**ѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕѕ**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту на тему:

|  |
| --- |
|  |
| Разработка системы реального времени в виде планировщика |
| исполнения заданий. |

**Москва 2004**

**Реферат.**

Проделана работа по проектированию системы реального времени. Созданная система содержит два основных компонента: планировщик задач реального времени и прикладное приложение – протокол A.415 ARINC. Работа содержит 39 страниц, 14 диаграмм, 3 таблицы и 2 рисунка. Использовано 13 ссылок на техническую литературу.

Раздел 1. Описываются отличия систем реального времени от обычных систем (разделения времени). Приведены характерные особенности управления задачами в подобных системах. Проведены классификация и анализ требований, предъявляемых к современным СРВ. Даны примеры систем данного класса (представленных в России). Рассмотрена необходимость использования специальной методологии разработки программного обеспечения.

Раздел 2. Заданы определения, используемые в данной работе. Рассмотрена принципиальная структура СРВ. Приведена классификация подходов к планированию и обзор методов его реализации. Рассмотрена объектно-ориентированная методология разработки программного обеспечения.

Раздел 3. Описана реализация планировщика задач реального времени: достигаемые возможности, используемые алгоритмы, общая схема функционирования. Приведена документация по приложению-протоколу, составленная в соответствии с требованиями методологии Real.

**Содержание.**

Реферат. 2

Содержание. 3

Введение. 5

1. Обзор требований проблемной области. 7

1.1. Особенности систем реального времени. 7

1.1.1. Ограниченное время ответа. 7

1.1.2. Статическая основа проектирования. 7

1.1.3. Портирование. 8

1.1.4. Встроенные системы реального времени. 8

1.1.5. Вывод. 9

1.2. Особенности управления задачами. 9

1.2.1. Управление временем. 9

1.2.2. Управление памятью. 9

1.2.3. Управление доступом (синхронизация). 9

1.2.4. Вывод. 10

1.3. Классификация систем реального времени. 10

1.3.1. Классификация по структурным характеристикам. 10

1.3.1.1. Исполнительные системы реального времени. 10

1.3.1.2. Ядра реального времени 11

1.3.1.3. UNIX'ы реального времени 11

1.3.2. Классификация по программной среде. 12

1.3.2.1. Программирование на уровне микропроцессоров. 12

1.3.2.2. Минимальное ядро системы реального времени. 12

1.3.2.3. Ядро системы реального времени и инструментальная среда. 12

1.3.2.4. ОС с полным сервисом. 12

1.3.3. Технические характеристики ОС РВ. 12

1.3.3.1. Время реакции системы. 12

1.3.3.2. Время переключения контекста. 13

1.3.3.3. Размеры системы. 13

1.3.3.4. Возможность исполнения системы из ПЗУ (ROM). 14

1.3.4. Вывод. 14

1.4. Современные представители рынка ОС РВ в России. 14

1.4.1. LynxOS® 4.x фирмы LinuxWorks, Inc. 14

1.4.1.1. Основные свойства LynxOS: 14

1.4.1.2. Поддержка приложений жёсткого реального времени. 15

1.4.2. OS-9/Hawk фирмы Microware Systems. 15

1.4.2.1. Основные свойства OS-9/Hawk. 15

1.4.2.2. Поддержка приложений жёсткого реального времени. 16

1.4.3. VxWorks фирмы Wind River Systems. 16

1.4.3.1. Основные свойства VxWorks. 16

1.4.4. QNX4 фирмы ОРАКУЛ. 17

1.4.4.1. Основные свойства QNX4. 17

1.4.4.2. Поддержка приложений жёсткого реального времени. 17

1.4.5. Вывод. 17

1.5. Методология разработки программного обеспечения. 17

1.5.1. История развития. 18

1.5.2. Разработка программного обеспечения систем реального времени 18

1.5.3. Вывод. 19

1.6. Постановка задачи курсового проекта. 19

2. Модели и методы предметной области. 21

2.1. Определения. 21

2.2. Принципиальная структура. 22

2.2.1. Среда исполнения. 22

2.2.2. Ядро систем реального времени. 22

2.2.2.1. Синхронизация ресурсов. 23

2.2.2.2. Межзадачный обмен. 23

2.2.2.3. Разделение данных. 23

2.2.2.4. Обработка запросов внешних устройств. 23

2.2.2.5. Обработка особых ситуаций. 23

2.2.3. Пикоядро. 24

2.3. Методы управления задачами в ОС РВ. 24

2.3.1. Классификация подходов. 24

2.3.1.1. Статическое планирование. 24

2.3.1.2. Динамическое планирование. 24

2.3.1.3. Планирование, основанное на времени. 25

2.3.1.4. Планирование апериодических задач 25

2.3.1.5. Планирование, управляемое приоритетами. 25

2.3.2. Обзор методов. 26

2.3.2.1. Rate-monotonic (RM). 26

2.3.2.2. Deadline Monotonic (DM). 26

2.3.2.3. Планирование апериодических задач. 27

2.3.2.4. EDF. 27

2.3.2.5. Сервер, допускающий задержку (DS) и Алгоритм обмена приоритетами (PE). 28

2.4. Методология разработки программного обеспечения. 28

2.4.1. Основы методологии Real. 28

2.4.2. Модель требований. 29

2.4.3. Динамическая модель. 29

2.4.4. Статическая модель. 30

3. Реализация прототипа системы реального времени. 31

3.1. Жизненный цикл разработки. 31

3.2. Планировщик заданий. 31

3.2.1. Выбор алгоритма планирования. 31

3.2.1.1. Виды требований РВ, поддерживаемые планировщиком. 31

3.2.1.2. Используемые алгоритмы. 32

3.2.2. Описание функционирования приложения. 33

3.2.2.1. Подготовка к запуску планировщика. 33

3.2.2.2. Работа. 33

3.2.2.3. Управление задачами. 34

3.3. Реализация протокола ARINC A.415 на основе разработанного модуля СРВ. 34

3.3.1. Модель требований к системе. 34

3.3.1.1. Описательная модель. 34

3.3.1.2. Модель случаев использования. 35

3.3.1.3. Функциональная модель. 35

3.3.2. Динамическая модель. 35

3.3.2.1. Модель объектов. 35

3.3.2.2. Модель взаимодействий. 35

3.3.2.3. Поведенческая модель. 36

3.3.3. Статическая модель. 37

3.3.3.1. Модель классов. 37

Заключение. 39

Литература. 40

Приложение 41

**Введение.**

Новый этап научно-технической революции был обусловлен повсеместным распространением вычислительной техники. Сейчас уже трудно найти вид деятельности, который тем или иным способом не поддерживался бы не просто автоматизированными, но и компьютеризированными устройствами. Такая организация жизнедеятельности позволяет не только выполнять заранее заданные алгоритмы управления производством, но и вносить в него элементы автоматизации интеллектуальной деятельности, элементы искусственного интеллекта. Использование таких технологий в жизненно важных отраслях, таких как авиация, банковское дело и других, требующих жёстко заданных требований к принятию решений, накладываемых на время, точность и безопасность деятельности данных систем, обуславливает необходимость создания особо надежных их видов – систем реального времени.

Управление процессом предоставления ресурсов системы задачам, нитям, процедурам обработки прерываний и т.д. является одной из основных функций любой операционной системы и осуществляется при помощи такого механизма, как планирование. Данный механизм обеспечивает системе возможность параллельного выполнения нескольких задач. В системах реального времени планирование должно также гарантировать предсказуемое поведение, безопасность, возможность длительной, безотказной работы, выполнение задач к поставленному сроку. От метода планирования во многом зависит успешная работа системы в целом.

С другой стороны, увеличение объемов производства и разнообразия средств микропроцессорной техники, расширение сфер их применения приводит к необходимости разработок различных операционных систем реального времени - от компактных, рассчитанных на обслуживание одночиповых микро-контроллеров, до мощных сетевых систем. Путь к удовлетворению требований высокой эффективности и надежности этих систем лежит через повышение ясности и стройности их логической организации.

Это обстоятельство выдвигает актуальные задачи разработки рационально организованных базовых структур, которые представляли бы в обобщенном виде ключевые принципы организации вариантов операционных систем, ориентированных на достижение того или иного типа эффективности. Для этой цели выдвигаются различные методологии разработки соответствующих систем. Особенную актуальность приобрели объектно-ориентированные методологии, опирающаяся на выгоды разработки при помощи объектных языков высокого уровня (в частности, С++).

В данной работе необходимо будет провести анализ предметной области ОС РВ. В виде фокус-группы логично было бы выбрать встраиваемые системы реального времени, предлагаемые в данный момент на рынке программного обеспечения России, сведения по которым размещены в сети Internet. Анализ проводится по результатам пресс-релизов подобных систем, в которых подчёркнуты опции, являющиеся наиболее важными для современных потребителей. Данное исследование позволит установить требования к системам реального времени, востребованные разработчиками в настоящее время, и общие методики удовлетворения этих требований.

В связи с обширностью проблемной области имеет смысл дать основополагающие определения и развёрнутые толкования отдельных терминов, имеющих особую важность. В этой связи будет рассмотрена принципиальная структура систем реального времени и выделена наиболее распространённая и общепризнанная в данный момент.

В данной работе будут рассмотрены подходы к задаче выбора приемлемого алгоритма планирования на основе сведений об алгоритмах, предполагаемой модели задач и структурных характеристик будущей системы. Предполагается выделить алгоритм планирования, ориентированный на разработку программного обеспечения систем контроля реального времени, и использовать его при создании прототипа модуля-диспетчера для заданий реального времени, который будет подключаться к программе пользователя. Данный модуль будет предоставлять интерфейс для формирования заданий с определёнными требованиями ко времени выполнения.

На основе спроектированного планировщика с использованием специальной методологии можно будет реализовывать прикладные приложения реального времени. В частности, будет реализован протокол A.415 ARINC, используемый во встроенных системах реального времени самолётов ведущих авиаперевозчиков. Это протокол опроса бортовых устройств, позволяющий в заранее обозначенный промежуток времени получить от них информацию и сигнализировать о неисправности в оборудовании. Такое приложение в наибольшей степени подходит как для анализа прототипа создаваемой СРВ, так и для используемой методологии.

При проектировании реализации протокола основной акцент планируется сделать на принципах его функционирования, соответствии заявленным требованиям и достигаемом уровне надёжности. Аппаратные требования, предъявляемые к используемому оборудованию, и, в целом, проблемы портирования рассмотрены не будут. В дальнейшем при претворении проекта в жизнь возможен анализ этих параметров для оптимизации вычислений в наиболее ресурсоёмких точках работы планировщика.

код

тестирование

верификация

архитектура

валидация

планировщик

требования/функции

приложение

системные требования

функциональные требования

архитектура

кодирование

интеграция

тестирование/верификация

Диаграмма 1. Этапы жизненного цикла разработки.

1. **Обзор требований проблемной области.**
	1. Особенности систем реального времени.

Для начала стоит дать определение операционных систем реального времени. Оно взято из [13]. Данное определение не является классическим, однако обладает тем преимуществом, что позволяется в общих чертах представить себе отличия ОС, рассматриваемых в данной работе от других аналогичных программ.

Операционные системы реального времени (ОС РВ) — управляющее ПО особого типа, часто используемое для организации работы встроенных компьютерных приложений, для которых характерны ограниченность ресурсов памяти, невысокая производительность, а также требования гарантированного времени отклика, высокого уровня готовности и наличия средств автомониторинга.

А теперь рассмотрим упомянутое в определении более подробно.

* + 1. Ограниченное время ответа.

По сути, система реального времени - это аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий. Это означает, что:

* Она должна успеть отреагировать на событие, произошедшее на объекте, в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается для СРВ ошибкой.
* Система должна успевать реагировать на одновременно происходящие события. Даже если два или больше внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий.

По последствиям выхода за пределы интервала СРВ делятся на мягкие и жёсткие.

Системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях, так как:

* результаты могут оказаться бесполезны в случае опоздания;
* может произойти катастрофа в случае задержки реакции;
* стоимость опоздания может оказаться бесконечно велика.

Системы мягкого реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличинию стоимости результатов и снижению производительности системы в целом.

Основное отличие между системами жесткого и мягкого реального времени можно выразить так: система жесткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени - не должна опаздывать с реакцией на событие.

В таблице 3 приведены времена отклика для нескольких ОС РВ.

* + 1. Статическая основа проектирования.

Кроме того, применение операционных систем реального времени всегда конкретно. Если ОС общего назначения обычно воспринимается пользователями (не разработчиками) как уже готовый набор приложений, то операционная система реального времени служит только инструментом для создания конкретного аппаратно-программного комплекса реального времени.

Для большинства СРВ предполагается, что основная часть обрабатываемых данных априорно известна. Поэтому наиболее широкий класс пользователей операционных систем реального времени - разработчики комплексов реального времени, люди проектирующие системы управления и сбора данных. Проектируя и разрабатывая конкретную систему реального времени, программист всегда знает точно какие события могут произойти на объекте, знает критические сроки обслуживания каждого из этих событий.

* + 1. Портирование.

Управление прокатными станами, роботами, движение на автомагистралях, контроль за состоянием окружающей среды, управление атомными и космическими станциями и многое другое - область задач реального времени. Для различных областей применения ОС РВ существуют разные аппаратные платформы и для каждой необходимо портирование, т.е процесс «состыковки» программной части ОС и её аппаратного обеспечения.

При выборе аппаратной платформы для систем реального времени основополагающими моментами являются жесткие требования к временным характеристикам и гибкости системы. Требования к аппаратному обеспечению в настоящее время довольно чётко определены. Большинство проектов реального времени осуществляется в рамках архитектурных решений магистрально-модульных систем (ММС).

Однако, как бы ни была важна сама ОС РВ, сейчас в условиях доступности совместимых аппаратных средств основное внимание уделяется разработке и отладке прикладного программного обеспечения, чья доля в затратах на разработку систем реального времени составляет до 70%.

По среде в которой функционируют системы их можно разделить на классы: программирование на уровне микроконтроллеров, минимальное ядро СРВ, ядро СРВ и инструментальная среда, ОС с полным сервисом.

* + 1. Встроенные системы реального времени.

Следующее отличие - применение операционной системы реального времени всегда связано с аппаратурой, с объектом, с событиями, происходящими на объекте. Система реального времени, как аппаратно-программный комплекс, включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. Операционная система реального времени ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям (по сравнению с ОС общего назначения) в структуре системы, в функциях ядра, в построении системы ввода-вывода. Операционная система реального времени может быть похожа по пользовательскому интерфейсу на ОС общего назначения (к этому, кстати, стремятся почти все производители операционных системах реального времени), однако устроена она совершенно иначе – более подробно об этом в пункте 2.2.

В последнее время высокопроизводительные микропроцессоры, а с ними и операционные системы реального времени, все чаще используются в так называемых "глубоко встроенных" (deeply embedded) применениях. К таким компьютерным системам предъявляются два основных требования: малые габариты и низкая стоимость. Поэтому глубоко встроенные микропроцессорные системы ставят две проблемы на пути применения серийных ОС РВ: небольшие объемы используемой памяти и отсутствие "лишних" интерфейсов, по которым можно было бы связать целевую и инструментальную машины на этапе разработки встроенного ПО.

По структурным характеристикам программно-аппаратные комплексы можно разделить на классы: исполнительные системы реального времени, ядра реального времени, Unix’ы реального времени.

* + 1. Вывод.

Были рассмотрены центральные отличия систем реального времени от систем разделения времени. В основе этих отличий лежит главное требование к подобным системам – предсказуемость. Пользователи СРВ должны быть заранее уверены в том, что отклик на внешнее воздействие будет получен в обозначенный интервал времени. Это влечет за собой необходимость представлять себе какие объёмы данных несут в себе внешние воздействия и какими аппаратными возможностями по обработке этих данных располагает система. Вполне логично, что системы реального времени не ориентированы на взаимодействие с человеком, а предполагают самостоятельную работу в критичных точках инженерных систем.

* 1. Особенности управления задачами.
		1. Управление временем.

Одним из основных свойств операционных систем реального времени является их способность изолировать друг от друга приложения, поэтому если в программе возникает сбой или выполняются какие-то нелегальные операции, ОС может быстро блокировать программу, инициировать восстановление и защиту других программ либо самой системы от серий вредоносных команд.

Если не выполняется обработка критических ситуаций либо она происходит недостаточно быстро, система жесткого реального времени прерывает операцию и блокирует ее, чтобы не пострадала надежность и готовность остальной части системы. Системы мягкого реального времени более «снисходительны» и «терпят» определенные, некритичные ошибки.

Особую важность приобретают такие инструменты как средства работы с таймерами, необходимые для систем с жестким временным регламентом. Развитость этих средств - необходимый атрибут операционных систем реального времени. Они, как правило, позволяют:

* измерять и задавать различные промежутки времени (от 1 мкс и выше),
* генерировать прерывания по истечении временных интервалов,
* создавать разовые и циклические будильники.
	+ 1. Управление памятью.

Система реального времени должна уметь управлять памятью в зависимости от критичности задач. Для устойчивой работы процессов требуются механизмы выделения памяти при их порождении, использования памяти при жизнедеятельности и освобождения.

ОС позволяет программистам изолировать совместно используемые библиотеки, данные и системное программное обеспечение, а также приложения. Та же самая защита предотвращает переполнение стеков памяти, вызываемое действиями любых программ.

* + 1. Управление доступом (синхронизация).

При одновременной работе нескольких процессов в многозадачной системе реального времени операционная система должна обеспечить устойчивый механизм для обмена информацией между запущенными процессами. Связь между процессами (Interprocess communication, сокращенно IPC) является ключом к разработке приложений как совокупности процессов, в которых каждый процесс выполняет отведенную ему часть общей задачи.

Для операционных систем реального времени характерна развитость IPC-механизмов. К таким механизмам относятся: семафоры, события, сигналы, средства для работы с разделяемой памятью, каналы данных (pipes), очереди сообщений. Многие из подобных механизмов используются и в ОС общего назначения, но их реализация в операционных системах реального времени имеет свои особенности - время исполнения системных вызовов почти не зависит от состояния системы, и в каждой операционной системе реального времени есть по крайней мере один быстрый механизм передачи данных от процесса к процессу.

* + 1. Вывод.

Так же как сами системы реального времени существенно отличаются от обычных ОС, так и способы выполнения задач в них имеют свою специфику. Работа по управлению их выполнением превращается в сложную инженерную задачу, которая включает в себя создание алгоритмов разделения ресурсов системы, планирования их независимого выделения и освобождения для задач системы.

* 1. Классификация систем реального времени.

Количество операционных систем реального времени, несмотря на их специфику, очень велико. В обзоре журнала "Real-Time Magazine" ещё за март 97 года было упомянуто около шестидесяти систем. За прошедшие годы этих систем стало ещё больше. Если же добавить к их числу некоммерческие операционные системы реального времени, то мы получим вполне солидное число, отражающее заинтересованность современного общества в подобных системах. Однако сама специфика применения операционных систем реального времени требует гарантий надежности, причем гарантий в том числе и юридических - этим, видимо, можно объяснить тот факт, что среди некоммерческих систем реального времени нет сколько-нибудь популярных.

На рис. 5 дано компактное представление классификации систем по трём различным признакам: класс (отсутствие РВ, мягкое РВ, жесткое РВ), сложность (одноадресное пространство, многоадресное/защищенное), стандартизация (частное решение, подмножество POSIX, только POSIX, UNIX и POSIX).

* + 1. Классификация по структурным характеристикам.

В мире операционных систем реального времени, как впрочем и в любой другой динамично развивающейся отрасли, в которой ещё нет установившейся достаточно строгой теории, существует несколько разнообразных подходов к построению подобных систем.

* + - 1. Исполнительные системы реального времени.

Признаки систем этого типа - различные платформы для систем разработки и исполнения. Приложение реального времени разрабатывается на host-компьютере (компьютере системы разработки), затем компонуется с ядром и загружается в целевую систему для исполнения. Как правило, приложение реального времени - это одна задача и параллелизм здесь достигается с помощью нитей (threads).

Системы этого типа обладают рядом достоинств, среди которых главное - скорость и реактивность системы. Главная причина высокой реактивности систем этого типа - наличие только нитей(потоков) и, следовательно, маленькое время переключения контекста между ними ( в отличие от процессов).

С этим главным достоинством связан и ряд недостатков: зависание всей системы при зависании нити, проблемы с динамической подгрузкой новых приложений.

Кроме того, системы разработки для продуктов этого класса традиционно дороги (порядка $20000). Хотя, надо отметить, что качество и функциональность систем разработки в этом классе традиционно хороши, так как они были изначально кроссовыми.

Наиболее ярким представителем систем этого класса является операционная система VxWorks. Область применения - компактные системы реального времени с хорошими временами реакций.

* + - 1. Ядра реального времени

В этот класс входят системы с монолитным ядром, где и содержится реализация всех механизмов реального времени этих операционных систем. Исторически системы этого типа были хорошо спроектированы. В отличие от систем других классов, которые появлялись как временные компромиссы и затем "наращивали мускулы" благодаря первым удачным реализациям (исполнительные системы реального времени и UNIX'ы реального времени), разработчики систем этого класса имели время для разработки систем именно реального времени и не были изначально ограничены в выборе средств (например фирма "Microware" имела в своем распоряжении три года для разработки первого варианта OS-9).

Одна из их особенностей - высокая степень масштабируемости. На базе этих ОС можно построить как компактные системы реального времени, так и большие системы серверного класса.

Как правило, ядра реального времени имеют два типа систем разработки - кроссовую и резидентную.

Системы этого класса, как правило, модульны, хорошо структурированы, имеют наиболее развитый набор специфических механизмов реального времени, компактны и предсказуемы. Наиболее популярные системы этого класса: OS9, QNX.

* + - 1. UNIX'ы реального времени

Исторически системы реального времени создавались в эпоху расцвета и бума UNIX'а и поэтому многие из них содержат те или иные заимствования из этой красивой концепции операционный системы (пользовательский интерфейс, концепция процессов и т.д.).

Часть разработчиков операционных систем реального времени попыталась просто переписать ядро UNIX, сохранив при этом интерфейс пользовательских процессов с системой, насколько это было возможно. Реализация этой идеи не была слишком сложной, поскольку не было препятствия в доступе к исходным текстам ядра, а результат оказался замечательным. Получили и реальное время, и сразу весь набор пользовательских приложений - компиляторы, пакеты, различные инструментальные системы.

В этом смысле создателям систем первых двух классов пришлось потрудиться не только при создании ядра реального времени, но и продвинутых систем разработки.

Однако Unix'ы реального времени не избавлены от следующих недостатков: системы реального времени получаются достаточно большими и реактивность их ниже, чем реактивность систем первых двух классов.

Наиболее популярным представителем систем этого класса является операционная система реального времени LynxOS.

* + 1. Классификация по программной среде.

Становится очевидным то, что задачи реального времени необходимо реализовывать в рамках специфической системной программной среды. В соответствии с [12] системы реального времени можно разделить на 4 класса.

* + - 1. Программирование на уровне микропроцессоров.

В данном случае программы для программируемых микропроцессоров, встраиваемых в различные устройства, очень небольшие и обычно написаны на языке низкого уровня типа ассемблера или PLM. Внутрисхемные эмуляторы пригодны для отладки, но высокоуровневые средства разработки и отладки программ не применимы. Операционная среда обычно недоступна.

* + - 1. Минимальное ядро системы реального времени.

На более высоком уровне находятся системы реального времени, обеспечивающие минимальную среду исполнения. Предусмотрены лишь основные функции, а управление памятью и диспетчер часто недоступны. Ядро представляет собой набор программ, выполняющих типичные, необходимые для встроенных систем низкого уровня функции, такие, как операции с плавающей запятой и минимальный сервис ввода/вывода. Прикладная программа разрабатывается в инструментальной среде, а выполняется, как правило, на встроенных системах.

* + - 1. Ядро системы реального времени и инструментальная среда.

Этот класс систем обладает многими чертами ОС с полным сервисом. Разработка ведется в инструментальной среде, а исполнение - на целевых системах. Этот тип систем обеспечивает гораздо более высокий уровень сервиса для разработчика прикладной программы. Сюда включены такие средства, как дистанционный символьный отладчик, протокол ошибок и другие средства. Часто доступно параллельное выполнение программ.

* + - 1. ОС с полным сервисом.

Такие ОС могут быть применены для любых приложений реального времени. Разработка и исполнение прикладных программ ведутся в рамках одной и той же системы.

Системы 2 и 3 классов принято называть системами "жесткого" реального времени, а 4 класса - "мягкого". Очевидно, это можно объяснить тем, что в первом случае к системе предъявляются более жесткие требования по времени реакции и необходимому объему памяти, чем во втором. Как мы видим, среда разработки и среда исполнения в системах реального времени могут быть разделены, а требования, предъявляемые к ним, весьма различны. Рассмотрим их более подробно.

* + 1. Технические характеристики ОС РВ.
			1. Время реакции системы.

Почти все производители систем реального времени приводят такой параметр, как время реакции системы на прерывание (interrupt latency).

В самом деле, если главным для системы реального времени является ее способность вовремя отреагировать на внешние события, то такой параметр, как время реакции системы является ключевым. Однако в настоящий момент нет, к сожалению, общепринятых методологий измерения этого параметра, поэтому он является полем битвы маркетинговых служб производителей систем реального времени. Есть надежда, что в скором времени положение изменится, так как уже стартовал проект сравнения операционных системах реального времени, который включает в себя в том числе и разработку методологии тестирования.

События, происходящие на объекте, регистрируются датчиками, данные с датчиков передаются в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события. Интервал времени - от события на объекте и до выполнения первой инструкции в программе обработки этого события и является временем реакции системы на события, и, проектируя систему реального времени, разработчики должны уметь вычислять этот интервал.

Время выполнения цепочки действий - от события на объекте до генерации прерывания - никак не зависит от операционных систем реального времени и целиком определяется аппаратурой, а вот интервал времени - от возникновения запроса на прерывание и до выполнения первой инструкции обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Причем это время нужно уметь оценивать в худшей для системы ситуации, то есть в предположении, что процессор загружен, что в это время могут происходить другие прерывания, что система может выполнять какие-то действия, блокирующие прерывания.

Неплохим основанием для оценки времен реакции системы могут служить результаты тестирования с подробным описанием архитектуры целевой системы, в которой проводились измерения, средств измерения и точным указанием, какие промежутки времени измерялись. Некоторые производители операционных систем реального времени результаты такого тестирования предоставляют. Их не увидишь в рекламных проспектах, но можно отыскать на WEB-страницах, в документах технической поддержки, в публикациях фирм, проводящих независимое тестирование.

Время реакции на прерывание, характерное для некоторых операционных систем реального времени, представлено на диаграмме 6.

* + - 1. Время переключения контекста.

В операционные системы реального времени заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все операционные системы реального времени являются многозадачными (многопроцессными, многонитиевыми). Для того чтобы уметь оценивать накладные расходы системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити), то есть время переключения контекста (диаграмма 7).

* + - 1. Размеры системы.

Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.). Хотя, надо признать, что с течением времени значение этого параметра уменьшается, тем не менее он остается важным и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики.

Примеры: размер ядра операционной системы реального времени OS-9 на микропроцессорах МС68xxx - 22 KB, VxWorks - 16 KB.

* + - 1. Возможность исполнения системы из ПЗУ (ROM).

Это свойство операционных систем реального времени - одно из базовых. Оно позволяет создавать компактные встроенные СРВ повышенной надёжности, с ограниченным энергопотреблением, без внешних накопителей.

* + 1. Вывод.

На столь широком поле деятельности как системы реального времени вполне закономерным оказалось возникновение множества подходов к их созданию. В основном они отличаются структурой создаваемой системы и аппаратной платформой, на которой ей предполагается функционировать. В настоящее время используются четыре основных параметра, которые могут характеризовать правильность выбранного подхода.

* 1. Современные представители рынка ОС РВ в России.

Среди коммерческих систем реального времени можно выделить группу ведущих систем - по объемам продаж и по популярности. Эти системы: VxWorks, OS9, LynxOS, QNX, pSOS, VRTX. В таблице 8 даны сведения о существующих в настоящее время СРВ и их характерных особенностях. В таблице 4 даны основные характеристики некоторых систем.

Четыре из перечисленных систем будут рассмотрены далее подробно. В системе, которая будет создана в рамках данной работы, не предусмотрены функции работы с работы с локальными или глобальными сетями. Поэтому в числе сравнительных параметров не были упомянуты эти возможности, которые являются немаловажной частью современных ОС РВ.

* + 1. LynxOS® 4.x фирмы LinuxWorks, Inc.

Предназначена для разработки ПО встроенных систем, работающих в режиме жёсткого реального времени, производителями комплектного оборудования (OEM) и телекоммуникационного оборудования (TEM), в частности, изготовителями бортовых систем военного применения.

* + - 1. Основные свойства LynxOS:
* Поддерживает многозадачные и многопотоковые приложения.
* LynxOS обеспечивает совместимость с Linux на уровне ABI (Application Binary Interface), уровне форматов объектных файлов, вызовов API, динамически подключаемых библиотек (DLL), компоновки и загрузки на этапе выполнения. Это свойство LynxOS является уникальным для систем реального времени и очень полезным для пользователей (например в случае отсутствия исходных текстов). Система работает так же с Unix и Java.
* Полностью поддерживается стандарт POSIX.1003-1, а также подразделы POSIX.1003-1b и POSIX.1003-1c, определяющие расширения реального времени и работы с нитями (потоками).
* Многоплатформенность. Поддерживает множество аппаратных архитектур (IA-32, PowerPC, MIPS, ARM, XScale, IBM) для оборудования различных фирм производителей.
* Разработка может осуществляться как на самой целевой системе (self-hosted), так и на инструментальном компьютере (host).
* Является ОС для ответственных приложений. Имеет всё необходимое для создания современных систем, обладающих свойствами "горячей замены" / "высокой доступности" (Hot Swap, High Availability), и устройств с высоким коэффициентом резервирования.
* LynxOS-178 - это версия LynxOS, сертифицированная в соответствии со стандартом DO-178. Это означает полное соответствие с точки зрения надежности строгим требованиям для мобильных систем военного и аэрокосмического применения. Кроме того, LynxOS-178 имеет сертифицированный стек TCP/IP для ответственных приложений в области авионики, медицины, атомной промышленности и связи.
* Большое количество средств разработок как в рамках самой LynxOS, так и host-систем (Linux, Windows, Solaris).
	+ - 1. Поддержка приложений жёсткого реального времени.
* количество задач: неограниченно;
* количество приоритетов: 256;
* диспетчеризация задач: вытеснение по приоритетам. 4 алгоритма диспетчеризации (FIFO, Priority Quantum, Round Robin, невытесняемый);
* детерминированное время переключения контекста благодаря эффективному алгоритму диспетчеризации реального времени;
* средства межзадачных взаимодействий как в стандарте POSIX (семафоры, разделяемая память, сокеты, сигналы, каналы, мьютексы, условные переменные), так и в терминах Unix SystemV (очереди сообщений, семафоры, разделяемая память);
* поддержка таймеров реального времени и часов POSIX;
* конфигурирование квантов времени для различных уровней приоритетов и для разрешения значения единицы (tick) таймера;
* выполнение задач в защищенном режиме, полная поддержка MMU (Memory Management Unit).
	+ 1. OS-9/Hawk фирмы Microware Systems.

Многозадачная, многопользовательская операционная система для встраиваемых приложений, работающих в режиме реального времени. Для производителей продуктов в таких областях, как мобильные телекоммуникационные устройства, встраиваемые терминалы доступа в Интернет, интерактивные цифровые телевизионные приставки.

* + - 1. Основные свойства OS-9/Hawk.
* Переносимая версия OS-9 позволяет применять в проекте наиболее подходящие микропроцессорные устройства (Motorola ColdFire; Motorola M-CORE; Intel Pentium; Intel StrongARM; PowerPC; ARM; Hitachi SuperH; MIPS; MicroSPARC).
* Система ввода-вывода ОС поддерживает различные форматы устройств массовой памяти и основных интерфейсов периферийных устройств: Raw, MS-DOS, True FFS, CardSoft PCMCIA, USB, IrDA.
* В среде OS-9 пользователь может выбирать несколько программных коммуникационных платформ: mwSoftStax (Microware), Harris & Jeffries, Trillium, - что ранее было исключительно прерогативой специализированных ОС.
* В инструментальный пакет Hawk встроена библиотека Tools.h из библиотеки Rogue Wave C++ Classes Lib.
* Hawk - интегрированная кросс-среда разработки приложений для OS-9 - функционирует на платформе MS Windows NT.
* Hawk является открытой средой и предоставляет сторонним разработчикам инструментальных средств более сотни API, позволяющих включать в рамках Hawk Partners Program в состав среды Hawk продукты известных фирм разработчиков инструментального ПО.
* Средство верификации программного обеспечения CodeTEST (Applied Microsystems) встроено в Hawk и представляет собой удобный и эффективный инструментарий трассировки встраиваемого ПО и контроля его характеристик, а также хода выполнения тестов и распределения памяти.
	+ - 1. Поддержка приложений жёсткого реального времени.
* масштабируемое, полностью вытесняемое ядро ОС;
* поддерживает функционирование до 65535 процессов;
* предоставляет 65535 уровней приоритета;
* обеспечивает работу до 255 пользователей;
* более 90 системных вызовов ядра предоставляют возможность управлять динамическими режимами диспетчеризации, распределением памяти, межпроцессорной коммуникацией и т.д. вплоть до управления встроенным в ядро ОС режимом экономичного потребления питания.
* характеристики производительности: 5.6 мкс Interrupt Latence Time, 14 мкс для времени переключения контекста процесса (MC68040, 30MHz).
	+ 1. VxWorks фирмы Wind River Systems.

ОС РВ VxWorks предназначена для применения на встроенных компьютерах, работающих в системах "жесткого" реального времени. VxWorks является системой с кросс-средствами разработки прикладного программного обеспечения.

* + - 1. Основные свойства VxWorks.
* Поддерживаемые целевые архитектуры (targets): Motorola 680х0 и CPU32, PowerPC; Intel 386/486/Pentium, Intel 960; Sparc, Mips R3000/4000; AMD 29K, Motorola 88110; HP PA-RISC; Hitachi SH7600; DEC Alpha.
* Поддерживаемые инструментальные платформы (hosts): Sun SPARCstation (SunOS и Solaris); HP 9000/400,700 (HP-UX); IBM RS6000 (AIX); Silicon Graphics (IRIX); DEC Alpha (OSF/1); PC (Windows).
* Все аппаратно-зависимые части VxWorks вынесены в отдельные модули для того, чтобы разработчик встроенной компьютерной системы мог сам портировать VxWorks на свою нестандартную целевую машину.
* В последней версии VxWorks 5.2 реализованы совместимые с расширениями POSIX для приложений реального времени (POSIX Real-Time Extensions 1003.1b) функции асинхронного ввода-вывода, счётные семафоры, очереди сообщений, сигналы, управление памятью (блокировка страниц), управление диспетчеризацией, часы и таймеры.
* Стандартным языком программирования в инструментальном комплексе VxWorks является язык С. Система программирования на языке C++ не входит в стандартную конфигурацию инструментального комплекса VxWorks и поставляется как дополнительный продукт. Система программирования на языке Ada для VxWorks поставляется почти всеми Ada-производителями.
* Возможность исследования динамики исполнения программ и изменения данных предоставляют специальные средства отладки в реальном масштабе времени, которые трассируют интересующие пользователя события и накапливают их в буфере для последующего анализа.
	+ - 1. Поддержка приложений жёсткого реального времени.
* Построена по технологии микроядра.
* Представляет собой архитектуру высокой готовности с распределенной передачей сообщений и поддержкой отказоустойчивости.
* ОС позволяет программистам изолировать совместно используемые библиотеки, данные и системное программное обеспечение, а также приложения.
	+ 1. QNX4 фирмы ОРАКУЛ.

QNX4 — многозадачная многопользовательская операционная система жесткого реального времени (ОСРВ) с архитектурой на основе микроядра и поддержкой ряда стандартов семейства POSIX.

* + - 1. Основные свойства QNX4.
* Состоит из микроядра и набора необязательных модулей.
* Предоставляет сервисы стандарта POSIX.1 и его расширения для систем реального времени POSIX.1b (POSIX.4).
* Можно использовать для расширения функциональных возможностей как штатные модули, так и свои собственные.
* Предоставляемое QNX4 окружение защищенного режима дает возможность легко и безопасно тестировать свои новые модули расширения.
* Возможности высокоскоростной трассировки диагностических событий.
* Позволяет запускать процессы по сети с полным наследованием окружения, включая открытые файлы, текущий каталог, файловые дескрипторы и идентификатор пользователя.
	+ - 1. Поддержка приложений жёсткого реального времени.
* Являясь истинно микроядерной ОС, QNX4 строится вокруг компактного высоконадежного «стержня» - имеет микроядро размером всего 10 Кбайт.
* Микроядро QNX4 обладает достаточно малыми размерами для встраивания в ПЗУ.
* Обладает достаточно большой мощностью для управления распределенной сетью, содержащей нескольких сотен процессоров.
* Менеджер устройств является высокопроизводительным и вносящим очень малые накладные расходы серверным процессом, обеспечивающим интерфейс между процессами и терминальными устройствами.
	+ 1. Вывод.

Системы реального времени в настоящий момент являются востребованным продуктом на рынке программного обеспечения. Существует целая гамма средств данного направления, покрывающая практически весь спектр возможных применений подобных систем повышенной надежности.

* 1. Методология разработки программного обеспечения.

Возрастающая сложность современного программного обеспечения привела к созданию специальной научной дисциплины — компьютерной инженерии (Software Engineering), основной задачей которой является создание эффективных методов разработки сложных программных систем.

* + 1. История развития.

Объектно-ориентированные методологии разработки программного обеспечения (первое направление) стали интенсивно развиваться с конца 80-х годов. В 1997 г. OMG (Object Management Group) приняла UML, появившийся в результате слияния ряда известных методологий, в качестве стандарта языка объектно-ориентированного моделирования. Еще одним объектно-ориентированным подходом является методология ROOM, созданная для разработки систем реального времени. Одновременно в течение последних 20 лет международным комитетом ITU развиваются стандарты для разработки телекоммуникационных систем (второе направление): SDL, MSC и т.д. Кроме того, с 70-х годов развиваются структурные методологии разработки программного обеспечения (третье направление): SADT, IDEF-стандарты, метод Йордана и т.д. В настоящее время эти методологии прочно закрепились в области разработки информационных систем. Они являются эффективным средством анализа систем в целом и успешно применяются.

В данной работе описывается объектно-ориентированная методология Real. Методология Real основывается, главным образом, на UML, SDL, ROOM и отражает перечисленные интеграционные тенденции. Помимо стандартных для объектно-ориентированного подхода черт в Real добавлены дополнительные возможности, направленные на две специальные области программного обеспечения: для информационных систем и для систем реального времени.

Естественно, что Real не претендует на то, чтобы покрыть все возможности программных продуктов соответствующих областей. В то же время, учитывая современный уровень развития локальных и глобальных информационных сетей и возрастающую сложность программного обеспечения, в информационных системах все большую популярность приобретает технология клиент-сервер, т.е. многие информационные системы приобретают ярко выраженный событийно-ориентированный аспект, который глубоко проработан в методологиях разработки программного обеспечения систем реального времени. С другой стороны, большие распределенные системы реального времени нуждаются, как правило, в хранении, доступе и передаче огромного количества информации (например, тарификационной и аутентификационной), а не только управляющих сигналов и данных трафика. Таким образом, методология Real подходит для разработки программного обеспечения обеих областей, но наиболее эффективна для их пересечения.

* + 1. Разработка программного обеспечения систем реального времени

Основное назначение Real применительно к системам реального времени — проектирование сложной управляющей логики с последующей возможностью автоматической генерации кода. Отметим, что методология Real не ориентирована специальным образом на разработку оборудования и программного обеспечения, непосредственно с ним контактирующего (драйверов устройств и т.п.), а также сетевых протоколов нижнего уровня. Однако мы считаем, что для этих задач она подходит примерно в той же степени, как и UML.

Как показывает практика, прямые ветки сложных алгоритмов достаточно удобно и наглядно определять с помощью сценариев. На начальных этапах разработки системы нужно четко определить логику всех взаимодействий. При этом правила поведения системы в ошибочных ситуациях в большинстве случаев можно доработать позднее. По сценариям можно сгенерировать STD- или SDL-диаграммы и продолжить создание спецификаций уже в этом стиле, учитывая все допустимые варианты поведения системы.

В терминах Real основной структурной единицей системы реального времени является объект. Объекты взаимодействуют друг с другом через интерфейсы. Под взаимодействием понимается посылка сообщений, вызов методов и обращение к атрибутам интерфейса. Поскольку в последнее время все большее число систем реального времени становятся распределенными и сетевыми, понятие интерфейса приобретает особую важность. Раньше ситуация была иной, примером чего служат ранние версии языка SDL, в которых интерфейсы отсутствовали. Интерфейсы Real сильно отличаются от портов (gate) SDL (составом, способом связи с классами) и интерфейсов UML (составом, способом связи с классами, способом изображения), а также интерфейсов в ROOM (составом и способом изображения).

В системах реального времени важную роль играют абстракции точек входа и выхода у различных компонент программного обеспечения. Поэтому в модель классов Real был добавлен из ROOM специальный элемент — порт.

Компонента программного обеспечения, определенная в виде класса с портами и интерфейсами, может иметь конечно-автоматное поведение, описываемое в терминах поведенческой модели Real. Поведенческая модель, в свою очередь, представляется двумя альтернативными нотациями: первая основана на варианте STD, представленном в ROOM, вторая — на расширенном конечном автомате SDL. С помощью STD-нотации удобно определять поведение компонент системы на ранних этапах разработки: множество мелких деталей можно временно убрать из поля зрения. В то же время на SDL-диаграммах можно наглядно изобразить мельчайшие подробности алгоритмов.

Эта возможность становится полезной на поздних этапах проектирования. При этом информацию, изображенную на STD-диаграммах, можно ”загрузить” на SDL-диаграммы, таким образом, результаты ранних этапов при переходе к более формальной спецификации не будут потеряны. В рамках Real, STD и SDL нотации предназначены для описания единой поведенческой модели, так что всегда возможно и обратное — загрузка на STD-диаграмму результатов работы с SDL-редактором. На поведенческую модель Real сильно повлияла технология SDL/PLUS 20: так же не используются типы данных и выражения SDL, но применяется более гибкая стратегия связи с языками реализации [7] вместо заранее фиксированного языка [11].

* + 1. Вывод.

Использование специальной событийно-ориентированной методологии разработки программного продукта поможет улучшить стройность организации прикладного приложения и, в целом, хорошо отразится на надежности создаваемой системы реального времени.

* 1. Постановка задачи курсового проекта.

Перед созданием программной системы реального времени был проведен анализ действующих в данный момент разработок данного направления. Он показал, что существующий спектр ОС может обеспечить все потребности задач реального времени. Выбор системы (если абстрагироваться от цены, условий поставки и т.д.) диктуется только тем обстоятельством, удовлетворяет ли она условиям стоящей задачи. Например, если необходима хорошая операционная поддержка сетевых средств, то целесообразно использовать QNX - многозадачную операционную систему жесткого реального времени с архитектурой на основе микроядра. Если необходима очень высокая скорость реакции системы, можно использовать VxWorks. В данной работе при создании очень небольшой, не слишком сложной, встроенной прикладной программы было решено ориентироваться LynxOS, относящуюся к классу Unix’ов реального времени.

В разработке будут использованы только базовые, обязательные механизмы ОСРВ. Кроме того, почти в каждой операционной системе реального времени можно найти целый набор дополнительных, специфических только для нее механизмов, касающихся системы ввода-вывода, управления прерываниями, работы с памятью. Каждая система содержит также ряд средств, обеспечивающих ее надежность, например, встроенные механизмы контроля целостности кодов.

Главный же вывод состоит в том, что любая задача реального времени требует операционной поддержки реального времени, и иные системные решения при этом неприемлемы. На основе ОС общего назначения UNIX, ориентированной на оптимальное распределение ресурсов компьютера между пользователями и задачами (система разделения времени) будет создана программная разработка планировщика задач, в котором главной целью является успеть среагировать на происходящие события в жестко заданный интервал времени (система реального времени).

На основе планировщика будет реализован протокол, требующий поддержки реального времени. Для проектирования его реализации будет использована объектно-ориентированная методология. Такая методология характеризует скорее подход к планированию приложения, чем используемые средства языка, т.к. в авиационных системах, к которым и принадлежит реализуемый протокол A415 ARINC, не разрешено использование памяти из «кучи». В то время как динамическое выделение памяти является немаловажной частью использования языков, подобных C++ или Java.

1. **Модели и методы предметной области.**
	1. Определения.

Сначала рассмотрим основное определение, вокруг которого и построена данная работа. Оно дано американским учёным Дональдом Гиллисом (Donald Gillies).

* Система реального времени – это система, в которой правильность вычислений зависит не только от логической корректности результата, но и от времени, в течение которого этот результат получен. Система работает неверно, если ее временные параметры не соответствуют заданным.

Рассмотрим простейшие определения, используемые в данной отрасли знаний. Приведено их неформальное определение, которое преследует своей целью дать общее представление об используемых терминах и уточнить подразумеваемое под ними в данной работе значение.

* Состояние — стабильное положение объекта, когда он готов к принятию запросов на взаимодействие со стороны других объектов. С состоянием может быть связана некоторая деятельность объекта (например, входная и выходная). Состояние может быть сложным, т.е. содержать подсостояния.
* Событие — получение сообщения или окончание срока действия таймера.
* Действие — посылка сообщения, установка таймера, блок кода на целевом языке.
* Время прибытия задачи – время, когда возникает необходимость в выполнении данной задачи.
* Вызов задачи – начало выполнения данной задачи после прибытия.
* Планировщик – механизм, который определяет, какой результат должен быть вычислен и в какое время. Результат работы планировщика называется расписанием.
* Политикой планирования (или управления) называется набор правил, которые определяют то, как планировщик выбирает следующий процесс для запуска, как образуется очередь процессов на выполнение и сколько времени выделяется каждому процессу на выполнение.

Определения, непосредственно затрагивающие специфику данной работы.

* Масштабируемая ОС – операционная система, характеризующаяся высоким уровнем модульности, при котором отдельные функции системы можно динамически отключать/подключать без ущерба для общей функциональности системы. Такая ОС в зависимости от решаемой задачи, может быть установлена как на FLASH-носитель, так и на гигабайтный жесткий диск.
* Микроядерная ОС – операционная система, в основу архитектуры которой положена специальная часть исполняемого кода – микроядро, реализующее базовые функции.
* Многопользовательская ОС – операционная система, ориентированная на одновременную работу с несколькими пользователями.
* Многозадачная ОС – операционная система, ориентированная на параллельное выполнение системой нескольких задач. Истинно многозадачной ОС может быть только при наличии более чем одного микропроцессора, но современные операционные системы обладают приемлемыми механизмами обеспечения псевдопараллельной работы.
* Сетевая ОС – операционная система, ориентированная не только на работу с использованием ресурсов своего аппаратного обеспечения, но и на взаимодействие с удалёнными ресурсами с использование специальных протоколов.
	1. Принципиальная структура.
		1. Среда исполнения.

Основное предназначение любой операционной системы - это рациональное управление ресурсами компьютера во время его работы. Все действия операционной системы по обеспечению успешного диалога с пользователем или пользователями сводятся к следующим простым действиям - управлению выполнением программ и работой служб, записи и чтению файлов с диска, обмену информацией по сети. Причем, все эти простые действия должны выполняться слаженно и не создавать конфликтных ситуаций при работе системы. Для этого нужно обратить внимание на среду, в которой функционирует приложение реального времени. Требования, предъявляемые к среде исполнения систем реального времени, следующие:

* небольшая память системы - для возможности ее встраивания;
* система должна быть полностью резидентна в памяти, чтобы избежать замещения страниц памяти или подкачки;
* система должна быть многозадачной - для обеспечения максимально эффективного использования всех ресурсов системы;
* ядро с приоритетом на обслуживание прерывания.

 Приоритет на прерывание означает, что готовый к запуску процесс, обладающий некоторым приоритетом, обязательно имеет преимущество в очереди по отношению к процессу с более низким приоритетом, быстро заменяет последний и поступает на выполнение. Ядро заканчивает любую сервисную работу, как только поступает задача с высшим приоритетом. Это гарантирует предсказуемость системы;

* диспетчер с приоритетом.

Дает возможность разработчику прикладной программы присвоить каждому загрузочному модулю приоритет, неподвластный системе. Присвоение приоритетов используется для определения очередности запуска программ, готовых к исполнению. Альтернативным такому типу диспетчеризации является диспетчеризация типа "карусель", при которой каждой готовой к продолжению программе дается равный шанс запуска. При использовании этого метода нет контроля за тем, какая программа и когда будет выполняться. В среде реального времени это недопустимо. Диспетчеризация, в основу которой положен принцип присвоения приоритета, и наличие ядра с приоритетом на прерывание позволяют разработчику прикладной программы полностью контролировать систему. Если наступает событие с высшим приоритетом, система прекращает обработку задачи с низшим приоритетом и отвечает на вновь поступивший запрос.

Сочетание описанных выше свойств создает мощную и эффективную среду исполнения в реальном времени.

* + 1. Ядро систем реального времени.

Кроме свойств среды исполнения, необходимо рассмотреть также сервис, предоставляемый ядром ОС реального времени. Ядро или диспетчер является основой любой среды исполнения в реальном времени. Микроядро реализует базовые функции операционной системы, на которые опираются системные сервисы и приложения. В системе реального времени диспетчер занимает место между аппаратными средствами целевого компьютера и прикладным программным обеспечением. В результате, такие важные компоненты ОС как файловая система, сетевая поддержка и т. д. превращаются в по-настоящему независимые модули, которые функционируют как отдельные процессы и взаимодействуют с ядром и друг с другом на общих основаниях. Все компоненты системы используют средства микроядра для обмена сообщениями, но взаимодействуют непосредственно. Предоставляемый ядром сервис дает прикладным программам доступ к таким ресурсам системы, как, например, память или устройства ввода/вывода.

Ядро может обеспечивать сервис пяти типов:

* + - 1. Синхронизация ресурсов.

Метод синхронизации требует ограничить доступ к общим ресурсам (данным и внешним устройствам). Наиболее распространенный тип примитивной синхронизации - двоичный семафор, обеспечивающий избирательный доступ к общим ресурсам. Так, процесс, требующий защищенного семафором ресурса, вынужден ожидать до тех пор, пока семафор не станет доступным, что свидетельствует об освобождении ожидаемого ресурса, и, захватив ресурс, установить семафор. В свою очередь, другие процессы также будут ожидать доступа к ресурсу вплоть до того момента, когда семафор возвратит соответствующий ресурс системе распределения ресурсов. Системы, обладающие большей ошибкоустойчивостью, могут иметь счетный семафор. Этот вид семафора разрешает одновременный доступ к ресурсу лишь определенному количеству процессов.

* + - 1. Межзадачный обмен.

Часто необходимо обеспечить передачу данных между программами внутри одной и той же системы. Кроме того, во многих приложениях возникает необходимость взаимодействия с другими системами через сеть. Внутренняя связь может быть осуществлена через систему передачи сообщений. Внешнюю связь можно организовать либо через датаграмму (наилучший способ доставки), либо по линиям связи (гарантированная доставка). Выбор того или иного способа зависит от протокола связи.

* + - 1. Разделение данных.

В прикладных программах, работающих в реальном времени, наиболее длительным является сбор данных. Данные часто необходимы для работы других программ или нужны системе для выполнения каких-либо своих функций. Во многих системах предусмотрен доступ к общим разделам памяти. Широко распространена организация очереди данных. Применяется много типов очередей, каждый из которых обладает собственными достоинствами.

* + - 1. Обработка запросов внешних устройств.

Каждая прикладная программа в реальном времени связана с внешним устройством определенного типа. Ядро должно обеспечивать службы ввода/вывода, позволяющие прикладным программам осуществлять чтение с этих устройств и запись на них. Для приложений реального времени обычным является наличие специфического для данного приложения внешнего устройства. Ядро должно предоставлять сервис, облегчающий работу с драйверами устройств. Например, давать возможность записи на языках высокого уровня - таких, как Си или Паскаль.

* + - 1. Обработка особых ситуаций.

Особая ситуация представляет собой событие, возникающее во время выполнения программы. Она может быть синхронной, если ее возникновение предсказуемо, как, например, деление на нуль. А может быть и асинхронной, если возникает непредсказуемо, как, например, падение напряжения. Предоставление возможности обрабатывать события такого типа позволяет прикладным программам реального времени быстро и предсказуемо отвечать на внутренние и внешние события. Существуют два метода обработки особых ситуаций - использование значений состояния для обнаружения ошибочных условий и использование обработчика особых ситуаций для прерывания ошибочных условий и их корректировки.

* + 1. Пикоядро.

Базовые требования современных систем реального времени стали столь обширны, что назрела необходимость в структуризации уже самого микроядра. Была выдвинута идея так называемого «пикоядра». Пикоядро – в данном случае это ядро, имеющее следующие свойства:

* Не имеет каких-либо состояний (начальных, конечных или промежуточных), ядро без состояния. Не требует инициализации и деинициализации.
* Реализует и содержит в себе очень малое количество функций и данных – только функции для работы с объектами.
* Предоставляет объектно-ориентированный интерфейс системе в виде небольшого количества системных вызовов для работы с объектами.
* Является полностью пассивной частью кода операционной системы – код ядра выполняется только во время системных вызовов.
* В большинстве случаев является обычной статической библиотекой, которая компонуется с главной системной частью ОС – менеджером процессов.
	1. Методы управления задачами в ОС РВ.
		1. Классификация подходов.

Существует большое количество различных методов управления задачами. Каждый из них предназначен для использования в определённом классе систем, каждая из которых основана на некотором множестве ограничений.

* + - 1. Статическое планирование.

Большинство всех существующих методов относятся к статическому планированию. В этом случае считается, что всё множество задач системы и все их характеристики известны заранее. В этом случае расписание работы задач строится до начала работы системы и остаётся постоянным во время её функционирования. В этом расписании определены времена старта для всех задач системы. В течение работы системы планировщик выбирает следующую задачу для запуска в соответствии с этим расписанием. Расписание циклически повторяется.

Однако в реальных системах одно подобное расписание не может предусмотреть все возможные ситуации, которые могут возникнуть. Кроме того, в системе может быть несколько независимых режимов работы, переключение между которыми может происходить в заранее не определённое время. Поэтому обычно на практике до начала работы составляется несколько расписаний для различных случаев. Затем во время функционирования системы расписания меняются. Это может происходить или в непредсказуемые или в заранее определённые моменты времени, когда потребовалась смена режима работы.

* + - 1. Динамическое планирование.

При динамическом же планировании планировщик в каждый момент времени обладает полными знаниями только о текущем множестве задач. В момент планирования данного множества, он не имеет никаких сведений о тех задачах, которые могут появиться в будущем. Поэтому расписание меняется с течением времени. Динамических алгоритмов планирования существует значительно меньше, чем статических.

* + - 1. Планирование, основанное на времени.

В этом случае производится статический анализ системы, в результате чего строится расписание, которое затем используется во время работы для того, чтобы решить, когда и какая задача должна начать своё выполнение. Это расписание содержит фиксированное время старта для каждого примера задач, основываясь на времени выполнения в худшем случае и всех взаимозависимостях между задачами. Затем это расписание может измениться.

Планировщик должен содержать всю дополнительную информацию обо всех примерах всех задач. Когда прибудет новая задача необходимо определить, основываясь на существующем расписании, можно ли её туда добавить, и если да, то построить новое расписание.

Данный класс методов применяются для периодических задач, или для задач, которые могут быть сведены к периодическим. Основным критерием для статического планирования периодических задач можно считать предсказуемость, то есть определение исполнимого расписания, в котором все задачи удовлетворяют всем своим ограничениям.

Так как в этом подходе, исходя из заданных характеристик, строится таблица, которая определяет время запуска и время выполнения для каждой задачи, после чего задачи располагаются в соответствии с этим расписанием, то, как следствие, то, когда и где выполняются задачи, строго фиксировано. Этот подход не является достаточно гибким, так как любое изменение характеристик какой-либо задачи может потребовать полной перестройки всей таблицы.

Так как задачи могут иметь множество различных ограничений, то для нахождения исполнимого расписания применяются различные методы, например, математического программирования. Чаще всего используется метод ветвей и границ.

* + - 1. Планирование апериодических задач

Используя данный принцип можно планировать и апериодические задачи. При этом они планируются во время работы. В начале крайние сроки всех примеров задач сортируются, после чего расписание делится на множество непересекающихся интервалов работы. Затем для этих интервалов определяются запасные промежутки времени, которые могут быть использованы для планирования вновь прибывших апериодических задач.

Другой метод основан на использовании того факта, что выполнение задачи может быть динамически сдвинуто влево или вправо по временной шкале до тех пор, пока все временные ограничения для всех задач всё ещё выполнимы. Задачи должны быть прерываемыми.

* + - 1. Планирование, управляемое приоритетами.

В этом случае также проводится статический анализ, но в отличие от предыдущего случая чёткое расписание не строится, а только устанавливаются приоритеты для всех задач. Во время работы системы активизируется первая готовая к запуску задача с наивысшим приоритетом. При этом если в этот момент выполняется задача с более низким приоритетом, то она приостановит своё выполнение и процессор будет отдан новой задаче с более высоким значением приората.

Приоритеты назначаются исходя из временных ограничений задач. Они могут быть статическими или динамическими. Статические приоритеты, в отличии от динамических, задаются один раз и не меняются с течением времени.

* + 1. Обзор методов.
			1. Rate-monotonic (RM).

Метод назначает статические приоритеты задачам основываясь на их периодах. В этом методе приоритеты определяются следующим образом: задача с самым маленьким периодом получает самый высокий приоритет.

Они также показали, что эта схема является оптимальной среди всех статических алгоритмов. Под оптимальным понимается то, что если множество задач может быть спланировано любым другим статическим алгоритмом, основанном на приоритетах, то оно также может быть спланировано и этим методом.

Исходный RM подход имеет ряд ограничений:

* Все задачи должны быть независимы друг от друга, т.е. между ними нет ни взаимодействия, ни общих ресурсов.
* Все задачи должны быть периодическими.
* Все задачи могут быть приостановлены другими задачами с более высокими приоритетами. Однако ни одна задача не может блокироваться, ожидая внешнего события.
* Время выполнения постоянно.
* Для задач определено время выполнения в худшем случае.
* Все задачи имеют крайний срок, эквивалентный их периоду.

Было проведено большое количество исследований для расширения этих методов. В результате этих работ были сняты или ослаблены ограничения, налагаемые на задачи в исходной модели.

Так в протоколе приоритетных границ (Priority Ceiling Protocol) и некоторых других похожих (Stack Resource Protocol) удалось избавиться от ограничения на взаимодействие задач. Также было предложено много методов приведения непериодических задач к периодическим.

* + - 1. Deadline Monotonic (DM).

Метод может быть использован для планирования задач, у которых крайние сроки меньше или равны периодам. Он ослабляет ограничение на величину крайнего срока в схеме планирования RM. В этом случае приоритет, назначенный задаче, обратно пропорционален величине её крайнего срока, то есть задача с самым коротким крайним сроком имеет самый высокий приоритет независимо от её периода. Если две задачи имеют одинаковые крайние сроки, то они получают приоритеты в произвольном порядке относительно друг друга. Метод может обслуживать как периодические, так и спорадические задачи.

Такой метод расстановки приоритетов будет оптимальным, если выполняются следующие условия:

* множество задач – фиксированное множество жёстких задач;
* задачи периодические или спорадические;
* задачи имеют определённое (известное) время выполнения в худшем случае;
* для задач определён критический момент, то есть время выполнения в худшем случае.

Оптимальность здесь также означает, что если любой планировщик с фиксированными приоритетами может спланировать множество задач, у которых крайние сроки меньше или равны периоду, и выполнены соответствующие ограничения, то и этот планировщик тоже может.

* + - 1. Планирование апериодических задач.
				1. Метод фонового выполнения.

Самый простой подход – это обрабатывать апериодические задачи в фоновом режиме и запускать их только тогда, когда процессор не занят выполнением какой-либо из периодических задач.

* + - * 1. Метод опроса.

Метод использует отдельную периодическую задачу с высоким приоритетом для поддержки выполнения апериодических задач.

Оба этих метода неэффективны, когда время ответа апериодической задачи важно.

* + - * 1. Алгоритм безотлагательного сервера (IS)

Это также подход сохранения пропускной способности. Он также использует периодический сервер, который имеет самый высокий приоритет, но не обязательно самый короткий период. Сервер приостанавливается, если не осталось ни одной апериодической задачи, и активизируется немедленно при прибытии апериодической задачи.

* + - * 1. Последний шанс.

Этот алгоритм является глобально оптимальным в том смысле, что обеспечивает минимальное время ответа для апериодических задач (при условии выполнения всех крайних сроков периодических задач) среди всех возможных методов планирования периодических и апериодических задач.

Планирование состоит в том, что если ещё остаётся апериодическая задача, которая должна быть выполнена, следующая периодическая задача не будет запущена до самого последнего возможного момента, называемого временем уведомления, когда ещё сохраняется гарантия выполнения её крайнего срока (также как и всех остальных периодических задач).

Этот метод гарантирует своевременность выполнения периодических задач и максимизирует ответную реакцию апериодических задач.

При использовании этого метода изначально применяется любой алгоритм с фиксированными приоритетами для планирования периодических задач до начала работы системы. Все периодические задачи имеют более высокие приоритеты, чем апериодические.

* + - 1. EDF.

В методе EDF приоритеты задачам назначаются исходя из их крайних сроков на текущий момент. В этом случае задача с ближайший крайним сроком получает наивысший приоритет. Этот метод также является оптимальным в том смысле, что если можно найти исполнимое расписание для данного множества задач с фиксированными приоритетами, то всегда можно найти исполнимое расписание и с использованием этого метода. Однако он является оптимальным только при недогрузке системы, но в условиях перегрузки ведёт себя довольно плохо.

Данный метод часто считается опасным из-за того, что при условии перегрузки системы он может показать нежелательное поведение. Однако во время работы жёстких систем реального времени перегрузок не должно возникать, потому что невыполнение крайнего срока задачи может привести к серьёзным последствиям. Поэтому для таких систем необходимо проводить априорное доказательство того, что когда у всех задач в системе одновременно возникнут потребности в системных ресурсах, то все их ограничения по времени по-прежнему будут выполнены.

* + - 1. Сервер, допускающий задержку (DS) и Алгоритм обмена приоритетами (PE).

Эти методы сохраняют доступными ресурсы системы, первоначально выделенные для апериодических задач.

Эти методы улучшают среднее время ответа системы и отличаются друг от друга способом сохранения пропускной способности. PE алгоритм раздаёт время выполнения, выделенное для работы высокоприоритетного периодического сервера, другим периодических задачам с меньшим приоритетом, если оно не нужно для работы апериодических задач.

В отличие от него DS не отдаёт время выполнения этой задачи, когда не осталось ни одной апериодической задачи. Вместо этого он хранит это высокоприоритетное время выполнения либо пока не прибудет апериодическая задача, либо пока не закончится период сервера.

Этот метод проще в реализации, но хуже в исполнении.

* 1. Методология разработки программного обеспечения.
		1. Основы методологии Real.

Не останавливаясь, в общем, на процессе разработки программного обеспечения, перечислим, какие модели используются в Real для описания разрабатываемой системы:

* Модель требований к системе:

Описательная модель — в текстовом виде описывает некоторые требования к системе.

Модель случаев использования — описывает требования, предъявляемые к системе ее окружением, т.е. отвечает на вопрос “что и для кого должна делать система?”.

Функциональная модель — описывает разбиение случаев использования и функций на подфункции. Дает ответ на вопрос “как должны реализовываться функции системы в терминах своих подфункций?”.

* Динамическая модель:

Модель объектов — описывает роли объектов системы и отвечает на вопрос “какие объекты взаимодействуют при выполнении функций системы?”.

Модель взаимодействий — описывает сценарии взаимодействия объектов системы между собой и с пользователями, т.е. дает ответ на вопрос “как объекты взаимодействуют друг с другом для выполнения функций системы?”.

Поведенческая модель — описывает алгоритмы поведения объектов системы, т.е. отвечает на вопрос “как должен вести себя каждый объект для реализации функций системы?”.

* Статическая модель:

Модель классов — описывает внутреннюю структуру системы, структуры данных, используемые в ней, т.е. отвечает на вопрос “как должна выглядеть система изнутри?”.

В Real большой упор был сделан на связность моделей, на контроль целостности информации о проекте, представленной внутри как одной модели, так и в нескольких.

* + 1. Модель требований.

Работа над системой в Real начинается с построения описательной модели, в которую, прежде всего, входят первичные требования заказчика. Среди них могут быть как функциональные требования, так и любые другие (эффективность, стоимость и т.п.). Описательная модель хранится в Real в виде обычного текста и формально не связана с остальными моделями. Эта модель может быть использована и для окончательной спецификации нефункциональных требований.

На основе требований заказчика формулируется полный список функциональных требований к системе, которые оформляются в терминах модели случаев использования и модели функций. Окончательное техническое задание на систему может быть сгенерировано по модели требований Real в том виде, который нужен заказчику (ГОСТ, какой-либо международный или внутрикорпоративный стандарт и т.п.).

*Модель случаев использования* в Real предназначена для описания стыка системы с окружением. В ее терминах описываются все пользователи системы, а также все ее функции (случаи использования), различимые с точки зрения этих пользователей. В дальнейшем с использованием могут быть связаны классы. Для случаев использования, в свою очередь, можно создавать диаграммы этого же типа, т.е. подвергать случаи использования дальнейшей декомпозиции в рамках той же модели.

*Функциональная модель* предназначена для дальнейшей декомпозиции функций системы. Она состоит из набора деревьев функций, корнями которых являются случаи использования. Дерево может содержать узлы двух видов: собственно функции и использование описанных ранее функций. Кроме того, функция может иметь свойство групповой, это означает, что ее ”дети” фактически находятся вместо нее на том же месте. Связь родительского узла с дочерними может иметь метку, описывающую характер в связи.

Отметим, что модель случаев использования в Real является подмножеством одноименной модели UML. То, что в UML делается с помощью не вошедшей в Real части модели случаев использования, в Real предлагается делать с помощью модели функций, которая является вариацией функциональной модели из структурных методологий разработки программного обеспечения. Модель функций Real основана на модели функций SDL, однако оттуда были убраны некоторые детали (в Real не предполагается так широко использовать модель функций, поскольку не хотелось бы подталкивать разработчика к алгоритмическому методу разработки системы) и добавлены использование функций, группы функций, а также связь с моделью случаев использования.

* + 1. Динамическая модель.

Она описывает поведение системы — взаимодействие между различными ее компонентами, взаимодействие системы с ее окружением и поведение самих компонент.

На начальных этапах разработки можно придерживаться одной из двух стратегий. Первая: сначала специфицировать классы системы, а затем объекты и сценарии взаимодействия. Она будет использоваться с большей вероятностью, если разработчикам хорошо знакома предметная область. Возможна и другая стратегия — в том случае, если на этапе анализа приходится изучать незнакомую предметную область. Основное назначение *модели объектов* — описание различных ролей, которые могут играть экземпляры классов системы. Каждой функции из функциональной модели Real можно сопоставить диаграмму объектов, назначение которой — описать типичную ”конфигурацию” объектов, задействованных в осуществлении данной функции, а также описать связи между ними. При использовании объектно-ориентированного подхода выполнение функций системы реализуется как совместная деятельность нескольких объектов. Основными ее элементами являются объекты-роли и отношения между ними.

Динамику взаимодействия объектов для реализации функции (*модель взаимодействия*) удобно представлять в виде сценариев. В этих сценариях принимают участие объекты-роли, определенные на диаграмме объектов для данной функции или ее надфункций. Сценарий представляет собой упорядоченную во времени последовательность событий, которыми, как правило, являются посылки и приемы сообщений объектами.

Построение сценариев для функции начинается с определения ”прямых веток”, т.е. идеального исполнения функции. При этом из рассмотрения исключаются граничные, ошибочные ситуации, частные случаи и т.п., для них впоследствии тоже строятся сценарии либо они специфицируются другими средствами.

*Поведенческая модель* описывает поведение составляющих систему классов с помощью расширенного конечного автомата и представлена в Real двумя нотациями: в стиле STD и SDL. Фактически, поведенческая модель определяет процессы, протекающие в системе в терминах состояний, событий и действий. В дальнейшем будем говорить о поведенческой модели отдельного класса. Построение такой модели можно начать с анализа всех сценариев, в которых участвуют объекты-роли данного класса. Проектирование поведения системы (поведения ее классов) на основе сценариев, а не напрямую, позволяет в более наглядном виде представлять общие процессы, протекающие в программном обеспечении, и, отталкиваясь от них, конструировать внутреннее поведение участников этих процессов.

* + 1. Статическая модель.

После того, как созданы основные сценарии системы, можно переходить к спецификации их участников — объектов, т.е. к построению модели классов. Эта модель классов строится на протяжении всего процесса разработки программного обеспечения.

В Real в модели классов могут быть следующие виды сущностей:

• класс — описание группы однородных объектов;

• шаблон — параметризованный класс с возможностью получения из него обычного класса подстановкой значений параметров;

• интерфейс — описание правил взаимодействия классов;

• представление — аналог конструкции VIEW языка SQL.

Модель классов Real реализует достаточно полное подмножество модели классов UML. Кроме того, в ней есть интерфейсы и порты из ROOM, при этом последние существенно расширены. Модель классов Real содержит также средства моделирования схемы баз данных.

1. **Реализация прототипа системы реального времени.**
	1. Жизненный цикл разработки.

Разработка состоит из двух основных частей: планировщика задач РВ и прикладного приложения. Прямых зависимостей между этапами проектирования данных систем нет. Однако, существуют логические связи. Приложение строится на основе созданного планировщика, что предполагает знание о предоставляемых им интерфейсах. Планировщик, в свою очередь, строится с учетом особенностей приложения, которое является приложением контроля, т.е. ориентированным на обработку внешних стохастических событий.

На диаграмме 1 представлены этапы разработки программной системы. Выполненные в рамках данной работы, выделены чёрным цветом, предполагаемые к исполнению в дальнейшем – серым.

Для планировщика выбрана V – образная модель жизненного цикла. Она применяется для приложений, при проектировании которых разработчикам приходится исследовать новую проблемную область. Отличительной особенностью этой модели жизненного цикла является наличие обратных связей уже на этапах тестирования и верификации. Предполагается, что это позволит создать более гибкую в плане предоставляемых возможностей систему.

Для приложения-протокола выбрана каскадная модель жизненного цикла. Она применяется для приложений в хорошо исследованных областях знаний. В данном случае системные требования на протокол уже описаны в технической документации.

В данной работе будут выполнены этапы создания системных и функциональных требований к планировщику, а также определение его архитектуры. Для протокола будет выполнена функциональная модель и модель классов.

* 1. Планировщик заданий.
		1. Выбор алгоритма планирования.
			1. Виды требований РВ, поддерживаемые планировщиком.

Во многих системах можно заранее установить множество задач, которые будут использоваться, и предположить их характеристики работы в худшем случае. При этом можно либо провести фиксированное планирование, которое будет удовлетворять требованиям системы, либо определить предварительные приоритеты задач.

Следующие ограничения будет возможно задавать с помощью создаваемого планировщика. Они основаны на временном поведении задач.

* + - * 1. Абсолютные ограничения.
* Интервал выполнения.

Выражает интервал, предоставляемый задаче для выполнения. Задаётся в микросекундах или как часть интервала выполнения всех задач. Определяет приоритетность определённой задачи на данном этапе вычислений. Может динамически изменяться.

* Время реакции.

Характеризует время, за которое должен быть получен отклик на внешнее воздействие. При превышении данного времени задаче выделяется больше ресурсов при помощи приостановки менее приоритетных задач.

* Время выполнения.

Выражает время выполнения задач в худшем случае. При превышении времени выполнения задача останавливается. В специальный стек «неуложившихся в срок» записывается её идентификатор. Для мягких задач данный параметр не фиксирован. Этот показатель также важен для распределённых (многопроцессорных) систем, где время выполнения задачи может зависеть от узла, на котором она выполняется.

* Периоды.

Период показывает, как часто задача должна выполняться. Период может ограничивать время реакции задачи, поэтому время реакции предполагается меньшим периода.

* + - * 1. Относительные ограничения.

Ограничения данного класса также называют локальными. Они выражают то, как две задачи связаны друг с другом.

* Приоритетные ограничения.

Данные ограничения определяют какие задачи предполагается блокировать при угрозе невыполнения срока данной задачи. В первую очередь блокируются мягкие задачи.

* Ограничения расстояния.

Определяют минимальное расстояние во времени между выполнением двух задач.

* Обновление.

Этот тип ограничений противоположен предыдущему. Данные ограничения влияют на то, в каком порядке должны выполняться задачи. Для взаимосвязанных задач можно задать выделение интервала выполнения первой перед второй. Эти ограничения также зависят от архитектуры системы, так как поведение системы моделируется как последовательность действий.

 Это может быть необходимо в случае, когда одна задача использует результаты работы другой, и если между ними пройдёт относительно большой промежуток времени, то результаты могут оказаться устаревшими.

* Гармонические ограничения

Эти ограничения связаны с периодами двух взаимодействующих задач. Они имеют место, например, когда период одной задачи (например, получателя) зависит от периода другой (отправителя).

* + - * 1. Неподдерживаемые ограничения.
* Отношения.

Данные ограничения выражают максимальный интервал времени между временами завершения двух задач.

* Разделительные ограничения.

Эти ограничения выражают интервал, которому должен принадлежать период задачи. Период может быть ограничен минимальным и/или максимальным значениями, которые гарантируют, что необходимые действия будут выполнены в полученный интервал.

* + - 1. Используемые алгоритмы.

Выбор того или иного метода планирования зависит от назначения системы. В системах контроля, на которые ориентирован планировщик, все данные должны быть чётко определены заранее. В этом случае статический алгоритм более уместен. Однако, статические методы планирования не являются достаточно гибкими, так как для обеспечения корректной работы системы заранее необходимо предусмотреть все возможные ситуации.

 Немного более гибким является подход, в котором расписание генерируется оперативно. В этом случае для каждой задачи заранее определяются: время реакции и время выполнения. В данной разработке будет использовано планирование, основанное на времени. Время реакции задаёт интервал, в течение которого предполагается ответ системы и после которого возникает угроза пропуска срока, требующая выделения дополнительных ресурсов для задачи. Время выполнения задаёт интервал, после выхода за границы которого задача считается невыполненной и требуется уже приостановка выполнения задачи для сохранения стабильности всей системы в целом.

Для динамического перераспределения ресурсов системы будет использована политика управления - round-robin. В этом случае процесс выполняется либо пока выделенный ему квант времени не истечёт, либо пока не будет приостановлен другим процессом с более высоким приоритетом. После того, как время, выделенное для данного процесса, истечёт, активируется следующий готовый к запуску процесс.

Когда процесс получает более высокий приоритет и приостанавливает выполнение текущего, планировщик сохраняет контекст приостановленного процесса для того, чтобы потом продолжить его выполнение с места останова. Приостановленный процесс остаётся готовым к запуску.

Процессы в списке готовых к запуску задач упорядочены, согласно методу EDF – в порядке возрастания времени реакции. При наличии блокировки запускаются только задачи с приоритетом выше или равным приоритету инициировавшей блокировку задачи.

* + 1. Описание функционирования приложения.

Схема взаимодействия объектов создаваемой системы показана на диаграмме 10.

* + - 1. Подготовка к запуску планировщика.

Главная программа запускает функцию инициализации планировщика, в которую передается массив структур, состоящий из указателей на планируемые процедуры и из параметров выполнения. В число параметров входят: идентификатор (задается главной программой), приоритет, интервал/квант выполнения, время реакции, время выполнения (для мягких задач может не определяться), период (для периодических функций), количество запусков или время работы (для спорадических задач), порядок, относительно других задач. Также возможна передача указателя на входные параметры процедуры.

В планировщике создается список задач. Создаётся процесс-таймер, который будет формировать последовательность сообщений для планировщика.

* + - 1. Работа.

Главная программа выполняет вызовы функции, отсылающей сообщения о начале работы определённой задачи. Планировщик создаёт отдельные процессы для каждого задания. Запущенные задачи сами могут вызывать функции планирования для себя или других задач.

Таймер получает от планировщика сообщения и создаёт соответствующие метки отсылки сообщений для планировщика. В дальнейшем по достижении метки таймер посылает планировщику сообщение инициировать работу определённой задачи. Если это метка периодической или спорадической задачи, то таймер создаёт через «период» следующую. Планировщик посылает процессам-заданиям сигналы начала кванта времени для выполнения.

При выходе определённой задачи за пределы времени реакции, таймер посылает сообщения о блокировании тех задач, у которых приоритет меньше. Если выход за пределы у двух одинаково приоритетных одновременно, то они продолжат выполняться вместе.

* + - 1. Управление задачами.

Во многих системах можно заранее установить множество задач, которые будут использоваться, и предположить их характеристики работы в худшем случае. При этом можно либо провести фиксированное планирование, которое будет удовлетворять требованиям системы, либо определить предварительные приоритеты задач. Однако неизбежно возникает необходимость изменения текущего режима/состояния системы. Можно выделить следующие типы операций по изменению режима:

* Добавление задачи.

Планировщик принимает сообщение об активации задачи, и помещает в число готовых к запуску с учётом абсолютных и относительных ограничений. Добавляется запись в процесс-таймер.

* Изменение интервала выполнения задачи.

Принимается сообщение об изменении интервала. Если задача, для которой он должен быть измен, активна, то она приостанавливается. Интервал изменяется. Затем цикл вычислений продолжается, начиная с этой задачи.

* Изменение периода выполнения периодической задачи.

Планировщик принимает сообщение и посылает к процессу-таймеру сигнал на изменение метки времени, соответствующей данной задаче.

* Изменение времени реакции, времени выполнения или приоритета.

Принимается сообщение о необходимости изменения параметра. Если задача, для которой он должен быть измен, активна, то она приостанавливается. Удаляется соответствующая метка в таймере. Параметр изменяется. Устанавливается новая метка в таймере. Если время реакции равно 0, то блокируются задачи с приоритетом меньшим, чем у данной.

* Удаление задачи.

Задача завершает своё выполнение и посылает планировщику сообщение на удаление её из списка готовых к выполнению. Или планировщик удаляет задачу, вышедшую за пределы выделенного ей времени выполнения.

* 1. Реализация протокола ARINC A.415 на основе разработанного модуля СРВ.
		1. Модель требований к системе.
			1. Описательная модель.

Протокола A.415 ARINC, используется во встроенных системах реального времени самолётов ведущих авиаперевозчиков, таких как Airbus, McDonnel Douglas и др. Это протокол опроса бортовых устройств, позволяющий в заранее обозначенный промежуток времени получить от них информацию и сигнализировать о неисправности в оборудовании.

Бортовые системы самолёта через жёстко заданные промежутки времени формируют специальные сообщения, в которых могут сообщать о возникновении внутри них неисправностей и описывать их. Специальные подсистемы самолёта должны получать сообщения бортовых систем и отправлять отчеты о неисправностях на диалоговую систему, предназначенную для взаимодействия с оператором в самолёте. Оператор, в свою очередь, должен иметь возможность акцентировать своё внимание на одной из контролируемых систем и вступить с ней во взаимодействие.

* + - 1. Модель случаев использования.

Данная модель представлена на рисунке 11.

У разрабатываемой системы будет 2 вида взаимодействий с «внешним окружением»: в Диалоговом режиме и в Обычном режиме. Диалоговый режим используется при взаимодействии с оператором в самолёте. Обычный режим используется при стандартной работе интерфейсной подсистемы по индикации неисправностей.

* + - 1. Функциональная модель.

Функциональная модель системы представлена в виде диаграмм 12 и 13.

В Обычном режиме система реализует следующие функции: начало работы (инициализация), тёплый старт, получение сообщения от бортовой системы, опрос APM (в случае необходимости), отсылка сообщений к CFDIU, переход в диалоговый режим.

В Диалоговом режиме система реализует следующие функции: получение сообщения от бортовой системы, отсылка сообщений к CFDIU, получение команд от CFDIU, переход в Нормальный режим.

* + 1. Динамическая модель.
			1. Модель объектов.
* Сообщение от бортовой системы.

Бортовая система ↔ OMSI И Бортовая система ↔ Энергонезависимая память

* Получить настройки из APM.

APM ↔ OMSI

* Отправить отчет к CFDIU.

OMSI ↔ Шина передачи данных ↔ CFDIU

ИЛИ

General Format Manager

* Запуск диалогового режима.

CFDIU ↔ Шина передачи данных ↔ OMSI

* Начало работы.

 Инициализация ИЛИ Тёплый старт

* General Format Manager.

Шина передачи данных ↔ OMSI

* Получить команду.

Шина передачи данных ↔ OMSI

* Инициализация.

OMSI ↔ APM

* Тёплый старт.

Энергонезависимая память ↔ OMSI И APM ↔ OMSI

* Получить команду от CFDIU.

CFDIU ↔ Шина передачи данных ↔ OMSI

* Запуск Обычного режима.

CFDIU ↔ Шина передачи данных ↔ OMSI

* + - 1. Модель взаимодействий.
* Сообщение от бортовой системы.

OMSI 🡪 начал работу. Бортовая система 🡪 начала работу. Бортовая система 🡪 сохранила сообщение о неисправности в Энергонезависимой памяти. Бортовая система 🡪 отправила сообщение о неисправности.

* Получить настройки из APM.

OMSI 🡪 начал работу. OMSI 🡪 запросил настройки у APM. APM 🡪 передало необходимые настройки.

* Отправить отчет к CFDIU.

CFDIU 🡪 начал работу. OMSI 🡪 начал работу. Шина передачи данных 🡪 активна. OMSI 🡪 проверил активность Шины передачи данных. OMSI 🡪 отправил отчет. Шина передачи данных 🡪 переправила отчет к CFDIU. CFDIU 🡪 получил отчет.

ИЛИ

General Format Manager.

* Запуск диалогового режима.

CFDIU 🡪 начал работу. OMSI 🡪 начал работу. Шина передачи данных 🡪 активна. CFDIU 🡪 отправил команду ENQ. Шина передачи данных 🡪 передаёт команду на нужный OMSI. OMSI 🡪 переходит в диалоговый режим.

* Начало работы.

 Инициализация ИЛИ Тёплый старт

* General Format Manager.

OMSI 🡪 начал работу. Шина передачи данных 🡪 неактивна. OMSI 🡪 проверил активность Шины передачи данных.

* Получить команду.

Шина передачи данных 🡪 активна. Шина передачи данных 🡪 передаёт команду на OMSI. OMSI 🡪 предпринимает действие в соответствии с командой.

* Инициализация.

OMSI 🡪 размещение в памяти, инициализация переменных, запрос настроек. APM 🡪 поиск и передача необходимых настроек.

* Тёплый старт.

OMSI 🡪 размещение в памяти, инициализация переменных, запрос настроек. APM 🡪 поиск и передача необходимых настроек. OMSI 🡪 восстановление ранее передававшегося сообщения из Энергонезависимой памяти.

* Получить команду от CFDIU.

CFDIU 🡪 начал работу. OMSI 🡪 начал работу. Шина передачи данных 🡪 активна. CFDIU 🡪 отправил команду. Шина передачи данных 🡪 передаёт команду на нужный OMSI. OMSI 🡪 предпринимает действие в соответствии с командой.

* Запуск Обычного режима.

CFDIU -> начал работу. OMSI -> начал работу. Шина передачи данных -> активна. CFDIU -> отправил команду Log Off. Шина передачи данных -> передаёт команду на нужный OMSI. OMSI -> переходит в номальный режим.

* + - 1. Поведенческая модель.
* OMSI.

Диаграмма 14. Старт 🡪 Проверить Энергозависимую память. 🡪 Выбор: Есть ли непереданные сообщения в Энергонезависимой памяти? Да – Инициализация; Нет – Тёплый старт. Тёплый старт 🡪 Забрать сообщения из Энергонезависимой памяти 🡪 Загрузить настройки из APM. Инициализация 🡪 Загрузить настройки из APM 🡪 Работа 🡪 Принять сообщения 🡪 Получить команду 🡪 Сформировать отчет 🡪 Проверить активность Шины передачи данных 🡪 Выбор: Шина активна? Да – Отослать вопрос; Нет – Ждать активизации Шины передачи данных. Отослать отчет 🡪 Принять сообщения. Ждать активизации шины 🡪 Отослать отчет.

* CFDIU.

Диаграмма 15. Старт 🡪 Включено 🡪 Направить команду (необязательное действие) 🡪 Получить отчет 🡪 Отобразить отчет 🡪 Выбор: Окончить работу? Да – Выключено; Нет 🡪 Направить команду (необязательное действие).

* APM.

Диаграмма 16. Старт 🡪 Получение запроса 🡪 Передать настройки 🡪 Получение запроса.

* Шина передачи данных.

Диаграмма 17. Старт 🡪 Неактивна 🡪 Включение CFDIU => Активна 🡪 Выбор: Перенаправление команды; Передача отчета. 🡪 Выбор: Выключение CFDIU => Неактивна; Активна.

* Бортовая система.

Диаграмма 18. Старт 🡪 Работает 🡪 Отправка сообщения в Энергонезависимую память 🡪 Отправка сообщения к OMSI 🡪 Работает.

* Энергонезависимая память.

Диаграмма 19. Старт 🡪 Активна 🡪 Записать сообщение от Бортовой системы 🡪 Передать сохраненные сообщения OMSI (необязательное действие)🡪 Уничтожить сообщения, не нужные OMSI 🡪 Активна.

* + 1. Статическая модель.
			1. Модель классов.

На диаграмме 9 представлены основные классы создаваемой системы: OMSI, CFDIU, Шина передачи данных, APM, Энергонезависимая память, Бортовая система.

* OMSI – интерфейсная система, обеспечивающая взаимосвязь функциональной системы, (например, T2CAS – системы предупреждения сближения самолетов, правильнее «предотвращения столкновения») с центральным устройством отображения данных CFDIU.
* Задача CFDIU предоставлять экипажу самолета данные о функционировании всех бортовых систем. С помощью меню экипаж (или техник на земле) может вступить во взаимодействие с конкретной функциональной бортовой системой (интерактивный режим). В остальных случаях CFDIU просто отображает (нормальный режим) состояние бортовых систем, которые через свои OMSI сообщают CFDIU свои состояния, посылая сообщения Label350.
* Процесс взаимодействия OMSI и CFDIU происходит через Шину передачи данных. Если шина не активна, то это может означать, что бортовая система к CFDIU не подключена, а, следовательно, некому слать сообщения.
* APM - это подсистема (таблица данных с интерфейсом) хранящая настройки данного самолета. Бортовая система типа T2CAS может ставиться на различные самолеты, и должна подстраиваться к работе конкретного борта. В частности, CFDIU не единственный вариант устройства отображения данных для экипажа. Могут быть и другие (на разных типах самолетов), тогда и протокол обмена реализуется с учетом соответствующей центральной системы.
* После сбоя в электропитании OMSI должен извлечь отчет о неисправности из энергонезависимой памяти, которая в данном случае будет представлена отдельным объектом создаваемой программной системы.
* Объект Бортовая система моделирует те сведения, которые должна иметь интерфейсная система о функциональной для взаимодействия.

**Заключение.**

Был проведён анализ предметной области систем реального времени. Определены основные отличия систем данного типа от других подобных систем и особенности управления исполнением задач. Были рассмотрены используемые классификации и отличительные особенности современных систем.

На основе проведенного анализа была спроектирована система, состоящая из двух основных подсистем: планировщика заданий реального времени и прикладного приложения – авиационного протокола.

Для обоих подсистем выполнены этапы создания системных и функциональных требований, определены используемые алгоритмы и архитектуры.

Для протокола использована современная методология разработки ПО и создана модель классов системы. В целом стоит отметить, что классы в проектируемой системе обладают простотой проектирования за счёт отсутствия иерархических связей, однако применяемый метод позволяет с относительной простотой усложнять структурные связи и расширять область проектирования.

На основе найденных при проектировании прикладного приложения недостатков используемой платформы в дальнейшем могут быть изменены функциональные или архитектурные особенности планировщика. Так же предполагается использование прикладного приложения для непосредственного тестирования планировщика.

**Литература.**

1. С. Кузнецов *«Механизмы IPC в операционной системе Unix». учебные материалы конференции «Индустрия Программирования 96», Центр Информационных Технологий, 1996.*
2. Алексей Быков *«Системное администрирование IBM AIX 4.x».*
3. Dr. Jurgen Sauermann, Melanie Thelen «*Real-time Operating Systems. Concepts and Implementation of Microkernels for Embedded Systems».*
4. See-Mong Tan, David K. Raila, Roy H. Campbell «*A case for nano-kernels».*  Department of Computer Science, University of Illinois at Urbana-Champaign, 1996, 11 стр.
5. Michel Gien *«Micro-kernel Architecture. Key to Modern Operating Systems Design».* Chorus systems, 1990, 10 стр.
6. Booch G. *«Object-oriented analysis and design with application, second edition»*. The Benjamin / Cummings Publishing Company, Inc, 1994, 589 стр.
7. Романовский К., Ивановский Б., Кознов Дм., Долгов П. «Обзор нотаций методологии Real». //http://www.tepcom.ru/produkts/real/Report\_Notations\_A .asp.
8. ITU *«SDL methodology guidelines and bibliography».* Appendices i to recommendation Z.100, 1993,107 стр.
9. Selic B., Gullekson G., Ward P.T. *«Real-time object-oriented modeling».* John Wiley & Sons. Inc, 1994, 525 стр.
10. ITU *«Recommendation Z.100: Specification and Description Language (SDL)».* 1993, 204 стр.
11. Бардзинь Я.М., Калкиньш А.А., Стродс Ю.Ф., Сыцко В.А. *«Язык спецификаций SDL/PLUS и его применения».* Рига, 1988, 313 стр.
12. IEEE Standards Project P1003.4a *«Thread Extension for Portable Operating Systems. Draft 6».* Draft 6.-IEEE, 1992.
13. Алан Джок «ОС реального времени».

**Приложение**

Диаграмма 2. Стандартные прикладные интерфейсы.

Таблица 3. Время отклика.

Таблица 4. Сравнение различных операционных систем.

Рисунок 5. ОС в пространстве "адресация-класс-стандартизация".

Диаграмма 6. Время реакции различных систем на прерывание

Диаграмма 7. Время переключения контекста

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ОСРВ** | **Разработчик** | **Область применения** | **Web-адрес** | **Комментарии** |
| **C Executive** | JIMI Software Systems | Коммерческая | *www.jmi.com* | Система реального времени для программ на Си; поддерживает процессоры архитектур CISC и RISC |
| **ITRON** | ITRON Committee, TRON Association | Коммерческая | *www.itron.gr.jp/home-e.html* | Спецификация разработана японской технологической ассоциацией; ориентирована на промышленные приложения |
| **LynxOS** | LynuxWorks | Коммерческая | *www.lynuxworks.com* | Совместима с Linux; поддерживает Unix и Java |
| **OS-9** | Microware Systems | Коммерческая | *www.microware.com* | Поддерживает микроархитектуру Intel XScale; модульная структура стимулирует добавление к системе новых устройств |
| **QNX** | QNX Software Systems | Коммерческая | *www.qnx.com* | Изолирует приложения, библиотеки, данные и системное программное обеспечение |
| **VxWorks, VxWorks AE** | Wind River Systems | Коммерческая | *www.windriver.com* | Позволяет изолировать совместно используемые приложения, библиотеки, данные и системное ПО |
| **Chimera** | Университет Карнеги- Меллона | Экспери-ментальная | *www.cs.cmu.edu/afs/ cs.cmu.edu/project/ chimera/www/chimera/ chimera.html* | Поддержка многозадачности и многопроцессорных систем; предназначена для роботов и автоматизированных систем |
| **Maruti** | Университет шт. Мэриленд | Экспери-ментальная | *www.cs.umd.edu/* | Поддерживает режимы "жесткого" и "мягкого" реального времени |

Таблица 8. Современные представители систем реального времени.

GeneralFormatManager

CFDIU

Энергонезависимая память

APM

Бортовая система

Шина активна

Шина неактивна

Шина передачи данных

Получение команды

Получена

Передача команды

Передана

Получение отчета

Получен

Получены

Получение сохраненного

Получено

Неполучено

Получение сообщения

Получено

Получение настроек

Сохранение сообщения

Сохранено

Несохранено

OMSI

Диаграмма 9. Основные классы системы протокола.

Процесс1

Таймер

Планировщик

ПроцессN

…

Главная програм ма

Событие-время

Запуск

ФункцияК

Запуск/Конец ФункцияК

Планировать точку

Функции

Активные

процессы

Диаграмма 10. Схема взаимодействия объектов СРВ.

Диалоговый режим

Обычный режим

CFDIU

OMSI

Рисунок 11. Модель случаев использования.

Обычный режим

Получить данные из APM

Отправить отчет к CFDIU

Запуск диалогового режима

Начать работу

Тёплый старт

Инициализация

General Format Manager

Получить команду

Получить сообщение от бортовой системы

Диаграмма 12. Обычный режим.

Диалоговый режим

Отправить отчет к CFDIU

Запуск обычного режима

General Format Manager

Получить команду от CFDIU

Получить сообщение от бортовой системы

Диаграмма 13. Диалоговый режим.

Старт

Есть ли непереданные сообщения?

Нет

Да

Забрать сообще-ния из Энергоза-висимой памяти

Загрузить настройки из APM

Тёплый старт

Инициализация

Проверить Энергозависимую память

Принять сообщения

Получить команду

Сформировать отчет

Проверить активность Шины передачи данных

Шина активна?

Нет

Да

Ждать активизации Шины передачи данных

Работа

Отправить отчет

Диаграмма 14. OMSI.

Старт

Да

Получить отчет

Отобразить отчет

Включено

Выключено

Направить команду

Окончить работу?

Нет

Диаграмма 15. CFDIU.

Старт

Передать настройки

Получение запроса

Диаграмма 16. APM.

Старт

Неактивна

Включение CFDIU

Активна

Перенаправление команды

Передача отчета

Выключение CFDIU

Диаграмма 17. Шина передачи данных.

Старт

Работает

Отправка сообщения в Энергозависимую память

Отправка сообщения к OMSI

Диаграмма 18. Бортовая система.

Старт

Активна

Записать сообщение от Бортовой системы

Передать сохраненные сообщения OMSI

Уничтожить сообщения, не нужные OMSI

Диаграмма 19. Энергозависимая память.