно. Таким образом, ERCOS обеспечивает объединение понятий аппаратной и программной активизации задач.

5. СООБЩЕНИЯ

Объектная модель ERCOS основана на сообщениях, используемых для передачи данных между объектами. Для достижения внутренней целостности данных в системе реального времени с вытесняющей многозадачностью, нужно гарантировать, что объекты не будут совместно использовать память. Механизм сообщения разделяет пространство памяти объектов и выполняет необходимый информационный обмен.

5.1. Семантика сообщений событий и состояний

Существует два возможных типа сообщений, семантика которых описана далее.

Семантика событийных сообщений:
Событийные сообщения [Tan92] характеризуются как операции разовых сообщений, т.е. принятое процессом сообщение перестает существовать и, таким образом, при следующей операции получения сообщения, процесс получает уже следующее (другое) сообщение. Таким образом, сообщение как бы связано с событием, которое будет обработано после получения сообщения. Событийные сообщения, как правило, реализуются очередями сообщений и, как обычно бывает, может быть отправлено сообщений больше, чем будет принято. Событийные сообщения представляют собой некую синхронизацию между отправителем и получателем, т.е. для каждого отправленного сообщения, есть точно один получатель. Эта синхронизированная один-в-один связь может быть осуществлена с блокировкой или без блокировки (процесса, ресурса, и т.п.).

Семантика сообщения состояния:
Сообщения состояний [KDK+89] очень похожи на глобальные переменные. Принимаемое сообщение состояния возвращает последнее отправленное значение. Эти сообщения не являются разовыми. Поэтому возможно не один раз получить значение отправленного сообщения состояния. Различие между глобальными переменными и сообщениями состояния состоит в том, что в глобальные переменные может быть осуществлена запись в произвольные моменты времени, тогда как получатель сообщения состояния получает копию сообщения, которая сохраняется неизменной, если нет далее операции приема, выполнение которой гарантирует целостность данных. Сообщение состояния, таким образом, отражает последнее состояние некоторого объекта, которое можно прочитать, получив сообщение, или обновить, отослав сообщение. У сообщений состояния нет никакой синхронизированной связи между отправителем и получателем. Скорее всего, сообщения могут использоваться для не синхронизированного 1:*n* или *m*:*n* информационного обмена между процессами. Сообщения состояния, как правило, реализуются пулами сообщений. Отправитель помещает сообщение в пул, в то время как получатели получают копии этого сообщения из пула.

Доступные коммерчески операционные

система реального времени, например, [Rea86, ISI93, TSK89] реализуют механизм событийных сообщений, который не подходит для применения в автомобильной электронике. Есть две основных вопроса. Во-первых, с функциональной точки зрения большинство действий в задаче выполняется периодически. Есть множество объектов, которые активизируются с различными периодами или частотами. А это не соответствует синхронизированной один-в-один связи между отправителем и получателем событийных сообщений. Во-вторых, реализация типичных операций приема/отправки недостаточно эффективна. В высокопроизводительных автомобильных приложениях может осуществляться передача и до 100000 сообщений в секунду, которые не смогут быть обработаны, если операциям приема/отправки требуется от 15 до 100 микросекунд на запрос. Это привело бы к теоретической загрузке процессора в 150 - 1000% лишь только задачами коммуникации.

5.2. Концепция сообщений ERCOS

Операционная система ERCOS предоставляет высоко оптимизированную реализацию событийных сообщений. Базирующийся на объектной модели ERCOS, процесс получает входные данные, выполняет действия обработки и посылает (ввод - обработка - вывод) выходные данные. Операциям приема/отправки, отображаются на объекты - сообщения. Между процессами нет никаких прямых путей информационного обмена данными (косвенный информационный обмен между процессами возможен при использовании функций с передачей параметров, в этом случае ответственность за целостность данных несет пользователь). Концептуально, реализация сообщения состояния в ERCOS обеспечивается копированием сообщения для каждого процесса получателя сообщения. Во время запуска процесса все входные сообщения копируются в private область для копий сообщений. По окончании работы процесса, все выходные сообщения, которые проведены в области для копий сообщений, копируются в global область сообщений. Принцип работы механизма сообщений состояния показан на рис. 8.

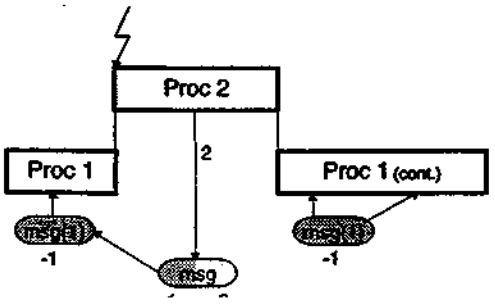

Рис. 8: Передача данных через сообщение состояния

При запуске процесса *Proc 1* будет скопировано входное сообщение *msg* в частное (private) сообщение *msg (1).* Все последующие операции чтения сообщения внутри процесса будут сделаны из частной копии сообщения. Даже притом, что процесс *Proc 1* будет вытеснен процессом *Proc 2,* который изменяет содержимое *msg* со значения “-1” на “2”. Эти изменений не затронут процесс *Proc 1*, и таким образом, гарантируется, что процесс *Proc 1* будет оперировать тем же самым содержимым своего входного сообщения во время всего выполнения. Это гарантирует целостность данных. Например, следующий алгоритм выполнится правильно независимо от того, будет процесс *Proc 1* вытеснен процессом *Proc 2* или нет (сравните с рис. 1).

```
if (x<0)
    {y= -x;}
else
    {y=x;}
```

5.2. Реализация оптимизации сообщений

Из-за высокой частоты транзакций, до 100000 сообщений в секунду, эффективность операций приема/отправки, имеет определенное значение. Вдобавок, требуемые объемы и конфигурация памяти для размещения копий сообщений должны быть ограничены абсолютным минимумом, так как память все еще является важным фактором, определяющим стоимость автомобильной электроники. Поэтому ERCOS обеспечивает мощный набор методов оптимизации, чтобы достигнуть эффективности в операциях приема/отправки сообщений. Эта оптимизация основана на статическом анализе исходного текста приложения. Чтобы предоставить дополнительную информацию для статического анализа, используется формальный язык описания, который дополняет языки Ассемблера и C исходного текста приложения.

Реализуемая стратегия оптимизации описывается следующим:

Вставка строки с операцией приема/отправки в код:
В общем случае при реализации операций приема/отправки нужно было бы решать, какая копия сообщения должна читаться или писаться в зависимости от фактического контекста процесса. В случае статического анализа исходного текста, операции приема/отправки, могут быть вставлены строкой в исходный текст, так как фактический контекст процесса известен. Это сокращает исходный текст до простых операций присваивания, например, *msg (1): = msg;* для *Proc 1* на рис. 8. Таким образом, для типичной длины сообщения время выполнения операций приема/отправки, становится меньше, чем 1 мксек.

Сокращение копий сообщений до потенциальных случаев нарушения целостности данных:
Нарушение целостности данных может возникнуть только в двух определенных случаях обмена информацией между процессами. Во-первых, если получатель сообщения может быть прерван отправителем, см. рис. 8. Во-вторых, если отправитель сообщения может быть прерван получателем, а отправитель не писал данные в сообщение атомарно. Таким образом, необходимо использовать копии сообщений и обеспечивать операции копирования сообщений только в этих двух определенных случаях. Это приводит к значительному сокращению требуемых объемов памяти и времени выполнения для операций приема/отправки сообщений.

Объединение операций приема/отправки сообщений:
Дальнейший потенциал оптимизации времени выполнения видится в объединении операций приема/отправки сообщений. Так как план-список процессов в пределах задачи выполняется строго последовательно, то становится возможным объединить все операции приема сообщений в начале, а все операции отправки сообщений в конце план-списка. Если, например, сообщение будет получено всеми четырьмя процессами из план-списка см. рис. 9, то достаточно использовать только одну операцию приема сообщения и позволить всем процессам совместно использовать одну ту же копию сообщения. Это экономит три операции по приему сообщения и три места под копии сообщения.

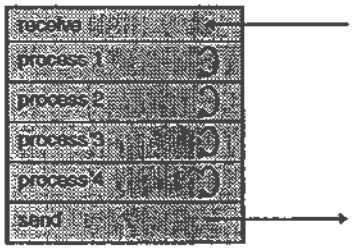

Рис. 9: Объединение операций приема/отправки сообщений

При использовании описанной оптимизации частота приема/отправки сообщений со 100000 сообщений в секунду может быть уменьшена до 26000 сообщений в секунду. Так как продолжительность выполнения этих операций - меньше чем 1 микросекунда, это приводит к загрузке процессора всего на 2.6% (имеется в виду только загрузка задачами информационного обмена между процессами). Что в итоге меньше на два порядка величины загрузки процессора, чем при других реализациях операций с сообщениями.

Таким образом, реализация механизма сообщений ERCOS, отвечая жестким требованиям высокой производительности для автомобильной электроники, гарантируя целостность данных. Это оказывает сильную поддержку объектной модели ERCOS, базирующейся на взаимодействии процессов посредством сообщений и допускающей повторное использование программного кода в многозадачных средах.

6. Взаимоисключающий доступ

Гарантия взаимоисключающего доступа к ресурсам - важное функциональное требование для систем реального времени. Например, для исключения несогласованности в работе системы, необходимо обеспечить взаимоисключающий доступ к файлу регистрации ошибок блока управления, чтобы гарантировать, что задача обновления файла регистрации ошибок не может быть вытеснена повторным обновлением файла регистрации ошибок.

6.1. Проблемы с семафорами

Обычно, требование взаимоисключающего доступа к ресурсам реализуется семафорами [Tan92]. Это решение не совсем подходит для требований автомобильной электроники по следующим причинам:
1. Семафоры могут блокировать вызывающую задачу, если к требуемому ресурсу уже обращаются. Это может привести к состоянию взаимной или оперативной блокировки, являющейся причиной серьезных программных ошибок.
2. Кроме того, семафоры являются причиной возникновения “проблемы смены приоритетов”. Это приводит к неограниченным задержкам, в попытке задачи получить доступ к ресурсу. Пример “смены приоритетов” показан на рис. 10.

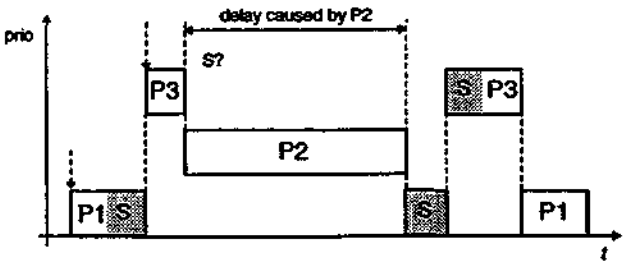

Рис. 10: Инверсия приоритетов

Процесс P1 с низким приоритетом получает монопольный доступ к семафору S. Процесс P1 вытесняется процессом P3, имеющим высокий приоритет, который также пытается обратиться к семафору S. Так как семафор S уже находится в пользовании процесса P1, то процесс P3 блокируется и вынужден ждать освобождения семафора S. Таким образом, становится возможным процессу P2, имеющему средний приоритет, выполняться с произвольной продолжительностью. Это явление называют сменой приоритетов, так как высокоприоритетный процесс P3, фактически вынужден ждать завершения среднеприоритетного процесса P2 (даже притом, что процесс P2 не

запрашивал доступ к семафору S).

3. Так как семафор может блокировать вызывающую программу, если требуемый ресурс занят, то операционная система должна обеспечить контекст (стек, регистры ...) для каждого процесса, чтобы сохранить его состояние в случае блокировки. Это приводит к недопустимому расходу ресурсов, особенно, для объектно-ориентированной системы с большим количеством процессов (>100).

6.2. Стек протокола максимального приоритета

Для преодоления проблем, описанных в предыдущем параграфе, операционная система ERCOS поддерживает оптимизированный вариант протокола наследования приоритета [SRL90], который называется стеком протокола максимального приоритет. Основная идея протокола наследования приоритета состоит в том, чтобы поднять приоритет низкоприоритетного процесса, который захватил критический ресурс, когда процесс с более высоким приоритетом пытается получить доступ к тому же самому ресурсу. Искусственное повышение приоритета низкоприоритетного процесса до уровня приоритета высокоприоритетного процесса, гарантирует отсутствие неограниченной инверсии приоритетов [SRL90]. Однако, этого всего еще недостаточно, чтобы гарантировать отсутствие взаимных блокировок и блокировок процессов. Протокол максимального приоритета [RSL88], является вариантом протокола наследования приоритета, где процесс наследует максимальный приоритет всех процессов, у которых есть потенциальный доступ к критическому ресурсу. Этот унаследованный приоритет процесса называется приоритетным потолком ресурса. Применение этого протокола гарантирует отсутствие взаимной блокировки, но все еще возможна блокировка выполнения процесса. Следующий протокол, являющийся вариантом протокола максимального приоритета, гарантирует отсутствие взаимной блокировки и отсутствие блокировки выполнения. Вместо того чтобы наследовать максимальный приоритет только в случае конфликта доступа к ресурсу максимальный приоритет может быть наследован непосредственно при доступе к ресурсу [Bak91].

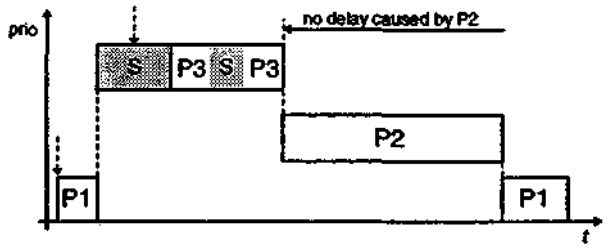

Рис. 11: Работа протокола максимального приоритета

Этот вариант протокола называется стеком протокола максимального приоритета. Таким образом, невозможно вытеснение процесса, который занимает ресурс, другим процессом, пытающимся получить доступ к этому ресурсу, см. рис. 11.

ERCOS поддерживает данный протокол, с помощью off-line инструментов, которые отвечают за статический анализ максимальных приоритетов для каждого ресурса. Кроме того, они обеспечивают следующую оптимизацию этого протокола. Статическим анализом определяется, имеет ли процесс самый высокий приоритет среди всех процессов, которые имеют доступ к определенному ресурсу. Тогда не нужно менять приоритет этого процесса при захвате и освобождение ресурсов. Другой путь оптимизации возможен, если время доступа к ресурсу или критической секции очень мало. Если продолжительность доступа составляет примерно такое же время, что потребуется для выполнения наследования приоритета и возврата к процессу, то наиболее эффективным решением будет запрет прерывания на время выполнения критической секции. Реализация этого варианта оптимизации является вставка строки в исходный текст off-line инструментами. Статический анализ исходного текста, кроме того, дает возможность обнаружения несанкционированного доступа к важнейшим ресурсам за пределами защищенного сегмента кода.

7. Таймеры

Лишь небольшое количество функций, в системах управления автомобильной электроники, вызывается внешними событиями. Основное большинство функций вызывается в определенные моменты времени программно. Эти синхронизированные по времени вызовы могут быть подразделены на вызовы с фиксированными периодами повторения, где периоды вызова сохраняются неизменными в течение всего операционного режима. И на вызовы с изменяемым периодом повторения. В связи с этим, ERCOS предоставляет два вида служб таймеров. Во-первых, *службу статического таймера* для синхронизированных по времени вызовов задач с фиксированным периодом повторения и, во-вторых, *службу динамического таймера* для синхронизированных по времени вызовов с изменяемым периодом повторения.

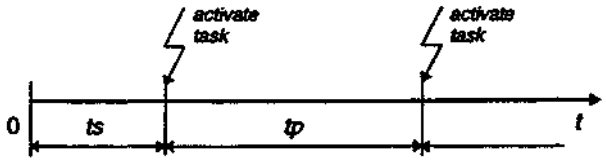

Рис. 12: Основная функция таймера

Между тем, обе службы таймеров функционирую

одинаково. Работа таймер описывается его периодом повторения *tp,* и стартовой задержкой *ts.* Стартовая задержка *ts* определяет задержку первого вызова задачи, а период повторения *tp* определяет дальнейшую синхронизацию вызовов. После истечения отсчета времени задача будет активизирована, см. рис. 12. Обе службы таймера обеспечивают достаточно хорошую разрешающую способность в несколько микросекунд.

7.1. Служба статического таймера

Служба статического таймера поддерживает синхронизированные по времени вызовы с фиксированными периодами повторения. Так как конечная информация по синхронизации вызовов для этого типа службы известна еще до времени выполнения, то возможно еще в off-line составить список периодов для данного набора вызовов. Соответствующая таблица создается для ERCOS off-line инструментами. При использовании этой off-line созданной таблицы, операционная система тратит значительно меньше усилий во время выполнения. Нет необходимости искать время следующей активации таймера и управлять структурами данных для таймеров. Операционная система должна только выбрать следующую активацию из отсортированной таблицы, что выполняется очень быстро. Таким образом, становится возможным поддерживать синхронизированные по времени вызовы задач с очень короткими периодами повторения, например две задачи - 1000 мксек и 800 мксек.

7.1. Служба динамического таймера

Вызов задач с изменяемым периодом повторения поддерживаются службой динамического таймера. Этот сервис таймера единственно учитывает задержку активации задачи, т.е. определяя стартовую задержку и не определяя период повторения. Дальнейшая возможность состоит в том, чтобы изменять период повторения во время пуска или остановки таймера отсчета периода, в зависимости от фактических условий во время выполнения. Для обеспечения необходимой гибкости, служба динамического таймера обслуживается операционной системой полностью on-line. Главное преимущество службы динамического таймера - своя гибкость, за которую приходиться платить большим временем выполнения и объемом памяти. Напротив, служба статического таймера обеспечивает очень высокую производительность, но меньшую гибкость. Поэтому ERCOS обеспечивает объединенную концепцию таймеров, где это возможно, чтобы выбрать самую соответствующую службу в соответствии с функциональным требованиям. Можно уменьшить время выполнения и требования к памяти, поддерживая статические и динамические таймеры.

8. Отказоустойчивость и обработка особых ситуаций

Надежность и отказоустойчивость - важные свойства для систем реального времени, так как система должна реагировать на изменение состояния среды с гарантируемым периодом ожидания. Такие гарантии, однако, могут быть даны только для определенного сценария пиковой загрузки системы. Сценарием пиковой загрузки называют определенную максимальную частоту поступления и некий шаблон событий, которые должны быть обработаны системой. Однако, как показывает опыт это предположение, сделанное о сценарии пиковой загрузки, часто не соответствует действительности. И для этого есть две причины: Во-первых, оценка пиковой загрузки была сделана неверно, и частота поступления событий оказалась выше, чем предполагалось. Во-вторых, ошибки в периферийных устройствах или в самой вычислительной системе приводят к непредвиденному "*потоку событий*". Операционная система обязана изящно обрабатывать подобные перегрузки. Система ERCOS обеспечивает ряд возможностей, позволяющих обрабатывать ситуации перегрузки и гарантирующих своевременную реакцию на событие. С точки зрения операционной системы ситуация перегрузки возможна, если число активаций задачи превышает ожидаемое для пиковой загрузки предположение.

8.1. Гарантируемое минимальное время между двумя последовательными активациями.

Для сценария пиковой загрузки может быть определен термин минимального времени между двумя активациями задачи. Индивидуально для каждой задачи этот параметр определяет минимальный интервал времени между двумя последовательными активациями, см. рис. 13.

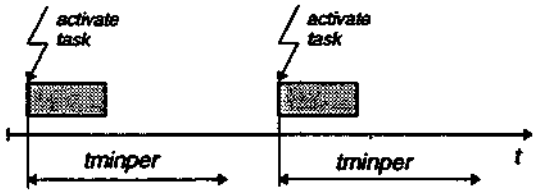

Рис. 13: Минимальный период между двумя активациями

В случае ошибок в периферии или ошибочного сценария пиковой загрузки, возможно, что период между двумя последовательными вызовами задачи станет меньше, чем был определен. Для предотвращения перегрузки системы, в подобной ситуации, ERCOS обеспечивает механизм, который отклоняет вызов задачи, следующий слишком рано за предыдущим вызовом этой задачи. Если задача активизирована программно, операционная система проверяет время между вызовами, начиная с

последнего вызова. Она отклоняет активацию задачи, если прошедшее время меньше, чем установленное минимальное время между двумя вызовами задачи. В случае задач, активизированных аппаратными прерываниями, используется другой подход. Операционная система запрещает, после запуска задачи, активизировавшее задачу прерывание на время минимального периода между двумя вызовами задачи. Таким образом, гарантируется наличие минимального времени между двумя последовательными активациями задачи. Точный анализ этого механизма отказоустойчивости дан в [Pol95b]. На рис. 14 показана обработка прерываний с использованием описанного механизма.

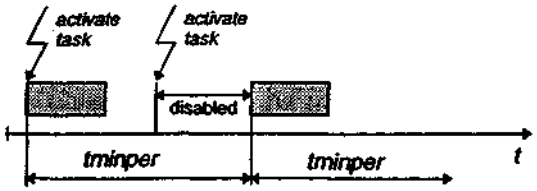

Рис. 14: Обработка прерываний

8.2. Ограничение числа одновременно готовых экземпляров задачи

В некоторых обстоятельствах, при программном вызове задач, необходимо иметь несколько активаций задач. Это приводит к нескольким одновременно готовым экземплярам задачи, которыми может управлять операционная система в порядке их поступления. Следовательно, возможно осуществить организацию очереди. Опять таки, по сценарию пиковой загрузки требуется гарантировать, что все активации задачи будут обработаны правильно операционной системой. Для этого ERCOS позволяет определить максимальное число одновременно готовых экземпляров задачи. Данный параметр удовлетворяет двум целям: Во-первых, во время конфигурации и анализа системы берутся все определения задачи, и создается структура данных, содержащая количество одновременно возможных экземпляров текущей задачи. Во-вторых, операционная система проверяет при каждом вызове задачи, будет ли максимальное число одновременно активных экземпляров превышен. В этом случае активация задачи будет отклонена. Использование данного механизма гарантирует работоспособность операционной системы в случай самого плохого сценария. Кроме того, операционная система устойчива к программным ошибкам во время выполнения, отклоняя чрезмерное количество активации задачи.

8.3. Контроль периода ожидания

Системы реального времени характеризует тот факт, что они должны ответить на события в операционной среде с конечным периодом ожидания. Этот период ожидания определяет, предоставлен ли результат своевременно, и будучи, таким образом, корректным или нет. Система ERCOS обеспечивает механизм, осуществляющий контроль периода ожидания. Он позволяет обнаружить поздние и соответственно некорректные ответы системы и дает возможность пользователю реагировать на эти ошибки. Механизм контроль периода ожидания предоставляет пользователю функции запуска контроля и проверки истечения начатого периода ожидания. Операционная система дополнительно контролирует период ожидания самостоятельно. Таким образом, нарушения периода ожидания будут обнаружены даже в случае, когда прикладная программа не использует функцию контроля периода ожидания. Кроме того, операционная система обеспечивает проверку целостности данных, обнаруживая искажения в данных внутренних структур службы контроля периода ожидания.

8.4. Обработка особых ситуаций

Кроме ошибок в синхронизации и вызовах задач, как упомянуто выше, существует широкое разнообразие ошибок, обнаруживаемых операционной системой во время выполнения. Например, ошибки переполнения стека или искажение данных в структурах. Кроме того, есть всевозможные ошибки, обнаруживаемые пользовательским программным обеспечением. Чтобы обработать эти ошибки в соответствии с требованиями к среде, операционная система обеспечивает механизм обработки особых ситуаций (исключений) [Cri89]. После возникновения ошибки операционной системой или пользователем будет сгенерированно исключение. В соответствии с прикладными требованиями, возможно, связать подпрограмму обработки особых ситуаций с исключением. Эта связь производится статически, во время конфигурирования и генерации системы, и позволяет встраивать написанные пользователем обработчики особых ситуаций в систему.

9. Заключение

В статье представлена концепция операционной системы ERCOS, разработанной специально для удовлетворения функциональных и эксплуатационных потребностей автомобильной электроники. Фирма Bosch использует ERCOS в качестве стандартной операционной системы для автомобильной электроники.

В главе 2 описываются общие цели и задачи для прикладного программного обеспечения в автомобильных блоках управления, которые в свою очередь влияют на проектирование операционных систем. Эти цели - повторное использование программного кода, модульность, эффективность, расширяемость и поддержка режима реального времени. Многие из этих целей соотносятся с

современными методами разработки программного обеспечения, и особенно ООП, и поэтому должны быть подробно рассмотрены для использования в процессе разработки операционных систем.

Глава 3 описывает проблемы с обычными реализациями объектов в системах реального времени и описывает объектную модель ERCOS, в которой преодолеваются эти проблемы. Показано, что реализация объектов на основе атрибутов (которые являются глобальными переменными) не подходит для достижения требуемых целей гибкости, надежности и повторного использования программного обеспечения в условиях многозадачности режима реального времени. Это связано с присущей для процессов (или функций, вызываемых этими процессами) параллельностью доступа к этим атрибутам. В частности, это приводит к проблеме целостности данных. Нарушение целостности данных происходит, когда процесс с высоким приоритетом изменяет данные, используемые процессом с низкий приоритетом, при выполнении последнего. Для этой проблемы, которая зависит от конфигурационных и системных аспектов, обычно применяют специальные решения, которые предотвращают повторное использование программного обеспечения. Поэтому, для достижения необходимой цели, важно гарантировать целостность данных неким общим механизмом. В ERCOS этих целей предусмотрен механизм сообщений. В основе объектной модели ERCOS - процессы, функции, сообщения и ресурсы, в качестве основных классов объектов. Функции соответствуют методам, сообщения заменяют атрибуты, гарантируя при этом целостность данных, процессы поддерживают реализацию автономно активных объектов, а ресурсы помогают гарантировать взаимоисключающий доступ к критическим секциям кода или ресурсам. Эта объектная модель подходит для параллельного выполнения процессов в многозадачной среде реального времени, поддерживая цели и задачи, указанные в главе 2.

Стратегия планирование ERCOS описана в главе 4. Для достижения высокой эффективности во времени выполнения, операционная система поддерживает сочетание статического и динамического планирования. Поддержка статического планирования значительно уменьшает требования к планировщику во время выполнения. Кроме того, можно использовать комбинации вытесняющего и кооперативные планирования, для минимизации требований к памяти.

Объекты - сообщения, для взаимодействия процессов, описаны в главе 5. ERCOS реализует сообщения состояния и предоставляет ряд новых методов оптимизации, для обеспечения очень высокую эффективность по отношению к времени выполнения и требованиям памяти. Эта концепция оптимизации позволяет систематическое применение сообщений и таким образом гарантирует целостность данных в общей конфигураций времени выполнения без потери эффективности.

Обработка взаимоисключающего доступа к важнейшим ресурсам, описана в главе 6. Многозадачные операционные системы реального времени, как правило, предоставляют семафоры для обеспечения взаимоисключающего доступа к ресурсам. У этого механизма, однако, есть серьезные недостатки, так как применение семафоров ведет к блокировкам, и могут быть причиной взаимной или оперативной блокировки. Семафоры также являются источником неограниченной инверсии приоритетов. Чтобы избежать этих проблем, ERCOS поддерживает стек протокола на основе максимального приоритета. Этот протокол гарантирует, отсутствие взаимной блокировки при выполнении, свобода от блокировок и ограничение задержки доступа к ресурсам.

Глава 7 описывает службы таймеров ERCOS. Для операционной системы реального времени необходимо эффективное обслуживание таймеров, поскольку многие задачи (функции) активируются синхронизированными по времени вызовами. Для достижения высокой эффективности операционная система поддерживает службы динамического и статического таймеров. Служба динамического таймера является очень гибкой. Служба статического таймера является наиболее эффективной, но не позволяет во время выполнения изменять статически запланированную таблицу активаций задач. В связи с этим можно придти к компромиссу в применении, между гибкостью и эффективностью.

К автомобильной электронике предъявляются очень серьезные требования в области надежности и безопасности применения. Для этого ERCOS обеспечивает механизмы поддержки отказоустойчивости и обработки особых ситуаций, описанные в главе 8. Важно обрабатывать перегрузки системы, вызванные неисправностями в периферии или программными ошибками, так как системе реального времени необходимо реагировать, на соответствующие события в среде, в пределах определенного времени ожидания. Операционная система обеспечивает механизмы контроля минимального времени между двумя последовательными активациями задачи. Кроме того, реализован механизм ограничения числа одновременно активных задач. Для обнаружения ошибок синхронизации, операционная система обеспечивает механизм контроля периода ожидания. Обработка остальных ошибок осуществляется механизмом обработки исключений, поддерживаемым операционной системой. В статье показан выбор механизмов операционной системы. Статические методы анализа исходного текста и оптимизация структуры делают ERCOS очень эффективной операционной системой для автомобильных приложений, не ставя под угрозу гибкость, надежность, расширяемость и поддержку многократного использования программного обеспечения. Добавляя

соответствующие службы ,операционная система с поддержкой статического планирования и сообщений, хорошо удовлетворяет требованиям для расширенной распределенной отказоустойчивой системе реального времени [KG94, MP96].

Список литературы

[Bak91] T.P. Baker. Stack-Based Scheduling of Realtime Processes. The Journal of Real-Time Systems. Nr. 3,1991, pp. 67-99.

[Cri89] F. Cristian. Exception handling. In *Dependability of Resilient Computers,* T. Anderson (Ed). Blackwell Scientific Publications, Oxford. 1989.

[CSR87] S. Cheng, J. Stankovic, and K. Ramamritham. Scheduling Algorithms for Hard Real-Time Systems—A Brief Survey. In *Tutorial of Hard Real-Time Systems.* J. Stankovic and K. Ramamritham (eds.), 1987, pp. 150-173.

[KI93] Integrated Systems, Inc. *pSOS System—System Concepts.* Release 2.0, PS2-000-O03, PSM2000-MAN, 6. Dec. 1993.

[JSM91] K. Jeffay, D.F. Stanat, and GU. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Proceedings of the Real-Time Systems Symposium.* San Antonio, Texas, 1991, pp. 129-139.

[KDK*89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, C. Senft and R. Zainlinger. The MARS approach. *IEEE Micro.* Vol. 9, Nr. 1, Feb. 1989, pp. 25-40.

[KG94] H. Kopetz and G. Grunsteidl. TTP—A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer.* Vol. 27, No. 1, Jan. 1994, pages 14-23.

[KRP^] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzales Harbour. A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Scheduling for Real-Time Systems. Kluwer Academic Publishers. 1993.

[LL73] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM,* Vol. 20, Nr. 1, Jan. 1973, pp. 46-61.

[Mey88] B. Meyer. Object-Oriented Software Construction. Prentice Hall Book Co., Inc. 1988.

[MP96] H.-J. Mathony and S. Poledna. Real-Time Software for In-Vehicle Communication. To appear in SAE International Congress, Michigan, USA. 1996.

[OSE95] OSEK (Open Systems and the Corresponding Interfaces for Automotive Electronics), Operating System. 1995.

[Pol95a] S. Poledna. Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism. Kluwer Academic Publishers. 1995.

[Pol95b] S. Poledna. Tolerating Sensor Timing Faults in Highly Responsive Hard Real-Time Systems. *IEEE Transactions on Computers.* Vol. 44, Nr. 2, Feb. 1995, pp 181-191.

[Rea86] JJF. Ready. VRTX: A Real-Tune Operating System for Embedded Microprocessor Applications. *IEEE Micro.* Vol. 4, Nr. 6, Jun. 1986, S. 8-17.

[RSL88] R. Rajkumar, L.Sha, and J. Lehoczky. Real-Time Synchronization Protocols for Multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium.* 1988, pp. 259—269.

[SRL90] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocol: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers.* Vol. 39, Nr. 9, 1990, pp. 1175-1185.

[SSN95] J. Stankovic, M. Spuri, and M. Di Natale. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer.* Vol. 28, Nr. 6, Jun. 1995, pp. 16-25.

[Str91] B. Stroustrup. The C-H- Programming Language. Second Edition. Addison-Wesley. 1991.

[Tan92] A.S. Tanenbaum. Modern Operating Systems. Prentice-Hall. 1992.

[TSK89] H. Takeyama, T. Shimizu, and M. Kobayakawa. Design Concept and Implementation of uITRON Specification for the H8/500 Series. In *Proceedings of the 34th IEEE Computer Society International Conference—COMPCON.* San Francisco, CA, 1989, pp. 48-53.