САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Бурдун Федор Викторович

Разработка инструментария уровня файловой системы для унифицированного управления периферийными устройствами контроллера ТРИК

Дипломная работа

Допущен к защите.
Зав. кафедрой:
д.ф.-м.н., проф. А.Н. Терехов

Научный руководитель:
ст. преп. Я.А. Кириленко

Рецензент:
вед. инж. Д.А. Дыдычкин

Санкт-Петербург

2014

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics and Mechanics Faculty

Software Engineering Chair

Fedor Burdun

Development of file system level framework for unified control of TRIK board peripheral devices

Graduation Thesis

Admitted for defence.
Head of the chair:
professor A.N. Terekhov

Scientific supervisor:
senior lecturer I.A. Kirilenko

Reviewer:
lead software engineer D.A. Dydychkin

Saint-Petersburg

2014

Оглавление

Введение **4**

1 Проблемы при разработке и проектировании моделей роботов **6**

1.1 Унификация периферии 6

1.2 Зависимость модельного приложения от выбранной периферии 7

1.3 Повторение кода (sensor fusion) 7

1.4 Сбор дополнительных данных для телеметрии/отладки 8

2 Постановка задачи **9**

3 Технологические решения **10**

3.1 Предоставление файлового интерфейса 10

3.1.1 Unix pipe 10

3.1.2 Linux VFS 10

3.1.3 FUSE 11

3.2 Прототипирование кода 12

3.2.1 Python 13

3.2.2 Guile 13

3.2.3 Lua 13

4 Архитектура, особенности реализации **15**

4.1 Место в стеке ПО 15

4.2 Внутреннее устройство 16

4.2.1 main 16

4.2.2 control 16

4.2.3 worker 17

4.2.4 fuse 17
4.3 /trikfs/config 18
4.4 prelude.lua 22
4.4.1 byte_array 22
4.4.2 id_create 22

5 Примеры использования **24**

6 Апробация **27**
6.1 Демон телеметрии 27

Заключение **28**

Введение

Ни для кого не секрет, что робототехника в последнее время претерпевает бурное развитие, давно не кажется удивительным ее применение в производстве, от пищевых продуктов, до тяжелой техники. Некоторые компании, такие как, например Tesla Motors, уже на данный момент добились почти полной автоматизации производства.

Повсеместное внедрение роботизированных технологий приходит не только в промышленность, но и в повседневную жизнь. Так появляется все больше интеллектуальных устройств, доступных массовому потребителю. Целый класс их занимает бытовая техника, роботы-пылесосы, которые являются его представителями, давно продаются, через интернет магазины пользуясь немалым спросом.

Многие эксперты индустрии не только прогнозируют рост рынка интеллектуальных и роботизированных систем, но и называют их next big thing [1].

На фоне интереса крупных компаний к данной отрасли наблюдается также является рост интереса к платформам прототипирования и моделирования роботизированных устройств среди любителей. Частично в эту нишу попадает контроллер Arduino[1], который породил довольно большое, активное сообщество.

Отдельной задачей стоит взращивание и подготовка качественных специалистов. В связи с чем становится ясно, что необходимо внедрять такие платформы в образовательный процесс. К сожалению Arduino, при всей своей популярности является довольно слабым контроллером, помимо этого обладает относительно сложной средой для программирования логики, зачастую недоступной для школьников, поскольку в ней приходится писать на языке

[1] `http://www.arduino.cc`

Си. К тому же при работе с ним возникают сложности с созданием самой физической модели (конструирование корпуса, каркаса отнимает время), а также с выбором и использованием внешних устройств: моторов, датчиков.

Закономерно, что возникла платформа LEGO Mindstorms[2], которая решает проблему построения физической модели, так как строится на основе известного конструктора LEGO, а также поставляется с удобными средами, доступными как студентам, так и самым младшим школьникам. В частности, на нашем факультете, некоторые задачи теории управления решаются и апробируются на данной платформе.

К сожалению, при всех своих достоинствах платформы, контроллер LEGO Mindstorm довольно слабый в смысле производительности, а прочности пластикового конструктора хватает не для всех задач.

С целью решения данных проблем, а также в надежде предоставить школьникам и студентам более мощную базу, возник ТРИК[3].

Рис. 1: ТРИК и примеры периферии

[2] http://mindstorms.lego.com

[3] http://blog.trikset.com/

1 Проблемы при разработке и проектировании моделей роботов

Не смотря на все достоинства контроллера ТРИК, при разработке новых моделей все еще остаются некоторые узкие места.

1.1 Унификация периферии

В отличие от брендовых тулкитов, таких как LEGO Mindstorm, которые буквально ограничены периферией, имеющейся в наборе (или китайскими клонами, дотошно повторяющими все параметры и интерфейсы), ТРИК более свободен в выборе периферийных устройств.

Но большой выбор периферии кроме того означает большое количество шин и протоколов взаимодействия с ними.

Так общение с некоторыми устройствами происходит по i2c, с другими по usb, третьи являются аналоговыми. Помимо этого, например, различные датчики освещенности (и расстояния) возвращают данные в разных величинах, а иногда обратно пропорциональное значение. Различные сервоприводы в свою очередь часто имеют отличающиеся частоты ШИМ.

В связи с данными обстоятельствами напрашивается задача унификации.

С другой стороны у Arduino вопрос унификации не поднимается ввиду скромных вычислительных мощностей контроллера, а также отсутствия предусмотренной поддержки серьезной периферии (чаще всего пользователю самому приходится писать драйвера для внешних устройств).

У LEGO эта проблема отпадает, поскольку их периферия ограничена конкретной моделью в своем классе (двигатель, датчик света и т.п.) от одного производителя.

1.2 Зависимость модельного приложения от выбранной периферии

Ввиду рассмотренных причин и в отсутствие средств унификации устройств, приложение задающее логику и поведение робота становится зависимым от выбранных конкретных устройств взаимодействия с внешним миром.

Данный факт выливается в множество накладных расходов при обновлении элементной базы проекта, а также сложностями в демонстрации обучающих моделей, т.к. в каждом отдельном случае их приходится переделывать и дорабатывать.

1.3 Повторение кода (sensor fusion)

Почти всегда разработка более-менее серьезной роботизированной модели начинается с реализации некоторых модулей-оберток, по сути комбинаторов, данных приходящих от датчиков робота, нужных для того, чтобы например считать направление перемещения по информации получаемой от акселерометра и гироскопа, или же модули совмещающие энкодер и мотор, позволяющие удерживать постоянные обороты вне зависимости от напряжения батареи и многие другие.

Задачи эти решаются снова и снова отчасти от отсутствия унификации периферии, отчасти от отсутствия четкого понимания на каком уровне конечного программно-аппаратного решения должен находиться их код.

1.4 Сбор дополнительных данных для телеметрии/отладки

Кроме того ввиду отсутствия общего метода доступа и получения данных о состоянии роботизированного устройства нетривиальной задачей является получение дополнительных данных для отладки и телеметрии. Хотя унифицированный механизм журналирования потока данных от периферийных устройств вероятно помог бы решить эти задачи.

2 Постановка задачи

Таким образом ввиду рассмотренных типичных проблем в разработке роботизированных устройств была поставлена цель работы: создать программного решения для унификации доступа к периферии, телеметрии, переиспользования фильтров из приложений реализованных на различных языках программирования.

В ходе работы сформулированы следующие задачи:

1. Выбор подхода и технологий для решения задачи
2. Разработка прототипа поддерживающего динамическое конфигурирование и написание пользовательских фильтров
3. Апробация на модельных приложениях контроллера ТРИК

3 Технологические решения

3.1 Предоставление файлового интерфейса

Поскольку одно из ключевых требований – обеспечение унификации для максимального количества языков программирования, принято решение строить унификацию посредством файлового взаимодействия (т.к. подавляющие большинство языков программирования имеют встроенные библиотеки для работы с ними), а также ввиду того, что данный подход нас освобождает от необходимости реализовывать библиотеки взаимодействия с приложением контроллера ТРИК для каждого нового языка в отдельности.

3.1.1 Unix pipe

Первым и очевидным решением является использование встроенных каналов (unix pipe) ОС Linux, т.е. построение цепочек следующего вида:

```
]$ cat /sys/.../device1 | ./filter1 | cat > /trikfs/outdev1
```

Где outdev1 был бы pipe соединенный с файловым узлом ОС. (сделан используя mkfifo)

Данный вариант был признан нецелесообразным ввиду избыточной нагрузки на ядро ОС, которая вызвана тем, что каждый фильтр по своей сути отдельный процесс ОС.

3.1.2 Linux VFS

Следующим способом реализации файлового интерфейса является Linux VFS [4] [5]. Использование данного подхода теоретически позволяет добиться

[4]Linux VFS – построение файловой системы на уровне ядра ОС Linux

максимальной производительности, но в свою очередь накладывает обязанности выдерживать повышенное качество кода.

Так как код написанный данным способом будет выполняться в kernel space, критически важна стабильность и производительность.

К сожалению поскольку одним из требований была возможность добавлять пользовательские фильтры-преобразователи, мы не можем заранее отвечать за качество их программного кода, а следовательно они могут в любой момент замедлить или сломать ядро ОС.

Что накладно даже для тестовых моделей, так как тормозит в том числе процесс разработки (необходимо дождаться перезагрузки системы).

Хоть и есть инструменты реализовать всю задуманную идею рассматриваемым способом, а именно работать с дескрипторами других устройств (чтение/запись) в рамках kernel module [4], этого всячески не рекомендуется делать.

3.1.3 FUSE

Следующим рассмотренным и выбранным в конечном итоге вариантом был FUSE[5][6].

FUSE реализован в виде модуля ядра ОС Linux. Позволяет отделить код реализации файловой системы от кода ядра и перенести его реализацию в user space.

Данный метод проигрывает предыдущему ввиду наличия дополнительных расходов на пробрасывание вызовов и данных в пространство пользователя, но с другой стороны позволяет обезопасить ядро ОС от потенциально небезопасного пользовательского кода, а также ускорить процесс разработки ФС.

[5]FUSE – фреймворк для реализации файловых систем в user space

[6]`http://fuse.sourceforge.net/`

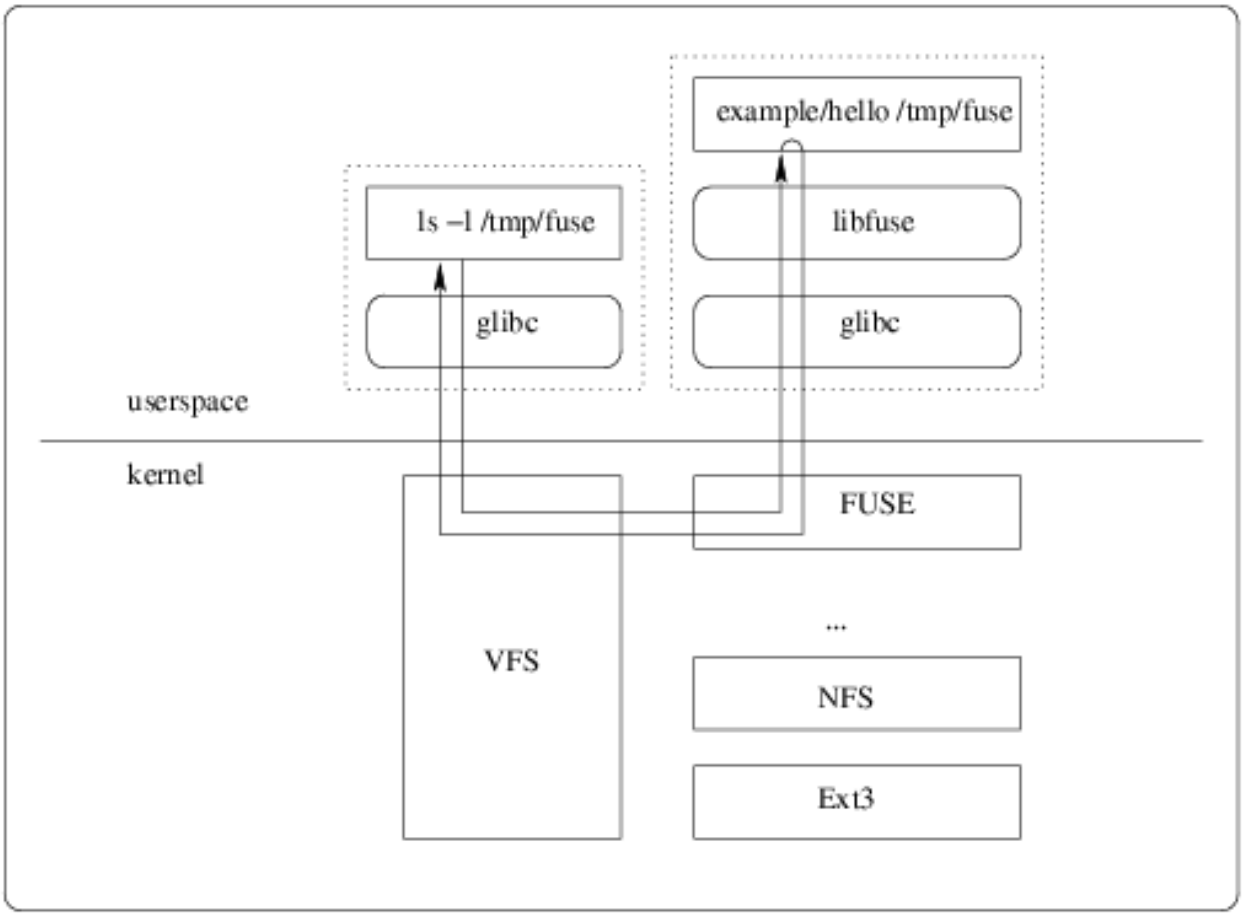

Рис. 2: устройство FUSE

3.2 Прототипирование кода

Для прототипирования кода фильтров заданных пользователем было решено выбрать скриптовый язык поддерживающий JIT компиляцию [7] Кроме того косвенно повлиял на выбор синтаксис языка и желание выполнять конфигурирование системы в целом на его подмножестве.

Далее рассмотрены наиболее вероятные кандидаты на эту роль.

[7]JIT компиляция – just in time (компиляция времени выполнения)

3.2.1 Python

Python[8] довольно популярный язык, который имеет множество реализаций, библиотек и обширное сообщество. Существует несколько реализаций поддерживающих JIT (PyPy, Pyston). Но к сожалению задача интеграции в приложение отнюдь не тривиальна [2], и помимо того, образ его VM[9] занимает по меньшей мере несколько мегабайт в оперативной памяти.

3.2.2 Guile

Еще один из вариантов – Guile[10]. Существует реализация guile-jit (последние изменения 2012). Образ в памяти тоже приближается к мегабайту. Несмотря на множество лестных отзывов, как о языке для расширения существующих приложений (в частности эта статья [6] стала причиной рассмотреть его как альтернативу в данной работе) было решено от него отказаться (отчасти из-за неоднозначной реакции на синтаксис lisp подобных языков большинством программистов, которые все-же являются целевой аудиторией нашего приложения)

3.2.3 Lua

В конечном итоге был выбран язык Lua[11] имеющий актуальную имплементацию поддерживающую JIT (LuaJIT[12]). Что замечательно, так то, что экземпляр VM этого языка помещается всего в 15 Kb. Язык обладает очень простым, но потенциально расширяемым синтаксисом, понятной документацией и очевидным способом интеграции с host-приложениями.

[8] `http://python.org`

[9] VM – virtual machine

[10] `http://www.gnu.org/s/guile` GNU scheme like language

[11] `http://www.lua.org`

[12] `http://luajit.org`

Существуют замечания [6], что работа со стековой машиной Lua в проектах некоторого масштаба может вызывать сложности. В нашем случае ввиду изолированности кода взаимодействующего с VM удалось избежать многих из перечисленных проблем.

4 Архитектура, особенности реализации

4.1 Место в стеке ПО

С точки зрения уровней ПО робота, TRIKFS является промежуточным слоем между системными файлами (отображающими реальные устройства подключеные к роботу) и их унифицируемым представлением (файлы порожденные FUSE) далее используемым в модельном приложении.

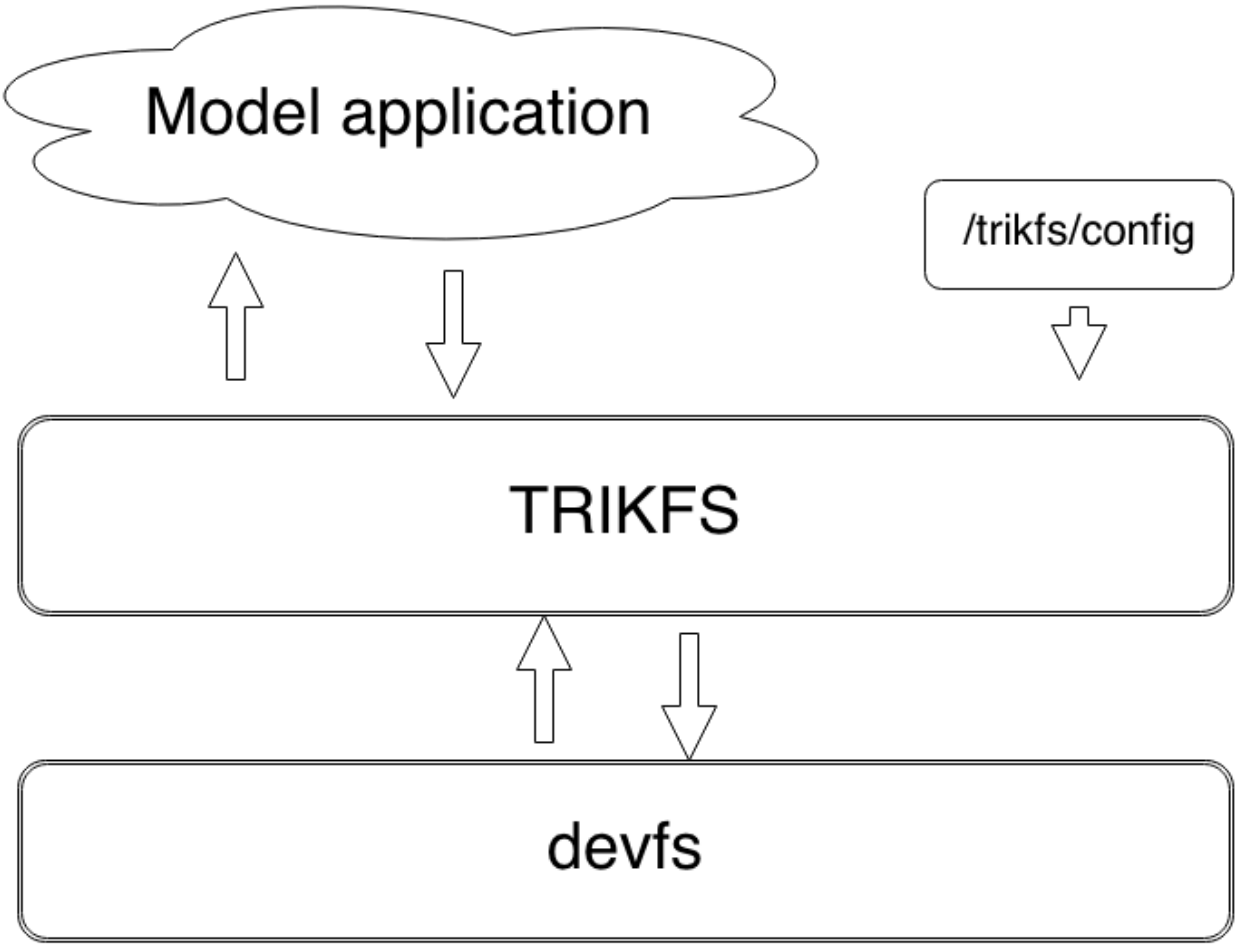

Рис. 3: место TRIKFS в стеке ПО робота

Конфигурация ФС производится записью конфигурационного файла в специальный файловый узел /trikfs/config. После записи структура (отобра-

жение системных устройств на виртуальные) будет незамедлительно обновлена.

4.2 Внутреннее устройство

TRIKFS условно может быть разбит на четыре составляющие.

4.2.1 main

Эта часть отвечает за инициализацию начального состояния приложения, fuse-specific частей, старт компоненты control, внутреннего логгера. Связывает компоненту fuse с control и workers. (А именно разделяет все fuse вызовы между ними)

Соответствует файл main.c

4.2.2 control

За текущее состояние и отображение системных файлов на файлы trikfs отвечает control. Эта компонента работает в контексте выделенной Lua VM. Поскольку конфигурационный файл TRIKFS является одновременно исполняемым Lua файлом, было довольно очевидным и логичным решением делегировать представление файловой структуры TRIKFS программному коду напрямую интерпретирующему содержимое конфигурационного файла. Поскольку операции восстанавливающую структуру ФС (opendir, readdir, getattr) вызываются в единичных случаях и не критичны к производительности, было решено оставить большую часть логики их обработки в контексте control Lua VM.

Соответствует файл config_file.c

4.2.3 worker

В контексте пользовательских операций чтения записи происходит определение фильтра-конвертера который будет обрабатывать данные приходящие от системных устройств. Далее в зависимости от конфигурации вызов может быть напрямую обработан с использованием кода пользовательской разделяемой библиотеки или же с использованием кода в контексте новой Lua VM. (Стоит заметить, что каждому открытому файлу TRIKFS в конфигурации для которого конвертер был задан на Lua инстанциируется отдельная Lua VM)

После обработки соответствующим конвертером порция данных пробрасывается дальше (пользователю либо системному устройству, в зависимости от направления потока)

Соответствует файл device_file.c/main.c

4.2.4 fuse

Задача этой компоненты связать fuse interface с внутренними представлением ФС.

Часть файла main.c

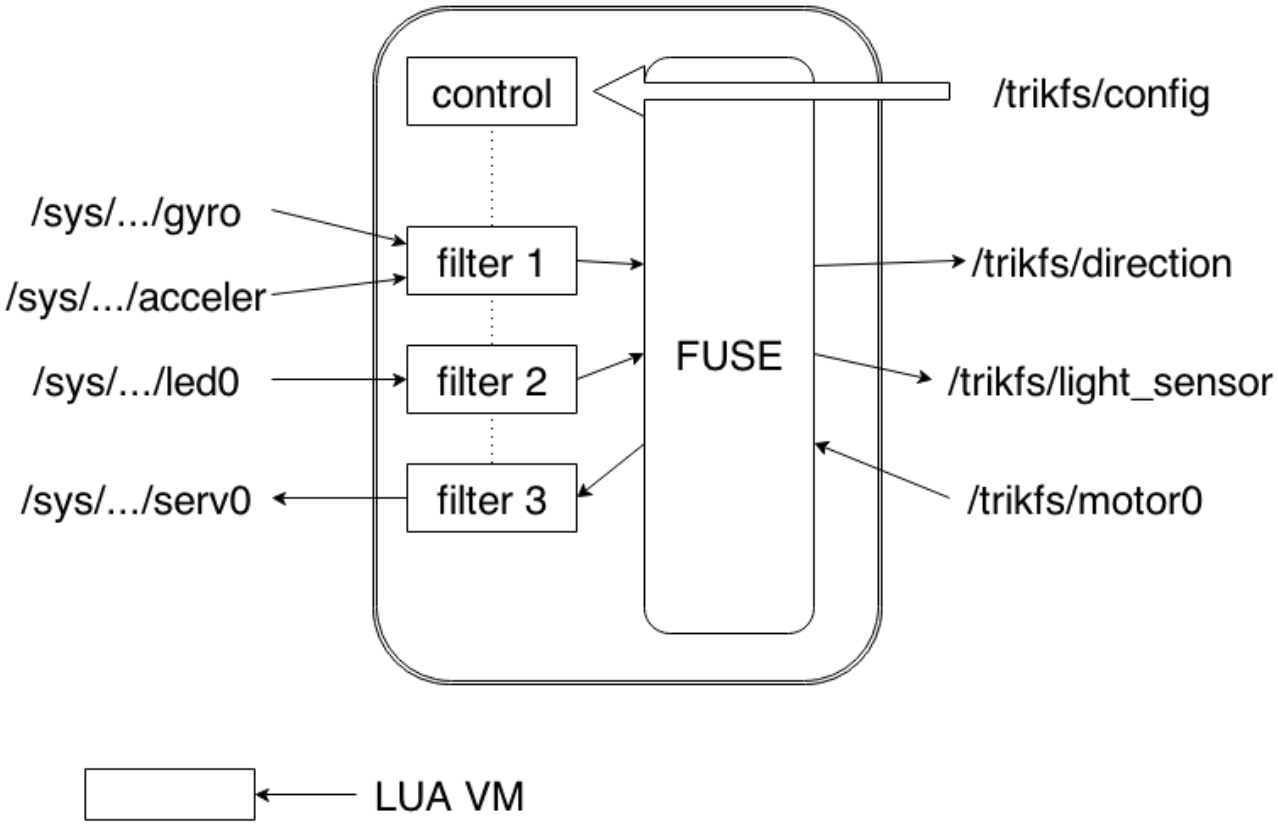

Рис. 4: Архитектура TRIKFS

4.3 /trikfs/config

Как уже было замечено ранее, конфигурационный файл TRIKFS является исполняемым Lua сценарием, что открывает возможности для генерации части конфигурационного скрипта на этапе “прочитывания”.

В частности таким образом мы можем генерировать код фильтров-преобразователей (прим: id_create), использовать в их реализации замыкания[13], а значит хранить контекст предыдущих вызовов.

После однопроходного вычитывания-исполнения предполагается конфигурационный сценарий устанавливает переменную trikfs, которая будет задавать корневую структуру TRIKFS.

[13] http://en.wikipedia.org/wiki/Closure_(computer_programming)

```
trikfs = { ..... }
```

В ней должна находиться таблица отображающая наименование TRIKFS файла к его описанию, т.е.:

```
trikfs = {
  random = {
    src = '/dev/urandom'
    input_tuple_size = 3,
    output_tuple_size = 3,
    converter = function (a_s, a, b_s, b)
      .....
    end
  },
  ....
}
```

После чего будет создан файл /trikfs/random/device, доступный для чтения и записи.

Здесь input_tuple_size и output_tuple_size задают размеры порции данных до применения фильтра-конвертера и после.

converter может принимать одно из следующих значений:

функция (как в примере выше)

```
converter = function (a_s, a, b_s, b)
  ...
end
```

где a_s задает размер входного вектора a, a b_s – размер выходного вектора b

строка

Задает путь к linux shared library:

```
converter = 'my_lib.so'
```

В этом случае библиотека должна экспортировать функцию со следующей сигнатурой:

```
int converter(int in_s, char* in, int out_s, char* out) {

  ....
  return 0;
}
```

Которая в случае успеха, должна в свою очередь возвращать 0.

таблица

Добавлена экспериментальная возможность собирать конвертер из кода на языке си находящегося в конфигурационном файле (при наличии на host ОС компилятора gcc)

Пример:

```
converter = {
        code = [[
                int converter(int in_s, char* in, int out_s, char* out)
                        out[0] = in[0] % 10 + '0' + 1;
                        out[1] = in[1] % 10 + '0' + 2;
                        out[2] = in[2] % 10 + '0' + 3;
                        out[3] = in[3] % 10 + '0' + 4;
                        return 0;
                }
        ]],
}
```

Разница с предыдущим в том, что .so файл собирается уже на целевой машине, а затем линкуется как описано ранее.

4.4 prelude.lua

Данный файл был создан с целью создания базовой библиотеки функций, которые пользователь может использовать при реализации собственных фильтров-конвертеров.

Сейчас в нем реализованы следующие вещи:

4.4.1 byte_array

Собственная реализация byte array выполненная с применением кода на языке C и Lua raw-data [14]

T.e. используемые в нашем случае byte_array с точки зрения Lua – raw-data (аллоцирование и деаллоцирование их происходит строго в контексте C).

Для манипулирования нашим byte_array при помощи FFI[15] зарегистрированы следующие две функции:

```
ba.get( <byte_array_variable>, <position> )

ba.set( <byte_array_variable>, <position>, <value> )
```

4.4.2 id_create

В качестве примера, чтобы показать возможность генерации кода конвертеров на этапе вычитывания конфигурационного файла реализована функция id_create

Пример:

```
converter = id_create(4)
```

[14] это значит, что для таких данных в Lua определены только операции с указателями на эти данные

[15] foreign function interface – механизм благодаря которому программа на одном языке может вызывать функции написанные на другом

Будет применен тривиальный фильтр, который перекладывает 4 байта со входа на выход.

```
id_create = function (len)

        return function(a_s, a, b_s, b)
                for i = 0, len - 1 do
                        ba.set(b, i, ba.get(a, i) )
                end
        end

end
```

5 Примеры использования

Поскольку на компьютере разработчика отсутствуют в чистом виде устройства специфичные для ТРИК, а именно периферия, т.е. те, ради которых данная ФС разрабатывалась, было выбрано минимальное множество устройств имеющихся в наличии и при этом претендующих на правдоподобность поведения периферии на контроллере.

В качестве таких устройств выбраны keyboard, mouse и urandom.

Тестирование проводилось при помощи следующего конфигурационного файла (доступен под названием hello.lua)

```
    input_tuple_size = 3,
   output_tuple_size = 3,
   converter = function (a_s, a, b_s, b)
    -- trikfs_log("hi, converter: ", a_s, " ", b_s)

    ba.set(b, 0, ba.get(a, 0) % 10 + 1 + 33)
    ba.set(b, 1, ba.get(a, 1) % 10 + 2 + 33)
    ba.set(b, 2, ba.get(a, 2) % 10 + 3 + 33)
    --ba.set(b, 3, ba.get(a, 3) % 10 + 4 + 33)
   end
},
random2 = {
  src = '/dev/urandom',
  input_tuple_size = 3,
  output_tuple_size = 3,
  converter = id_create(3)
},
      zero = {
              src = '/dev/zero',
              input_tuple_size = 4,
              output_tuple_size = 4,
```

```
                converter = {
                        code = [[
                                int converter(int in_s, char* in, int out_s, char* out)
                                        out[0] = in[0] % 10 + '0' + 1;
                                        out[1] = in[1] % 10 + '0' + 2;
                                        out[2] = in[2] % 10 + '0' + 3;
                                        out[3] = in[3] % 10 + '0' + 4;
                                        return 0;
                                }
                        ]],
                }
        },
zero2 = {
  src = '/dev/zero',
  input_tuple_size = 4,
  output_tuple_size = 4,
  converter = '/home/fijiol/koning/trikfs/tmp.so',
  --converter = '/tmp/a001.so',
},

mouse = {
        src = '/dev/input/mouse0',
        input_tuple_size = 4,
        output_tuple_size = 4,
        converter = '/home/fijiol/koning/trikfs/tmp.so'
},
keyboard = {
  src = '/dev/input/event2',
  input_tuple_size = 3,
  output_tuple_size = 3,
  converter = id_create(3)
},
```

```
}
```

6 Апробация

6.1 Демон телеметрии

ФС была портирована и инсталирована на контроллер ТРИК. Проведенная итерация тестирования на контроллере, к сожалению выявила пробему следующего рода: для интеграции с ТРИК необходимо было поддержать poll[16] события, что к сожалению на момент тестирования сделано не было. Из-за чего интеграцию с ТРИК пока нельзя считать успешной.

[16] http://en.wikipedia.org/wiki/Polling_(computer_science)

Заключение

В рамках данной работы разработана файловая система TRIKFS [17] с использованием FUSE и LuaJIT, унифицирующая доступ к периферии, поддерживающая

1. Динамическое конфигурирование
2. Задание пользовательских фильтров
3. В процессе апробации были выявлены некоторые проблемы

[17] код доступен здесь `https://bitbucket.org/fijiol/trikfs`

Список литературы

1. А. Ефимов *Области прорывных исследований в робототехнике* Control engineering Россия, No 4 (46), 2013

2. Python.org *Extending and embedding the python interpreter* Python documentation `https://docs.python.org/2/extending/index.html`

3. Aditya Rajgarhia *Performance and Extension of User Space File Systems* ACM Symposium on Applied Computing, SAC'2010, 206-213 (2010), ISBN: 978-1-60558-639-7

4. Linuxtopia *The Linux Kernel Module Programming Guide* `http://www.linuxtopia.org/online_books/Linux_Kernel_Module_Programming_Guide/x773.html`

5. Andries Brouwer *The Linux Virtual File System* The Linux Kernel `http://www.win.tue.nl/~aeb/linux/lk/lk-8.html`

6. Julien Danjou *Why not Lua* `https://julien.danjou.info/blog/2011/why-not-lua`