МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВПО «КубГУ»)**

**Физико-технический факультет**

**Кафедра теоретической физики и компьютерных технологий**

**КУРСОВАЯ РАБОТА**

**СПЕЦИФИКА ПОСТРОЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ**

**РЕАЛЬНОГО ВРЕМЕНИ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Левченко Никита Андреевич

Курс 2

Направление 09.03.02 Информационные системы и технологии

Научный руководитель

старший преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. Н. Значко

Нормоконтролер

канд. физ-мат. наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Мартынов

**СОДЕРЖАНИЕ**

Обозначения и сокращения…………………………………………………………3

Введение……….……………………………………………………………………..4

1 Операционные системы реального времени …………………………………5

1.1 Понятие операционных систем реального времени ……..…………….5

1.2 АрхитектураОСРВ….……...…………….……………………………….6

1.3 Планирование задач....………...………………………………………….8

1.4 Выделение памяти в ОСРВ..…………………………………………….11

1.5 Стандартизация ОСРВ…………………………………………………..11

1.6 Стандарты безопасности………………………………………….……..12

2 Варианты реализации ОСРВ............................................................................16

2.1 Краткие характеристики наиболее распространенных ОСРВ.…….…16

### 2.2 VxWorks……………………………………………………………….….16

### 2.3 QNX Neutrino RTOS……………….…...………………………………..21

### 2.4 ChorusOS…………………………………………………………………23

### Заключение…...……………………………………………………………………..28

### Список использованных источников……………………………….......................29

**ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

|  |  |
| --- | --- |
| ОС | Операционная система |
| ОСРВ | Операционная система реального времени |
| POSIX | Portable Operating System Interface for Computer Environments |
| RTOS | Real Time Operation System |

**ВВЕДЕНИЕ**

Развитие технологий производства promPC (промышленных ПК) привели к ужесточению конкуренции на рынке систем автоматизации производства между аппаратными решениями на базе микроконтроллеров и программными комплексами, работающими под управлением специализированных операционных систем (ОС) на интегрированных promPC платформах.

Операционные системы реального времени (ОСРВ) предназначены для обеспечения интерфейса к ресурсам критических по времени систем реального времени. Основной задачей в таких системах является своевременность (timeliness) выполнения обработки данных.

Целью данной курсовой работы является изучение специфики построения операционных систем реального времени и их технических характеристик.

Задачами курсовой работы являются:

* изучение требований, выдвигаемых ОСРВ;
* исследование архитектуры построения ОСРВ;
* рассмотрение распределения памяти ОСРВ;
* изучение характеристик наиболее распространенных ОСРВ;
* нахождение оптимальной ОСРВ.

В данной работе объектом исследования являются операционные

системы QNX, VxWorks и ChorusOS.

1. **Операционные системы реального времени**

**1.1 Понятие операционных систем реального времени**

В качестве основного требования к операционным системам реального времени (ОСРВ) выдвигается требование обеспечения **предсказуемости** или **детерминированности** поведения системы в наихудших внешних условиях, что резко отличается от требований к производительности и быстродействию универсальных ОС. Хорошая ОСРВ имеет предсказуемое поведение при всех сценариях системной загрузки (одновременные прерывания и выполнение потоков).

Операционные системы реального времени иногда делят на два типа — системы жесткого реального времени и системы мягкого реального времени. В системах жесткого реального времени неспособность обеспечить реакцию на какие-либо события в заданное время ведет к отказам и невозможности выполнения поставленной задачи. В большинстве русскоязычной литературы такие системы называют системами с детерминированным временем. При практическом применении время реакции должно быть минимальным. Системами мягкого реального времени называются системы, не попадающие под определение «жесткие». Системы мягкого реального времени могут не успевать решать задачу, но это не приводит к отказу системы в целом. В системах реального времени необходимо введение некоторого директивного срока (deadline), до истечения которого задача должна обязательно (для систем мягкого реального времени – желательно) выполниться. Этот директивный срок используется планировщиком задач как для назначения приоритета задачи при ее запуске, так и при выборе задачи на выполнение.

Системы жёсткого реального времени не допускают задержек реакции системы, так как это может привести к:

* потере актуальности результатов;
* большим финансовым потерям;
* авариям и катастрофам.

Большинство программного обеспечения ориентировано на «мягкое» реальное время. Для подобных систем характерно:

* гарантированное время реакции на внешние события (прерывания от оборудования);
* жёсткая подсистема планирования процессов (высокоприоритетные задачи не должны вытесняться низкоприоритетными, за некоторыми исключениями);
* повышенные требования к времени реакции на внешние события или реактивности (задержка вызова обработчика прерывания не более десятков микросекунд, задержка при переключении задач не более сотен микросекунд)[1].
	1. **Архитектура ОСРВ**

В своем развитии ОСРВ строились на основе следующих архитектур:

* **Монолитная архитектура (рисунок 1)**. ОС определяется как набор модулей, взаимодействующих между собой внутри ядра системы и предоставляющих прикладному ПО входные интерфейсы для обращений к аппаратуре. Основной недостаток этого принципа построения ОС заключается в плохой предсказуемости её поведения, вызванной сложным взаимодействием модулей между собой.



Рисунок 1 – Монолитная архитектура.

* **Уровневая (многослойная) архитектура (рисунок 2)**. Прикладное ПО имеет возможность получить доступ к аппаратуре не только через ядро системы и её сервисы, но и напрямую. По сравнению с монолитной такая архитектура обеспечивает значительно большую степень предсказуемости реакций системы, а также позволяет осуществлять быстрый доступ прикладных приложений к аппаратуре. Главным недостатком таких систем является отсутствие многозадачности.



Рисунок 2 – Уровневая(многослойная) архитектура

* **Архитектура «клиент-сервер» (рисунок 3)**. Основной её принцип заключается в вынесении сервисов ОС в виде серверов на уровень пользователя и выполнении микроядром функций диспетчера сообщений между клиентскими пользовательскими программами и серверами — системными сервисами. Преимущества такой архитектуры:
* Повышенная надёжность, так как каждый сервис является, по сути, самостоятельным приложением и его легче отладить и отследить ошибки;
* Улучшенная масштабируемость, поскольку ненужные сервисы могут быть исключены из системы без ущерба к её работоспособности;
* Повышенная отказоустойчивость, так как «зависший» сервис может быть перезапущен без перезагрузки системы[2].



Рисунок 3 – Архитектура "клиент-сервер"

* 1. **Планирование задач**

В связи с проблемой дедлайнов главной проблемой в ОСРВ становится планирование задач (scheduling), которое обеспечивало бы предсказуемое поведение системы при всех обстоятельствах. Процесс с дедлайнами должен стартовать и выполняться так, чтобы он не пропустил ни одного своего дедлайна. Если это невозможно, процесс должен быть отклонен.

В связи с проблемами планирования в ОСРВ изучаются и развиваются два подхода – статические алгоритмы планирования (RMS – Rate Monotonic Scheduling) и динамические алгоритмы планирования (EDF – Earliest Deadline First).

RMS используется для формального доказательства условий предсказуемости системы. Для реализации этой теории необходимо планирование на основе приоритетов, прерывающих обслуживание. В теории RMS приоритет заранее назначается каждому процессу. Процессы должны удовлетворять следующим условиям:

* процесс должен быть завершен за время его периода,
* процессы не зависят друг от друга;
* каждому процессу требуется одинаковое процессорное время на каждом интервале;
* у непериодических процессов нет жестких сроков;
* прерывание процесса происходит за ограниченное время.

Процессы выполняются в соответствии с приоритетами. При планировании RMS предпочтение отдается задачам с самыми короткими периодами выполнения.

В EDF приоритет присваивается динамически, и наибольший приоритет выставляется процессу, у которого осталось наименьшее время выполнения. При больших загрузках системы у EDF имеются преимущества перед RMS.

Во всех системах реального времени требуется политика планирования, управляемая дедлайнами (deadline – driven scheduling). Однако этот подход находится в стадии разработки.

Обычно в ОСРВ используется планирование с приоритетами, прерывающими обслуживание, которое основано на RMS. Приоритетное прерывание обслуживания (preemption) является неотъемлемой составляющей ОСРВ, т.к. в системе реального времени должны существовать гарантии того, что событие с высоким приоритетом будет обработано перед событием более низкого приоритета. Все это ведет к тому, что ОСРВ нуждается не только в механизме планирования на основе приоритетов, прерывающих обслуживание, но также и в соответствующем механизме управления прерываниями. Более того, ОСРВ должна быть способна запрещать прерывания, когда необходимо выполнить критический код, который нельзя прерывать. Длительность обработки прерываний должна быть сведена к минимуму.

ОСРВ должна обладать развитой системой приоритетов. Во-первых, это требуется потому, что система сама может рассматриваться как набор серверных приложений, подразделяющихся на потоки, и несколько высоких уровней приоритетов должно быть выделено системным процессам и потокам. Во-вторых, в сложных приложениях необходимо все потоки реального времени помещать на разные приоритетные уровни, а потоки не реального времени помещать на один уровень (ниже, чем любые потоки реального времени). При этом потоки не реального времени можно обрабатывать в режиме циклического планирования (RRS – round-robinscheduling), при котором каждому процессу предоставляется квант времени процессора, а когда квант заканчивается, контекст процесса сохраняется, и он ставится в конец очереди. Во многих ОСРВ для планирования задач на одном уровне используется RRS.

При планировании на основе приоритетов необходимо решить две обязательные проблемы:

* обеспечить выполнение процесса с наивысшим приоритетом;
* не допустить инверсии приоритетов, когда задачи с высокими приоритетами ожидают ресурсы, захваченные задачами с более низкими приоритетами.

Для борьбы с инверсией приоритетов в ОСРВ часто используется механизм наследования приоритетов, однако при этом приходится отказываться от планирования на основе RMS, поскольку приоритеты становятся динамическими.

**1.4 Выделение памяти в ОСРВ**

Следующим проблемам выделения памяти в ОСРВ уделяется больше внимания, нежели в операционных системах общего назначения.

Во-первых, скорости выделения памяти. Стандартная схема выделения памяти предусматривает сканирование списка неопределённой длины для нахождения свободной области памяти заданного размера, а это неприемлемо, так как в ОСРВ выделение памяти должно происходить за фиксированное время.

Во-вторых, память может стать фрагментированной в случае разделения свободных её участков уже запущенными процессами. Это может привести к остановке программы из-за её неспособности задействовать новый участок памяти. Алгоритм выделения памяти, постепенно увеличивающий фрагментированность памяти, может успешно работать на настольных системах, если те перезагружаются не реже одного раза в месяц, но является неприемлемым для встроенных систем, которые работают годами без перезагрузки.

Простой алгоритм с фиксированной длиной участков памяти очень хорошо работает в несложных встроенных системах.

Также этот алгоритм отлично функционирует и в настольных системах, особенно тогда, когда во время обработки участка памяти одним ядром следующий участок памяти обрабатывается другим ядром.

 **1.5 Стандартизация ОСРВ**

Большие различия в спецификациях ОСРВ и огромное количество существующих микроконтроллеров выдвигают на передний план проблему стандартизации в области систем реального времени.

Наиболее ранним и распространенным стандартом ОСРВ является стандарт **POSIX** (IEEE Portable Operating System Interfacefor Computer Environments IEEE 1003.1). Первоначальный вариант стандарта POSIX появился в 1990 г. и был предназначен для UNIX-систем, первые версии которых появились в 70-х годах прошлого века. Спецификации POSIX определяют стандартный механизм взаимодействия прикладной программы и операционной системы и в настоящее время включают набор более чем из 30 стандартов. Для ОСРВ наиболее важны семь из нихно широкую поддержку в коммерческих ОС получили только три первых.

Несмотря на явно устаревшие положения стандарта **POSIX** и большую востребованность обновлений стандартизации для ОСРВ, заметного продвижения в этом направлении не наблюдается.

Военная и аэрокосмическая отрасли предъявляют жесткие требования к вычислительным средствам, влияющим на степень безопасности целевой системы. В настоящее время имеются следующие стандарты для ОСРВ в авиации – стандарт **DO-178B** и стандарт **ARINC-653**. Поскольку эти стандарты разработаны в США, стоит отметить еще европейский стандарт **ED-12B**, который является аналогом **DO-178B.**

Распространенным также является стандарт **OSEK/VDX**, который первоначально развивался для систем автомобильной индустрии.

**1.6 Стандарты безопасности**

В связи со стандартами для ОСРВ стоит отметить широко известный стандарт критериев оценки пригодности компьютерных систем (Trusted Computer System Evaluation Criteria – TCSEC). Этот стандарт разработан Министерством обороны США и известен также под названием "Оранжевая книга" (Orange Book – из-за цвета обложки).

В ряде других стран были разработаны аналогичные критерии, на основе которых был создан международный стандарт “Общие критерии оценки безопасности информационных технологий” (далее просто – Общие критерии) (Common Criteriafor IT Security Evaluation, ISO/IEC 15408).

В "Оранжевой книге" перечислены семь уровней защиты:

* А1 – верифицированная разработка. Этот уровень требует, чтобы защиту секретной и другой критичной информации средствами управления безопасностью гарантировали методы формальной верификации;
* В3 – домены безопасности. Этот уровень предназначен для защиты систем от опытных программистов;
* В2 – структурированная защита. В систему с этим уровнем защиты нельзя допустить проникновение хакеров;
* В1 – мандатный контроль доступа. Защиту этого уровня, возможно, удастся преодолеть опытному хакеру, но никак не рядовым пользователям;
* С2 – дискреционный контроль доступа. Уровень С2 обеспечивает защиту процедур входа, позволяет производить контроль за событиями, имеющими отношение к безопасности, а также изолировать ресурсы;
* С1 – избирательная защита. Этот уровень дает пользователям возможность защитить личные данные или информацию о проекте, установив средства управления доступом;
* D – минимальная защита. Этот нижний уровень защиты оставлен для систем, которые проходили тестирование, но не смогли удовлетворить требованиям более высокого класса.

Что касается общих критериев, то в них введены похожие требования обеспечения безопасности в виде оценочных уровней (Evaluation Assurance Levels – EAL). Их также семь:

* EAL7 – самый высокий уровень предполагает формальную верификацию модели объекта оценки. Он применим к системам очень высокого риска;
* EAL6 определяется, как полуформально верифицированный и протестированный. На уровне EAL6 реализация должна быть представлена в структурированном виде, анализ соответствия распространяется на проект нижнего уровня, проводится строгий анализ покрытия, анализ и тестирование небезопасных состояний;
* EAL5 определяется, как полуформально спроектированный и протестированный. Он предусматривает создание полуформальной функциональной спецификации и проекта высокого уровня с демонстрацией соответствия между ними, формальной модели политики безопасности, стандартизованной модели жизненного цикла, а также проведение анализа скрытых каналов;
* EAL4 определяется, как методически спроектированный, протестированный и пересмотренный. Он предполагает наличие автоматизации управления конфигурацией, полной спецификации интерфейсов, описательного проекта нижнего уровня, подмножества реализаций функций безопасности, неформальной модели политики безопасности, модели жизненного цикла, анализ валидации, независимый анализ уязвимостей. По всей вероятности, это самый высокий уровень, которого можно достичь на данном этапе развития технологии программирования с приемлемыми затратами;
* EAL3 определяется, как методически протестированный и проверенный. На уровне EAL3 осуществляется более полное, чем на уровне EAL2, тестирование покрытия функций безопасности, а также контроль среды разработки и управление конфигурацией объекта оценки;
* EAL2 определяется, как структурно протестированный. Он предусматривает создание описательного проекта верхнего уровня объекта оценки, описание процедур инсталляции и поставки, руководств администратора и пользователя, функциональное и независимое тестирование, оценку прочности функций безопасности, анализ уязвимостей разработчиками;
* EAL1 определяется, как функционально протестированный. Он обеспечивает анализ функций безопасности с использованием функциональной спецификации и спецификации интерфейсов, руководящей документации, а также независимое тестирование. На этом уровне угрозы не рассматриваются как серьезные.

В соответствии с требованиями Общих критериев, продукты определенного класса (например, операционные системы) оцениваются на соответствие ряду функциональных критериев и критериев доверия – профилей защиты. Существуют различные определения профилей защиты в отношении операционных систем, брандмауэров, смарт-карт и прочих продуктов, которые должны соответствовать определенным требованиям в области безопасности. Например, профиль защиты систем с разграничением доступа (Controlled Access Protection Profile) действует в отношении операционных систем и призван заменить старый уровень защиты С2, определявшийся в соответствии с американским стандартом TCSEC. В соответствии с оценочными уровнями доверия сертификация на соответствие более высокому уровню означает более высокую степень уверенности в том, что система защиты продукта работает правильно и эффективно, и, согласно условиям Общих критериев, уровни 5-7 рассчитаны на тестирование продуктов, созданных с применением специализированных технологий безопасности.

Следует отметить, что большинство усилий по оценке продуктов безопасности сосредоточены на уровне 4 стандарта Общих критериев и ниже, что говорит об ограниченном применении формальных методов в этой области.

С точки зрения программиста Общие критерии можно рассматривать как набор библиотек, с помощью которых пишутся задания по безопасности, типовые профили защиты и т.п. Следует отметить, что требования могут быть параметризованы.

**2 Варианты реализации ОСРВ**

### 2.1 Краткие характеристики наиболее распространенных ОСРВ4

Большинство распространенных ОСРВ являются проприетарными, поэтому информация о них не всегда доступна. В этом разделе описаны наиболее распространенные ОСРВ в порядке объема собранных о них сведений.

### 2.2 VxWorks

Операционные системы реального времени семейства **VxWorks** корпорации **Wind River Systems** предназначены для разработки программного обеспечения (ПО) встраиваемых компьютеров, работающих в системах жесткого реального времени. Операционная система **VxWorks** обладает кросс-средствами разработки программного обеспечения (ПО), т.е. разработка ведется на инструментальном компьютере (host) в среде Tornado для дальнейшего ее использования на целевом компьютере (target) под управлением системы **VxWorks**.

Операционная система **VxWorks** имеет архитектуру клиент-сервер и построена в соответствии с технологией микроядра, т.е. на самом нижнем непрерываемом уровне ядра (**WIND** *Microkernel*) обрабатываются только планирование задач и управление их взаимодействием/синхронизацией. Вся остальная функциональность операционного ядра – управление памятью, вводом/выводом и пр. – обеспечивается на более высоком уровне и реализуется через процессы. Это обеспечивает быстродействие и детерминированность ядра, а также масштабируемость системы.

**VxWorks** может быть скомпонована как для небольших встраиваемых систем с жесткими ограничениями для памяти, так и для сложных систем с развитой функциональностью. Более того, отдельные модули сами являются масштабируемыми. Конкретные функции можно убрать при сборке, а специфические ядерные объекты синхронизации можно опустить, если приложение в них не нуждается.

Хотя система **VxWorks** является конфигурируемой, т.е. отдельные модули можно загружать статически или динамически, нельзя сказать, что в ней используется подход, основанный на компонентах. Все модули построены над базовым ядром и спроектированы таким образом, что не могут использоваться в других средах.

Ядро **VxWorks** обладает следующими параметрами:

* количество задач не ограничено;
* число уровней приоритетов задач – 256;
* планирование задач возможно двумя способами – вытеснение по приоритетам и циклическое;
* средствами взаимодействия задач служат очереди сообщений, семафоры, события и каналы (для взаимодействия задач внутри CPU), сокеты и удаленные вызовы процедур (для сетевого взаимодействия), сигналы (для управления исключительными ситуациями) и разделяемая память (для разделения данных);
* для управления критическими системными ресурсами обеспечивается несколько типов семафоров: двоичные, вычислительные (counting) и взаимно исключающие с приоритетным наследованием;
* поддерживается детерминированное переключение контекста.

В **VxWorks** обеспечивается как основанный на POSIX, так и собственный механизмы планирования (windscheduling). Оба варианта включают вытесняющее и циклическое планирование. Различие между ними состоит в том, что windscheduling применяется на системном базисе, в то время как алгоритмы POSIX-планирования применяются на базисе процесс-за-процессом.

В **VxWorks** все задачи системы и приложений разделяют единственное адресное пространство, что чревато нарушением стабильности системы из-за неисправности какого-либо приложения. Необязательный компонент VxVMI дает возможность каждому процессу иметь свою собственную виртуальную память.

Чтобы достичь быстрой обработки внешних прерываний, программы обработки прерываний (ISRs – interruptserviceroutines) в **VxWorks** выполняются в специальном контексте вне контекстов потоков, что позволяет выиграть время, которое обычно тратится на переключение контекстов. Следует отметить, что C-функция, которую пользователь присоединяет к вектору прерывания, на самом деле не является фактической ISR. Прерывания не могут непосредственно обращаться к C-функциям. Адрес ISR запоминается в таблице векторов прерываний, которая вызывается аппаратно. ISR выполняет некую начальную обработку (сохранение регистров и подготовку стека), а затем вызывается C-функция, которая была присоединена пользователем.

**VSPWorks**– это весьма популярная и достаточно мощная ОС на основе **VxWorks**. **VSPWorks** спроектирована специально для систем, основанных на DSP. Она обеспечивает многозадачный режим с приоритетами и поддержку быстрых прерываний на процессорах DSP и ASIC. ОСРВ **VSPWorks** следует модели единственного виртуального процессора, что значительно упрощает распределение приложений в многопроцессорной системе, сохраняя при этом производительность жесткого реального времени. **VSPWorks** является модульной и масштабируемой.

ОСРВ **VSPWorks** обладает многослойной структурой (рисунок 4), что служит хорошей основой для абстрагирования и переносимости. Центром системы служит сильно оптимизированное наноядро (nanokernel), которое способно управлять совокупностью процессов. Ниже наноядра находятся программы, обслуживающие прерывания, выше наноядра располагается микроядро, которое управляет многозадачным режимом с приоритетами C/C++ задач.



Рисунок 4 – Многослойная архитектура VSPWorks

Управление прерываниями имеет два уровня. Нижний уровень (уровень 1) используется для обработки аппаратных прерываний. Во время обработки таких прерываний все остальные прерывания блокируются. Код, выполняющийся на этом уровне, написан на языке ассемблера, и ответственность за сохранение соответствующих регистров в стеке ложится на программиста. На этом уровне может быть обработано прерывание, которое требует малого времени для обработки. Если обработка прерывания является более сложной и требует большего времени, то прерывание обрабатывается на более высоком уровне (уровень 2), где разрешено прерывание прерывания и, таким образом, они могут быть вложенными. Переход на более высокий уровень прерываний происходит по системному вызову.

Процессы наноядра (уровень 3) пишутся на языке ассемблера и имеют сокращенный контекст (т.е. используют меньше регистров). Эти процессы могут быть загружены и разгружены с процессора очень быстро. Каждому процессу присваивается приоритет. Уровень 3 идеален для написания драйверов для интерфейсов аппаратуры низкого уровня.

Микроядро находится на уровне 4. Микроядро написано на языке C и имеет свыше 100 сервисов. Обработка задач на этом уровне ведется в режиме приоритетного прерывания, и планирование управляется приоритетами.

**Средства построения мультипроцессорных систем. VxWorks** поддерживает два вида мультипроцессинга: слабосвязанный – через распределенные очереди сообщений и сильносвязанный – через объекты в разделяемой памяти. Слабосвязанный мультипроцессинг через распределенные очереди сообщений реализован в библиотеке VxFusion, которая является отдельным продуктом. VxFusion применяется для обмена между процессорами, не имеющими общей памяти (например, между узлами сети). Сильносвязанный мультипроцессинг через объекты в разделяемой памяти реализован в библиотеке VxMP, которая также является отдельным продуктом. VxMP применяется для обмена между процессорами, имеющими общую область памяти (например, находящимися на одной шине).

**Средства портирования.** Все аппаратно-зависимые части **VxWorks** вынесены в отдельные модули для того, чтобы разработчик встраиваемой компьютерной системы мог сам портировать **VxWorks** на свой нестандартный целевой компьютер. Этот комплект конфигурационных и инициализационных модулей называется BSP (BoardSupportPackage) и поставляется для стандартных компьютеров (VME-процессор, PC или Sparcstation) в исходных текстах. Разработчик нестандартного компьютера может взять за образец BSP наиболее близкого по архитектуре стандартного компьютера и портировать**VxWorks** на свой компьютер путем разработки собственного BSP с помощью BSPDeveloper'sKit.

**Промежуточное ПО (middleware).** Модель компонентных объектов COM (ComponentObjectModel) и ее расширение для распределенных систем DCOM (Distributed COM) являются стандартными интерфейсами обмена между приложениями для Windows. VxDCOM – DCOM для операционной системы **VxWorks** – это первая реализация модели распределенных компонентных объектов для систем реального времени. Теперь нет необходимости в разработке специализированных драйверов ввода/вывода при интеграции нижнего и верхних уровней распределенной системы управления. VxDCOM поддерживает также OPC-интерфейсы (OLE forProcessControl), что позволяет разрабатывать OPC-серверы для встраиваемых систем, работающих под управлением ОСРВ **VxWorks**.

### 2.3 QNX Neutrino RTOS

Операционная система QNX Neutrino Realtime Operating System (RTSO) корпорации QNX SoftwareSystems является микроядерной операционной системой, которая обеспечивает многозадачный режим с приоритетами. QNX Neutrino RTOS имеет клиент-серверную архитектуру. В среде QNX Neutrino каждый драйвер, приложение, протокол и файловая система выполняются вне ядра, в защищенном адресном пространстве. В случае сбоя любого компонента он может автоматически перезапуститься без влияния на другие компоненты или ядро. Хотя система QNX является конфигурируемой, т.е. отдельные модули можно загружать статически или динамически, нельзя сказать, что она использует подход, основанный на компонентах. Все модули полагаются на базовое ядро и спроектированы таким образом, что не могут использоваться в других средах.

**QNX Neutrino RTOS** состоит из ядра, планировщика процессов (processmanager) и расширенных сервисов на уровне пользователя. Как истинная микроядерная операционная система, **QNX Neutrino RTOS** реализует в ядре ОС только наиболее фундаментальные сервисы, такие как передача сообщений, сигналы, таймеры, планирование потоков, объекты синхронизации. Все другие сервисы ОС, драйверы и приложения выполняются как отдельные процессы, которые взаимодействуют через синхронную передачу сообщений.

Ядро **QNX Neutrino RTOS** выполняется на уровне 0, управляющие программы и драйверы устройств выполняются на уровнях 1 и 2, совершая операции ввода/вывода. Приложения выполняются на уровне 3.

Планировщик процессов строится на базисе ядра и обеспечивает дополнительную семантику уровня процессов, управление памятью и путями доступа к файлам. Все другие компоненты – файловые системы, набор протоколов, очереди сообщений, приложения – выполняются в защищенном адресном пространстве и являются расширенными сервисами. Взаимодействие компонентов осуществляется через передачу сообщений. Передача сообщений играет роль виртуальной “программной шины”, которая позволяет оперативно динамически подгружать и отгружать любой компонент. Как следствие, любой модуль, даже драйвер устройства, может быть замещен или перезапущен оперативно, для чего в большинстве ОСРВ требуется перезапуск системы. Сообщения передаются прозрачно через границы процессора, обеспечивая бесшовный доступ к любому ресурсу в сети.

Обладая вытесняющим микроядром и планировщиком с приоритетным обслуживанием, **QNX Neutrino RTOS** способна быстро и с высокой предсказуемостью реагировать на события реального времени. Высокоприоритетные потоки обрабатывают дедлайны своевременно даже при большой загрузке системы (рисунок 5).



Рисунок 5 ­­– Производительность реального времени QNX Neutrino RTOS.

**QNX Neutrino RTOS** имеет малые времена обработки прерываний, быстрое переключение контекстов. Инверсия приоритетов преодолевается с помощью распределенного наследования приоритетов. Упрощенное моделирование активностей реального времени проводится через синхронную передачу сообщений. Вложенные прерывания и фиксированная верхняя граница времени обработки прерывания гарантируют, что высокоприоритетные прерывания обрабатываются быстро с предсказуемым временем[3].

### 2.4 ChorusOS

Операционная система **ChorusOS –** это масштабируемая встраиваемая ОС, широко применяемая в телекоммуникационной индустрии. В настоящее время этот бренд развивается и распространяется корпорацией SunMicrosystems. Для компоновки и развертывания ОС **ChorusOS** на конкретных телекоммуникационных платформах SunMicrosystems предлагает использовать среду разработки SunEmbeddedWorkshop. Корпорация SunMicrosystems представляет ОС **ChorusOS** как встраиваемую основу для Sun’овской сети, управляемой сервисами (Sun'sService-DrivenNetwork). В сочетании с широким набором сервисов, полной интеграцией ПО и аппаратуры, удобным администрированием и поддержкой Java-технологии, которая посвящена нуждам телекоммуникации, ОС **ChorusOS** дает возможность эффективно развертывать новые возможности и приложения, поддерживая надежность и функциональность современных сетей.

ОС **ChorusOS** поддерживает на одной аппаратной платформе широкий набор телекоммуникационных протоколов, унаследованных приложений, приложений режима реального времени и Java-технологии.

ОС **ChorusOS** моделирует три сорта приложений:

* POSIX-процессы составляют большинство приложений **ChorusOS**; эти приложения имеют доступ к чисто POSIX API, нескольким POSIX-подобным расширенным API и небольшому числу ограниченных системных вызовов микроядра;
* Акторы **ChorusOS –** эти приложения выполняются над микроядром и ограничиваются API микроядра, акторы включают драйверы, события подсистем и протокольные стеки;
* Унаследованные приложения **ChorusOS** поддерживаются для совместимости с приложениями, разработанными для более ранних версий **ChorusOS.**

Архитектура ОС **ChorusOS** является многослойной, основанной на компонентах (component-based). Микроядро содержит минимальный набор компонентов, необходимых для функционирования ОС

* kern – реализует интерфейс микроядра и содержит актор KERN, вспомогательную библиотеку и заголовочные файлы;
* менеджер приватных данных (pd) реализует интерфейс между подсистемами микроядра;
* менеджер постоянной памяти (pmm) реализует интерфейс постоянной памяти;
* coreexecutive обеспечивает существенную часть поддержки реального времени.

Компонент диспетчера ядра (coreexecutive) обеспечивает следующую функциональность

* поддержка многочисленных независимых приложений;
* поддержка пользовательских и системных приложений;
* поддержка актора – единицы модуляризации приложений;
* поддержка единицы исполнения – потока;
* операции управления потоками;
* управление Local Access Point (LAP);
* сервисы управления исключительными ситуациями;
* минимальный сервис управления прерываниями.

В coreexecutive отсутствует управление такими сущностями, как синхронизация, планирование, время, память. Политики управления этими понятиями обеспечиваются дополнительными компонентами, которые выбираются пользователем в зависимости от требований аппаратных и программных средств. Coreexecutive всегда присутствует в исполняемом экземпляре ОС **ChorusOS,** остальные компоненты конфигурируются и добавляются по необходимости. Размер резидентной часть ядра составляет 10Kb[4].

Понятие “актор” в **ChorusOS** определяется как единица загрузки для приложения. Оно также служит единицей инкапсуляции для того, чтобы сопоставить все системные ресурсы, используемые приложением, и потоки, выполняющиеся внутри актора. Примерами таких ресурсов являются потоки, регионы памяти и конечные точки взаимодействия[5].

ОС **ChorusOS 5.0** лежит в основе операционной среды Solaris и поддерживает следующие целевые платформы:

* UltraSPARC II (CP1500 и CP20x0);
* Intel x86, Pentium, MotorolaPowerPC 750 и семейство процессоров 74x0 (mpc7xx).

Таблица 1 – Сравнительная характеристика ОСРВ

|  |  |  |  |
| --- | --- | --- | --- |
|  | **VxWorks** | **QNX** | **ChorusOS** |
| **Архитектура** | Клиент-сервер, микроядро WIND Microkernel | Клиент-сервер, микроядро и взаимодействующие процессы | Многослойная |
| **Предсказуемая производитель–ность реального времени** | Приоритетное планирование в двух вариантах, наследование приоритетов | Приоритетное планирование с выбором методов планирования. Наследование приоритетов | Приоритетное планирование, таймеры с высокой разрешающей способностью, MIPC |
| **Что реализует микроядро** | Многозадачность, планирование, переключение контекста, взаимодействие /синхронизация задач, управление разделяемой и динамической памятью, управление прерываниями | Потоки, сигналы, передача сообщений, синхронизация, планирование, временные сервисы | Многозадачность, поддержка факторов, управление потоками, управление LAP, управление исключительными ситуациями, минимальное управления прерываниями |
| **POSIX** | POSIX 1003.1, .1b, .1c (включая Threads) | POSIX 1003.1-2001, с потоками и расширенным. РВ | POSIX-сигналы, сигналы реального времени, потоки,  |
| Продолжение таблицы 1 |
|  |  |  | таймеры, очереди сообщений, семафоры. сокеты, разделяемая память |
| **Целевые платформы** | x86, PowerPC, ARM, MIPS, 68K, CPU 32, ColdFire, MCORE, Pentium, i960, SH, SPARC, NEC V8xx, M32  | ARM, MIPS, PowerPC, SH4, Strong ARM, XScale, x86 | UltraSPARC II (CP1500 и CP20x0), Intel x86, Pentium, Motorola PowerPC 750 |
| **Модель защиты** | -без защиты-защита виртуальной памяти (VxVMI) | Защита виртуальной памяти | -защищенная память-защита виртуальной памяти |
| **Подкачка** | Нет  | Да | Да |
| **Поддержка MMU** | Не требуется | Да | В зависимости от конфигурации |
| **Виртуальная память** | Да (для VxVMI) | Да | Да  |

**ЗАКЛЮЧЕНИЕ**

Основные результаты курсовой работы состоят в следующем:

1 Изучена специфика построения ОСРВ. Проведено исследование их характеристик.

2 Была найдена оптимальная, на мой взгляд, ОС реального времени – ChorusOS. На этой есть несколько причин:

– она обладает всеми видами защиты памяти;

­– предсказуема;

– некоторые версии имеют поддержку MMU

– является легко интегрируемой.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Бурдонов И.Б., Косачев А.С., Пономаренко В.Н. Операционные системы реального времени – Спб.: Институт системного программирования РАН, 2006.– 78 с.

2 ВикипедиЯ, Операционная система реального времени-(Рус)-URL: http://ru.wikipedia.org/wiki/RTOS [6 мая 2015].

3 Кондукова Е., Кузьмина А. Операционная система реального времени QNX Neutrino 6.5.0. Системная архитектура – Спб.: БХВ-Петербург, 2014.– 400с.

4 Интегрированные системы управления и проектирования-(Рус)- URL: http://www.studfiles.ru/dir/cat32/subj1451/file16334/view156454/page2.html [1 мая 2015].

5 Операционные системы реального времени для начинающих-(Рус)- URL: http:// http://habrahabr.ru/post/208780 [30 апреля 2015].