Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Кижнеров Павел Александрович

Разработка программного обеспечения для контроллера ЙоТик v2.0

Бакалаврская работа

Научный руководитель:
зав. каф. СП, д.ф.-м.н., проф. А. Н. Терехов

Рецензент:
инженер-программист М. М. Киселев

Санкт-Петербург
2020

SAINT PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software engineering chair

Kizhnerov Pavel

Software development for IoTik v2.0 controller

Bachelor's Thesis

Scientific supervisor:
chair head, D.Sc, prof. Andrey Terekhov

Reviewer:
software engineer Kiselev Mikhail

Saint-Petersburg
2020

Оглавление

Введение

Разработка программного обеспечения невозможна без компетентных программистов, способных качественно и своевременно выполнять поставленные задачи, поэтому обучение программированию и информатике является важной и актуальной задачей на сегодня.

Еще в школе люди примерно определяют вектор своего развития, увлекаясь теми или иными предметами. Зачастую информатика и программирование не входят в число таких увлечений, так как выглядят для большинства обучаемых сложными и запутанными. Возможно таким образом мир теряет потенциальных специалистов.

Существует несколько концепций, которые позволяют повысить восприятие этих дисциплин и упростить процесс обучения для школьников. Одна из них – использование графического языка. Предлагается составлять программы путем соединения друг с другом визуальных блоков, обозначающих, например, какое-то действие или логический оператор. Такой подход позволяет снизить порог вхождения до уровня начальной школы, на котором обучаемые еще посредственно владеют раскладкой клавиатуры.

Следующая концепция – плавный переход от графического языка к текстовому. В какой-то момент из-за размеров трудность понимания программы на графическом языке начинает превышать трудность понимания эквивалентной программы на текстовом языке. По этой причине этот переход неизбежен в перспективе, и для того, чтобы он произошел плавно, текстовый язык должен быть простым и указывать на как можно больший спектр ошибок пользователя.

Как правило обучаемый видит только результат выполнения программы – текстовый вывод, при этом процесс получения результата, например отладочной печатью, не всегда можно наглядно представить, поэтому предлагается в качестве исполнителя программы использовать робота. К тому же, большинство школьников испытывают интерес к роботам, что подкрепляет оправданность подобного подхода.

Ключевым пунктом является интуитивно понятная среда обучения,

которая позволит пользователю хранить и использовать инструментарий обучения в одном месте.

Цель и задачи

Таким образом, целью данной работы является упрощение процесса обучения школьников путем реализации программного обеспечения для контроллера IoTik, которое позволяет исполнять программы, разработанные средствами среды разработки TRIK Studio. Для этого были сформулированы следующие задачи:

- спроектировать архитектуру системы;
- интегрировать виртуальную машину RuC;
- реализовать программное окружение для удобной работы с системой: интерфейс командной строки по UART; файловые утилиты; сетевой обмен файлами.

1. Обзор

1.1. Описание всеобъемлющего проекта

Проект РуСи [7] – инструмент для обучения алгоритмической грамотности, язык программирования, разработанный А.Н. Тереховым. Мотивацией его создания послужило практическое отсутствие хороших средств обучения программированию на текстовом языке младших и средних школьников в России. При составлении программ на текстовом языке ключевую роль играют легкость использования и информативность сообщений об ошибках, поэтому именно на этих аспектах и было сконцентрировано внимание. Целевая аудитория еще плохо владеет английским языком, поэтому в РуСи есть возможность использовать ключевые слова на родном языке, а компилятор максимально полно и информативно сообщает о допущенных синтаксических ошибках. Проект состоит из двух частей: компилятора РуСи в коды виртуальной машины и самой виртуальной машины.

TRIK Studio [6] – среда программирования, позиционирующая себя как универсальное программное обеспечение для преподавания основ программирования и информатики. Предоставляет возможность составлять программы как с помощью последовательности пиктограмм, так и с помощью текстовых языков, а также передавать исполняемый файл в память робота. Поддерживает РуСи: реализована генерация кода на РуСи из диаграмм и интегрирован компилятор РуСи в коды виртуальной машины.

Завершающим звеном проекта является робот с интегрированной виртуальной машиной РуСи, благодаря которой появляется возможность исполнять программы на графическом языке и текстовом языке РуСи, составленные в среде разработки TRIK Studio.

Рис. 1: Принцип работы программного продукта

1.2. Контроллер IoTik

Контроллер IoTik – плата на базе микроконтроллера esp32, разработанная и производящаяся российской компанией MGBOT для образовательных целей в сфере робототехники, а также интернета вещей.

В отличие от аналогичных плат, IoTik более устойчив к нагрузкам и способен работать на более высоких показателях напряжения и силы тока. Это позволяет подключать большее количество периферийных устройств, что расширяет спектр возможных конфигураций робота.

Цена за сам контроллер на момент написания данной работы составляет 4 тыс. рублей, а за стартовый набор – 14 тыс. рублей.

Также важным аспектом является тот факт, что производство контроллера производится на территории Российской Федерации, что позволяет юридическим лицам закупать такие наборы существенно проще, чем аналогичные зарубежные.

1.3. ESP-IDF

ESP-IDF [2] – официальный фреймворк разработки под esp32 от Espressif – создателя этого микроконтроллера. Предоставляет API в виде библиотек и исходного кода для взаимодействия с esp32, а также скрипты для работы с toolchain – программное обеспечение для инструментальной операционной системы, необходимое для компиляции под esp32.

Данный фреймворк поставляет компоненты, которые вошли в основу архитектуры программного обеспечения, разрабатываемого в контексте данной работы. Среди них: FreeRTOS – операционная система реального времени для встраиваемых систем; esp_wifi – драйвер беспроводного интерфейса; lwip – TCP/IP стек для встраиваемых систем;

FAT – файловая система; vfs – виртуальная файловая система.

ESP-IDF включает в себя скрипты, предоставляющие удобные функции для языка Cmake, которые позволяют гибко настраивать процесс сборки. Основное: редактирование директорий поиска единиц трансляции, конфигурирование списка включенных в сборку компонентов; регистрация новых компонентов; установка зависимостей между компонентами.

1.4. Аналогичные решения

1.4.1. Контроллеры

Всеобъемлющий проект уже имеет поддержку контроллеров TRIK [8], Lego NTX, Lego EV3, однако ценовая политика данных продуктов не позволяет достичь желаемого уровня доступности. Цена за стартовый набор TRIK составляет около 40 тыс. рублей [9], Lego – около 25 тыс, Lego NXT нет в наличии во многих магазинах ввиду того, что этот контроллер считается устаревшим, однако можно найти цены в районе 25 тыс. рублей. При этом NXT по характеристикам во многом уступает контроллеру IoTik. Вычислительные ресурсы EV3 и TRIK являются исчерпывающими для изучения основ информатики. Практика показала, что IoTik тоже может справляться с поставленными задачами не хуже. Далее приведена сравнительная таблица характеристик контроллеров.

	NXT	EV3	TRIK	IoTik
Цена	от 25000	от 25000	от 40000	от 14000
RAM	64KB	64MB	256MB	520KB
Flash	256KB	16MB	16MB	4MB
Wi-Fi	Нет	Опционально	Да	Да
Bluetooth	Да	Да	Да	Да
USB	Нет	Да	Да	Да
частота CPU	48MHz	300MHz	375MHz	240MHz

Рис. 2: Основные характеристики контроллеров

1.4.2. Решение для IoTik

В 2018 году Фадеевым Виктором и Шитовым Егором было реализовано программное обеспечение, позволяющее программировать робота на базе контроллера IoTik с помощью графического языка TRIK Studio и текстового языка RuC.

Взаимодействие с роботом происходит следующим образом: байт-код программы передается в память контроллера средствами TRIK Studio с помощью UART либо Wi-Fi, через интерфейс командной строки по UART вызывается интерпретатор РуСи. Однако данное решение трудно сопровождать ввиду отсутствия документации и запутанности исходного кода: например, часто встречаются дублирования и неочевидные зависимости, в разных частях проекта используются разные стайлгайды. Также во время разработки не было уделено должного внимания критическим секциям и утечкам памяти, что часто приводит к сбою во время использования данного программного обеспечения.

Было принято решение полностью переписать данную реализацию ввиду трудности сопровождения, не допуская при этом обнаруженных несовершенностей. Также важно учитывать отзывы пользователей, испытывавших прошивку Фадеева Виктора и Шитова Егора. Для комфортной работы необходимо улучшить интерфейс командной строки: добавить возможность автоматически дополнять команды, просматривать историю команд, выводить список всех команд. Более того, оказалось, что есть потребность в сохранении байт-кодов разных программ в файлах и их организации с помощью директорий. По этой причине в новой реализации должна присутствовать файловая система, и соответствующие команды интерфейса командной строки для работы с ней: создание директорий, смена текущей директории, листинг файлов и директорий, а также копирование, перемещение и удаление файлов.

2. Архитектура системы

Система проектировалась с учетом использования средств фреймворка ESP-IDF, который позволяет разрабатывать с помощью языков C и C++. Основным языком был выбран C++, так как он позволяет гибко структурировать код с помощью пространств имен и предоставляет множество удобных высокоуровневых абстракций.

В качестве операционной системы в данном проекте был выбран FreeRTOS ввиду того, что данный компонент предоставляется официальным фреймворком разработки ESP-IDF с 2016 года, следовательно, можно считать его надежным. Более того, ввиду ограниченности времени, решение об использовании готового к эксплуатации программного обеспечения являлось более приемлемым, чем интеграция иных операционных систем.

Одним из важнейших компонентов в данной системе, является виртуальная машина РуСи. Для его успешной интеграции необходима реализация POSIX threads интерфейса, которая не предоставляется операционной системой FreeRTOS.

По этой причине в данную систему интегрирован проект FreeRTOS + POSIX от разработчиков FreeRTOS, реализующий стандарт POSIX для работы с потоками.

Виртуальная машина РуСи во время работы читает данные из файла, поэтому требуется файловая система. ESP-IDF предоставляет FAT и SPIFFS. Было отдано предпочтение файловой системе FAT, так как в ней, в отличие от SPIFFS, можно работать с директориями, что является одним из требований заказчика. Взаимодействие с файловой системой предполагается с помощью стандартных средств языка C++, которые в свою очередь используют vfs – виртуальную файловую систему, предоставляющую единообразный доступ к файловым системам. Поэтому система открыта для расширения в данном направлении и при необходимости есть возможность использовать иную файловую систему с помощью интерфейса, предоставляемого компонентом vfs.

Ключевое требование – иметь возможность управлять системой с

помощью интерфейса командной строки по UART. Для этой цели используется компонент console.

В процессе эксплуатации прототипа стало понятно, что для комфортной работы с виртуальной машиной РуСи нужно иметь возможность просматривать список хранимых в памяти контроллера файлов, переименовывать их, удалять, перемещать, создавать иерархию директорий и так далее. Поэтому было сформулировано требование: добавить утилиты работы с файлами. Был реализован компонент fs_utils, удовлетворяющий это требование.

Исследование прототипа показало, что программное обеспечение контроллера выходило из строя в частности из-за доступа нескольких задач FreeRTOS к общим ресурсам драйвера беспроводного интерфейса. Также к сбою приводил недостаток памяти ввиду отсутствия возможности освободить кучу после работы драйвера Wi-Fi. В связи с этим появилась необходимость в программной обертке над драйвером Wi-Fi с критическими секциями и возможностью завершения работы и освобождения памяти.

По просьбе А.Н. Терехова развитие данного проекта породило другой проект – мини-робофутбол, для которого необходимо реализовать программное обеспечение, позволяющее управлять роботом-тележкой с помощью команд, пересылаемых через беспроводной интерфейс. В это же время в основном проекте требуется реализация передачи файлов с байт-кодом для виртуальной машины РуСи по Wi-Fi. В перспективе планируется реализовать интерфейс командной строки по беспроводной сети. То есть предполагается использование драйвера Wi-Fi из esp_wifi и TCP/IP стека lwip.

Все это удобно реализуется в абстракциях "клиент-сервер". Для экономии трудозатрат в условиях ограниченного времени, а также для предупреждения дублирования кода было принято решение реализовать и использовать в обоих проектах компонент, включающий в себя обертку над драйвером Wi-Fi, реализацию интерфейсов клиента/сервера. Такой компонент был назван network.

На следующей диаграмме представлены основные компоненты и их

зависимости.

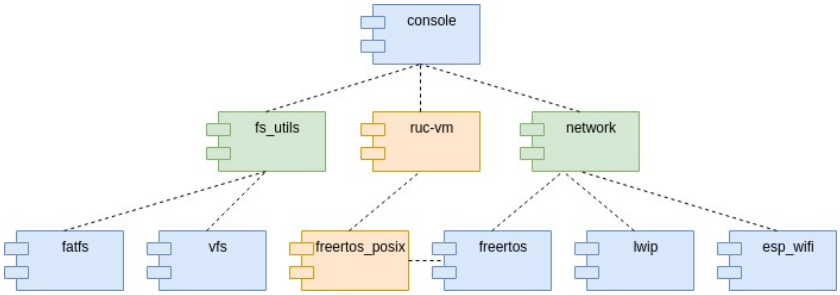

Рис. 3: Диаграмма компонентов

Дерево проекта представлено на следующей картинке. В директории components располагаются все компоненты, которые реализуются самостоятельно: cmd – команды для интерфейса командной строки; external – компоненты, необходимые для работы периферийных по отношению к контроллеру устройств, напримр драйверы двигателей; native – для компонентов, отвечающих за взаимодейтсвие с внутренними устройстами контроллера IoTik.

Директория scripts содержит скрипты, например для быстрой настройки окружения инструментальной операционной системы.

Стороннее программное обеспечение содержится в директории thirdparty. Именно там в располагаются FreeRTOS + POSIX и виртуальная машина РуСи. Важно отметить, что эти проекты являются подмодулями git относительно основного проекта, то есть исходный код загружается в первозданном виде. Для обеспечения совместимости необходимо внести в него правки, для этого во время сборки применяются patch-файлы.

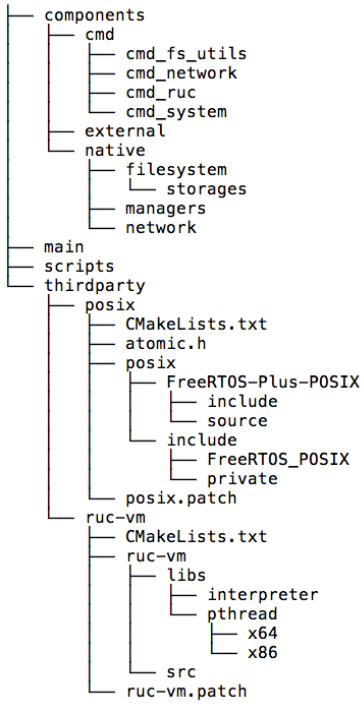

Рис. 4: Дерево проекта

Взаимодействие с системой происходит с помощью интерфейса командной строки по UART и наглядно описывается диаграммой состояний.

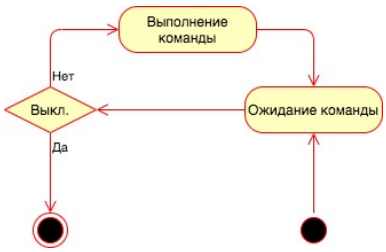

Рис. 5: Диаграмма состояний системы

По требованию заказчика контроллер должен пытаться переподключиться к беспроводной сети при разрыве соединения. Программная обертка над драйвером Wi-Fi, являющаяся частью компонента network, удовлетворяет следующей диаграмме состояний.

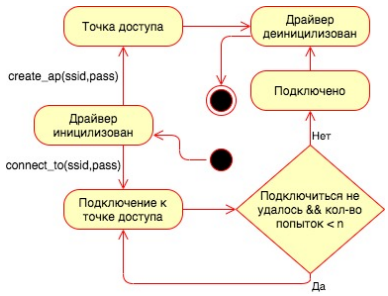

Рис. 6: Диаграмма состояний network::wifi

Таким образом, сформирован конкретный перечень компонентов, которые должны быть интегрированы, реализованы или позаимствованы из ESP-IDF, а также продумана связь между ними.

3. Интеграция виртуальной машины РуСи

Виртуальная машина РуСи разрабатывалась как программное обеспечение, переносимое на различные платформы, реализующие интерфейс POSIX Threads, стандарт для работы с потоками. Используемая в контексте данной работы операционная система FreeRTOS не реализует этот интерфейс – она предоставляет принципиально другие API для организации параллельных вычислений.

Существуют два пути решения данной проблемы: адаптировать виртуальную машину РуСи под API операционной системы FreeRTOS; использовать программную обертку над API FreeRTOS, которая предоставляет POSIX Threads интерфейс. Первый вариант предполагает кардинальные изменения в исходном коде внешнего проекта, что нецелесообразно, так как при необходимости обновления виртуальной машины все действия придется воспроизводить заново. Поэтому второй вариант, использование адаптера с FreeRTOS Tasks API на POSIX Threads, является более предпочтительным, так как такое решение избавляет от необходимости существенно менять код виртуальной машины, а также потенциально уменьшает усилия, затраченные на интеграцию иных проектов, использующих POSIX Threads интерфейс.

Такой адаптер был найден в репозитории FreeRTOS Labs от разработчиков FreeRTOS. Проект называется FreeRTOS + POSIX [4].

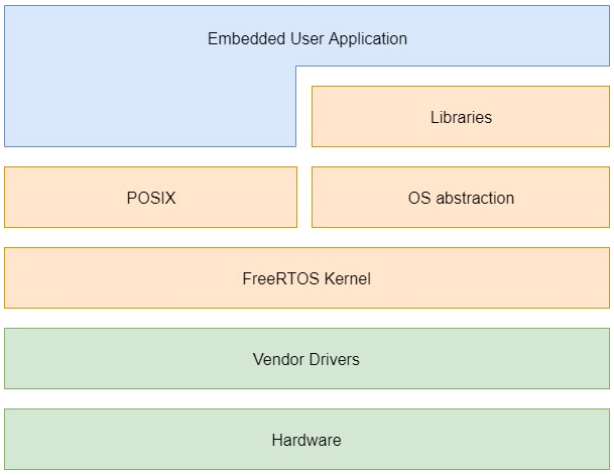

Рис. 7: Расположение FreeRTOS + POSIX относительно FreeRTOS [4]

Интеграция виртуальной машины РуСи началась с интеграции адаптера. Необходимый проект был подключен в качестве подмодуля git в основной репозиторий в директорию thirdparty/posix. Далее были рассмотрены функции и структуры из интерфейса POSIX Threads используемые в виртуальной машине. Как оказалось, некоторые структуры и глобальные переменные определены и в FreeRTOS + POSIX и в компоненте newlib из ESP-IDF: `errno`, `clock_t`, `clockid_t`, `mode_t`, `pthread_attr_t`, `pthread_cond_t`, `pthread_condattr_t`, `pthread_mutex_t`, `pthread_mutexattr_t`, `pthread_t`, `time_t`, `timer_t`, `timespec`, `timer`. Более того, в newlib в заголовочном файле pthread.h определены необходимые для виртуальной машины РуСи функции, однако реализации некоторых нет.

Проанализировав исходный код остальных компонентов из ESP-IDF, было сделано предположение, что эти структуры и функции были добавлены для интеграции внешних по отношению к ESP-IDF проектов. Действительно, например компонент nghttp, реализация HTTP протокола, использует структуру `pthread_attr_t` и функцию `pthread_mutex_lock()`.

Изменение исходного кода ESP-IDF по очевидным причинам не целесообразно – можно нарушить совместимость имеющихся там компонентов, которые в перспективе могут понадобиться в данном проекте. По этой причине было принято решение изымать лишние структуры, переменные и функции из адаптера с помощью patch-файла при сборке проекта.

В одном из заголовочных файлов виртуальной машины РуСи определяются константы MAXREPRTAB, MAXIDENTAB, MAXTREESIZE, MAXMODETAB, MAXBOUNDS, MAXMEMSIZE, которые задают максимальный размер различных таблиц, дерева трансляции и виртуальной памяти, представленных в виде статических массивов. Это означает, что виртуальная машина занимает некоторый фиксированный объем памяти для работы. При запуске системы возникало переполнение памяти, что привело к решению об уменьшении вышеуказанных констант. Эмпирическим путем значения были подобраны, а виртуальная машина успешно интегрирована. Так же как и в FreeRTOS + POSIX интерпретатор РуСи является подмодулем git, и во время сборки к нему

применяется patch-файл, изменяющий необходимые константы.

На данном этапе интегрирована виртуальная машина, однако у пользователя все еще нет возможности пользоваться ее функционалом. Необходимо организовать передачу байт-кода программ на контроллер и их сохранение в виде файлов в энергонезависимой памяти. Тогда можно будет организовать процесс программирования робота следующим образом: пользователь составляет программу средствами TRIK Studio, отправляет байт-код программы контроллеру, контроллер исполняет байт-код сразу после загрузки. Однако при таком сценарии нет возможности загрузить несколько исполнимых файлов и выбрать один конкретный для исполнения, что не является удобным для проверяющего преподавателя. Поэтому необходимо добавить интерфейс для взаимодействия с системой.

4. Реализация программного окружения

4.1. Интерфейс командной строки по UART

Одной из основных задач является предоставление возможности пользователю взаимодействовать с роботом на базе контроллера IoTik после составления программы средствами TRIK Studio. По отзывам клиентов, использовавших прототип с целью преподавания основ информатики, интерфейс командной строки по UART является удачным решением. Поэтому в данном проекте был задействован компонент ESP-IDF console, который позволяет легко и оперативно добавлять команды, подсказки и описания к ним, разбирать аргументы командной строки, а также генерировать справку по всем доступным командам, которая вызывается с помощью команды help. Более того, console поддерживает историю введенных команд и автодополнение по нажатию на tab.

Был сформулирован список команд, необходимых для эффективного взаимодействия пользователя с контроллером:

Команда	Описание
`help`	вывести список всех команд
`ruc <path>`	исполнить байт-код
`connect_to <ssid> [<pass>]`	подключиться к точке доступа
`create_ap <ssid> [<pass>]`	создать точку доступа
`wifi_download [<path>]`	загрузить байт-код по Wi-Fi
`uart_download [<path>]`	загрузить байт-код по UART
`mkdir <path>`	создать директорию
`cd <path>`	сменить директорию
`ls [<path>]`	вывести список файлов/директорий
`cat <path>`	показать содержимое файла
`rm <path>`	удалить файл/ директорию
`cp <source_path> <dest_path>`	скопировать файл
`mv <source_path> <dest_path>`	переместить файл
`rename <source_path> <dest_path>`	переименовать файл/директорию

Таблица 1: Основные команды

Также для удобства ориентации в иерархии директорий было решено добавить вывод текущей директории в приветствии командной

строки.

В структуре проекта обработчики команд для интерфейса командной строки располагается в поддиректориях директории components/cmd в соответствии с их предназначением. Команды были разделены на файловые, сетевые, связанные с виртуальной машиной РуСи и системные.

Для добавления команды необходимо реализовать обработчик и сконфигурировать разборщик аргументов командной строки из библиотеки argtable3 [1].

Сначала определяется структуру, описывающая принимаемые командой аргументы. Далее приведен пример.

```
struct
{
struct arg_str *first_arg;
struct arg_int *second_arg;
struct arg_end *end;
} example_args;
```

Структуры `arg_str`, `arg_int` и `arg_end` предоставляются библиотекой `argtable3` и нужны для описания строкового, числового аргумента и символа конца строки соответственно.

Далее необходимо реализовать сам обработчик команды с сигнатурой `int command_handler(int argc, char **argv)`, в его теле произвести разбор аргументов и выполнить непосредственно обработку.

```
int command_handler(int argc, char **argv)
{
int nerrors = arg_parse(argc, argv, (void **) &command_args);
if (nerrors != 0)
{
arg_print_errors(stderr, cat_args.end, argv[0]);
return 1;
}

// Обработка

return 0;
}
```

После этого конфигурируется вывод аргументов в описании команд при вызове `help` и регистрируется сама команда. Следующий исходный код описывает этот процесс наглядно.

```
void register_command()
{
comamnd_args.first_arg = arg_str1(NULL, NULL, ''<string_arg>'', ''String arg'');
command_args.second_arg = arg_int1(NULL, NULL, ''<int_arg>'', ''Int arg'')
command_args.end = arg_end(3);

const esp_console_cmd_t example_cmd = {
.command = ''example_command'',
.help = ''Example command'',
.hint = NULL,
.func = &comamnd_handler,
.argtable = &command_args
};

ESP_ERROR_CHECK(esp_console_cmd_register(&example_cmd));
}
```

Все описанные ранее команды были добавлены аналогичным образом.

Для дальнейшего использования интерфейса командной строки по UART необходимо настроить системное программное окружение контроллера: сбросить поток вывода, отключить буферизацию потока ввода, сконфигурировать UART, сконфигурировать интерфейс командной строки, настроить библиотеку linenoise для автодополнения и истории команд. Фрагмент кода, в общих чертах описывающий эти действия, представлен далее.

```
fflush(stdout);
fsync(fileno(stdout));
setvbuf(stdin, NULL, _IONBF, 0);
const uart_config_t uart_config = { };
uart_param_config(UART_NUM, &uart_config));
uart_driver_install(UART_NUM, ...);
esp_vfs_dev_uart_use_driver(UART_NUM);
esp_console_config_t console_config = { ... };
esp_console_init(&console_config);
linenoiseSetCompletionCallback(&esp_console_get_completion);
linenoiseHistoryLoad(DEFAULT_HISTORY_PATH);
```

Остается только в бесконечном цикле считывать строку и передавать ее в обработчик.

```
while(true)
{
char* line = linenoise();
if (line == NULL)
{
continue;
}
linenoiseHistoryAdd(line);
linenoiseHistorySave(DEFAULT_HISTORY_PATH);
int ret;
esp_console_run(line, &ret);
linenoiseFree(line);
}
```

Далее необходимо реализовать функции, которые будут использоваться обработчиками команд.

4.2. Файловые утилиты

Файловые утилиты предполагается использовать в обработчиках команд `mkdir`, `cd`, `ls`, `cat`, `rm`, `cp`, `mv`, `rename`, которые нецелесообразно использовать без файловой системы. ESP-IDF предоставляет готовые к использованию компоненты, реализующие файловые системы FAT [3] и SPIFFS [5]. Основное их отличие заключается в том, что первая файловая система имеет поддержку работы с директориями, а вторая – нет. А преимущество SPIFFS в виде более высокой скорости чтения данных является сомнительным в рамках данной работы. По этой причине в системе по умолчанию используется FAT.

Прежде чем монтировать файловую систему, необходимо создать для этого раздел на внутреннем flash-накопителе. Для этого в файле `sdkconfig.defaults`, находящимся в корне проекта, нужно добавить следующие строки.

```
CONFIG_PARTITION_TABLE_CUSTOM=y
CONFIG_PARTITION_TABLE_CUSTOM_FILENAME=''partitions_example.csv''
CONFIG_PARTITION_TABLE_FILENAME=''partitions_example.csv''
```

В данном примере файл `partitions_example.csv` содержит разметку разделов внутреннего flash-накопителя. Его содержимое выглядит следующим образом.

```
Name, Type, SubType, Offset, Size, Flags
nvs, data, nvs, 0x9000, 0x6000,
phy_init, data, phy, 0xf000, 0x1000,
factory, app, factory, 0x10000, 1M,
fat_storage, data, fat, , 1M,
```

После создания раздела необходимо сконфигурировать и монтировать файловую систему. Данный пример в общих чертах описывает этот процесс.

```
const esp_vfs_fat_mount_config_t mount_config =
{
cur_format_if_mount_failed,
cur_max_files,
cur_allocation_unit_size
};

esp_vfs_fat_spiflash_mount(PATH, LABEL, &mount_config);
```

Далее можно использовать стандартные функции C++ для работы с файлами и директориями, с помощью которых и были реализованы файловые утилиты.

Анализ исходного кода компонентов FAT и VFS показал, что отсутствует поддержка относительных путей и текущей рабочей директории, что вынудило бы пользователя вручную прописывать пути до нужных файлов и директорий. Такой способ взаимодействия с файловой системой не является удобным, поэтому недостающий функционал был добавлен. Все вышеописанное было реализовано в компоненте fs_utils, интерфейс которого продемонстрирован далее.

```
bool list(std::string path);
bool remove(std::string path);
bool move(std::string source_path, std::string dest_path);
bool cat(std::string path);
bool copy(std::string source_path, std::string dest_path);
bool make_directory(std::string path);
bool is_correct(const std::string path);
std::string to_absolute_path(std::string path);
std::string get_cwd();
bool change_directory(std::string path);
```

Отдельно стоит рассмотреть последние четыре функции, с помощью которых была реализована поддержка относительных путей и теку-

щей директории: `is_correct` принимает на вход путь в виде строки и возвращает true, если он не содержит недопустимых символов и/или последовательности символов; `to_absolute_path` принимает на вход относительный путь в виде строки и возвращает абсолютный путь, например строка `/FLASH/A/..` преобразуется в `/FLASH/A`; `get_cwd()` возвращает значение глобальной переменной, хранящей текущую рабочую директорию; `change_directory` устанавливает текущую рабочую директорию, если она существует.

На данном этапе реализованы все необходимые для обработчиков команд функции, которые позволяют удобно работать с файлами. Осталось реализовать получение файлов.

4.3. Сетевой обмен файлами

Для исполнения байт-кода виртуальной машиной необходимо загружать его в память. В прототипе существует наивная реализация загрузки файлов по UART, но, как показала практика, для передачи файлов в память контроллера чаще использовался беспроводной интерфейс ввиду возможности централизованной раздачи данных, поэтому его добавление в разрабатываемую систему обоснованно.

Реализация загрузки файлов по беспроводному интерфейсу началась с разработки безопасной программной обертки над драйвером Wi-Fi. Ее необходимость обосновывается тем, что подключение к точке доступа может инициироваться из разных мест системы, и отсутствие функций, выполняющих конфигурацию беспроводного драйвера, привело бы к множественному копированию кода. Более того, интерфейсные функции драйвера не имеют критических секций, поэтому их вызов в параллельных задачах приводит к критическим ошибкам. Язык РуСи позволяет описывать параллельные вычисления и инициировать подключение к беспроводной сети, поэтому вероятен исход параллельного вызова функций драйвера беспроводного интерфейса посредством виртуальной машины РуСи. Программная обертка должна избавлять

от подобных потенциальных проблем. Ее интерфейс представлен далее.

```
bool create_ap(const std::string ssid, const std::string password);
bool connect_to(const std::string ssid, const std::string password);
bool stop();
bool deinit();
```

Проблема с одновременным доступом была решена путем добавления критических секций в тела функций обертки с помощью семафоров FreeRTOS.

На данном этапе можно было бы уже реализовать загрузку файлов с помощью TCP/IP стека lwip. Однако, например реализация получения команд для робота-футболиста по беспроводной сети в некоторых участках повторяла бы код загрузки файлов: создание сокета, подключение к удаленному сокету или принятие входящих соединений, закрытие сокета после работы. Любая другая сущность, описываемая терминами "клиент" или "сервер" повторяла бы эти участки кода, поэтому было принято решение реализовать интерфейсы клиента и сервера с возможностью выбора транспортного протокола для предупреждения дублирования кода. Функции в обоих случаях получились одинаковые.

```
bool start(uint16_t port, int32_t type, user_function function);
bool stop(uint16_t port);
bool stop_all();
```

Для взаимодействия с сокетом необходимо знать только дескриптор соединения, поэтому предлагается вынести требуемую логику работы в функцию с сигнатурой `void func(int32_t descriptor)` и передавать ее в качестве параметра функции `start` из интерфейса клиента или сервера.

Передача файлов по беспроводному интерфейсу была реализована следующим образом: на стороне компьютера запускается сервер для передачи данных на желаемом порту; контроллер подключается к серверу в качестве клиента и, отсылая символ $ выражает готовность к получению данных; сервер отсылает имя и размер файла и ожидает готовности от клиента; клиент создает файл с необходимым именем и снова выражает готовность; сервер передает данные, пока не встретит символ конца файла; клиент записывает получаемые данные в поток файлового ввода и завершает соединение после получение всех данных.

Заключение

В конечном итоге получилось программное обеспечение, в общих чертах повторяющее функциональность решения Фадеева Виктора и Шитова Егора. Однако, так как ошибки в коде прототипа, приводящие к сбоям, были внимательно изучены, их удалось избежать в новой версии. Особое внимание было уделено сопровождаемости проекта: была добавлена документация, весь код удовлетворяет единому стайлгайду, потенциальное обновление сторонних проектов, например виртуальной машины РуСи, потребует минимальных усилий. Также были учтены просьбы и пожелания пользователей старой прошивки, поэтому был усовершенствован интерфейс командной строки.

В дальнейшем планируется расширить возможности взаимодействия с роботом путем добавления интерфейса командной строки по Wi-Fi, добавить возможность использовать arduino-совместимые устройства с помощью реализации программного адаптера для их драйверов.

Предполагается, что получившееся программное обеспечение найдет применение в образовательных организациях для обучаемых школьного возраста, где преподается информатика и программирование. Также возможно использование и частными лицами, решившими заняться самообучением.

Таким образом, в ходе данной работы были достигнуты следующие результаты:

- спроектирована архитектура системы;
- интегрирована виртуальная машина РуСи;
- реализовано программное окружение для эффективной работы с контроллером: интерфейс командной строки, файловые утилиты, файловый обмен по беспроводной сети.

Список литературы

[1] A Cross-Platform, Single-File, ANSI C Command-Line Parsing Library // Argtable.org. — 2020. — URL: https://www.argtable.org/ (online; accessed: 5.05.2020).

[2] ESP-IDF Programming Guide // Espressif. — 2020. — URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/ (online; accessed: 20.01.2020).

[3] FAT Filesystem Support // Espressif. — 2020. — URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/fatfs.html (online; accessed: 3.03.2020).

[4] FreeRTOS + POSIX // FreeRTOS. — 2020. — URL: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_POSIX/index.html (online; accessed: 4.05.2020).

[5] SPIFFS Filesystem // Espressif. — 2020. — URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/spiffs.html (online; accessed: 3.03.2020).

[6] TRIK Studio // TRIK Studio. — 2020. — URL: https://trikset.com/products/trik-studio (дата обращения: 6.02.2020).

[7] А.Н.Терехов М.А.Терехов. Проект РуСи для обучения и создания высоконадежных систем // CYBERLENINKA. — 2017. — URL: https://cyberleninka.ru/article/n/proekt-rusi-dlya-obucheniya-i-sozdaniya-vysokonadezhnyh-programm (дата обращения: 6.02.2020).

[8] Контроллер ТРИК // TRIK. — 2020. — URL: https://trikset.com/products/trik-controller (дата обращения: 21.05.2020).

[9] Наборы ТРИК // TRIK. — 2020. — URL: https://trikset.com/prices (дата обращения: 21.05.2020).