МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по научно-исследовательской работе

Тема: “Исследование производительности Real Time Linux для встроенных систем”

Студент гр. 3303 _____ Мирошниченко Е.А.

Руководитель _____ Кринкин К.В.

Санкт-Петербург

2018

ЗАДАНИЕ
НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ

Студент Мирошниченко Е.А.

Группа 3303

Тема НИР: Исследование производительности Real Time Linux для встроенных систем.

Задание на НИР:

В ходе выполнения данной научной исследовательской работы предстоит провести исследование производительности микроядерной операционной системы реального времени – RT Linux, для специализированной микропроцессорной системы управления. Однако стандартное ядро ОС Linux не обеспечивает такие возможности "жёсткого" реального времени, как гарантированное время отклика и микросекундные задержки, которые требуются во многих встраиваемых устройствах. Исходя из этого были определены следующие задачи научной исследовательской работы:

- Изучение основ систем реального времени;
- Исследование производительности операционной системы реального времени для встроенных систем;
- Проведение сравнительного анализа методов вычислений в реальном времени;

Сроки выполнения НИР: 13.02.2018 – 27.05.2018

Дата сдачи отчета: 24.05.2018

Дата защиты отчета: 24.05.2018

Студент _____ Мирошниченко Е.А.

Руководители _____ Кринкин К.В.

АННОТАЦИЯ

Целью научно-исследовательской работы является исследование способов улучшения производительности системы реального времени RT Linux для встроенных систем. В ходе данного исследования был произведен обзор предметной области, в частности были рассмотрены понятия систем реального времени и встроенных систем, а также были рассмотрены способы применения систем реального времени для встроенных систем.

SUMMARY

The purpose of the research work is to study ways to improve the performance of the RT Linux real-time system for embedded systems. In the course of this research, the domain was surveyed, in particular, the concepts of real-time systems and embedded systems were considered, and ways of using real-time systems for embedded systems were considered.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В ходе выполнения данной научной исследовательской работы предстоит провести исследование производительности микроядерной операционной системы реального времени – RT Linux, для специализированной микропроцессорной системы управления. Linux позволяет работать с многочисленными разработчиками, использовать большой объём существующего исходного кода и интерфейсы прикладного программирования (API) стандарта POSIX. Однако стандартное ядро ОС Linux не обеспечивает такие возможности "жёсткого" реального времени, как гарантированное время отклика и микросекундные задержки, которые требуются во многих встраиваемых устройствах. Исходя из этого были определены следующие задачи научной исследовательской работы:

- Изучение основ систем реального времени;
- Исследование производительности операционной системы реального времени для встроенных систем;
- Проведение сравнительного анализа методов вычислений в реальном времени;

1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Операционная система реального времени

Тип операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций для работы систем реального времени на конкретном аппаратном оборудовании, либо это способность операционной системы обеспечить требуемый уровень сервиса в определённый промежуток времени.

Операционные системы реального времени иногда делят на два типа — системы жёсткого реального времени и системы мягкого реального времени.

Операционная система, которая может обеспечить требуемое время выполнения задачи реального времени даже в худших случаях, называется операционной системой жёсткого реального времени. Система, которая может обеспечить требуемое время выполнения задачи реального времени в среднем, называется операционной системой мягкого реального времени.

Системы жёсткого реального времени не допускают задержек реакции системы, так как это может привести к потере актуальности результатов, большим финансовым потерям или даже авариям и катастрофам. Ситуация, в которой обработка событий происходит за время, большее предусмотренного, в системе жёсткого реального времени считается фатальной ошибкой. При возникновении такой ситуации операционная система прерывает операцию и блокирует её, чтобы, насколько возможно, не пострадала надёжность и готовность остальной части системы. Примерами систем жёсткого реального времени могут быть бортовые системы управления (на самолёте, космическом аппарате, корабле, и пр.), системы аварийной защиты, регистраторы аварийных событий.

В системе мягкого реального времени задержка реакции считается восстановимой ошибкой, которая может привести к увеличению стоимости

результатов и снижению производительности, но не является фатальной. Примером может служить работа компьютерной сети. Если система не успела обработать очередной принятый пакет, это приведёт к остановке на передающей стороне и повторной посылке (в зависимости от протокола). Данные при этом не теряются, но производительность сети снижается.

Основное отличие систем жёсткого и мягкого реального времени можно охарактеризовать так: система жёсткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени не должна опаздывать с реакцией на событие.

Часто операционной системой реального времени считают лишь систему, которая может быть использована для решения задач жёсткого реального времени. Это определение означает наличие у ОСРВ необходимых инструментов, но также означает, что эти инструменты необходимо правильно использовать.

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).

Классическим примером задачи, где требуется ОСРВ, является управление роботом, берущим деталь с ленты конвейера. Деталь движется, и робот имеет лишь маленький промежуток времени, когда он может её взять. Если он опоздает, то деталь уже не будет на нужном участке конвейера, и,

следовательно, работа не будет выполнена, несмотря на то, что робот находится в правильном месте. Если он подготовится раньше, то деталь ещё не успеет подъехать, и он заблокирует ей путь.

Также для операционных систем иногда используется понятие «интерактивного реального времени», в котором определяется минимальный порог реакции на события графического интерфейса, в течение которого оператор — человек — способен спокойно, без нервозности, ожидать реакции системы на данные им указания.

В своем развитии ОСРВ строились на основе следующих архитектур:

- **Монолитная архитектура**. 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.
- **Уровневая (слоевая) архитектура**. 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.
- **Архитектура «клиент-сервер»**. 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:

1. Повышенная надёжность, так как каждый сервис является, по сути, самостоятельным приложением и его легче отладить и отследить ошибки.
2. Улучшенная масштабируемость, поскольку ненужные сервисы могут быть исключены из системы без ущерба к её работоспособности.
3. Повышенная отказоустойчивость, так как «зависший» сервис может быть перезапущен без перезагрузки системы.

Ядро ОСРВ обеспечивает функционирование промежуточного абстрактного уровня ОС, который скрывает от прикладного ПО специфику технического устройства процессора (нескольких процессоров) и связанного с ним аппаратного обеспечения.

Указанный абстрактный уровень предоставляет для прикладного ПО пять основных категорий сервисов:

Управление задачами. Самая главная группа сервисов. Позволяет разработчикам приложений проектировать программные продукты в виде наборов отдельных программных фрагментов, каждый из которых может относиться к своей тематической области, выполнять отдельную функцию и иметь свой собственный квант времени, отведенный ему для работы. Каждый такой фрагмент называется *задачей*. Сервисы в рассматриваемой группе обладают способностью запускать задачи и присваивать им приоритеты. Основной сервис здесь — *планировщик задач*. Он осуществляет контроль над выполнением текущих задач, запускает новые в соответствующий период времени и следит за режимом их работы.

Динамическое распределение памяти. Многие (но не все) ядра ОСРВ поддерживают эту группу сервисов. Она позволяет задачам заимствовать области оперативной памяти для временного использования в работе приложений. Часто эти области впоследствии переходят от задачи к задаче, и посредством этого осуществляется быстрая передача большого количества

данных между ними. Некоторые очень малые по размеру ядра ОСРВ, которые предполагается использовать в аппаратных средах со строгим ограничением на объём используемой памяти, не поддерживают сервисы динамического распределения памяти.

Управление таймерами. Так как встроенные системы предъявляют жёсткие требования к временным рамкам выполнения задач, в состав ядра ОСРВ включается группа сервисов, обеспечивающих управление таймерами для отслеживания лимита времени, в течение которого должна выполняться задача. Эти сервисы измеряют и задают различные промежутки времени (от 1 мкс и выше), генерируют прерывания по истечении временных интервалов и создают разовые и циклические будильники.

Взаимодействие между задачами и синхронизация. Сервисы данной группы позволяют задачам обмениваться информацией и обеспечивают её сохранность. Они также дают возможность программным фрагментам согласовывать между собой свою работу для повышения эффективности. Если исключить эти сервисы из состава ядра ОСРВ, то задачи начнут обмениваться искаженной информацией и могут стать помехой для работы соседних задач.

Контроль устройства ввода-вывода. Сервисы этой группы обеспечивают работу единого программного интерфейса, взаимодействующего со всем множеством драйверов устройств, которые являются типичными для большинства встроенных систем.

В дополнение к сервисам ядра, многие ОСРВ предлагают линейки дополнительных компонентов для организации таких высокоуровневых понятий, как файловая система, сетевое взаимодействие, управление сетью, управление базой данных, графический пользовательский интерфейс и т. д. Хотя многие из этих компонентов намного больше и сложнее, чем само ядро ОСРВ, они, тем не менее, основываются на его сервисах. Каждый из таких

компонентов включается во встроенную систему, только если её сервисы необходимы для выполнения встроенного приложения и только для того, чтоб свести расход памяти к минимуму.

Многие операционные системы общего назначения также поддерживают указанные выше сервисы. Однако ключевым отличием сервисов ядра ОСРВ является *детерминированный*, основанный на строгом контроле времени, характер их работы. В данном случае под детерминированностью понимается то, что для выполнения одного сервиса операционной системы требуется временной интервал заведомо известной продолжительности. Теоретически это время может быть вычислено по математическим формулам, которые должны быть строго алгебраическими и не должны включать никаких временных параметров случайного характера. Любая случайная величина, определяющая время выполнения задачи в ОСРВ, может вызвать нежелательную задержку в работе приложения, тогда следующая задача не уложится в свой квант времени, что послужит причиной для ошибки.

В этом смысле операционные системы общего назначения не являются детерминированными. Их сервисы могут допускать случайные задержки в своей работе, что может привести к замедлению ответной реакции приложения на действия пользователя в заведомо неизвестный момент времени. При проектировании обычных операционных систем разработчики не акцентируют своё внимание на математическом аппарате вычисления времени выполнения конкретной задачи и сервиса. Это не является критичным для подобного рода систем.

1.2. Встроенные системы

Современные встраиваемые системы управления реального времени (Embedded Systems или ВсС,) представляют собой результат междисциплинарного проектирования, в котором условно можно выделить три основных составляющих.

- Этап решения задачи на прикладном уровне, когда необходимо найти правильные методы и алгоритмы без деталей реализации. Это сфера деятельности прикладных специалистов из соответствующих областей (физика, энергетика, медицина, лингвистика, биология и др.).
- Процесс программирования, в ходе которого требуется отобразить полученное прикладное решение на технологическую базу информатики и вычислительной техники (ВТ). Это работа специалистов из области информатики, сегодня все чаще ее называют архитектурным, высокоуровневым или системным проектированием.
- Фаза реализации, в ходе которой инженеры, программисты и прикладные специалисты обеспечивают выполнение ранее сформулированных требований, таких как необходимая функциональность, динамика поведения, надежность и безопасность функционирования, габариты, энергопотребление, стоимость и технологичность при тиражировании.

Первые опыты использования цифровых вычислительных машин для управления физическими объектами начались в 60-х годах XX в. В следующем десятилетии ЭВМ начинают активно применяться в составе информационно-управляющих систем (ИУС) для задач сбора информации о состоянии различных технических систем, а также для управления ими. Это определяется развитием интегральных микросхем и, особенно, появлением микропроцессоров. На данном этапе развития для ИУС с небольшими габаритами, конструктивно объединенными с физическим объектом контроля (управления), удачным стало название «встраиваемые системы управления

реального времени». Термин «реальное время» здесь означает, что ВсС должна выполнять определенные действия, например, считывание данных с датчиков и выдачу команд на исполнительные устройства, не раньше и не позже, а в строго заданные моменты и интервалы времени.

Основными особенностями ВсС считаются:

- работа в реальном масштабе времени (почти всегда);
- различные, часто тяжелые, условия эксплуатации;
- автономность работы (отсутствие оператора, ограничения электропитания);
- высокие требования по надежности и безопасности функционирования;
- ограниченные ресурсы;
- критические применения (Dependable Applications), связанные со здоровьем и жизнью человека.

ВсС относятся к категории систем с преимущественно программной реализацией (Software-Intensive или Software-Dominated Systems). Это означает, что большая часть функциональности системы реализуется программным способом. Программируемость и конфигурируемость пронизывают все уровни и компоненты ВсС во все большей степени. Сложность и удельный вес программной составляющей в ВсС стремительно растет. Также появился термин «встроенное программное обеспечение» (Embedded Software), подчеркивающий особые свойства такого ПО и технологий его создания.

Элементную базу ВсС составляют электронные, оптические, механические и иные физические компоненты (элементы, модули, блоки), из которых складывается физическая реализация ВсС. Сегодня в перечене таких компонентов — сложные микросхемы процессоров (микропроцессоры), контроллеров, акселераторов, системные платы вычислителей. В свою очередь, в состав таких элементов входят программные средства (загрузчики, стеки

протоколов и другие), размещаемые во встроенных блоках постоянной памяти. Таким образом, даже традиционное представление вычислительной элементной базы выходит далеко за границы описания только конструкции и схемотехники, затрагивая все больше вопросы системотехники, программирования, архитектуры.

2. ОПИСАНИЕ РАЗЛИЧНЫХ СПОСОБОВ ВЗАИМОДЕЙСТВИЯ

2.1. Введение

В данном разделе будут рассмотрены различные подходы приминения RT Linux для встроенных систем. Будут рассмотрены следующие подходы:

- Подход на основе тонкого ядра
- Подход на основе наноядра
- Подход на основе ядра ресурсов

2.2. Подход на основе тонкого ядра

Подход на основе тонкого ядра (или микроядра) использует второе ядро для абстрагирования интерфейса между аппаратными устройствами и ядром Linux (см. рисунок 1). То ядро Linux, которое не работает в режиме реального времени, выполняется в фоновом режиме как задача с низким уровнем приоритета для тонкого ядра и на нем выполняются все задачи, не относящиеся к реальному времени. Задачи реального времени выполняются непосредственно в тонком ядре.

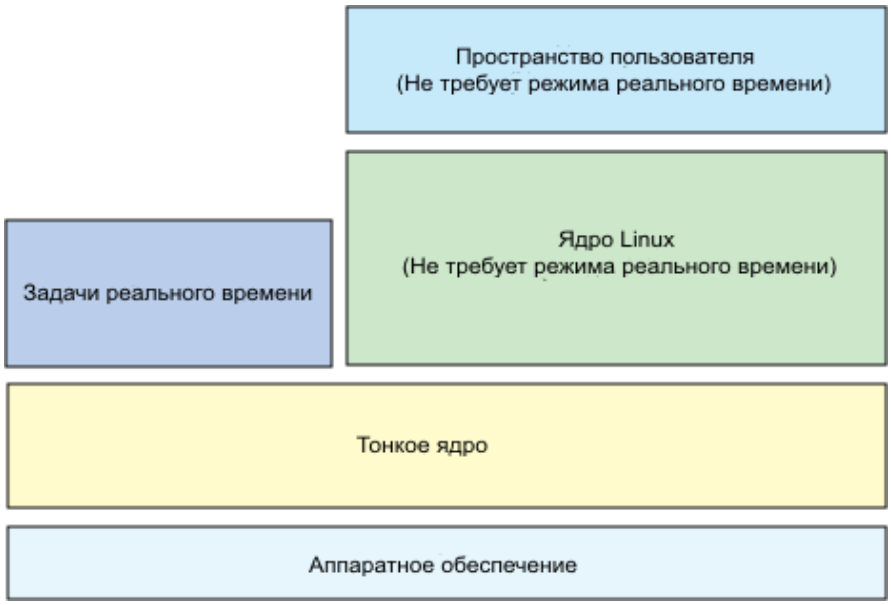

Рисунок 1 – Подход на основе тонкого ядра к жесткому режиму реального времени.

Основным назначением тонкого ядра (кроме выполнения задач реального времени) является управление прерываниями. Тонкое ядро перехватывает

прерывания, благодаря этому гарантируется, что работа тонкого ядра не будет прервана другим ядром, которое не выполняет задачи реального времени. Это позволяет тонкому ядру обеспечить жесткую поддержку режима реального времени.

Хотя подход на основе тонкого ядра имеет свои преимущества (жесткая поддержка реального времени вместе со стандартным ядром Linux), он имеет и недостатки. Обычные задачи и задачи реального времени не зависят друг от друга, это может значительно усложнить отладку. Кроме этого, обычные задачи не обеспечивают полную поддержку платформы Linux (именно по этой причине выполнение ядра носит название *тонкого*).

Примерами реализации такого подхода являются RTLinux (теперь закрытая система, принадлежит Wind River Systems), Real-Time Application Interface (RTAI) и Xenomai.

2.3. Подход на основе наноядра

В то время как тонкое ядро использует минимальное ядро, которое включает управление задачами, подход на основе наноядра двигается далее в этом направлении, еще более минимизируя размеры ядра. Такое ядро уже является в меньшей степени ядром, а в большей степени уровнем абстракции аппаратного обеспечения (HAL). Нано-ядро обеспечивает возможность совместного использования аппаратных ресурсов для нескольких операционных систем, которые выполняются на более высоком уровне (см. рисунок 2). Так как нано-ядро позволяет абстрагировать аппаратное обеспечение, то оно способно управлять приоритетами операционных систем, исполняемых на более высоком уровне, и благодаря этому, обеспечивать жесткую поддержку режима реального времени.

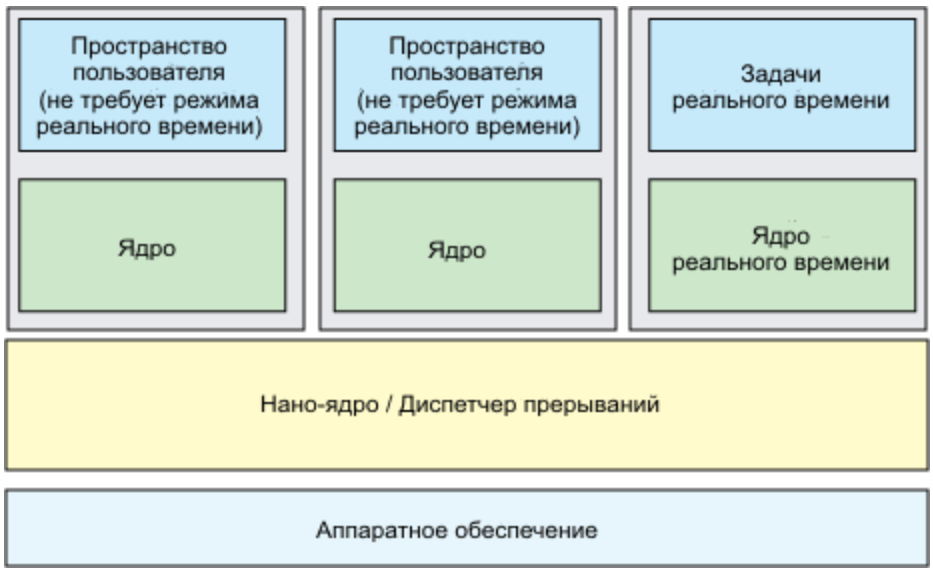

Рисунок 2 – Нано –ядро для поддержки абстрагирования аппаратного обеспечения.

Схожесть между данным подходом и виртуализацией, которая обеспечивает возможность выполнения нескольких операционных систем. В нашем случае нано-ядро позволяет абстрагировать аппаратные устройства от обычного ядра и ядра реального времени. Это аналогично тому, как гипервизоры абстрагируют голое аппаратное обеспечение от выполняемых операционных систем.

Примером подхода на основе нано-ядра является Adaptive Domain Environment for Operating Systems (ADEOS). ADEOS обеспечивает возможности одновременного выполнения операционных систем. При возникновении определенного аппаратного события, ADEOS по очереди обращается ко всем операционным системам и определяет, кто из них будет осуществлять обработку события.

2.4. Подход на основе ядра ресурсов

Еще один вариант архитектуры реального времени - подход на основе ядра ресурсов. В этом подходе к ядру добавляются модули, которые обеспечивают резервирование ресурсов различного типа. Такое резервирование обеспечивает доступ к системным ресурсам с временным разделением (центральный процессор, сеть или доступ к диску). Эти ресурсы имеют несколько параметров

резервирования, таких как период повторного доступа, необходимое время обработки (то есть, период времени, необходимый для выполнения обработки), а также временные ограничения.

Ядро ресурсов предоставляет набор прикладных интерфейсов программирования (API) которые позволяют задачам запрашивать подобное резервирование (см. рисунок 3). Затем ядро ресурсов объединяет эти запросы с целью, определяя график, который обеспечивает гарантированный доступ с учетом временных ограничений, действующих для конкретных задач (или же возвращается ошибка, если невозможно гарантировать такой доступ). Используя алгоритм планирования, такой как Earliest-Deadline-First (EDF), ядро затем может производить динамическое планирование нагрузки.

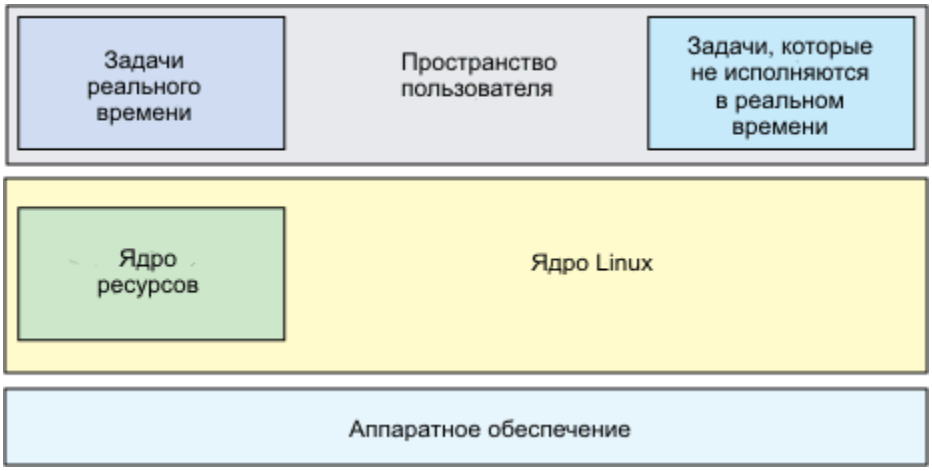

Рисунок 3 – Подход на основе ядра ресурсов обеспечивает резервирование ресорсов.

Одним из примеров реализации подхода на основе ядра ресурсов является Linux/RK от компании CMU, который интегрирует переносимое ядро ресурсов в Linux в качестве загружаемого модуля. Сегодня данный подход воплощен в коммерческой системе TimeSys Linux/RT.

ЗАКЛЮЧЕНИЕ

Результатом научно-исследовательской работы является обзор методов реализации вычислений в реальном времени для ядра операционной системы Linux. В многочисленных ранних попытках для выделения из стандартного ядра задач реального времени использовался подход на основе тонкого ядра. Затем стали использоваться подходы на основе нано-ядра, которые действовали во многом аналогично гипервизорам, использующимся сегодня в решениях по виртуализации. Наконец, современное ядро Linux обеспечивает собственные средства для поддержки как мягкого, так и жесткого режима реального времени.

Также в результате проведения научной исследовательской работы были выполнены следующие задачи:

- Изучена предметная область: операционные системы реального времени;
- Исследована производительности операционной системы реального времени для встроенных систем;
- Проведен анализ методов реализации вычислений в реальном времени для операционной системы Linux для выявления возможности оптимизации производительности операционной систем реального времени при использовании в встроенных системах;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. John L. Hennessy, David A. Patterson. Computer architecture: a quantitative approach. San Francisco. Morgan Kaufmann publishers. 2007.
2. Кузнецов С. Миром правят встроенные системы // Открытые системы. 2009. № 4.
3. Actor-Oriented Design of Embedded Hardware and Software Systems// Journal of Circuits, Systems, and Computers. 2003. № 12.
4. Платунов А. Роль и проблемы высокоуровневого этапа проектирования встраиваемых вычислительных систем // Компоненты и технологии. 2009. № 4.
5. www.date-conference.com/
6. Edward A. Lee, Sanjit A. Seshia. Introduction to Embedded Systems, A Cyber-Physical Systems Approach. 2011. http://LeeSeshia.org.
7. http://embedded.ifmo.ru
8. http://lmt.ifmo.ru
9. https://www.ibm.com