**Санкт-Петербургский Государственный университет**
**Математико-механический факультет**

Кафедра системного программирования

# Внедрение средства проверки сетевых протоколов в ОС Embox

Курсовая работа студента 344 группы
Мухатова Тимура Мухатовича

Научный руководитель
А. П. Козлов
асп. каф. сис. прог.

Санкт-Петербург
2014

# Оглавление

# Введение

Сетевой протокол - это набор правил, необходимых для того, чтобы два или более вычислительных процессов могли общаться друг с другом. Эти процессы могут выполняться на одной или на разных машинах, соединенных различными видами связи. Они могут быть процессами операционной системы, под управлением различных программ, виртуальными, модульными частями одной программы или могут быть компонентами операционной системы. Ключевым фактором являеся то, что имеется несколько процессов, которые должны взаимодействовать друг с другом .

Как вычислительные процессы, они общаются обмениваясь четко определенными сообщениями по каналу связи, и характер этого обмена определяет правила, составляющие сетевой протокол. Наиболее заметным аспектом этих правил является формат сообщений, которыми обмениваются процессы. Однако, более важны правила описывающие вычисления, которые должны сделать каждый из процессов, чтобы отправить сообщение, содержащее правильные значения в нужное время .

В процессе программирования сетевого протокола возникает множество различных проблем, помимо тех, с которыми сталкивается программист обычных программ. Многие из этих проблем можно отнести к одному из трех классов:

1. Внутренние ошибки
2. Внешние ошибки
3. Ошибки совместимости

Внутренние ошибки можно описать независимо от внешней среды. К ним относятся ошибки коммуникации, например, потеря или искажение сообщенией и ошибки безопасности: нарушение конфиденциальности, целостности, ошибки авторизации и аутентификации.

Внешние ошибки зависят от внешней среды. Примером может служить следующая ситуация: рассмотрим процесс, который посылает множество коротких сообщений без задержки. Во внешней среде может содержаться коммуникатор,

который при переполнении буфера сообщений, новые сообщения игнорирует. При том, что протокол реализован корректно, нужного результата не достигается.

Реализация протокола или отдельный компонент протокола могут быть разработаны в разное время и даже в разными людьми. Отсюда вытекают ошибки совместимости. Необходимо вести разработку вокруг четко оговоренной, централизованной спецификации.

Помимо вышеописанных проблем, разработчик может так же столкнуться с тем, что RFC (Request for Comments), документ содержащий спецификацию, может быть большим и не точным. В них часто содержатся только формальное описание спецификации.

В процессе эволюции, разрабатываются новые протоколы и создаются новые версии старых протоколов, чтобы удовлетворять новым требованиям. Изменения сделанные в одном протоколе, могут повлечь изменения в общей базе, из-за чего могут появиться ошибки. Например, RFC TCP 4614[1] от 2006 г. насчитывает описания из 32 RFC. Отсюда вытекает тот факт, что важно проверять корректность старых реализаций или осуществлять test driven development разработку протокола.

Для облегчения создания сетевых протоколов, поиска и устранения неисправностей используются различные фреймворки. Некоторые из них будут рассмотрены в следующих главах.

1 https://tools.ietf.org/html/rfc4614

# Постановка задачи

В настоящее время, в операционной системе реального времени Embox, существует небольшая реализация стека протоколов TCP/IP. Специальную проверку на корректность она не проходила. Моя задача выбрать средство, которое поможет проверить корректность существующих реализаций сетевых протоколов и внедрить его в ОС Embox. А так же составить набор тестов для последующего регрессионного тестирования. Средство должно быть простым для написания тестов, иметь платформенную совместимость с ОС Embox.

# Обзор решений

RFC 2398[2] содержит описание 12 инструментов для тестирования TCP. Эти инструменты конца 90-х годов и никакие из них активно не используются в тестировании современного TCP. Мной были рассмотрены 5 существующих решения: tcptest, synscan, packetdrill, unitesk и beep.

Tcptest - мультиплатформенный инструмент тестирования TCP/IP. Он предоставляет инструменты для регрессионного, аттестационного и фаззинг тестирования. Позволяет тестировать 3, 4 и 5 слои по OSI. Использует набор Python модулей PCS (Packet Construction Set), который позволяет создать классы которые описывают пакеты.

Synscan - инструмент для тестирования стека TCP и снятия “отпечатков” TCP/IP (так называемое определение ОС). Этот инструмент более подходит в сочетании с другими инструментами, и как отдельный, не служит для аттестационного тестирования. Для скриптов он использует язык похожий на tbit (TCP Behavior Inference Tool).

Packetdrill - инструмент, который позволяет разработчикам легко написать точные, воспроизводимые, автоматизированные тестовые скрипты для стека протоколов TCP/ UDP/ IP. Он работает на POSIX-совместимых ОС. Имеет язык скриптов похожий на tcpdump и strace, любимые инструменты сетевых инжинеров. Хорошо подходит для разработки новых функций, регрессионного тестирования, поиска и исправления неисправностей.

Unitesk - унифицированное решение для промышленного тестирования и обеспечения качества программного обеспечения. Разработанна в Институте системного программирования РАН. Позволяет создать комплексную среду тестирования и сопровождения ПО: тесты, средства взаимодействия тестовой платформы с тестируемой системой, методики тестирования (регрессионное тестирования, различные варианты “ручного” тестирования). Язык сценариев может быть реализован на C, C# и Java.

2 http://tools.ietf.org/html/rfc2398

BEEP (Blocks Extensible Exchange Protocol) - фреймворк для создания сетевых протоколов. Это основа, которая предоставляет механизмы асинхронной коммуникации, безопасности транспортного уровня, аутентификации, мультиплексирования канала, механизм создания фреймов, управления полосой пропускания канала и другие.

Ниже представлена таблица сравнительного анализа рассмотренных инструментов:

| | POSIX совместимость | Целевые протоколы | Тестирование сетевого интерфейса | Язык сценариев |
|---|---|---|---|---|
| tcptest | + | TCP/IP | + | Python |
| synscan | + | TCP/IP | - | tbit, activemap |
| packetdrill | + | TCP, UDP, ICMP | + | похож на tcpdump, strace |
| unitesk | - | IPv6, IPsec, UDP | + | C, C#, Java |
| beep | + | любой | - | нет |

В результате сравнительного анализа было выбрано два подходящих решения: tcptest и packetdrill. Они имеют совместимость с ОС Embox, тестируют целевые протоколы и имеют удобный язык сценариев. В итоге было выбрано средство - packetdrill. Основным фактором который повлиял на решение стал язык сценариев.

# Описание решения

Разработчики могут быстро изучить синтаксис packetdrill без понимания внутренних протоколов или самого packetdrill. Синтаксис также позволяет разработчику скриптов легко перевести поведение пакетов в тестовые скрипты. Инструмент работает в режиме реального времени и тесты часто завершаются в пределах одной секунды , что позволяет быстро совершать итерации.

Язык сценариев packetdrill имеет следующие типы операторов:

- Пакеты, использующие синтаксис похожий на tcpdump, включая TCP, UDP и ICMP, а также распространенные TCP опции: SACK, временные метки, MSS, масштаб окна и Fast Open.
- Системные вызовы, использующие Strace-похожий систаксис.
- Команды оболочки, которые позволяют конфигурировать систему или делать утверждения о состоянии сетевого стека.
- Скрипты Python, которые позволяют выводить информацию или делать утверждения о состоянии tcp_info, которое ОС выставляет на TCP сокеты.

Packetdrill работает с пакетами и системными вызовами, тестируя точные последовательности реальных событий. Инструмент тестирует точный образ ядра используемый в разработке, работает в режиме реального времени на физической машине. Может надежно воспроизводить время тестового сценария с менее чем одним ложным отказом в 2500 тестовых прогонов. Позволяет скрипту работать в IPv4, IPv6 или IPv4-mapped режиме IPv6. Он работает на Linux, FreeBSD, OpenBSD и NetBSD, и переносим на POSIX-совместимые операционные системы.

Packetdrill анализирует весь тестовый скрипт, а затем выполняет каждую отмеченную временем строку в режиме реального времени – со скоростью, описываемой временными метками, для проигрывания и проверки сценария. Для каждой строки системного вызова, packetdrill выполняет системный вызов и проверяет, что он возвращает ожидаемый результат. Для каждой командной строки, packetdrill выполняет команду оболочки. Для каждого входящего пакета, packetdrill строит пакет и внедряет его в ядро. Для каждого исходящего пакета, packetdrill

исследует следующий исходящий пакет и проверяет, что сроки и содержание данного пакета соответствуют сценарию.

Packetdrill допускает два режима тестирования: в локальном режиме, используя виртуальное сетевое устройство TUN, или в удаленном режиме, используя физическую сетевую карту. В локальном режиме, интсрумент использует одну машину и виртуальное сетевое устройство TUN в качестве источника и приемника пакетов. Этот вариант позволяет тестировать системные вызовы, сокеты, слои TCP и IP, и проще в использовании, так как тут меньше вариации времени, и пользователям не нужно координировать доступ к нескольким машинам. В удаленном режиме, пользователи запускают два экземпляра packetdrill, один из которых находится на удаленной машине и взаимодействует с тестируемой системой через локальную сеть. Этот подход тестирует всю сетевую систему: системные вызовы, сокеты, TCP, IP, программное и аппаратное обеспечение, драйвер сетевой карты, сетевое оборудование и саму сеть. Тем не менее, из-за присущей изменчивости многих компонентов при испытании удаленный режим может привести к более большим изменениям во времени, что может привести к ложным исходам тестирования.

# Реализация и внедрение

Как было описано выше, инструмент packetdrill использует виртуальное сетевое устройство TUN. В ОС Embox реализиции данного адаптера не было предусмотрено. Поэтому в мою задачу так же вошло программирование виртуального TUN адаптера.

TUN обеспечивает прием и передачу пакетов программам запущенным в пользовательском режиме. Это можно представить как простое устройство, которое, вместо того чтобы получать пакеты с физической сети, получает их от пользовательской программы и вместо отправки пакетов через физическую сеть записывает их в пользовательскую программу. Схему действия можно посмотреть на рисунке 1.

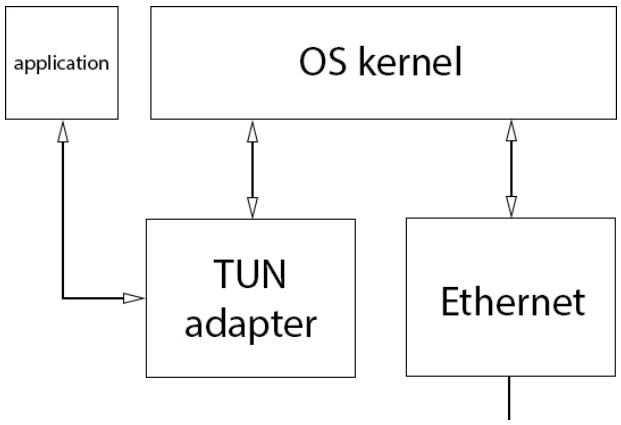


Рис. 1: Принцип работы виртуального TUN-адаптера.

Для того чтобы использовать драйвер программа должна открыть /dev/net/tun и выполнить функцию регистрации сетевого устройства в ядре. Сетевое устройство появится в формате tun%d (%d - номер устройства). После создания экземпляра, программа может работать с устройством: передавать и получать пакеты. Когда программа закрывает файловый дескриптор, сетевое устройство и все соответствующие пути исчезнут.

Внутренняя функциональность выглядит так: когда в ядре дело доходит до посылки пакета по сети, он передается в пользовательскую программу, а когда нужно передать ядру пакет, то мы делаем это через адаптер.

Обмен информацией в TCP соединении происходит через сокеты: клиентский или серверный. Сокет в каждый момент времени имеет определенное состояние:

LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, CLOSING и т.д. Схему описывающую взаимодействие можно увидеть на рисунке 2.

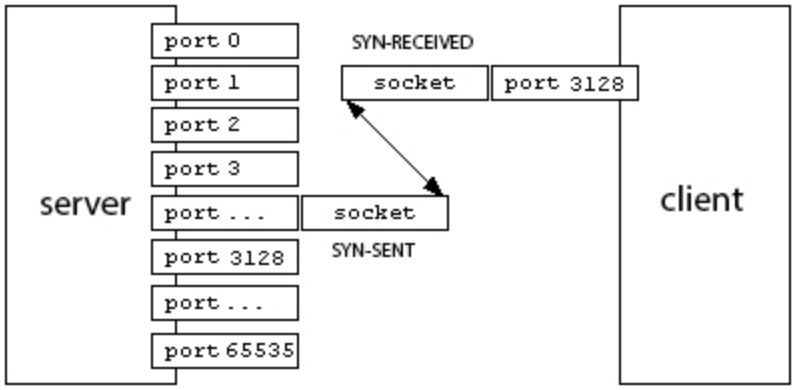


Рис. 2: Схема обмена информацией через TCP сокеты.

Описанные состояния система хранит во внутренней памяти и пользовательским программам они не доступны. Мне потребовалось реализовать интерфейс получения состояний пользовательскими программами.

Также целью было составить набор тестов TCP для регрессионного тестирования. Рассмотрим составленный мной тест на рисунке 3, на котором изображен скрипт, который тестирует быстрый повтор (Fast Retransmit) TCP. Сценарий начинается с создания сокета (строки 1-4) и установления соединения (линии 5-8). После записи данных в сокет (строка 9), сценарий предполагает, что тестируемый сетевой стек отправит пакет данных (строка 10), а затем выполняет команду внедрить подтверждение (ACK) (строка 11), которые стек будет обрабатывать. Скрипт в конечном счете подтверждает, что быстрый повтор происходит после прибытия трех дубликатов подтверждений. Символами “...” указываются не важные для примера параметры.

```
1   0  socket(..., SOCK_STREAM, IPPROTO_TCP) = 3 // создание сокета
2  +0  setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0 //
   избегание проблем со связыванием
3  +0  bind(3, ..., ...) = 0 // связывание сокета
4  +0  listen(3, 1) = 0 // начало прослушивания

5  +0  < S 0:0(0) win 32792 <mss 1000,sackOK,nop,nop,nop,wscale 7> //
   ввод команды SYN
6  +0  > S. 0:0(0) ack 1 <...> // ожидаем SYN/ACK.
7  +.1 < . 1:1(0) ack 1 win 257 // ввод команды ACK.
```

```
8  +0  accept(3, ..., ...) = 4 // подтверждение соединения

9  +0  write(4, ..., 1000) = 1000 // запись данных в размере 1 MSS
10 +0  > P. 1:1001(1000) ack 1 // предположение об отправке
11 +.1 < . 1:1(0) ack 1001 win 257 // ввод ACK после 100мс

12 +0  write(4, ..., 4000) = 4000 // запись данных в размере 4 MSS
13 +0  > . 1001:2001(1000) ack 1 // ожидаем передачу
14 +0  > . 2001:3001(1000) ack 1
15 +0  > . 3001:4001(1000) ack 1
16 +0  > P. 4001:5001(1000) ack 1

17 +.1 < . 1:1(0) ack 1001 win 257 <sack 2001:3001,nop,nop> // ввод 3
   ACK с SACK
18 +0  < . 1:1(0) ack 1001 win 257 <sack 2001:4001,nop,nop>
19 +0  < . 1:1(0) ack 1001 win 257 <sack 2001:5001,nop,nop>

20 +0  > . 1001:2001(1000) ack 1 // предполагаем быстрый повтор
21 +.1 < . 1:1(0) ack 6001 win 257 // ввод ACK для всех данных
```

Рис. 3: Скрипт Packetdrill проверяющий быстрый повтор TCP.

# Результаты

В ходе работы выбрано средство, проверяющее корректность существующих реализаций сетевых протоколов операционной системы реального времени Embox и успешно внедрено. Средство полностью удовлетворяет требованиям и будет использоваться в дальнейшем при разработке новых и усовершенствовании старых сетевых протоколов. Некоторые тесты в наборе packetdrill не были предусмотрены, т.к. эти части имеют низкую вероятность ошибки. Мной было составлено 4 таких теста для дальнейшего регрессионного тестирования.

Реализация TCP в проекте Embox имеет только основные функции и некоторые тесты попусту не запустились. Тесты которые запустились не выявили ошибок в существующих реализациях протоколов.

# Список литературы

McGuire T. M. Network Protocols [Электронный ресурс] // Correct Implementation of Network Protocols. 2004. URL: http://www.crsr.net/files/CorrectImplementation.pdf (дата обращения: 11.04.2014).

Bilouro V. H., Neville-Neil G. Tcptest: A multi-platform TCP/IP v4 Stack Testing Tool [электронный ресурс] // Google Summer of Code project. 2008. URL: https://code.google.com/p/tcptest/ (дата обращения: 28.03.2014).

Taleck G. SYNSCAN: Towards Complete TCP/IP Fingerprinting [Электронный ресурс]. 2004. URL: http://synscan.sourceforge.net/taleck-synscan-2004.pdf (дата обращения: 19.04.2014).

Кознов Д. В., Арчак Н. А. Апробация технологии тестирования UniTESK // сборник Системное программирование. СПб.: Изд-во С.-Петерб. ун-та, 2004.

Cardwell N., Cheng Y., Brakmo L., Mathis M., Raghavan B., Dukkipati N., Hsiao-keng Jerry Chu, Terzis A., Herbert T. Packetdrill: Scriptable Network Stack Testing, from Sockets to Packets [Электронный ресурс] // USENIX Annual Technical Conference. 2013. URL: http://goo.gl/5t1nn2 (Дата обращения: 25.05.2014).