Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Курсовая работа

“Методы тестирования сетевого стека ARINC 653-совместимой операционной системы реального времени”

Автор:
Русецкий Илья Владиславович

Научный руководитель:
Чепцов Виталий Юрьевич

Москва, 2022

Содержание

1 Введение

Тестирование является обязательной частью разработки любой системы, ведь перед ее эксплуатацией необходимо быть уверенным в том, что она соответствует определенным требованиям, будет вести себя корректно, а также не содержит ошибок и дефектов.

Методы тестирования различаются для всех систем и зависят от множества факторов: от целей тестирования, от структуры тестируемой системы, от доступных инструментов и т.д. Поэтому разработка для новой системы конкретного метода тестирования, возможно, основанного на более общем методе или на аналогичных, является актуальной задачей.

Целью данной курсовой работы является проектирование, обоснование и реализация метода тестирования сетевого стека SpaceWire в операционной системе реального времени JetOS, целью которого является проверка соответствия сетевого стека SpaceWire его спецификации, а также некоторым другим требованиям, сформулированным при разработке сетевого стека.

Так как конкретных методов тестирования систем, аналогичным данной, найдено не было, проектирование метода основано на общем плане тестирования, описанном в [1]:

1. Определение требований, свойств и ограничений, соблюдение и выполнение которых проверяет тестирование
2. Определение критерия полноты тестирования
3. Построение набора тестов
4. Выполнение тестов, получение их результатов
5. Анализ результатов тестирования

В соответствии с этим планом можно выделить следующие задачи, решение которых необходимо для достижения цели:

1. Изучить систему, в которой будет проводится тестирование, а именно
    - Понятие операционной системы реального времени
    - Структуру и принципы работы конкретной ОСРВ – JetOS (стандарт ARINC-653, модульный подход и т.д.)
    - Спецификацию SpaceWire, реализацию этого сетевого стека в JetOS

2. Определить набор требований к сетевому стеку, исходя из спецификации SpaceWire и постановки задачи, а также критерий полноты для построения набора тестов
3. Построить набор тестов для проверки всех сценариев использования системы, затрагивающих различные требования
4. Реализовать набор тестов в JetOS
5. Запустить тесты
6. Проанализировать результаты тестирования

Из перечисленных задач следует структура данной курсовой работы: в пункте 3 кратко описывается система, в которой будет проводится тестирование, а пункт 4 описывает разработку и реализацию метода.

2 Описание системы, в которой проводится тестирование

2.1 Операционные системы реального времени

Операционные системы реального времени – это операционные системы, в которых, в отличие от пользовательских систем, самым важным параметром является время. Такие ОС используются на производстве, где требуются точность, строгая детерминированность и безопасность. Часто для ОСРВ очень важно, чтобы при любых нагрузках и любых сценариях использования система вела себя заранее определенным образом.

2.2 ОСРВ JetOS и стандарт ARINC 653

В данной работе рассматривается ОСРВ JetOS и ее реализация для интегральной микросхемы 1892ВМ15Ф [2]. Данная операционная система разработана в соответствии со стандартами ARINC SPECIFICATION 653 (PART 1-5) – REQUIRED SERVICES в редакции 2019 г. и ARINC SPECIFICATION 653 (PART 2-4) – EXTENDED SERVICES в редакции 2015 г. Стандарт ARINC 653 определяет интерфейс прикладного программного обеспечения на основе идеи временного и пространственного разделения ресурсов.

Согласно этому стандарту все программное обеспечение, исполняющееся на микросхеме называется интеграционным модулем, который состоит из разделов. Раздел является единицей планирования, а в его рамках может выполняться несколько параллельных процессов. Разделы делятся на прикладные и системные. Системным разделам, в отличие от прикладных, ядро предоставляет особые функции. Интерфейс прикладных разделов однозначно определен в спецификации ARINC, а вот на системные разделы почти не накладывается ограничений. Контактировать друг с другом разделы могут исключительно через каналы.

Также очень важной составляющей стандарта ARINC-653 является то, что все разделы и выделяемые им ресурсы определяются перед запуском системы, во время ее конфигурирования. После этого происходит загрузка и инициализация разделов и перераспределение ресурсов уже невозможно. [3]

Разумеется, стандарт ARINC 653 гораздо шире, однако здесь были лишь кратко перечислены его особенности важные в контексте данной работы.

2.3 Модульный подход построения JetOS

Системные разделы JetOS строятся в соответствии с модульным подходом [4], согласно которому они представляют собой набор модулей (компонент), каждый из которых имеет свой процесс инициализации и т.н. “активность” (activity). Перечень этих модулей, их параметры инициализации и конфигурирования описываются в файле composition.yaml в директории с кодом системного раздела. Взаимодействие модулей осуществляются через порты, которые также определяются этим файлом.

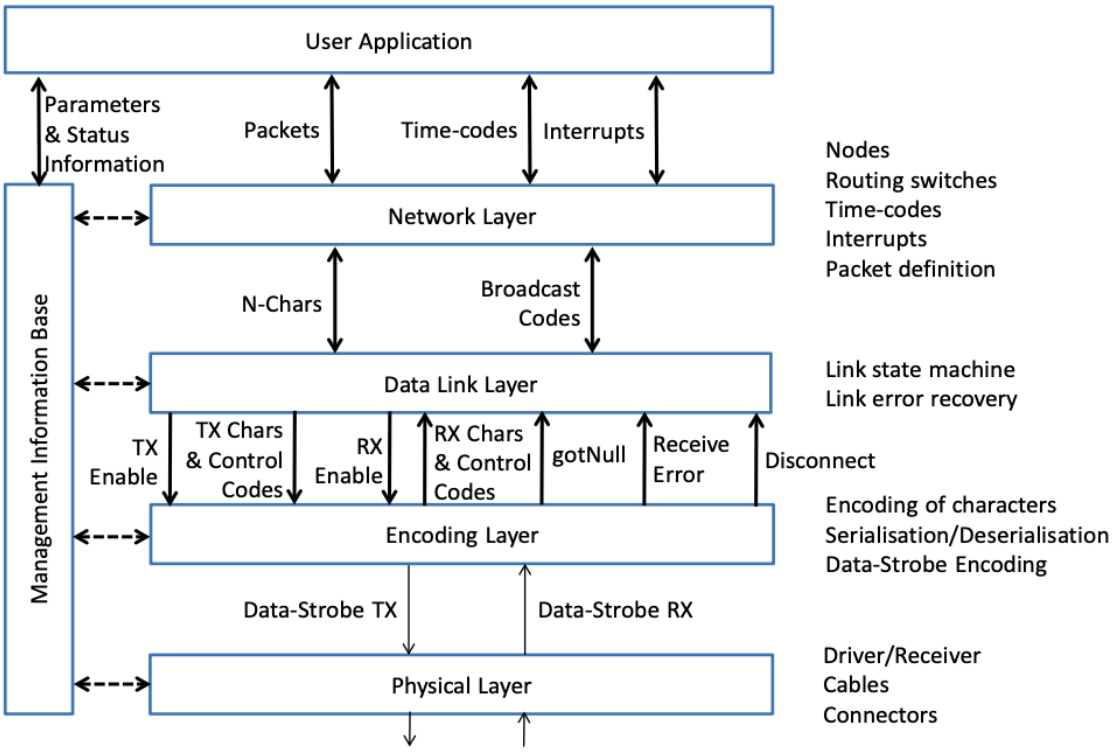
Системный раздел собирается согласно этому файлу из кода модулей. После его запуска он сначала обратится к функции инициализации каждого модуля, а после этого будет выполнять их активности. С точки зрения данной работы важным представляется то, что драйвер и планировщик SpaceWire будут представлять собой модули VM15_SPACEWIRE_SWITCH и SPACEWIRE_NET_SCHEDULER.

2.4 Сетевой стек SpaceWire

Протокол SpaceWire описывается в спецификации стандарта ECSS-E-ST-50 -12C [5]. Согласно ему сеть SpaceWire состоит из следующих уровней:

1. Физический. Отвечает за передачу информации по кабелю.
2. Символьный. Отвечает за передачу символов с проверкой четности, передачу символов контроля потока передачу маркеров времени.
3. Канальный. Отвечает за контроль состояния канала.
4. Сетевой. Отвечает за объединение символов в пакеты, а также за организацию адресации. Первый байт пакета может являться номером физического порта в коммутаторе, он отбрасывается при отправке, а может содержать логический адрес из таблицы коммутатора. Заканчивается пакет специальным символом.

Схематично данная структура изображена на следующей схеме [5]:



Однако сетевой стек построенный только в соответствии с этим стандартом не будет отвечать требованиям детерминированности времени передачи информации по сети, выполнение которых необходимо в ОСРВ. Поэтому разработка сетевого стека SpaceWire в JetOS также основывалась на протоколе SpaceWire-D [6], согласно которому отправка пакетов по сети регулируется планировщиком строго в соответствии с расписанием и текущим временем.

2.5 Реализация SpaceWire в JetOS

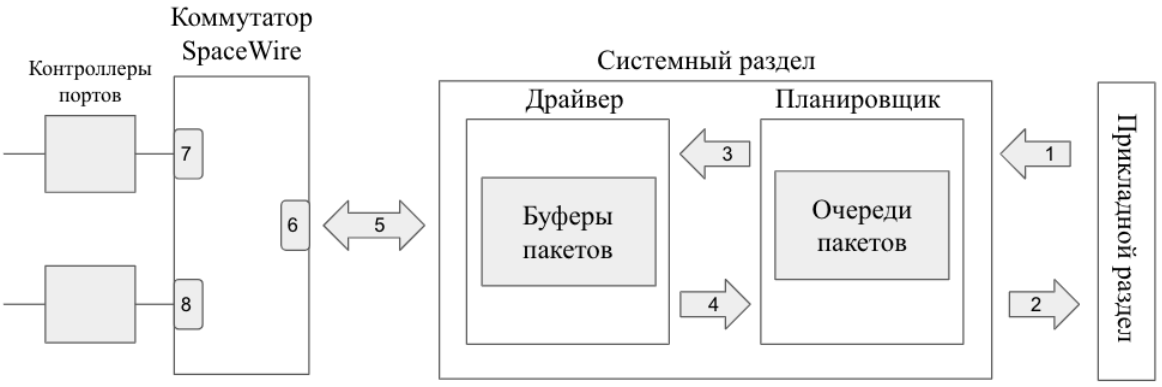
Сетевой стек SpaceWire в JetOS в случае модульных проектов для микросхемы 1892ВМ15Ф состоит из программной части и аппаратной. В программную часть входят планировщик и драйвер, представляющие собой модули VM15_SPACEWIRE_SWITCH и SPACEWIRE_NET_SCHEDULER. Аппаратная часть состоит из коммутатора и контроллеров портов.

Отправка сообщений по сети SpaceWire происходит следующим образом:

1. Прикладной раздел через ARINC Queueing Port отправляет сообщения системному разделу в планировщик.

2. Планировщик помещает сообщение в очередь, а далее на основании адреса сообщения принимает решение о моменте отправки сообщения.
3. Планировщик через порт связи между компонентами передает сообщение драйверу
4. Драйвер через DMA канал передает сообщение аппаратному коммутатору
5. Коммутатор на основании адреса физического порта принимает решение о коммутации и передает сообщение либо обратно в DMA канал, либо контроллеру канального уровня SpaceWire.
6. Контроллер осуществляет отправку сообщения по кабелю.

Прием сообщения происходит согласно такому же алгоритму. Кратко он представлен на следующей схеме:



1, 2 – ARINC Queueing Ports; 3, 4 – порты между компонентами; 5 – DMA канал; 6, 7, 8 – порты коммутатора

Далее следует основная часть данной курсовой работы – создание метода тестирования. Приведенное выше краткое описание системы позволяет понять, в каком виде будут конструироваться тесты, а также проанализировать их результаты.

3 Реализация задач

3.1 Составление перечня требований

Согласно цели разрабатываемого метода тестирования, заявленной во введении, будем формулировать требования к реализации сетевого стека на основании стандарта ECSS-E-ST-50-12C Rev.1 протокола SpaceWire и на основании стандарта SpaceWire-D. Требования следующие:

1. Сетевой стек должен предоставлять пользователю сервисы для отправки и приема пакетов по сети в определенный физический порт и по определенному адресу
2. Пакеты должны доходить до верного адресата, если логические и физические адреса заданы корректно, либо сбрасываться, если таких адресов не существует
3. Все пакеты с верными адресами должны доходить до адресата, если очередь планировщика не заполнена, в противном случае пакеты сбрасываются
4. Пакеты должны приходить адресату с корректным содержимым
5. Приоритезация пакетов должна осуществляться в строгом соответствии с конфигурацией планировщика
6. Время доставки пакеты должно зависеть только от маршрута доставки, то есть на одном и том же пути время не должно отличаться
7. При обрыве соединения отправка пакетов должна останавливаться, до его восстановления

В качестве критерия полноты в данном случае выберем так называемый “критерий покрытия элементов требований” [1], то есть для каждого требования будет спроектирован тест, моделирующий основные сценарии, в которых затрагивается данное требование.

3.2 Проектирование набора тестов

Тесты для проверки выполнения приведенных требований реализуют сценарии, их затрагивающие, в виде модульных проектов для JetOS. Модульные проекты есть следующих двух видов:

Модульный проект 1-го типа
Содержит прикладной и системный разделы. Системный раздел состоит из 4 компонент: планировщика сети, драйвера SpaceWire, а также двух портов для связи с прикладным разделом (читающий и пишущий). Прикладной раздел содержит два параллельных процесса: send_process и rcv_process. Первый отправляет в сеть определенное число сообщений, а второй принимает их.

Модульный проект 2-го типа
Также содержит прикладной и системный разделы, причем последний совпадает с системным разделом проекта 1-го типа. Прикладной раздел же содержит единственный процесс, который отправляет в сеть сообщение и сам же принимает его, после чего отправляет сообщение снова.

На основании общего вида модульных проектов для каждого требования был построен тест, реализующий основные сценарии, затрагивающие данные требования:

Тест для тр. 2: 1-ый тип проекта. В файле конфигурации системного раздела проекта процессу rcv_process присваивается несколько логических адресов. Процесс send_process отправляет по каждому адресу, а также через разные физические порты коммутатора, некоторое число сообщений. В rcv_process проверяется, что все сообщения дошли. Далее send_process отправляет сообщения по несуществующим адресам. С помощью логов планировщика проверяется, что эти сообщения были сброшены из-за неверных адресов.

Тест для тр. 3: 1-ый тип проекта. В файле конфигурации указывается размер очереди планировщика. Процесс send_process отправляет сообщения rcv_process, каждое сообщение содержит в себе его номер. Принимающий процесс сверяет порядковые номера сообщений и проверяет, что были не доставлены только те сообщения, порядковые номера которых больше размера очереди. С помощью логов планировщика проверяется, что сообщения были сброшены из-за переполнения очереди.

Тест для тр. 4: 1-ый тип проекта. Процесс send_process отправляет некоторое число сообщений с определенным содержимым. Процесс rcv_process проверяет корректность содержимого и выводит информацию о неверных сообщениях.

Тест для тр. 5: 1-ый тип проекта. В файле конфигурации системного раздела резервируется несколько слотов планировщика, каждый содержит свой логический адрес. Процесс send_process по очереди отправляет сообщения по разным логическим адресам. Процесс rcv_process проверяет, что сообщения доставляются в соответствии с приоритетами, установленными в файле конфигурации, а с помощью логов проверяется, что сообщения были отправлены в определенные для них слоты.

Тест для тр. 6: 2-ой тип проекта. Сообщения отправляются единственным процессом по трем маршрутам: через нулевой порт в коммутаторе обратно в процесс, через порт 1 по кабелю в порт 2 обратно процессу и наоборот через порт 2 в порт 1. По каждому маршруту отправляется определенное количество сообщений, для каждого сообщения считается время с момента отправки до момента получения. Для всех трех маршрутов считается среднее время доставки и дисперсия времени доставки сообщения.

Тест для тр. 7: 1-ый тип проекта. Процессом send_process отправляется определенное число сообщений, содержащих порядковые номера, процессу rcv_process. Однако в процессе передачи из порта 1 выдергивается провод, тем самым моделируется разрыв соединения. rcv_process фиксирует, какие сообщения не были доставлены, а с помощью логов определяется, какие сообщения и почему были сброшены.

3.3 Анализ результатов тестов

Тесты были реализованы согласно описанию, представленному выше. Для каждого требования были получены следующие результаты:

Тр. 1: Для него не потребовалось тестов, требование выполняется

Тр. 2: Все сообщения с правильными адресами были получены, сообщения с неправильными адресами были сброшены планировщиком из-за некорректности адреса. Требование выполняется.

Тр. 3: Все сообщения, попавшие в очереди, были получены, остальные сброшены планировщиком из-за отсутствия места в очереди. Требование выполняется.

Тр. 4: Все полученные сообщения имеют корректное содержимое. Требование выполняется.

Тр. 5: Все полученные сообщения были доставлены в порядке, соответствующем конфигурации планировщика. Требование выполняется.

Тр. 6: По всем трем маршрутам время доставки пакета близко к постоянному, дисперсия очень мала относительно времени доставки сообщения. Однако время доставки сообщений излишне велико. Был проанализирован код планировщика и установлено, что перед отправкой сообщения он просматривает все слоты, зарезервированные в памяти, тогда как в реальности задействуются только те, которые были описаны в файле конфигурации. Требование выполняется частично.

Тр. 7: При обрыве соединения, согласно логам планировщика, пакеты продолжают отправляться в аппаратный коммутатор и, следовательно пропадают. Требование не выполняется.

Таким образом, можно заключить, что сетевой стек SpaceWire в JetOS для микросхемы 1892ВМ15Ф удовлетворяет не всем требованиям стандарта и требованиям, сформулированным при разработке.

4 Заключение

В процессе написания данной курсовой работы были выполнены все поставленные задачи:

1. Была изучена ОСРВ JetOS и реализация в ней сетевого стека SpaceWire
2. Были сформулированы требования к сетевому стеку
3. Был определен вид и набор тестов для проверки этих требований
4. Тесты были реализованы и запущены
5. Результаты тестов были проанализированы

После выполнения задач были получены следующие результаты:

1. Метод тестирования сетевого стека в ОСРВ, построенной в соответствии со стандартом ARINC 653 и модульным подходом к построению системных разделов
2. Метод был применен к стеку SpaceWire в ОСРВ JetOS, с его помощью были выявлены некоторые несоответствия требованиям, заявленным при его проектировании

За рамками данной курсовой работы остались, однако, тестирование сетевого стека в других сетевых конфигурациях, модульное тестирование отдельных составляющих сетевого стека, тестирование механизма отправки маркеров времени и другие вопросы. Ими, а также применением метода к другим сетевым стекам в JetOS, планируется заняться в дальнейшем.

5 Источники

1. Кулямин В.В. Методы верификации программного обеспечения. – 2008

2. Микросхема интегральная 1892ВМ15Ф - Руководство Пользователя. – 04.08.2021

3. Годунов А.Н., Солдатов В.А., Хоменков И.И. Реализация каналов спецификации ARINC 653 в операционной системе реального времени Багет 3 // Программные продукты и системы. 2017. №3. URL: https://cyberleninka.ru/article/n/realizatsiya-kanalov-spetsifikatsii-arinc-653-v-operatsionnoy-sisteme-realnogo-vremeni-baget-3 (дата обращения: 03.06.2022).

4. Mallachiev K.A., Pakulin N.V., Khoroshilov A.V., Buzdalov D.V. Using modularization in embedded os // Труды ИСП РАН. 2017. №4. URL: https://cyberleninka.ru/article/n/using-modularization-in-embedded-os (дата обращения: 03.06.2022).

5. ECSS-E-ST-50-12C Rev.1: SpaceWire - Links, nodes, routers, and networks. – 15 мая 2019.

6. SpaceWire-D: Deterministic Control and Data Delivery Over SpaceWire Networks. – апрель 2010.