Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра Автоматизации систем и вычислительных комплексов

Магистерская диссертация

Структурный синтез и планирование вычислений в системах с архитектурой ИМА

Научный руководитель: Костенко В.А.

Студент: Смирнов А. С.

Москва 2019

Содержание

1 Цель работы

Разработка и реализация алгоритмов для построения расписания и синтеза вычислительных систем в комплексах с архитектурой ИМА.

Подзадачи:

- Провести обзор архитектуры ИМА с целью выявления ее особенностей
- Провести обзор применяемых алгоритмов построения расписаний и синтеза архитектур
- Провести обзор решаемых близких задача
- Разработать алгоритм построения расписаний, как адаптацию известных алгоритмов под особенности комплексов ИМА
- Разработать алгоритм синтеза вычислительной системы ИМА, гарантированно выполняющей все заданные работы
- Реализовать разработанные алгоритмы как единую систему, которой удобно пользоваться
- Провести экспериментальное исследование разработанных алгоритмов

2 Введение.

Тут немного истории про переход с федеративной на модульную архитектуру. Плюсы, минусы и развитие.

Бортовые вычислительные системы реального времени летательных аппаратов предыдущих поколений имели федеративную архитектуру. Работы каждой бортовой систе-

мы (например, локационной, навигационной, управления двигателями) выполнялись на своих вычислителях, очень часто специализированных. На одном вычислителе не допускалось выполнение работ разных бортовых систем. Это позволяло изолировать работы систем друг от друга, но приводило к большим затратам аппаратных средств.

Вычислительные системы реального времени летательных аппаратов нового поколения имеют интегрированную модульную архитектуру. Единый бортовой вычислитель строится из набора стандартизированных вычислительных модулей. Аппаратные ресурсы одного вычислительного модуля могут разделять прикладные работы различных бортовых систем. Это позволяет сократить аппаратные затраты. Однако, при этом требуется обеспечить изоляцию работ различных систем. Изоляция распространяется на все ресурсы, включая регистровую память, кэши центральных процессоров, шины ввода-вывода.

В операционных системах, работающих в соответствии со стандартом ARINC 653, изоляция работ достигается введением разделов и окон. Для работ каждой системы выделяется свой раздел и набор временных окон (интервалов времени). Работы раздела могут выполняться только в рамках своих окон, и каждому разделу выделяется необходимая память, к которой не могут обращаться программы других разделов. Взаимодействие работ различных разделов осуществляется только путем передачи сообщений.

Для построения вычислительной системы реального времени с интегрированной модульной архитектурой требуется решить задачу построения статико-динамических расписаний [1]. Статико-динамическое расписание построено, если определены: привязка разделов к ядрам; набор окон для каждого раздела; время открытия и закрытия каждого окна. Работы раздела внутри окна запускаются на выполнение по мере готовности данных в соответствии с приоритетами. Допустимо прерывание программы и ее последующее выполнение в этом окне или в одном из следующих окон раздела.

Современные авиалайнеры, такие как Airbus A320, Boeing 787, перспективный отечественный самолёт МС-21, используют новую архитектуру построения комплекса бортового оборудования, получившую название Интегрированная модульная авионика (ИМА). В её основе лежит объединение приборов и бортовых вычислителей в единую сеть реального времени, что позволяет существенно снизить количество кабелей на борту и, тем самым, уменьшить взлётный вес лайнера. В ИМА разделяются функции сбора информации (датчики), воздействия (актуаторы) и логики оказания управляющих воздействий, которая реализуется специализированным прикладным ПО в бортовых вычислительных модулях. Международный стандарт ARINC 653 описывает требования к операционной системе реального времени, устанавливаемой на таких модулях, и программный интерфейс между прикладным авиационным ПО и операционной системой. Данный стандарт регламентирует временное и пространственное разделение прикладного ПО в соответствии с принципами ИМА. Большинство ОСРВ соответствующих стандарту ARINC 653 являются коммерческим ПО. В данной статье представляется JetOS - ОСРВ с открытым исходным кодом полностью соответствующую требованиям ARINC 653 части 1 версии 3. JetOS была основана на открытом проекте французских исследователей POK. Некогда POK была единственной ОСРВ с открытым исходным кодом, которая хоть сколько-нибудь соответствовала требованиям стандарта ARINC 653, однако была непригодна для практического использования: POK не удовлетворяла ряду фундаментальных требований ARINC 653 и работала только в эмуляторе. При разработке JetOS код POK был существенно переработан. В статье мы обсуждаем недостатки POK и показываем, как нам удалось решить эти проблемы и какие изменения были внесены в архитектуру и реализацию POK и отдельным подсистем. В частности, был полностью переписан планировщик реального времени, сетевой стек и управление памятью. Также в JetOS были добавлены новые возможности. Наиболее интересной является поддержка системных разделов. Системный раздел - специальное прикладное ПО с расширенным набором возможностей, таких как прямой доступ к отдельным аппаратным средствам (сетевой карте,

PCI контроллеру и т.п.). Наличие системных разделов позволяет вынести крупные подсистемы из ядра ОС и оставить в ядре минимальный набор задач, связанных с переключением контекстов, планировщиком и обменом сообщениями между компонентами ПО. В частности, в системный раздел вынесена подсистема, отвечающая за взаимодействие через сеть. Данное перемещение кода позволяет уменьшить размер ядра ОС, что теоретически уменьшает вероятность наличия ошибки в ядре и упрощает процесс верификации ядра.

В данной работе рассматривается алгоритм построения однопроцессорных и многопроцессорных статико-динамических расписаний, основанный на нахождении максимального потока в транспортной сети. Наиболее близкими задачами, для которых известны алгоритмы, основанные на нахождении максимального потока в транспортной сети, являются задачи построения расписаний с прерываниями работ.

Основными проблемами применения известных алгоритмов построения расписаний с прерываниями, которые основаны на нахождении максимального потока в сети, для построения статико-динамических расписаний являются: проблема учета принадлежности работ к разделам и проблема определения набора окон. .

Любая система построенная на архитектуре ИМА должна придерживаться следующи принципов.

- *Унифицированные процессорные подсистемы*. Они должны позволить нескольким приложениям делить и повторно использовать одни и те же вычислительные ресурсы. Это приводит к сокращению количества развертываемых подсистем и более эффективному использованию системных ресурсов, что оставляет свободное пространство для будущих расширений
- *Абстракция ПО*. Она должна сделать приложения независимыми от нижележащей аппаратной архитектуры. Это повысит уровень переносимости приложений между различными платформами, а также позволит вводить новую аппаратуру

для замены устаревших архитектур.

- *Максимизировать повторное использование*. Архитектура ИМА должна позволять повторное использование ранее разработанного кода. Это сокращает время разработки, предоставляя разработчику возможность использования уже существующих приложений без значительной модификации.

- *Стоимость изменений*. Архитектура ИМА должна понизить стоимость внесения изменений и повторного тестирования и упростить анализ последствий путем изолирования отдельных частей платформы, которые исполняются на одном и том же процессоре.

Одним из широко применяемых на практике пакетов для создания систем авионики и критичных по безопасности приложений является пакет WxWorks Platform производства компании Wind River [**?**] . Он поддерживает две важные концепции, широко используемые в ИМА: пространственное и временное разделение.

Пространственное разделение.

Пространственное разделение определяет требования по изоляции нескольких пработ, исполняемых одновременно на одной и той же вычислительной платформе, называемой «модулем». Согласно этим требованиям, работы, исполняемые в разделе ИМА, не должны отбирать друг у друга разделяемые ресурсы, предоставляемые ядром ОСРВ. Чаще всего это реализуется через различные контексты виртуальной памяти, обеспечиваемые процессорным устройством управления памятью MMU (Memory Management Unit). В спецификации ARINC 653 эти контексты называются разделами. Каждый раздел содержит работы со своей динамически распределяемой областью памяти типа «куча» и стеком для процессов приложения (процесс – это исполняемая единица в ARINC 653). Эти требования влияют на конструкцию и реализацию ядра ОСРВ и исполнительной системы языка программирования. Например, VxWorks 5.5 использует разделяемое виртуальное адресное пространство для приложений и

обеспечивает через MMU базовую защиту программного кода от доступа со стороны ошибочных приложений, не применяя полную модель процессов, влияющую на производительность. Операционные системы VxWorks 6.x и VxWorks 653 обеспечивают через MMU среду с полной изоляцией контекстов. Однако одна только защита памяти в среде ИМА не может воспрепятствовать ошибочной работе, исполняемой в разделе, занимать системные ресурсы, что может иметь вредоносное влияние на работу, исполняемую в другом разделе. Это может иметь серьезные последствия, когда несколько работ разных уровней критичности исполняются на одном и то же процессоре. Поэтому требуется разработка специальной ОСРВ, предназначенной именно для ИМА. Операционная система VxWorks 653 было разработана специально для этой цели и поддерживает модель ARINC 653 на уровне реализации архитектуры ядра.

- ОС модуля (Module OS) непосредственно взаимодействует с аппаратной платформой (core-модуль), обеспечивая управление общими ресурсами, планирование исполнения и мониторинг состояния каждого из разделов. Она также использует пакет поддержки модуля BSP (Board Support Package) – аппаратно-зависимые описания, необходимые для исполнения на различных процессорных и аппаратных конфигурациях.

- ОС раздела (Partition OS) реализована на микроядре VxWorks и обеспечивает планирование исполнения и управление ресурсами внутри раздела. Коммуникация с ОС модуля для обеспечения надежности происходит через внутренний закрытый интерфейс передачи сообщений. ОС раздела также обеспечивает программный интерфейс ARINC 653 APEX (Application/Executive), используемый работами.

Временное разделение.

Временное разделение определяет требования по изоляции нескольких работ, исполняющихся одновременно на одной и той же вычислительной платформе. Одна работа не может занимать процессор дольше, чем ему предназначено, чтобы не навредить

другим работам. В ARINC 653 уделено внимание этой проблеме и определена реализация, в которой используется планирование исполнения разделов. Каждому разделу выделяется для исполнения временной интервал определенной длительности, и интервалы различных разделов могут быть одинаковой или различной длительности. Внутри своего интервала раздел может применять собственную политику планирования, по окончании временного интервала ARINC-планировщик переключит контекст на следующий по расписанию раздел. Это модель достаточно универсальна, чтобы разместить на core-модуле существующие федерированные приложения или ИМА-приложения, разработанные независимо друг от друга. Но планирование исполнения разделов и проверка целостности расписания и границ исполнения, а также необходимые корректирующие действия, неизбежно вносят дополнительную сложность.

В VxWorks 653 ОС модуля выполняет планирование исполнения отдельных разделов в соответствии с ARINC 653. Внутри каждого временного интервала ОС раздела использует планировщик VxWorks для выполнения планирования на основе вытеснения по приоритету. Это значит, что все планирование на уровне работ происходит в пространстве раздела, что обеспечивает более высокую масшабируемость и стабильность в системе даже при высокой частоте системных тактов). Также полностью реализован протокол силинга приоритета для предотвращения неограниченной инверсии приоритетов. Существует потенциальная возможность использование других типов планирования таких, как:

1. *Циклическое планирование*.

   Каждой работе предоставляется квант времени процессора. Когда квант заканчивается, работа переводится планировщиком в конец очереди.

2. *RMS*.

   Работы должны удовлетворять условиям:

   - Работа должна быть завершена за время ее периода.

- Одна работа не должен зависеть от другой.
- Прерывание работы происходит мгновенно.

Приоритет в этом алгоритме пропорционален частоте.

3. *EDF*.

   Наибольший приоритет выставляется работе, которой осталось наименьшее время выполнения.

Важным пунктом является тот факт, что хоть в названии и звучит термин авионика, данный подход к построению вычислительной системы реального времени нашел себе применение далеко за границами авиация. Данная архитектура оказалось достаточна удобна и надежна для испоьзования ее в системах управления автомобилями и реальных системах контроля производства. Так например компания WindRiver создает свои продукты на базе архитектуры ИМА для автомобилей и производства.

2.1 Концепция ИМА.

Появление ИМА стало естественным процессом движения мысли человека и усложнения систем управления и обеспечения воздушных судов [21]

В [19] описано понимание ИМА : Под интегрированной модульной авионикой понимается концепция построения бортового комплекса, базирующаяся на открытой сетевой архитектуре и единой вычислительной платформе. Вычислительная платформа реализуется в виде базовой конструкции (крейта) с набором сменных электронных модулей (процессорных модулей, модулей памяти, модулей сетевого коммутатора, модулей электропитания). Конструкция модуля строится на базе единого стандарта, обеспечивающего принцип унификации и взаимозаменяемости. Функции систем комплекса в этом случае выполняют программные приложения, разделяющие общие

вычислительные и информационные ресурсы. Понятие функции является ключевым понятием ИМА. Под функцией понимаются функциональные возможности, которые могут быть обеспечены аппаратными и программными средствами систем, установленными на воздушном судне, например, самолетовождение, связь, индикация. Переход к ИМА позволил перейти от идеи «система – одна функция» к мультифункциональной структуре – «много функций в одном вычислительном ядре». Практически интеграция функций, которые ранее воспринимались как интеграция систем, сводится в новом поколении КБО к созданию базы данных функций и сигналов, а также коммуникатора функций на программном уровне. Потребность в снижении стоимости авиационных комплектующих благодаря расширению числа производителей и повышению эксплуатационной эффективности за счет более мелкого, чем блок, сменяемого в эксплуатации элемента объективно привела в новом поколении КБО к модульности аппаратного и программного обеспечения. Интегрированная модульная авионика позволила перенести все функции управления на уровень программного обеспечения. Это обеспечило аппаратное построение вычислительной системы в виде набора ограниченного числа стандартных элементарных модулей. Наличие современных операционных систем реального времени, в свою очередь, позволило построить и программное обеспечение в виде отдельных функционально-программных модулей. Модульность аппаратной и программной части – это ключ к унификации, стандартизации и, как следствие, снижению затрат в разработке и производстве.

Перспективный КБО должен иметь открытую сетевую отказоустойчивую функционально-ориентированную архитектуру на базе масштабируемой интегрированной модульной авионики с использованием вычислительной среды (платформы), реализованной на одних и тех же принципах. Причем, в составе КБО ВС могут одновременно функционировать несколько платформ ИМА, тем самым обеспечивая возможность реализации распределенной архитектуры КБО. В платформе ИМА для организации информационного обмена между функциями, датчиками и исполнительными элемен-

тами применяются перспективные протоколы связи AFDX, обеспечивающие эффективное построение динамических структур с сетевой организацией (рис. 5). Открытая архитектура комплекса предполагает подключение различных по своему назначению устройств, например датчиков информации, через стандартные концентраторы к вычислительному ядру системы. Распределение ресурсов функционального программного обеспечения осуществляется под управлением операционной системы реального времени. Важной особенностью такой архитектуры является отсутствие «жестких», раз и навсегда установленных связей между датчиками бортового оборудования (информационными каналами) и вычислительными средствами. Это позволяет реализовать динамическую реконфигурацию структуры КБО с соответствующим перераспределением ресурсов. Внутри вычислительной среды формируются (с подключением к необходимым информационным каналам комплекса) структуры для оптимального выполнения каждой функции КБО. Каждая возникающая при этом структура формируется только на время выполнения заданной функции. Таким образом, общая конфигурация вычислительной среды динамически перестраивается в процессе функционирования комплекса.

3 Особенности ИМА.

Конкретно рассмотреть как работают системы на основе ИМА, в пример ОС привести, технологии рассмотреть - четко объяснить какие задачи ею решаются и как система выглядит.

В [20] описаны основные сложности работы с такими системами - одна из очень важных - это планирование вычислительных ресурсов системы и ее эффективный синтез.

3.1 Спецификация ИМА

как никак это стандарт написанный кровью и потом многих людей и его надо указать и основные особенности для нас и как они с реализацией связаны. Вообще в такой области без спецификаций и стандартов далеко не зайдешь и все начинается от них как в WiFi.

3.2 Критичность работ по DO что-то

Выделяют разные уровни критичности

Оценка времени выполнения работ

Нужно отдельно отметить, что оценка времени выполнения работ - это отделльная сложная тема со своими подводными амнями, но в этой работе мы считаем, что кто-то умный нам посчитал уэе эти оценки и мы работаем с готовыми значениями для сложности работ. Они посчитаны так

3.3 Наиболее часто встречающиеся работы

Периодические, периодические с запасом, индивидуальные.

Периодическими с запасом можно моделировать как индивидуальные так ки периодическкие работы. В первом случае можно не ставить не какие функциональные интервалы, а во втором выставить период равный периоду планирования и поставить функкциональные интервалы как директивный срок задачи. Поскольку моделируется система реального времени, то у нее все планируется периодически.

Если в нашей системе есть случайности в поступающих работах, то для них можно выделить время как для задачи и забронировать место для них. А система раздела

уже сама внутри себя решит, акие именно ей задачи размещать исходя из своего планировщика.

Однако планировщик раздела может принять расписание от общего планировщика. Такие функкции поддерживают современный операционные системы реального времени, такие как WxWorks.

Для проведения эккспериментов наиболее близких к действительности будут использоваться данные из открытых источников о длительностях и периодах работ в системах реального времени.

Все это подробно будет описано тут. В главе с экспериментальным исследованием будет дана ссылка сюда + как именно генерировались данные.

Время появления в системе.

Особая черта - детерминированность.

Таким образом в нашей задаче основные работы - периодические. Хотя подходы рассматриваемые в данной работе могут быть применены с число периодическим и индивидуальным работам.

3.4 Организация вычислителей

Виды процессоров и их производительности. Например возможен уст или она статично. Обычно за счет изменения тактовой частоты меняется производительность - насколько это верно для встроенных систем.

2 системы через кор-модули [18].

Так же нужно провести анализ процессоров, которые могут использоваться в архитекуре ИМА, а так же необходимые специальные процессоры.

Наиболее часто встречающие процессоры.

Как происходит выполнения поставленных задач и какие подсистемы могут быть встроены в архитектуру, например система вооружения, система РЛС, система метеонаблюдения, система защиты, система развлекательного назначения, управления двигателем, управления крылом, мониторинг системы, резервирование

Как там проводки подсоединяются и как осуществляется взаимодействиеО

Основные составляющие работы встроенной системы - процессоры, оперативная память, сеть и датчики с сенсорами + дополнительные модули и их связь с реальными объектами.

Процессоры: - Типы пооцессоров при синтезе - или дискретные производительности или непрерывные засчет тактовой частоты (в первом случае расширение числа процессоров, во втором корректировка производительности в конце)

3.5 Дисциплины планированиия

RM, EDF, выделение квантилей времени и другие — описать математически и сказать об оптимальности и возможности использования.

Возможны несколько дисциплин планирования - как с учетом прерываний так ки без их учета

3.6 Про синтез

В [22] и [23] показано, что часто для обеспечения надежности всего вычислительного комплекса необходимо делать резервирование определенных видов ресурсов и если они не в точности повторяют заменяемый, то требуется перераспределение рас-

писания налету или переход в другой режим работы - такая задача может решать наличием предпланированных вариантов работы системы, учитывающая максимально возможные допустимые изменения в ней.

3.7 WxWorks 653

В [18] описана система WxWorks 653 - корпоративная разработка системы реального времени для систем модульной авионики, успешно используемая в таких гигантах как Boeing. Особенности именно этой системы

4 Постановки задач.

4.1 Задача построения расписания.

4.1.1 Исходные данные.

Для задачи построения статико-динамических расписаний для систем реального времени с архитектурой интегрированной модульной авионики задаются следующие исходные данные:

- n - число работ.
- m - число процессоров.
- q - число разделов.
- $M = \{p_j = \langle s_j, R_j \rangle \mid j = \overline{1, m}\}$ - набор процессоров, где s_j - проиводительность процессора j в операциях в единицу времени, R_j - множество функциональных фозможностей процессора

- $A = \left\{a_k = \left\langle b_k^A, f_k^A, c_k^A, d_k^A, r_k^A \right\rangle \mid k = \overline{1,n}\right\}$ - набор работ b_k^A - начало директивного интервала, f_k^A - конец директивного интервала, c_k^A - сложность выполнения работы, d_k^A - номер раздела, к которому привязана работа, r_k^A - множество функциональных возможностей процессора, необходимых для выполнения работы

- w - время на переключение окна

Вычислительная система.

На системе функционирует несколько видов операционных систем. Операционные системы разделов и основная (модульная) операционная система.

Модульная операционная система отвечает за планирование временных окон для каждой операционной системы раздела. Операционной системе раздела она так же передает информацию о кванте времени, который нужно выделить на исполнение каждой работы внутри этого окна.

Вычислительная система состоит из процессоров, на которые размещаются окна разделов. В рамках одного окна могут выполняться работы только одного раздела. И планировщик операционной системы раздела внутри окна ставит работы на выполнение в соответствии с приоритетом по ближайшему времени завершения и снимает работы с выполнения по истечению кванта времени, выданной этой работе на исполнение в данном окне.

4.1.2 Статико-динамическое расписание.

Статико-динамическое расписание построено, если определен набор окон для каждого процессора $l = \overline{1,m}$:

m_l - число окон на процессоре $W_l = \left\{w_i^l = \left\langle b_i^{W_l}, f_i^{W_l}, d_i^{W_l}, A_i^{W_l} \right\rangle \mid i = \overline{1,m_l}\right\}$ - набор окон. $b_i^{W_l}$ - время открытия окна. $f_i^{W_l}$ - время закрытия окна. $d_i^{W_l}$ - номер раздела,

к которому привязано окно. $A_i^{W_l} = \{(a_j, c_j) \mid a_j \in A, j \subset \overline{1,n}, c_j \leq c_j^A\}$ - набор работ, выполняемых в окне (с указанием времени, отведенном работе на исполнение в данном окне).

И выполнены условия корректности расписания для каждой системы окон W_l, $l = \overline{1,p}$:

1. Окна не перекрываются и учтено минимальное время переключения:

$$b_{i+1}^{W_l} - f_i^{W_l} \forall i \in \overline{1, m-1}. \tag{1}$$

2. Сумма значений длительностей частей работ, размещенных в одном окне, не превышает значения длительности окна:

$$\sum_{a_j \in A_i^{W_l}} \frac{c_{ji}}{p_l} f_i^{W_l} - b_i^{W_l} \forall i \in \overline{1,m}. \tag{2}$$

3. В окне могут выполняться только части работы или целые работы из того раздела, к которому это окно привязано:

$$\forall a_j, a_k \in A_i^{W_l} \rightarrow d_j^{W_l} = d_k^{W_l} \forall i \in \overline{1,m}. \tag{3}$$

4. Работа может выполняться на процессоре, если процессор удовлетворяет ее функциональные ограничения

$$\forall a_k \in A_i^{W_l} \rightarrow r_k^A \subseteq R_l \tag{4}$$

5. Работа размещена, если она выполнена полностью на одном процессоре:

$$\forall a_k \in A_i^{W_l} \sum_{A_j^{W_l} | a_k \in A_j^{W_l}} c_{kj}^l = c_k^A. \tag{5}$$

Запись c_{ij}^l обозначет число операция выполненных на процессоре l для задачи i в окне j.

4.1.3 Варианты задачи.

Однопроцессорный вариант. Построить статико-динамическое расписание для работ для системы состоящей только из одного процессора.

Периодические работы. Вместо набора работ задан набор периодических задач:

$AP = \left\{a_k^P = \left\langle T_k^{AP}, c_k^{AP}, d_k^{AP}, r_k^{AP}\right\rangle \mid k = \overline{1, n^P}\right\}$ - набор периодических задач. T_k^{AP} - период выполнения задач. c_k^{AP} - время выполнения экземпляра задачи. d_k^{AP} - номер раздела, к которому привязана задача. r_k^{Ap} - необходимые функциональные возможности процессора.

В этой задаче большой цикл планирования являтся наибольшим общим кратным всех периодов задач. Каждой задаче соответствует набор работ с индивидуальными директивными интервалами, которые вычисляются следующим образом. Задача разбивается на N работ, где N - длина большого цикла, деленного на период задачи. Работы нумеруются начиная с 0. И для каждой работы с номером $i = 0, ..., N-1$ задается следующий директивный интервал $(T_k^A \cdot i,\ T_k^A \cdot (i+1)]$. Раздел работы является разделом задачи. Время выполнения работы является временем выполнения задачи. Далее будем рассматривать алгоритм для построения работ с индивидуальными директивными интервалами, но будем помнить, что нам важно получать результаты для периодических работ.

4.2 Задача синтеза архитектур.

4.2.1 Формальная постановка.

Пусть у нас есть набор типов процессоров, которые различаются выполняемыми задачами и производительность. Требуется найти такую минимальную по количеству процессоров систему, что все задача планируемого интервала будут размещены.

4.2.2 Математическая постановка.

Дан набор работ $A = \left\{a_k = \left\langle b_k^A, f_k^A, c_k^A, d_k^A, r_k^A \right\rangle \mid k = \overline{1,n}\right\}$ - набор работ как в задаче построения расписания.

Задано множество типов возможных процессоров $Z = \left\{z_j = \left\langle s_j^z, R_j^z, v_j^z \right\rangle \mid j = \overline{1,m^z}\right\}$ где s_j^z - проиводительность процессоров типа j в операциях в единицу времени, R_j^z - множество функциональных фозможностей процессора типа j, v_j^z - стоимость процессора..

Необходимо построить систему $M^* = (m_1, \ldots, m_{m^z})$. Такую что

$$M^* = arg \min_M \sum_{j=1}^{m^z} m_j \cdot v_j^z.$$

И в данной системе возможно построение статико-динамического расписания для набора работ A (см. п 4.1).

Сложность задачи — что NP задача класса.

5 Обзор

5.1 Алгоритмы построения расписаний

Что рассматриваем возможные подходы к решению задачи.

Нормальных алгоритмов нет, есть идеи, но они не подходят по тем или иным причинам.

В [15] рассмотрены подходы для организации вычислений в системах реального времени в разных конфигурациях по прерыванию ресурсов - что важно для систем рельаного времени. И системы без прерываний могут работать надежнее своих собратьев.

В [16] рассмотрено оптимальное решение на основе поиска максимального потока в сети.

В [24] представлен алгоритм ?

5.2 Подходы к решению задач построения статико-динамическ расписаний.

5.2.1 Алгоритмы основанные на декомпозиции задачи.

В [24] описан алгоритм построения статико-динамического расписания на основе метода декомпозиции задачи, который заключается в разделении задачи на 2 подзадачи:

1. Назначение разделов на процессоры вычислительной системы.
2. Создание набора окон для каждого процессора системы для назначенных для него разделов.

Первый шаг алгоритма основывается на минимизации определенного критерия, например минимизации передачи сообщений по сети. Второй шаг использует алгоритмы для построения статико-динамического расписания в однопроцессорной системе.

5.2.2 Алгоритмы основанные на схеме муравьиных колоний.

Идея муравьиных алгоритмов основана на моделировании поведения муравьев при нахождении кратчайшего пути от муравейника к источнику пищи. Муравьи при перемещении оставляют особое вещество феромон, который используется в дальнейшем другими муравьями при выборе маршрута. Чем выше концентрация феромона на том или ином маршруте, тем выше вероятность того, что муравьи будут выбирать

для движения именно этот маршрут. Общую схему работы муравьиных алгоритмов можно представить в следующем виде:

1. Задание начального количества феромона на ребрах графа.
2. Определение количества и начального положения муравьев.
3. Поиск муравьями решений в соответствии с заданным алгоритмом построения маршрута.
4. Обновление количества феромона на ребрах в зависимости от качества полученных решений.
5. Если условие останова не выполнено, то переход к п.2.

Для решения задачи построения статико-динамического расписания используется алгоритм, описанный в [22]. Строится граф $G =< N, A >$, где:

- $N = \{n_i^+, n_i^- | i \in [1..n]\} \cup O$ – множество вершин;
- $A = \{(n_i^+, n_j^-)(n_i^+, n_j^+)(n_i^-, n_j^-) | i, j \in [1..n], i \neq j\} \cup (O, n_i^+)(O, n_i^-) | i \in [1..n]$ – множество ребер.

Каждой размещаемой работе соответствует две вершины в графе - размещение с закрытием окна n_i^- и размещение без закрытия окна n_i^+.

Муравьиный алгоритм строит маршруты, в которых все вершины включены ровно один раз. Каждому такому маршруту соответствует последовательность работ, такая что, в нее включены все работы из исходно заданного набора и каждая работа включена в данную последовательность один раз.

Таким образом, при помощи описанного алгоритма по заданной последовательности работ однозначно строится корректное расписание. Однако оно не допускает прерывания работ и может быть построено только для однопроцессорной системы.

5.3 Алгоритмы основанные на нахождении максимального потока для задач построения расписаний с прерываниями.

Методика использования алгоритмов нахождения максимального потока в сети для построения расписаний с прерываниями была предложена в работах [3, 4].

В работе [5] был рассмотрен алгоритм построения расписаний для неоднородной многопроцессорной системы (процессоры имеют разную производительность). Прерывания и переключения работ не требуют временных затрат. В [6] был предложен алгоритм для задачи, в которой учитывался ограниченный объем памяти процессоров. В работах [7, 8] предложен алгоритм для случая, когда длительности выполнения работ линейно зависят от количества выделенного им ресурса и учитываются затраты на обработку прерываний и переключений.

Рассматривается вычислительная система, состоящая из m процессоров. Каждый процессор $j = 1, ..., m$ характеризуется производительностью s_j. Предполагается, что процессоры упорядочены по невозрастанию производительностей ($s_1 \geq s_2 \geq \ldots \geq s_m$). Задан набор работ $N = 1, 2, ..., n$, подлежащих выполнению. Каждая работа i характеризуется своим директивным интервалом $A_i = [s_i, f_i]$ (т.е. работа i может быть начата не ранее момента времени s_i и должна быть выполнена не позднее момента времени f_i), а также сложностью (или объемом работы процессоров по ее выполнению) Q_i , $i = 1, ..., n$.

Работа сложности Q_i может быть выполнена на процессоре j за время Q_i/s_j. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором и каждый процессор может выполнять не более одной работы. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Предполагается, что прерывания и переключения не требуют временных за-

трат. Задача состоит в том, чтобы определить, существует ли допустимое расписание (т.е. расписание, позволяющее выполнить все работы в их директивные интервалы на имеющихся процессорах) и, если оно существует, указать такое расписание.

Алгоритм, находящий точное решение рассматриваемой в этом разделе задачи, был разработан и реализован как комбинация полиномиальных алгоритмов, предложенных в [3]. Будем предполагать, что имеется t типов процессоров, каждый из которых характеризуется скоростью s_j , $j = 1, ..., t$, всего имеется m_j , $j = 1, \ldots, t$, процессоров j-го типа, $m = \sum_{j=1}^{t}(m_j)$

В [3] доказывается, что такая задача может быть сведена к задаче поиска максимального потока в сети, построенной специальным образом. Для поиска максимального потока в построенной сети применялся алгоритм «поднять-и-в-начало», описанный в [9]. Предложенный алгоритм находит точное решение задачи о многопроцессорном расписании за время $O(t^3n^3)$, допуская при этом не более $2(n^2 + 2mn - 3n - m + 1)$ прерываний.

5.3.1 Похожие задачи

В [16] рассмотрено оптимальное решение на основе поиска максимального потока в сети задачи потроения расписания с прерываниями. Там все отлично за полиномиальное время решается и если бы у нас был заранее известный набор окон, то задача решалась бы легко, но этот набор окон нам как раз нужно найти, что делает прямое применение данного алгоритма невозможным для решения поставленной задачи.

5.3.2 Обзор.

1 путь от минимального числа процессоров к макс (и наоборот + и минусы)+ стоимость ввести

5.3.3 Существующие алгоритмы .

У Фуругяна [14] рассмотрен алгоритм построения оптимальной конфигурации на заданном числе процессоров - идет подбор значений производительностей методами линейного программирования с ограничениями.

5.4 Похожие задачи

???

5.5 Предложенный алгоритм

5.6 Выводы.

Существующие алгоритмы построения статико-динамического расписания либо не допускают прерывания работ, которые современные системы реального времени могут обеспечивать, или не имеют высокой точности решения. В их основе лежит алгоритм построения однопроцессорного статико-динамического расписания. Существует подход к решению близкой задаче построения расписания работ с прерываниями в многопроцессорной системе, отличающейся от задачи построения статико-динамического расписания отсутствием учета затрат на переключение между окнами разных разделов, основанный на поиске максимального потока в сети и находящий оптимальное решение.

В [17] описан классический алгоритм составления расписания.

Муравьиный алгоритм. Линейное программирование. Динамическое программирование. Генетические алгоритмы. Эвристические алгоритмы.

6 Алгоритм построения многопроцессорного стати динамического расписания, основанного на методе поиска максимального потока в сети

Алгоритм состоит из трех основных этапов: 1. Построение транспортной сети (ориентрованного графа). 2. Нахождение максимального потока в транспортной сети. 3. Восстановление расписания из полученного потока.

Общая схема алгоритма. 1. Выполняется операция Инициализация сети. 2. Строится поток: 2.1.выбирается раздел, список переполненных вершин которого не пуст (если такого раздела нет, то переход к пункту 3, далее для этих вершин , выполняются операции: Возможная разрядка вершины . 2.2. пока списки переполненных вершин раздела и вершин-интервалов не пусты: 2.2.1. Для всех переполненных вершин-интервалов-процессоров v выполняется операция Разрядка вершины v . 2.2.2. Для одной переполненной вершины раздела u выполняется операция Разрядка вершины u . 2.3. переход к пунтку 2. 3. Если есть не полностью размещенные работы, то для первой такой работы выполняется операция Удаление работы из сети и переход к пунтку 2. 4. Выполняется операция Восстановление расписания.

6.1 Свойства алгоритма построения статико-динамического расписания на основе алгоритма поиска максимального потока в сети.

Результат работы алгоритма удовлетворяет условиям корректности расписания. В алгоритме основной операцией является операция разрядки вершины, которая в свою очередь состоит из операций проталкивания из вершины в вершину и операций подъема вершины. Операция подъема влияет только на очередность проталкиваний и не

влияет на корректность расписания. Операция проталкивания устроена таким образом, что при отсутствии избыточного потока в сети будут выполнены все условия корректности расписания:

Операция проталкивания, связанная с вершиной-интервалом, выполняет операции корректировки окон, которая добавляет в соответствующую вершину переключения окон между разделами — резервирует время для переключения между разделами, чтобы другие работы не могли его использовать. Окна строятся по длительностям работ одного раздела, следующих подряд. Повторный анализ сети на не полностью размещенные задачи исключает таковые из сети.

Так как размещение происходит сразу после первого проталкивания работы на вершину интервал-процессор данного процессора, то дальнейшее размещение работ этого раздела возможно только на этом процессоре (так как пропускные способности к остальным процессорам равны). До того момента, пока работа не захочет сделать проталкивание в исток, что равносильно тому, что не удается разместить весь раздел на этом процессоре. Тогда выполняется вторая операция. Поток отзывается из всех вершин от всех вершин-работ данного раздела (корректность расписания остается, так как учитываются все переключения окон). И пропускные способности до остальных процессоров восстанавливаются. Поэтому не может быть нарушено условие: Работы одного раздела выполняются на одном процессоре. Построение сети можно осуществить за $O(pn^2)$. Восстановление расписания осуществляется за $O(pn)$.

Для одного процессора верно следующее: Если n — число работ, то $V \leq 3n + 2$ и $E \leq 3n + 2n^2$. Первый шаг алгоритма заключается в возможной разрядке всех вершин одного раздела, который в худшем случаем занимает $2n^2$ операций проталкивания. За этим следует разрядка всех вершин-интервалов, что занимает в худшем случае $2n(n + 1)$ операций протакивания. Эти действия выполняются для каждого раздела, значит нужно $n(2n^2 + 2n(n + 1))$ операций проталкивания. Далее покажем, что для каждой пары вершина-работа – вершина-интервал возможны не более 10 про-

талкиваний. Рассмотрим функцию высоты для вершин-работ, она всегде больше 0 . И если она равна 11, то протакивание осуществляется в исток и этот поток больше в сети не появляется. Высота истока, равная 10 обеспечивает 5 попыток проталкивания из одной и той же вершины-работы в вершину-интервал. Так как высоты вершин- интервалов не уменьшаются: когда проталкивание возвращает поток в вершину-работу, то высота становится больше высоты вершины-работы из которой было осуществлено проталкивание. Следовательно для всех пар вершин-работ и вершин-интервалов- процессоров возможно не больше проталкиваний. При удалении работы нужно повторить алгоритм с начала, максимум можно удалить n работ, что соответствует n итерациям алгоритма —$n(2n^2 + 2n(n + 1) + 20n^2)$. Общая сложность поиска потока для одного процессора алгоритмом равна $O(n^4)$, если требуется удаление работ, и $O(n^3)$, если получается спланировать все работы. Тогда сложность для всех процессоров $O(pn^3)$.

Так как имеется только K попыток перераспределения раз дела, то сложность $O(Kpn^3)$ при полном планировании расписания и $O(Kpn^4)$ — при удалении некоторых работ.

Алгоритм подходит для построения и однопроцессорного расписания

7 Алгоритм архитектурного синтеза

Существует два подхода для решения задачи синтеза архитектур.

Первый подход — используя алгоритм из п 5.3 начать строить расписание, когда в системе есть один только процессор. Далее по мере того, как будут появляться задачи, которые невозможно разместить на этом процессоре — добавлять новые, до тех пор, пока станет возможным выполнение расписания.

Второй подход — построить систему, в которой гарантировано может быть построено расписание. Дальше, используя определенные оценки пытаться снизить общую стоимость синтезируемой системы.

7.1 Первый вариант

Алгоритм начинает из состояния, когда нет ни одного процессора.

Шаги алгоритма:

1. Оценка системы и выбор расширенной системы.
2. Построение расписания в данной системе,
3. Если расписание невозможно построить, то перейти к шагу 1, иначе решение найдено

Как видно из схемы алгоритма - самым важным шагом в этом алгоритме является выбор расширенной системы. Именно в нем будет заложен фундамент того, что мы получим хорошее решение в смысле стоимости готовой системы.

Оценка системы и выбор расширенной системы Для выбора расширенной системы мы будем использовать итеративные изменения, это значит, что общее число процессоров в системе не может меняться произвольно, а только добавлением или заменой процессора (или сразу несколькими такими операциями за раз). То есть в ходе работы нашего алгоритма число процессоров не будет уменьшатся и в конце концов он завершится.

Для оценки наиболее подходящего процессора мы будем использовать следующие метрики:

Суммарная сложность каждого раздела (напомню, что раздел целиком должен размещаться на процессоре) $C_A^i = \sum\limits_{a_k \in A} c_k^A, i = \overline{1,q}$

Суммарная сложность, выполняемая на процессоре типа $j = \overline{1,m^z}$ $C_M^j = p_j^z \cdot T$ за интервал планирования T

Ресурсы, необходимые для задач раздела $R_A^i = \cup_{a_k \in A} r_k^A$

Ресурсы, предоставляемые процессором типа $j = \overline{1, m^z}$ $R_M^j = R_j^z$

При появлении ситуации, когда задача раздела i не размещается в системе будем выделять процессор типа j такой, что $R_A^i \subseteq R_M^i$, $C_A^i \leq C_M^j$ и v_j^z - минимальна среди возможных.

Это жадный алгоритм, его просто реализовать. Алгоритм не смотрит на ситуацию в целом, а только в конкретный момент, когда определенный раздел не может быть размещен на процессор. Могут возникать ситуации, когда можно заменить 2 процессора на один и этот алгоритм этого не заметит.

7.2 Второй вариант

В этом варианте мы будем отталкиваться от системы в которой гарантировано можно построить расписание, а потом будем сокращать ее.

Так как в нашей задаче каждый раздел должен выплняться целиком на процессоре, то для начала достаточно выделить каждому разделу свой процессор минимальной стоимости, на котором раздел может выполниться. Вопрос о возможности построения расписания проверяется несколькими критериями:

1. Совпадение необходимых и предоставляемых ресурсов.

2. $C_A^i \leq C_M^j$ - необхдимое условие возможности построения расписания

3. Если 2 предыдущих пункта выполнены, то с помощью алгоритма [16] построением максимального потока можно дать ответ о возможности построения расписания для однопроцессорной системы с прерываниями (тут не будет времен переключения между ними, поэтому это достаточно просто)

После выполнения этих шагов заполним таблицу H_{ij}, где каждая строка отвечает за раздел, а каждый столбец за процессор определенного типа. Заполним единицами те

ячейки в которых раздел i может быть выполнен на процессоре j.

Для начала в каждой строке выберем процессор с 1 и с минимальным весом - это будет наша начальная конфигурация, состоящая из числа процессоров, равных числу разделов в задаче.

Шаги алгоритма:

1. Оценra минимального набора процессоров (оценка снизу)
2. Построение расписания на наборе построенном выше
3. Расширение числа процессоров засчет обнаружения неразмещенных работ
4. Если удалось построить расписание, то мы нашли ответ на поставленную задачу

Теоремы про ограничения.

2 пункт описан в ?, остается подробнее описать пункт 1 и пункт 2.

7.3 Оценка минимального набора процессоров

Для оценки минимального набора процессоров производится две операции: оценка необходимых функциональных возможностей процессоров. Далее подсчет общей сложности выполнения задача по функциональным возможностям и на основе построение оценки общего числа процессоров и их возможностей

7.4 Расширение

Когда работы не может быть размещена в системе - ее поток направляется обратно в источник. Далее (отличие от алгоритма построения расписания) в зависимости от

этой работы и раздела к которому она привязана выделяется еще один процессор и работы размещается на него. Далее алгоритм работает с расширенным набором процессоров.

8 Экспериментальное исследование

8.1 Общее

Когда дело доходит до экспериментального исследования неточных алгоритмов очень важно понимать, на ккаких данных проверять его работоспособность. Так например может оказаться, что алгоритм прекрасно справляется с тестовыми данными, однако они далеки от реальности, и потому при внедрении в реальную систему результаты алгоритма сильно отличаются от тестовых.

Так как большинство проектов реального времени связано с военной тематикой, то получить о них хоть какие-нибудь данные - сложная задача. Однако системы реального времени используются не только в военной сфере, она распространена в технологическом производстве, в космических программах и др.

Для анализа структуры кокличества работ и информации о их длительности выполнения, а так же о возможностях процессоров, используемых в реальных системах были использованы открытые источники:

Так же при учете генерируемых данных мы будем пользоваться теоремой о верхней границе коэффиицента исопльзования. Которая гласит, что при монотонном планировании периодических работ их утилизация ресурсов процессорного времени не превышает $n(2^{\frac{1}{n}} - 1)$, где n - число периодических задач . Таким образом мы будем выбирать нагрузкук на процессоры таким образом, чтобы занимать примерно 70 - 80 процентов процессорного времени.

8.2 Экспериментальное исследование построения расписания.

Экспериментальное исследование проводилось в системе с ОС Windows 7 64bit, процессором Intel Core i3-2370M 2.39GHz, ОЗУ 8Гб. На сгенерированных исходных данных алгоритм построения однопроцессорного статико-динамического расписания разместил все работы. При этом время построения решения не сильно зависит от загрузки процессора и не превышает 1, 2 секунды для 1000 работ. Так же число разделов почти не влияет на точность и время решения. На сгенерированных исходных данных алгоритм продемонстрировал полную планируемость работ при загрузке процессоров до 90% для 2-х процессоров. Увеличение числа процессоров плохо влияет на число планируемых работ, так для 3-х процессоров планируемость всех работ возможна при загрузках процессора до 70%, при больших значениях — возможно размещение 99% работ. Увеличение числа процессоров до 8 приводит к размещению 99% процентов работ при загрузке до 70% процентов, и к размещению 90% работ при загрузке до 90%. Время выполнения алгоритма построения многопроцессорного статико-динамического расписания растет с увеличением загрузки процессоров для 8 процессоров, это связано с тем, что число размещенных работ меньше 100% и требуется дополнительное время на исключение работ, для которых не нашлось места в статико-динамическом расписании. И время построения такого расписания может занимать 20 мин. Для числа процессоров от 2 до 4 время построения статико-динамического расписания не превосходит 4 мин и точность находится на уровне 99-100% при загрузках процессора до 90%.

8.3 Экспериментальное исследование построения архитектур

Экспериментальное исследование проводилось на искусственно сгенерированных данных, на основе реально существующих длительностей задач в системах реального времени.

8.4 Выводы

Что в итоге из двух алгоритмов можно составить.

9 Программная реализация

Некоторые особенности программной реализации. Форматы входных и выходных данных. Используемые структуры. Используемые языки - обоснование использования тех или иных средств разработки и применяемых подходов.

10 Заключение

Задача построения статико-динамических расписаний и синтеза архитектур возникает при проектировании информационно-управляющих систем реального времени с архитектурой интегрированной модульной авионики. Для ряда задач построения расписаний с прерываниями высокую эффективность по критериям точность и вычислительная сложность показали алгоритмы, основанные на нахождении максимального потока в транспортной сети. Основными проблемами, препятствующими применению известных алгоритмов, основанных на нахождении максимального потока в

транспортной сети, для построения статико-динамических расписаний являются: проблема учета принадлежности работ к разделам и проблема построения набора окон. Проблемы учета принадлежности работ к разделам и построения набора окон в предложенном алгоритме решаются за счет модификации алгоритма нахождения максимального потока в сети. Экспериментальное исследование свойств алгоритма показало его высокую эффективность по критериям точность и вычислительная сложность на многих классах исходных данных. Для построения архитектур учитываются те же особенности, поэтому за счет использования конструктивного создания архитектуры на основе предложенного алгоритма построения расписаний получается построить систему близкую к оптимальной.

Список литературы

[1] Костенко В. А. Архитектура программно-аппаратных комплексов бортового оборудования // Изв. вузов. Приборостроение. 2017. **60**. № 3. С. 229–233.

[2] VxWorks 653 multi-core edition [PDF] (https://www.windriver.com/-products/product-overviews/vxworks-653-product-overview-multi-core/vxworks-653-product-overview-multi-core.pdf)

[3] Federgruen A., Groenevelt H. Preemptive scheduling of uniform machines by ordinary network flow technique // Management Science. 1986. **32**. N 3. P. 341–349.

[4] Gonzales T., Sanhi S. Preemptive scheduling of uniform processor systems // J. Assoc. Comput. Mach. 1978. **25**. N 1. P. 92–101.

[5] Фуругян М.Г. Некоторые алгоритмы анализа и синтеза многопроцессорных вычислительных систем реального времени // Программирование. 2014. №1. С. 36–44.

[6] Гуз Д.С., Фуругян М.Г. Планирование вычислений в многопроцессорных АСУ реального времени с ограничениями на память процессоров // Автоматика и телемеханика. 2005. №2. С. 138–147.

[7] Косоруков Е.О., Фуругян М.Г. Некоторые алгоритмы распределения ресурсов в многопроцессорных системах //Вестн. Моск. ун-та. Сер.15. Вычисл. матем. и киберн. 2009. № 4. С. 34–37.

[8] Фуругян М.Г. Составление расписаний в многопроцессорных системах с несколькими дополнительными ресурсами // Изв. РАН. ТиСУ. 2017. №2. С. 57–66.

[9] Кормен Т., Лейзерсон Ч., Ривест Р. и др. Алгоритмы: построение и анализ. М.: МЦНМО, 2005.

[10] Балаханов В.А., Костенко В.А. Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007. – С.148-156.

[11] Balashov V. V., Balakhanov V. A., Kostenko V. A. Scheduling of computational tasks in switched network-based ima systems // Proc. International Conference on Engineering and Applied Sciences Optimization. — National Technical University of Athens (NTUA) Athens, Greece, 2014. — P. 1001–1014.

[12] Ford L. R., Fulkerson D. R. Maximal Flow through a Network // Canad. J. Math. 1956. - P. 399-404.

[13] Dinic E. A. Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation // Soviet Math Doklady. 1970. V. 11. P. 1277-1280.

[14] Фуругян М. Г. СИНТЕЗ МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ ПРИ ПОСТРОЕНИИ РАСПИСАНИЙ С ПРЕРЫВАНИЯМИ И ДИРЕКТИВНЫМИ ИНТЕРВАЛАМИ // Известия РАН Сер.: Теория и системы управления. — 2019. — № 2. — С. 41–46.

[15] Гончар Дмитрий Русланович, Фуругян Меран Габибуллаевич Эффективные алгоритмы планирования вычислений в многопроцессорных системах реального времени // УБС. 2014. №49.

[16] FEDERGRUEN A., GROENEVEL H.T. Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique // Management Science. - 1986. - Vol. 32, №3. - P. 341-349.

[17] GONZALES T., SAHNI S. Preemptive Scheduling of Uniform Processor Systems // Journal of the Association for Computing Machinery, January. - 1978. - Vol. 25, №1. - P. 92-101.

[18] WIND RIVER VXWORKS 653 3.0 MULTI-CORE EDITION

[19] Федосов Е. Косьянчук В. Сельвесюк Н. Интегральная модульная авионика

[20] Smeliansky, Ruslan B.A, Костенко. (2018). Проблемы построения бортовых комплексов с архитектурой интегрированной модульной авионики. 3. в журнале Радиопромышленность

[21] Хакимов Дмитрий Валерьевич, Киселев Сергей Константинович Историческое развитие и современное состояние комплексов бортового оборудования летательных аппаратов (Окончание) // Вестник УлГТУ. 2017. №3 (79).

[22] Дегтярев Алексей Робертович, Киселев Сергей Константинович Отказоустойчивые реконфигурирующиеся комплексы интегрированной модульной авионики // Электротехнические и информационные комплексы и системы. 2016. №1.

[23] Дегтярев Алексей Робертович, Киселев Сергей Константинович Отказоустойчивые реконфигурирующиеся комплексы интегрированной модульной авионики // Электротехнические и информационные комплексы и системы. 2016. №1.

[24] Дегтярев Алексей Робертович, Киселев Сергей Константинович АЛГОРИТМ РАСПРЕДЕЛЕНИЯ ЗАДАЧ В МНОГОПРОЦЕССОРНЫХ КОМПЛЕКСАХ ИНТЕГРИРОВАННОЙ МОДУЛЬНОЙ АВИОНИКИ

[25] https://habr.com/ru/post/395973/

[26] http://www.computer-museum.ru/articles/po-i-os-dlya-sistem-realnogo-vremeni/1925/

[27] А.М.СУЛЕЙМАНОВА СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ Учебное пособие

A Результаты экспериментального исследования

Подробные графики и таблицы.

B Программная реализация

UML диаграмма разработанной системы.

TODO Как разделы формируются