Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

по курсу «Проектирование ОС и компонентов»

по теме «SRS спецификация для TNeo RTOS»

Выполнил студент гр. 13541/2:
Влкова М.Д.

Проверил преподаватель:
Душутина Е. В.

Санкт-Петербург
2019 г.

Содержание

1 Введение

1.1 Назначение

Данная спецификация требований программного обеспечения создана в рамках курса "Проектирование ОС и компонентов"для ознакомления с операционной системой TNeo RTOS, выработкой навыков ведения документации для программного обеспечения, а также ознакомлением со стандартом IEEE 830-1998

Документ иллюстрирует цель и полную спецификацию для разработки продукта. В нем также приведены системные ограничения, интерфейсы и взаимодействие с другими внешними приложениями.

1.2 Область применения

TNeo RTOS - многозадачная операционная система реального времени для встраиваемых систем. Портирована на 9 микропроцессорных архитектур. Распространяется без лицензиии.

TNeo RTOS является операционной системой с открытым исходным кодом.

Операционная система реального времени TNeo RTOS рассчитана на работу в таких, задаваемых особенностями массовых микроконтроллеров, условиях, как низкое быстродействие аппаратуры и малый объём оперативной памяти, отсутствие поддержки на аппаратном уровне таких механизмов операционных систем, как блок управления памятью и механизмы реализации многозадачности, такие, как быстрое переключение контекста.

Ядро системы умещается в несколько файлов.

1.3 Определения, акронимы и сокращения

Определения и термины:

- Обработчик прерываний (ISR) — специальная процедура, вызываемая по прерыванию для выполнения его обработки. Обработчики прерываний могут выполнять множество функций, которые зависят от причины, которая вызвала прерывание.
- Round-robin (циклический) — алгоритм распределения нагрузки распределённой вычислительной системы методом перебора и упорядочения её элементов по круговому циклу.
- Переключение контекста (context switch) — процесс прекращения выполнения процессором одной задачи с сохранением всей необходимой информации и состояния, необходимых для последующего продолжения с прерванного места, и восстановления и загрузки состояния задачи, к выполнению которой переходит процессор.

Акронимы:

- IPC (inter-process communication, межпроцессное взаимодействие) — обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC.
- MMU (memory management unit, блок управления памятью) - компонент аппаратного обеспечения компьютера, отвечающий за управление доступом к памяти, запрашиваемым центральным процессором.
- RTOS (real-time operating system, операционная система реального времени, ОСРВ) — тип операционной системы, основное назначение которой — предоставление необходимого и достаточного набора функций для работы систем реального времени на конкретном аппаратном оборудовании.
- UART (универсальный асинхронный приёмопередатчик, УАПП,Universal Asynchronous Receiver-Transmitter, UART)- узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству.
- SPI (Serial Peripheral Interface) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии.

1.4 Ссылки

[1] https://dmitryfrank.com [электронный ресурс] - информационный портал разработчика (дата обращения 09.04.2019)

[2] https://dmitryfrank.com/projects/tneo [электронный ресурс] - репозиторий с открытым исходным кодом операционной системы (дата обращения 09.04.2019)

[3] https://dfrank.bitbucket.io/tneokernel_api [электронный ресурс] - спецификация проекта (дата обращения 09.04.2019)

2 Общее описание

2.1 Перспектива изделия

Ядро расчитано для 32 и 16 битных микропроцессоров. Выполняет упреждающее планирование на основе приоритетов и циклическое планирование для задач с одинаковым приоритетом. Так что может использоваться для встраевымых систем.

Ядро было представлено на ежегодном семинаре Microchip MASTERS 2014, где команда отдела разработки StarLine заинтересовалась проектом, после чего ядро портировалось под архитектуру Cortex-M. Сейчас ядро используется в их последних продуктах.

Ядро TNeo компактное и бысрое, но на рынке есть альтернатива - FreeRTOS. FreeRTOS также является многозадачной операционной системой реального времени. ОС распространяеться под лицензией MIT с 2017 года. Эта операционная система поддерживает множество архитектур (более 30) и дистрибутивы можно скачать с официального сайта разработчика.

2.2 Архитектура и интерфейсы системы

2.2.1 Ядро

Работа с микроядром осуществляется так же, как и с монолитным ядром — через системные вызовы. Микроядра предоставляют лишь небольшой набор низкоуровневых примитивов, для:

- управления памятью компьютерной системы, как физической так и виртуальной (выделение памяти процессам, обеспечение её изоляции/защиты);
- управления процессорным временем (сервисы для работы с потоками процессов);
- управления доступом к устройствам ввода-вывода;
- осуществления межпроцессной коммуникации и синхронизации (inter process communications, IPC) (управляемое и контролируемое нарушение изоляции памяти процесса для организации обмена данными).

TNeo имеет стандартный для RTOS набор функций и некоторые дополнительные. Большинство возможностей опциональны, так что их можно отключить, тем самым сэкономив память и слегка увеличив производительность.

- Задачи, или потоки: самая основная возможность, для которой ядро вообще было написано;
- Мютексы: объекты для защиты разделяемых ресурсов:
  - Рекурсивные мютексы: опционально, мютексы позволяют вложенную блокировку
  - Определение взаимной блокировки (deadlock): если deadlock происходит, ядро может оповестить вас об этом, вызвав произвольную коллбэк-функцию
- Семафоры: объекты для синхронизации задач
- Блоки памяти фиксированного размера: простой и детерминированный менеджер памяти
- Группы флагов: объекты, содержащие биты событий, которые потоки могут ожидать, устанавливать и сбрасывать;

  - Соединение групп флагов с другими объектами PTOC: очень полезная возможность соединить другие объекты PTOC с какой-либо группой флагов: например, когда потоку нужно ожидать сообщения сразу из нескольких очередей

- Очереди сообщений: FIFO буфер сообщений, которые потоки могут отправлять и принимать
- Таймеры: позволяют попросить ядро вызвать определенную функцию в будущем.
- Отдельный стек для прерываний: такой подход значительно экономит RAM
- Программный контроль переполнения стека: очень полезно, когда нет аппаратного контроля переполнения стека
- Динамический тик: если системе нечего делать, то можно не отвлекаться на периодическую обработку системных тиков
- Профайлер: позволяет узнать, сколько времени каждый из потоков выполнялся, максимальное время выполнения подряд, и другую информацию

2.2.2 Запуск ядра

- Сначало надо выделить массивы для стека незанятых задач и стека прерываний, для этого есть вспомогательный макрос TN_STACK_ARR_DEF (). Можно обратиться к TN_MIN_STACK_SIZE для определения размеров стека.
- Объявляем callback-функцию: void init_task_create (void) ..., в которой должны быть созданы и активированы задачи. Эти задачи должны выполнить инициализацию приложения и создать все остальные задачи.
- Предоставляем функцию обратного вызова в режиме ожидания, которая будет периодически вызываться из задачи в режиме ожидания. Можно оставить его пустым.
- В main ():
  - отключить системные прерывания, вызвав tn_arch_int_dis ();
  - выполнить некоторые важные настройки процессора;
  - установить системное прерывание по таймеру;
  - вызвать tn_sys_start (), предоставив всю необходимую информацию: указатели на стеки, их размеры и ваши функции обратного вызова.
- Ядро действует следующим образом:
  - выполняет всю необходимую деятельность;
  - создает idle задание;
  - вызывает обратный вызов TN_CBUserTaskCreate;
  - выполняет первое переключение контекста (для задачи с наивысшим приоритетом).
- На этом этапе система работает нормально. Первоначальная задача действует следующим образом:
  - инициализация различных встроенных периферийных устройств;
  - инициализация программных модулей, используемые приложением;
  - создание остальных задач.

2.2.3 Задачи

Каждая существущая задача в системе имеет свой собственный дескриптор: struct TN_Task, объявленный в файле src/core/tn_tasks.h. Это довольно большая структура.

Первый элемент дескриптора задачи — указатель на вершину стека задачи: stack_cur_pt. Этот факт активно используется ассемблерными подпрограммами переключения контекста: имея указатель на дескриптор задачи, мы можем просто разадресовать его, и полученное значение будет указывать на вершину стека задачи.

В ядре есть два указателя: на выполняемую в данный момент задачу, и на следующую задачу, которая должна быть запущена.

Задачи имеют разные приоритеты. Максимальное количество доступных приоритетов определяется размерностью процессорного слова: на 16-битных микроконтроллерах у нас есть 16 приоритетов, и на 32-битных — 32.

Для каждого приоритета существует связанный список готовых к запуску задач, которые имеют этот приоритет.

2.2.4 Контекст задачи

Для каждой поддерживаемой ядром архитектуры (MIPS, ARM Cortex-M, и т.д.) есть определенная структура контеста. Когда задача только что создана и готовится к запуску, начало ее стека заполняется «инициализационным» контекстом, так что когда задача, наконец, получает управление, этот инициализационный контекст восстанавливается. Таким образом, каждая задача запускается в изолированном, чистом окружении.

2.2.5 Состояния задач

Задача может находиться в одном из следующих состояний:

- Runnable: задача готова к запуску (это не означает, что задача фактически выполняется в данный момент)
- Wait: задача ждет чего-то (сообщения из очереди, мютекса, семафора, и т.д.)
- Suspended: задача был приостановлена другой задачей
- Wait + Suspended: задача была в состоянии Wait, после чего была приостановлена другой задачей
- Dormant: задача еще не была активирована после создания, или была уничтожена с помощью вызова tn_task_terminate. Следующий раз, когда задача будет активирована, любое предыдущее ее состояние будет обнулено, и она будет запущена в чистом окружении

Когда задача выходит из какого-либо состояния или, наоборот, входит в него, есть определенный набор действий, которые необходимо выполнить.

2.2.6 Переключение контекста

Процедура переключения контекста всегда выполняется в специальном ISR, имеющим низший приоритет. Так что, когда нужно переключить контекст, ядро устанавливает бит соответствующего прерывания. Если в данный момент выполняется пользовательская задача, то ISR, переключающий контекст, вызывается процессором тут же. Если же этот бит установлен из какого-то другого ISR (независимо от приоритета прерывания), то переключение контекста будет запущено позже: когда все выполняемые в данный момент ISR вернут управление.

Когда ISR ядра, переключающий контекст, вызывается, он делает следующее:

- Сохраняет значения всех регистров в стек текущей задачи, один за другим. Кроме прочего, сохраняется и адрес в программной памяти, на котором задача была прервана;
- Когда все регистры сохранены в стек, ядро сохраняет текущий указатель стека в дескриптор текущей задачи;
- Выполняет on-context-switch handler (он нужен для профайлера и для программного контроля переполнения стека, если любая из этих функций включена);
- Переходим к последовательности, указанной в предыдущей секции «Запуск задач».

Переключение контекста происходит, когда какая-либо задача с более высоким приоритетом, чем текущая задача, становится готова к выполнению; или же когда текущая задача переходит в состояние Wait.

2.2.7 Задача Idle

В TNeo есть одна специальная задача: Idle. Она имеет самый низкий приоритет (пользовательские задачи не могут иметь такой низкий приоритет), и она всегда должна быть готова к запуску, так что _tn_ready_to_run_bmp всегда имеет как минимум один установленный бит: бит для низшего приоритета задачи Idle. Ядро передает управление этой задаче тогда, когда нет никаких других готовых к запуску задач.

Приложение может реализовать callback-функцию, которая будет вызываться из задачи Idle постоянно. Это может быть использовано для разных полезных целей:

- MCU sleep. Когда системе нечего делать, часто имеет смысл перевести процессор в sleep, чтобы уменьшить потребление. Конечно, приложение должно настроить какое-то условие, по которому процессор должен выйти из sleep: чаще всего, это какое-нибудь прерывание;
- Вычисление загрузки системы. Простейшая реализация: просто увеличивать на единицу значение какой-нибудь переменной в бесконечном цикле. Чем быстрее изменяется значение — тем меньше нагрузка на систему.

Т.к. задача Idle должна всегда быть готова к запуску, запрещено вызывать из этого callback любые сервисы ядра, которые могут перевести задачу в состояние Wait.

2.2.8 Таймеры

Ядро должно иметь представление о времени. Доступны две схемы: статический тик и динамический тик.

2.2.9 Статический тик

Статический тик — простейший способ реализовать таймауты: нам нужен какой-либо аппаратный таймер, который будет генерировать прерывания с определенной периодичностью. Период этого таймера определяется пользователем. В ISR этого таймера нам только нужно вызвать специальный сервис ядра.

Каждый N-й системный тик, ядро проходит по всем таймерам из «общего» списка, и для каждого таймера производит следующие действия:

- Таймаут уменьшается на N;
- Если результирующий таймаут меньше, чем N, то таймер перемещается в соответствующий tick-список.

А на каждый системный тик, мы проходимся по одному соответствующему tick-списку, и безусловно выполняем callback всех таймеров из этого списка. Это решение более эффективно.

2.2.10 Динамический тик

Ядро должно иметь возможность пообщаться с приложением, для того чтобы избавиться от бесполезных вызовов tn_tick_int_processing():

- Чтобы запланировать следующий вызов tn_tick_int_processing() через N тиков;
- Чтобы спросить, сколько времени.

Когда режим динамический тик активен (TN_DYNAMIC_TICK установлен в 1), приложение должно предоставить указатели на callback при старте системы. Конечно, фактическая реализация callback полностью зависит от типа MCU, поэтому корректная реализация callback лежит на приложении.

2.2.11 Циклическое планирование

TNKernel имеет возможность выполнять циклическое планирование для задач с одинаковым приоритетом. По умолчанию циклическое планирование отключено для всех приоритетов. Чтобы включить циклическое планирование для задач с определенным уровнем приоритета и установить временные интервалы для этих приоритетов, пользователь должен вызвать функцию tnë_sys_tslice_set (). Значение временного интервала одинаково для всех задач с одинаковым приоритетом, но может быть различным для каждого уровня приоритета. Если планирование циклического перебора включено, каждое системное прерывание по времени увеличивает счетчик текущих временных срезов задачи. Когда интервал

временного интервала завершен, задача помещается в конец очереди готовности к запуску ее уровня приоритета (эта очередь содержит задачи в состоянии RUNNABLE), и счетчик временного интервала очищается. Тогда задача может быть прервана задачами с более высоким или равным приоритетом.

В большинстве случаев нет причин включать циклическое планирование.

2.3 Функции изделия

2.3.1 Интерфейсы пользователя

Механизм взаимодействия человека с программой называют интерфейсом пользователя. В это понятие включают внешний вид программы на экране, основные принципы управления и даже конкретные команды. Иными словами, интерфейс пользователя — это способ представления программ.

2.3.2 Многопоточность

В TNeo RTOS несколько потоков могут работать одновременно. Система контролирует переключение между задачами. Каждый процесс может самостоятельно обращаться к устройствам компьютера. Операционная система принимает запросы, выделяет программам запрошенные ресурсы и устраняет возникшие конфликты.

2.3.3 Обработка аварийных ситуаций

И аппаратные, и программные средства не застрахованы от сбоев. Задача операционной системы — свести последствия таких сбоев к минимуму. В операционной системе Fuchsia ошибка отдельной программы не должна влиять на работу других активных программ и на саму операционную систему.

2.3.4 Автоматизация действий

Повторяющиеся, однотипные, рутинные операции можно автоматизировать, например автоматически запускать программы и обслуживать компьютер. Обслуживание — это вспомогательные операции для проверки системы и поддержания ее работоспособности.

2.3.5 Поддержиываемые устройства

В настоящее время он доступен для следующих архитектур:

- Микрочип:
  - PIC32
  - PIC24
  - dsPIC
- ARM Cortex-M ядер:
  - Cortex-M0
  - M0 +
  - M1
  - M3
  - M4
  - M4F