Санкт-Петербургский государственный университет

Кафедра системного программирования

Свирин Евгений Александрович

Портирование Embox на Raspberry Pi

Выпускная квалификационная работа

Научный руководитель:
д. ф.-м. н., профессор Д. В. Луцив

Консультант:
А. В. Бондарев

Рецензент:
Разработчик ООО «Embox» А. И. Калмук

Санкт-Петербург
2021

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information SystemsSoftware Engineering

Svirin Evgeny

Port Embox to Raspberry Pi

Graduation Thesis

Scientific supervisor:
Sr. lecturer, PhD Dmitry Luciv

Scientific advisor:
Anton Bondarev

Reviewer:
Aleksandr Kalmuk

Saint-Petersburg
2021

Оглавление

Введение

Последнее время встраиваемые системы набирают большую популярность, например, потому что обеспечивают автоматизацию промышленного оборудования и бытовых предметов. Актуализируется и соответствующее программное обеспечение, операционные системы.

Разработка программного обеспечения для встраиваемых систем тесно связана со специфичным оборудованием, последнее время растет известность у Raspberry Pi – модель одноплатного компьютера размером с банковскую карту. Основная сфера применения этих компьютеров – обучение информатике. Рост популярности можно частично объяснить доступной ценой при широким диапазоне применения: от WiFi точки доступа до медиа-сервера. В некоторых отраслях Raspberry Pi уже незаменим, например, в роботехнике.

Есть множество операционных систем работающих на представленной платформе, в большинстве своем это Linux-системы. Соответственно список некоторых:

- Raspberry Pi OS (Raspbian),
- Pidora,
- Arch Linux ARM.

Raspberry Pi OS на дистрибутиве debian является главной системой для этой платформы. Linux-системы обладают тем серьёзным недостатком для Raspberry Pi, что сложны, что негативным образом сказывается на основной сфере использования платформы – обучение. Кроме того, они не являются операционными системами реального врмени, что сужает возможные сферы использования этого устройства.

Частично перенесена на платформу операционная система реального времени freeRTOS. Но она не поддерживает набор стандартов POSIX, что не позволяет переиспользовать программное обеспечение, написанное на Linux. Это также недостаток для Raspberry Pi, так как, в основном, ПО, используемое на этой платформе, разработано на указанном семействе систем.

И для переноса была выбрана система Embox, которая как Raspberry Pi используется для обучения, обладающая более простым устройством чем Linux-системы и являющаяся операционной системой реального времени, частично поддерживающей POSIX. Помимо этого система сборки Embox позволяет пользователю широко конфигурировать конечный образ системы, что предоставляет или облегчает такие актуальные возможности для платформы как:

- регулирование нагрузки на систему,
- тестирование устройств,
- сертификация устройств.

1. Постановка задачи

Цель данной работы – добавить Raspberry Pi в список поддерживаемых архитектур ОС РВ Embox. Для реализации этой цели были поставлены следующие задачи.

1. Провести обзор предметной области.
2. Реализовать архитектуро-зависимые части ядра:
    - bootcode,
    - карта памяти,
    - exception table,
    - context switch,
    - vfork,
    - обработчик прерывний.
3. Реализовать драйвера основных устройств:
    - Arm PrimeCell PL190 VIC,
    - BCM2835 Systimer
    - SP804 timer,
    - ARM Generic Timer,
    - ARM PrimeCell PL011 UART,
    - DWC2.
4. Провести апробацию полученного решения на Raspberry Pi.

2. Обзор

2.1. Raspberry Pi

Raspberry Pi[3] – семейство моделей небольших одноплатных компьютеров, которые изначально разрабатывались как учебное пособие по информатике. Несмотря на свои размеры их производительность соизмерима со станционарными ПК. Есть множество интересных проектов по использованию этой платформы, как отдельно, так и вкупе с другим оборудованием, вот список некоторых:

- медиа-сервер,
- метеостанция,
- FM-радиостанция,
- сервер печати,
- Google дом на Raspberry Pi,
- Tor маршрутизатор.

Этот компьютер имеет ARM-архитектуру процессора, отличающуюся низким энергопотреблением. На данный момент разработано уже 11 моделей Raspberry Pi, и модели с индексом 3 и выше могут поддерживать 64-разрядные системы. В данной работе рассматривается перенос на модели Pi 1b и Pi 2b.

2.2. ОС РВ Embox

Embox[1] – это операционная система реального времени для встренных систем. Разрабатывается с 2010 года в сотрудничестве с кафедрой Системного Программирования Математико-Механического факультета СПбГУ, что позволяет студентам приобретать практические навыки

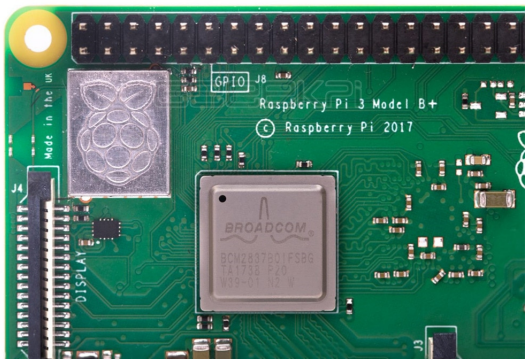

Рис. 1: Raspberry Pi

в различных дисциплинах: архитектура операционных систем, встроенные системы, программирование микроконтроллеров и так далее. Также на базе проекта проводятся различные исследования в областях построения ОС и вычислительных систем. Результаты этих исследований применяются в различных коммерческих проектах в областях: АСУ, встроенных и телекоммуникационных систем.

Embox является модульной конфигурируемой системой. Весь код ядра максимально структурирован, располагается в различных пакетах (библиотеках), реализующих отдельный функционал, как, например: интерфейсы для диспетчера памяти или стратегию планирования.

Специальная система сборки – Mybuild, разработанная в рамках проекта Embox, позволяет выбирать части этой системы для запуска конечного образа, что позволяет подстраивать его под конкретную задачу, эффективно использовать аппаратные ресурсы, соответственно некоторые свойства Embox:

- является системой тестирования,
- облегчает процесс проектирования и отладки аппаратного обеспечения системы,
- облегчает процесс сертификации системы.

Модули этой системы относительно слабо связаны и разделение их на архитектуро-зависимые и нет облегчает разработку переноса на очередную платформу. В Embox уже есть поддержка следующих архитектур:

- SPARC v8 (LEON3),
- ARM,
- RISC-V,
- MicroBlaze,
- x86,
- MIPS,
- PowerPC,
- E2K.

Также система частично поддерживает Posix и программное обеспечение под Linux: SSH, Qt Embedded, Dropbear, PJSIP.

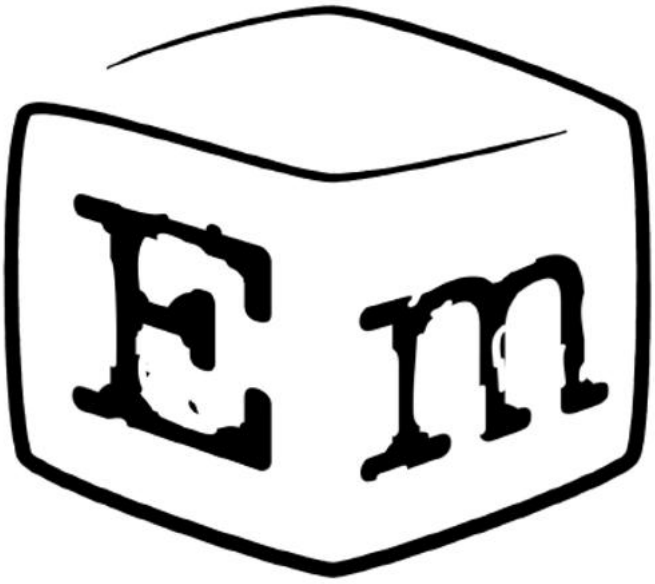

Рис. 2: Логотип Embox

2.3. Архитектура Embox

Для понимания, что необходимо для разработки переноса ОС РВ Embox на очередную платформу необходимо обратиться к её архитек-

туре, на Рисунке 3 она представлена диаграммой компонент UML. Зеленым цветов выделены компоненты, зависящие от архитектуры платформы, они разбиваются на 2 группы: архитектуро-зависимая часть ядра и драйвера устройств. Синим цветом выделены либо интерфейсы работы с архитектуро-зависимыми компонентам, либо части системы с этими интерфейсами работающие, как например, планировщик работает с таймером через его интерфейс. Соответственно, для разработки переноса системы на Raspberry Pi необходимо было реализовать все архитектуро-зависимые компоненты.

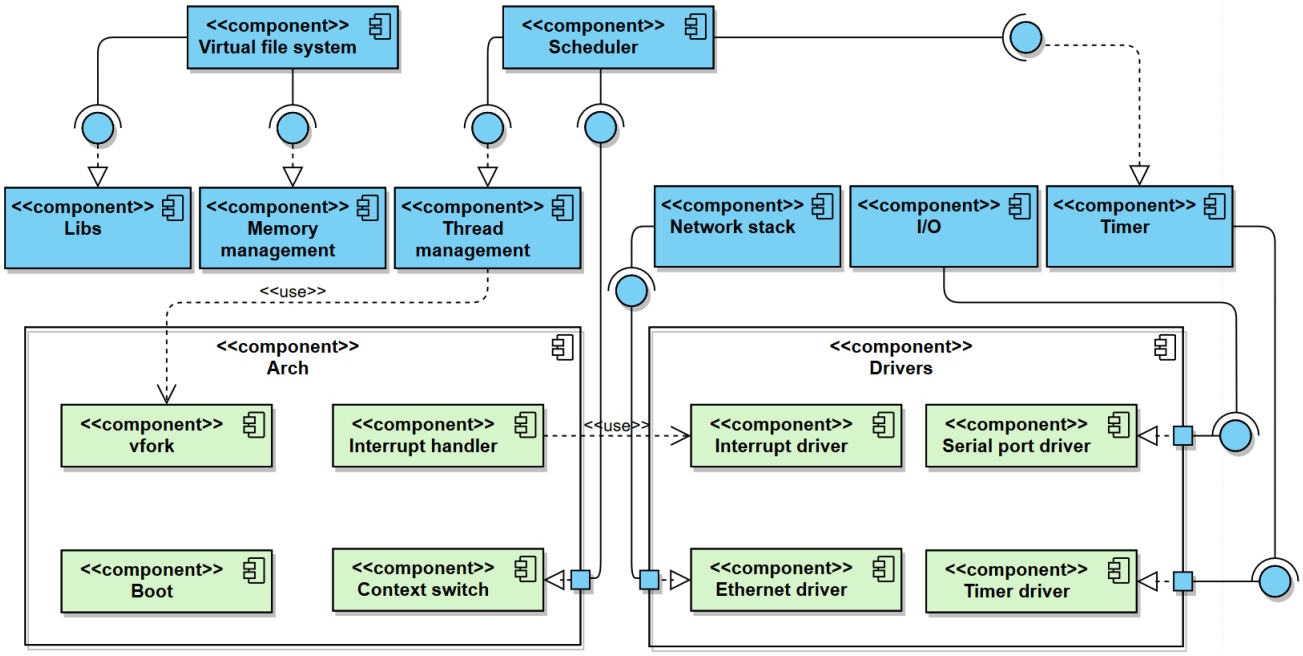

Рис. 3: Архитектура системы Embox

2.4. Метод отладки

Для отладки разработки широко использовалась виртуальная машина QEMU. QEMU – открытый виртуалайзер и эмулятор, предоставляет qemu-console для детализации состояния системы, кроме того, открытость этого проекта повышает прозрачность его кода, что повышает понимание устройства виртуализированного аппаратного обеспечения и даёт самому разработчику возможность его менять, что сильно облегчает отладку.

Рис. 4: Логотип QEMU

3. Архитектуро-зависимая часть ядра

3.1. Архитектура процессора

В Raspberry Pi выбранных моделей используется процессор ARM Cortex-A7. Это процессор поддерживает архитектуру ARMv7A. Свойства этой архитектуры:

- Реализует традиционную ARM архитектуру с несколькими режимами,
- поддерживает архитектуру системы виртуальной памяти (VMSA),
- поддерживает ARM и Thumb инструкции.

Эта архитектура уже поддерживалась Embox, соответственно для реализация большинства компонент архитектуро-зависимой части ядра для Raspberry Pi достаточно было перенастроить и переиспользовать уже реализованные компоненты.

3.2. Карта памяти

Карта памяти определяет положение регионов памяти: ROM, RAM, I/O устройств, расположение сегментов кода и данных. В адресном пространстве ARM нет ROM, соответственно нужно было настроить только RAM под платформу. И разместить в ней скомпилированный машинный код, глобальные и статические переменные, их неизменяемый вариант и глобальные и статические переменные. На Рисунке 5 представлена разметка карты памяти.

```
/*
 * Linkage configuration.
 */

RAM (0x8000, 512M)

/* section (region[, lma_region]) */
text    (RAM)
rodata  (RAM)
data    (RAM)
bss     (RAM)
```

Рис. 5: Разметка карты памяти

В Embox определение I/O устройств возложено на модули для удобства конфигурации системы и возможном их изменении при подключении устройств в некоторых архитектурах.

3.3. bootcode

bootcode, или загрузочный код, – программа запускаемая при старте системы, в том числе она выполняет следующие функции:

- Инициализирует регистры,
- инициализирует режим процессора,
- включает прерывания.

В конце своей работы загрузочный код вызывает функцию инициализации ядра системы.

3.4. exception table

Была переиспользована таблица исключений, она хранит адреса обработчиков исключений. В ней присутствуют адреса следующих обработчиков:

- перезапуска,

- неопределенного поведения,
- преждевременного окончания,
- сброса данных,
- исключение программного обеспечения,
- fiq (быстрое прерывание),
- прерывания.

3.5. Механизм context switch

Многозадачность реализуется через механизм context switch[4]. Этот механизм позволяет системе возобновлять и приостанавливать процессы сохраняя их состояния в момент остановки или перезапуска. Чтобы сохранить состояние процесса используются структуры данных – контексты.

Функция context_init вызывается при создании процесса, она создаёт и инициализирует структуру-контекст процесса. Эта структура используется, чтобы однозначно задавать процесс, поэтому вызываемая функция сохраняет в ней:

- lr (регистр-адрес возврата функции),
- sp (регистр-указатель на стек),
- CPSR (регистр-статус программы),
- все данные с fpu.

Функция context_switch приостанавливает текущий процесс, сохраняя его контекст, и возобновляет работу другого процесса, восстанавливая его ход работы по его сохраненному контексту.

3.6. Функция vfork

Функция vfork создаёт дочерний процесс, который разделяет адресное пространство с родительским. Вызывающий процесс при этом приостанавливается до завершения дочернего. Эта функция используется для управления процессами, получения информации о них. На использование функции накладываются строгие ограничения, поведение функции не определено, если порожденный ею процесс сделает хотя бы одно из следующих действий:

- произведет возврат из функции, в которой был вызван vfork,
- вызовет функцию отличную от exit или exec,
- изменит любые данные кроме переменной возвращаемого vfork значения.

4. Прерывания

Прерывания – сигнал от аппаратного или программного обеспечения, который сообщает процессу о наступлении некоторого события, требующего обработки. При возникновении прерывания текущий процесс необходимо приостановить, сохранив его состояние, вызвать соответствующий обработчик прерываний, после работы которого возобновить работу процесса. Соответственно работу прерываний обеспечивают две программные компоненты – низкоуровневый обработчик прерываний и драйвер контроллера прерываний.

ARM процессор имеет два типа источников прерываний: прерывания, исходящие из GPU перифирии и локальной подконтрольной ARM перифирии. И различает три типа прерываний: специальные прерывания из ARM перефирии, прерывания от GPU и прерывания специальных событий.

Контроллер прерываний это внешнее устройство в архитектуре, он принимает сигнал прерывания от других устройств и позволяет процессору их последовательно обрабатывать. В Raspberry Pi используется контроллер модели Arm PrimeCell PL190 VIC[2]. Его регистры-включения и регистры-выключения разрешают или запрещают прерывания соответствующие номерам бит, а биты регистров ожидания выставляются, когда соответствующая им подсистема возбуждает прерывание.

4.1. Низкоуровневый обработчик прерываний

Низкоуровневый обработчик прерываний при возникновении прерывания выполняет следующую последовательность действий.

1. Приостаналивает текущий процесс.
2. Сохраняет состояние процесса.
3. Вызывает соответствующий обработчик.
4. Возобновляет работу процесса.

Address offset[7]	Name	Notes
0x200	IRQ basic pending	
0x204	IRQ pending 1	
0x208	IRQ pending 2	
0x20C	FIQ control	
0x210	Enable IRQs 1	
0x214	Enable IRQs 2	
0x218	Enable Basic IRQs	
0x21C	Disable IRQs 1	
0x220	Disable IRQs 2	
0x224	Disable Basic IRQs	

Рис. 6: Регистры контроллера прерываний

4.2. Драйвер контроллера

Обработчик прерываний контроллера занимается тем, что считывает из регистров ожидания информацию о прерываниях и, если нужно, сигнализирует о них, вызывая соответствующий обработчик, и обнуляет соответствующий бит в регистре. Полный список функциональности драйвера контроллера:

- инициализирует контроллер,
- настраивает обработчик контроллера,
- разрешать конкретные прерывания,
- запрещать конкретные прерывания.

5. Драйвера остальных устройств

5.1. Таймеры

Для работы системы также необходимо время, расчет которого идет от прерываний-тиков, генерируемых таймерами. В Raspberry Pi используются 3 таймера:

- BCM2835 Systimer,
- SP804 timer,
- ARM Generic Timer.

BCM2835 Systimer – простой таймер с 64-разрядным счетчиком, младшие 32 бита которого сравниваются с одним из 32-разрядных компараторов, в случае совпадения которых вызывается прерывание-тик.

SP804 timer – по сути 2 таймера со схожей логикой.

ARM Generic Timer доступен через сопроцессор CP15, на Raspberry Pi работа с этим таймером реализуется через архитектуру системы виртуальной памяти(VMSA), представляет 4 возможных таймера, основным различием которых являются параметры доступа: небезопасный PL1 физический таймер, безопасный PL1 физический таймер, небезопасный PL2 физический таймер и виртуальный таймер.

```
static inline uint32_t el1_phy_timer_get_ctrl(void) {
        uint32_t val;
        asm volatile("mrc p15, 0, %0, c14, c3, 1" : "=r" (val) : : "cc");
        return val;
}

static inline void el1_phy_timer_set_ctrl(uint32_t val) {
        asm volatile("mcr p15, 0, %0, c14, c3, 1" : : "r" (val));
}
```

Рис. 7: Работа с контрольным регистром ARM Generic Timer через VMSA

5.2. Устройство ввода/вывода

Для системы также необходимо устройство посимвольного вывода. Стандартным устройством, предназначенным для этого является UART. В Raspberry Pi UART совместим с PL011 (ARM PrimeCell PL011 UART).

PL011[5] относительно просто устроен: пересылка в PL011 идет через 32-битный регистр UART_DR. Информация о том пустой этот регистр или нет, то есть, есть ли какие-то непринятые сообщения регулируется через 32-ой, последний, бит UART_FR. Если он 1, то пусто, иначе – нет.

Настройка UART позволяет регулировать частоту пересылки. Полный список функций драйвера:

- отправка символа,
- чтение символа,
- проверка наличия символа,
- настройка устройства.

На Рисунке 8 показан правильный вывод введенных с клавиатуры последовательных чисел.

5.3. Поддержка сети

На плате Raspberry Pi есть интерфейс Ethernet. Но так как на кристалле нет MAC контроллера, то поддержку сети реализуют, используя шину USB, то есть, используя стек технологий Ehternet через USB, представленный на Рисунке 9.

Соответственно, для работы стека Ethernet через USB со стороны платформы сверху вниз: необходимы были реализация сетевого стека, Ethernet-протокола, поддержка устройства класса ECM-CDC. Эти три перечисленные компоненты уже были реализованы в Embox.

Для работы USB компонент необходим USB хост-контроллер и его драйвер. В Raspberry Pi используется хост-контроллер модели DWC2.

```
        unit: initializing embox.driver.clock.arm_el1_phy_timer: done
        test: running embox.test.math.math_test .. done
runlevel: init level is 1
        unit: initializing embox.fs.buffer_cache: done
        unit: initializing embox.driver.flash.flash_nofs: done
        unit: initializing embox.driver.tty.serial: done
        unit: initializing embox.driver.serial.pl011: done
        test: running embox.test.math.fpu_context_consistency_test . done
        test: running embox.test.posix.getopt_test ............. done
        test: running embox.test.posix.environ_test ......... done
        test: running embox.test.stdlib.qsort_test ....... done
        test: running embox.test.stdlib.bsearch_test ...... done
        test: running embox.test.posix.ppty_test ... done
        test: running embox.test.stdio.printf_test ......... done
        test: running embox.test.recursion . done
        test: running embox.test.critical .. done
        unit: initializing embox.fs.rootfs_dvfs: done
runlevel: init level is 2
runlevel: init level is 3
Default IO device[ttyS0]
>export PWD=/
>export HOME=/
>tish
root@embox:/#12345
```

Рис. 8: Ввод с клавиатуры

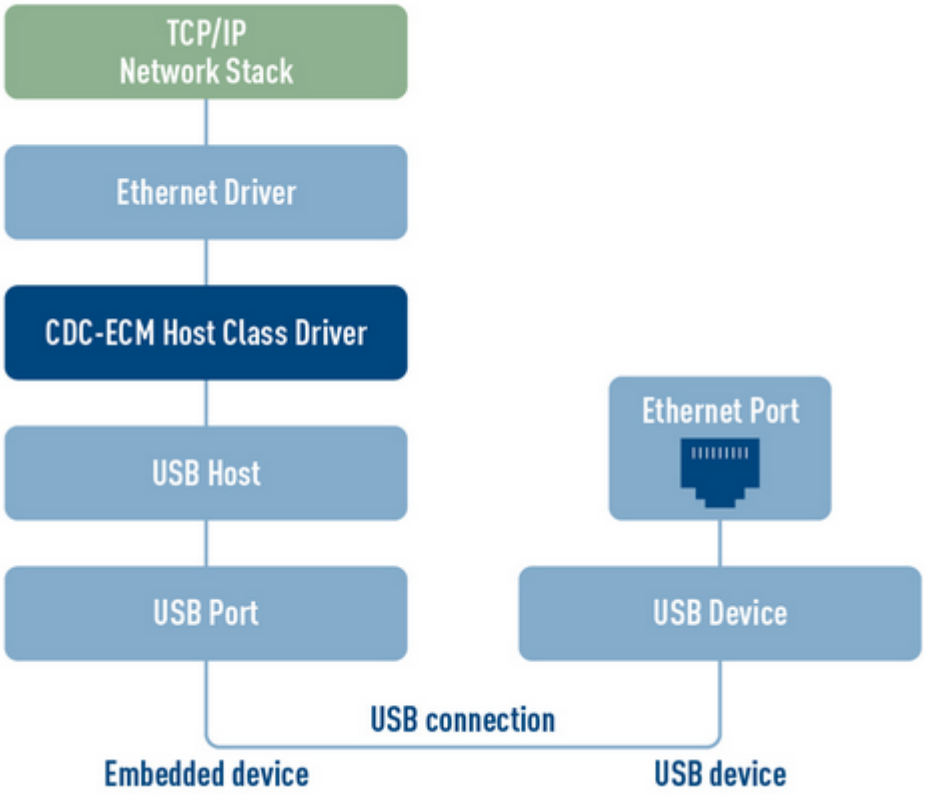

Рис. 9: Ethernet over USB

5.4. USB хост-контроллер

Драйвер хостконтроллера также уже присутствовал в Embox, но его было необходимо привести к актуальному интерфейсу и обновленному USB стеку. Для этого были восстановлены сущности usb_port_status и usb_hub_status, хранящие подробную информацию о портах и хабах соответственно.

Функциональность хост-контроллера весьма обширна, туда входят:

- сигнализация о подключения устройства,
- сигнализация об отключении устройства,
- сигнализация о состоянии устройств,
- работа с питанием,
- разбор синхронных пакетов,
- разбор асинхронных пакетов,
- осуществление процесса пересылки,
- установка каналов.

Большинство из указанных задач является большими составными задачами.

Драйвер на основе получаемой информации выполняет остальную работу, необходимую для работы USB стека:

- настраивает контроллер,
- обслуживает прерывания контроллера,
- обслуживает USB-запросы,
- реагирует на ошибки,
- следит за пересылкой пакетов.

6. Тестирование и апробация

Arm PrimeCell PL190 VIC, BCM2835 Systimem, SP804 timer, ARM Generic Timer, ARM PrimeCell PL011 UART были протестированы как необходимые составляющие тестов других компонент систем. Аналогично и архитектуро-зависимые компоненты ядра. Помимо этого, все таймеры были протестированы командой ticker. Для глубокой отладки использовалась виртуальная машина QEMU, её конфигурации raspi1 и raspi2. На ней DWC2 был успешно протестирован, эмулируя usb-storage.

Была составлена полная конфигурация Embox для Raspberry Pi, которая была запущена на QEMU. После, система была запущена на самой платформе Raspberry Pi и апробирована.

Рис. 10: Работа Embox

Заключение

В результате данной работы ОС РВ Embox была запущена на платформе Raspberry Pi. Были выполнены следующие задачи.

1. Была изучена предметная область:
2. Были реализованы необходимые архитектуро-зависимые компоненты:
   - bootcode,
   - карта памяти,
   - exception table,
   - context switch,
   - vfork,
   - низкоуровневный обработчик прерывний.
3. Были реализованы драйвера основных устройств:
   - Arm PrimeCell PL190 VIC,
   - BCM2835 Systimer
   - SP804 timer,
   - ARM Generic Timer,
   - ARM PrimeCell PL011 UART,
   - DWC2.
4. Проведена апробация на виртуальной машине QEMU и на самой платформе Raspberry Pi.

Список литературы

[1] Embox. Embox. — 2021. — URL: https://github.com/embox/embox (online; accessed: 10.04.2021).

[2] Foundation Raspberry Pi. BCM2835-ARM-Peripherals Reference Manual. — 2021. — URL: https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf (online; accessed: 03.03.2021).

[3] Foundation Raspberry Pi. Raspberry Pi WebSite. — 2021. — URL: https://www.raspberrypi.org/products/ (online; accessed: 03.03.2021).

[4] specification group ARM. ARM Compiler Software Development Guide. — 2021. — URL: https://developer.arm.com/documentation/dui0471/m/handling-processor-exceptions/context-switch (online; accessed: 03.03.2021).

[5] specification group ARM. PrimeCell UART (PL011) Technical Reference Manual. — 2021. — URL: https://developer.arm.com/documentation/ddi0183/latest/ (online; accessed: 03.03.2021).