Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет по преддипломной практике

Студент	А.В. Патеенок
Руководитель	В.В. Хилько
Консультант от кафедры ЭВМ	А.Г. Третьяков
Нормоконтролер	А.С. Сидорович

МИНСК 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С каждым днём промышленность развивается и одним из самых острых вопросов для человечества остаётся вопрос экологии. Крупные предприятия тратят огромные суммы на поддержание экологичности своего процесса изготовления, так как они имеют связь с общественностью, и её мнение определяет количество новых клиентов и им это выгодно. В то же время, начинающие корпоративные машины не могут позволить себе средств на такие роскоши, и, тем самым, экономят на общей окружающей среде.

Причин загрязнения может быть очень много, и очень радует тот факт, что в последнее время эта тема вновь стала актуальна, и многие производители начали задумываться над этим вопросом.

Как бы это не было печально, но эта проблема крайне остра в Беларуси. Хоть количество производства ниже, чем в других Европейских сранах, но небольшая прибыльность предприятий не позволяет им обеспечить своему производственному процессу достаточную экологичность.

На сегодняшний день встраиваемые системы являются неотъемлемой частью нашей жизни. Их используют практически повсеместно – начиная от автомобилей (системы климат-контроля, автопилотирования) и заканчивая холодильниками, стиральными машинами, телевизорами и другой бытовой техникой, которыми можно управлять, например, с помощью встраиваемых решений или смартфонов.

Встраиваемая система (англ. embedded system) – это система, которая выполняет определенный производителем ограниченный набор функций, который не предназначен для изменения конечным пользователем [1]. К данным системам обычно устанавливаются требования высокой отказоустойчивости, обработки данных в режиме реального времени, практически мгновенной реакции на входные воздействия. Из-за этого разработчики вынуждены использовать специализированные операционные системы, которые соответствуют вышеупомянутым требованиям. Их называют операционными системами реального времени (англ. RTOS), примерами которых могут служить VxWorks, FreeRTOS, TI-RTOS, eCos, RTLinux.

После описания логики работы встраиваемой системы на каком-либо языке программирования необходимо проверить описание на соответствие вышеописанным требованиям, в том числе и требованиям времени. Для этого приходится либо использовать какие-либо готовое решения, либо создавать и применять собственный профилировщик задач. При использовании какой-либо операционной системы необходима поддержка профилирования самой операционной системой, иначе может потребоваться правка ее исходных кодов либо добавление в каждую задачу кода для сбора статистики по профилированию (что, практически, сопоставимо с имлементацией собственной системы профилирования).

Целью данного дипломного проекта является создание модуля сбора статистики в режиме реального времени для операционной системы TI-RTOS, который позволит производить отладку в условиях, наиболее приближенных к реальным. Работа данного проекта не будет требовать наличие подключенного отладчика или программатора (например, по интерфейсу JTAG).

Для достижения поставленной цели нужно выполнить следующие задачи:

– запустить пример работы с операционной системой TI-RTOS, содержащий несколько псевдозадач, проверить корректность его работы (данный пример в дальнейшем будет использоваться в качестве примера для тестирования);

– добавить в пример несколько тестовых HWI и SWI для более полной проверки модуля;

– исследовать возможности профилирования, предоставляемые операционной системой;

– разработать модуль сбора статистики для каждого ядра процессора на основе возможностей, предоставленных операционной системой;

– исследовать возможности межъядерного взаимодействия;

– разработать модуль для отправки собранной информации для ее внешней обработки и отображения.

Дипломный проект будет реализовать следующие функции:

– сбор статистики по задачам;

– сбор статистики по прерывания (HWI и SWI);

– сбор системных и пользовательских сообщений;

– частичное конфигурирование системы профилирования во время ее работы;

– получение данных из системы профилирования для их дальнейшего отображения;

– отображение полученных данных на веб-странице в режиме реального времени.

Таким образом, можно утверждать, что тема актуальна для реализации и может быть в дальнейшем использована для получения коммерческой выгоды.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Операционные системы реального времени

Сегодня все больше и больше встраиваемых систем окружают нас. «Умная» бытовая техника, повсеместные системы видеонаблюдения – и этот список еще можно продолжать и продолжать. При этом, большинству этих систем требуется максимально быстро реагировать на воздействия (например, система видеонаблюдения должна начать запись при обнаружении движения). Из-за этого, разработчкики вынуждены использовать операционные системы реального времени. В отличие от других типов операционных систем (пакетных и интерактивных), основная часть задач выполняется достаточно быстро, чтобы не задерживать работу остальных компонентов [2]. Остальным задачам задают низкий приоритет, что позволяет им работать в фоновом режиме и практически не влиять на работу высокоприоритетных задач.

Операционные системы реального времени имеют еще ряд особенностей [2], [3]:

– временные ограничения на обработку данных;
– предсказуемость поведения во времени;
– простота внутренней реализации.

Временные органичения на обработку данных подразумевают, что скорость обработки данных составляет не меньше скорости их поступления. Это обусловлено тем, что система должна полностью обработать данные до поступления новой порции, поскольку если такого не будет происходить, необработанные данные будут накапливаться, что приведет к затору.

Иногда системы реального времени допускают частичные заторы путем отбрасывания необработанных данных. В зависимости от типа обрабатываемой информации может быть разрешена разная степень игнорирования входных данных [2].

Простота внутренней реализации связана с тем, что обычно данная операционная система не программируется извне (конечным пользователем). Поэтому она не требует поддержки расширяемости во время исполнения, различных уровней привелегий, имеет более примитивную внутреннюю структуру и, обычно, ей практически не требуется время для запуска [3].

Отсюда также следуют и различия в архитектуре. Архитектура операционной системы реального времени упрощена, из нее убраны излишние слои. Она визуально отмечена на рисунке 1.1 желтым цветом. Зеленым цветом отмечено управляющий микропроцессор или микроконтроллер, который исполняет код, а также периферийные устройства, которые предназначены для ввода и вывода информации. Красным отмечено приложение. Оно управляет аппаратным обеспечением (зеленой компонентой) с помощью операци-

Рисунок 1.1 – Взаимодействие приложения с операционной системой реального времени

онной системы реального времени (желтой компоненты) [4].

Таким образом, операционные системы реального времени имеют серьезные различия с другими видами операционных систем и должны выделяться в их отдельную категорию.

1.2 Обзор операционной системы реального времени TI-RTOS

TI-RTOS – операционная система реального времени, созданная и поддерживаемая компанией Texas Instruments для микропроцессоров [5] и микроконтроллеров [6] собственной разработки. Она распространяется бесплатно по лицензии BSD с полным набором исходных кодов, что позволяет использовать ее для ускорения разработки, не публикуя при этом собственных исходных кодов, но не позволяет переиздовать продукт с изменениями. Для сравнения: Linux распространяется по лицензии GPLv2 и требует публикации исходных кодов, FreeRTOS распространяется по лицензии MIT и не имеет никаких ограничений.

Характеристики ядра TI-RTOS [7]:

– предоставление сервисов операционной системы не нарушая реального времени;

– планировщик задач работает в режиме вытесняющей многозадачности;

– ядро управляется событиями;

– ядро имеет объектную архитектуру;

– большинство вызовов API ядра имеют деретминированную длительность;

– API для отладки в режиме реального времени небольшие и быстрые.

Из вышеописанных характеристик можно сделать следующие выводы:

– TI-RTOS минимально воздействует на пользовательское приложение;

– всегда выполняет задачи в соответствии с их приоритетом;

– система отладочных сообщений спроектирована так, чтобы ее можно было сохранить даже в конечной версии продукта из-за ее незначительного воздействия.

Задачей (или процессом, потоком) в операционных системах реального реального времени обычно называют поток или псевдопоток, который выполняют одну простую функцию, запускается в выделенном контексте со определенным приоритетом [7]. На основании приоритета планировщик операционной системы производит выбор текущей задачи для исполнения среди тех, которые имеют статус готовности.

Контекстом в операционных системах реального времени называют минимальный набор данных, используемых задачей, которые должны сохраняться. Это позволяет прерывать процесс выполнения задачи (как по специальной команде от самой задачи, так и по специальной команде извне) и восстанавливать его с остановленной позиции.

Прерыванием (англ. interrupt) в операционных системах реального времени называется предупреждение процессора о произошедшем событии. Оно может быть сгенерировано как с помощью инструкций процессора, так и периферийными устройствами, и требует наиболее быстрой обработки. Предупреждение о произошедшем событии генерирует контроллер прерываний, после чего процессор вызывает обработчик прерывания (англ. ISR) [3], [7].

В TI-RTOS прерывания, описанные выше, называются аппаратными прерываниями (англ. HWI). Они вызывают специальный обработчик прерываний, который поставляется с TI-RTOS, который по завершению пользовательской функции для обработки сбросит флаг прерывания.

Также в TI-RTOS существуют программные прерывания (англ. SWI). Они имеют следующие особенности [7], [8]:

– имеют повышенный приоритет по сравнению с задачами;

– имеют пониженный приоритет по сравнению с аппаратными прерываниями;

– не имеют собственного контекста выполнения, а использу;

– имеют внутреннюю приоритезацию (до 32 уровней приоритетов в зависимости от используемого микроконтроллера или микропроцессора);

– предназначены для вынесения «тяжелой» обработки данных из HWI, которая имеет менее жесткие временные требования;

Таким образом, TI-RTOS является бесплатной современной поддерживаемой операционной системой реального времени, которая содержит в себе интересные решения и может быть использована для ускорения разработки прошивок для устройств.

1.3 Инструментирование

Инструментирование – это возможность отслеживания и измерения уровня производительности продукта, определения ошибок, записи информации трассировки [9].

Большинство современных сред разработки и компиляторов поддерживают инструментирование «из коробки»: после установки программного средства у программиста имеется в наличии большинство необходимых ему компонентов для инструментирования. В основном набор инструментирования представлен в минимальной конфигурации, то есть имеет очень ограниченный набор функций. Инструменты для более глубокого анализа кода обычно разрабатываются отдельными компаниями и интегрируются в привычные разработчику среды. Примером может послужить ReSharper – продукт от компании JetBrains, который улучшает статический анализ кода для языков C, C++ и C# для интегрированной среды разработки от компании Microsoft [10].

Обычно инструментирование проводится по следующим направлениям:

– трассировка кода;
– отладка и обработка исключений;
– профилирование;
– подсчет производительности;
– логирование данных.

Трассировка – это процесс использования логирования для записи информации о процессе исполнения. Обычно ее использует среда разработки совместно с отладчиком для предоставления возможности отладки кода и обработки исключений [11].

Отладка – это процесс поиска и разрешения дефектов программы, которые мешают ее корректной работе.

Профилирование – это способ анализа программного обеспечения во времени его выполнения, которое позволяет измерять различные параметры работы программы.

Подсчет производительности – процесс предоставления информации о качестве работы операционной системы, приложения, драйвера или службы с помощью счетчиков производительности. Последние позволяют определить «узкие места» в производительности тестируемого программного обеспечения. Эти данные могут предоставляться и в графическом виде [12].

Логирование данных – процесс сбора и хранения данных на протяжении определенного периода времени с целью записи произошедших в системе событий.

Единственным особенным требованием, выдвигаемым к программным продуктам такого рода является минимальное влияние на работу системы,

которая подвергается анализу с помощью этих утилит. Это можно объяснить желанием и необходимостью разработчиков проверке и исследовании корректности работы реализованной системы в максимально приближенных к реальным условиях. Остальные требования не отличаются от предъявляемых ко всем программам. Например, промышленным стандартом по качеству разрабатываемого программного обеспечения является стандарт ISO/IEC 25010.

Таким образом, все утилиты инструментирования разрабатываются для сбора информации о работе какой-либо системы на этапе ее разработки или использования с целью дальнейшего улучшения качества ее работы.

1.4 Сравнение с аналогами

Операционная система TI-RTOS содержит компоненты, которые позволяют разработать свою собственную систему профилирования. Они называются унифицированной измерительной архитектурой (англ. UIA) и являются частью TI-RTOS SDK. UIA может быть скачана с веб-сайта разработчика [13] как вместе с SDK для конкретного семейства процессоров, так и в виде отдельного модуля.

Также TI реализовала свой собственный компонент для профилирования на основе предоставленного API. Он называется System Analyzer и является частью UIA. Поскольку он является неотъемлемой частью UIA, под UIA будет пониматься как сам UIA, так и его компонент System Analyzer.

UIA содержит в себе следующие основные компоненты [14]:

– интерфейсы прикладного программирования (англ. API);
– набор предопределенных программных событий и метаданных;
– регистратор событий (англ. event logger);
– коммуникатор (англ. transport);
– захватчик событий операционной системы;
– поддержка нескольких ядер.

UIA является очень гибким и необходимым компонентом на этапе написания приложений на базе данной операционной системы. Совместно с системой сборки XDCTools он позволяет конфигурировать вывод отладочных сообщений еще на этапе компиляции, что влечет за собой уменьшение объема памяти, которую будет занимать прошивка. На этапе компиляции также можно выбрать способ связи с прибором (через интерфейс JTAG или Ethernet, с использованием протокола TCP или UDP и так далее). Но, к сожалению, для работы компонента UIA в полной мере требуется установленная и сконфигурированная Code Composer Studio, предлагаемая TI для разработки и отладки конечной версии программы.

Code Composer Studio является интегрированной средой разработки (англ. IDE) для разработки приложений на базе встаиваемых решений от компании Texas Instruments, основана на платформе Eclipse, может рабо-

тать на современных операционных системах из семейств Windows, Linux и macOS [15].

С точки зрения конечного пользователя аппаратно-программного продукта у Code Composer Studio есть следующие проблемы:

– большой объем занимаемого пространства;
– имеет большое количество непонятных и неиспользуемых функций;
– его требуется устанавливать;
– оно не доступно для мобильных устройств;
– не является интуитивно понятным.

Решить любую из вышеперечисленных проблем можно с помощью разработки собственной системы профилирования. Таким образом, разрабатываемая система сбора статистики будет основана на компоненте UIA, поскольку разработка аналогичного компонента займет большое количество времени и не будет являться целесообразным решением. При этом она будет лишена описанных выше недостатков использования Code Composer Studio.

1.5 Выбор способа получении информации с прибора в режиме реального времени

На сегодняшний день в мире очень активно развиваются веб-технологии. Казалось, еще только вчера интернет был очень медленным (1 Мб/с), сайты загружались длительное время и верстались постранично. Сегодня же, сайты верстаются уже поблочно и подсчет денежных затрат идет уже за какие-либо конкретные функциональные требования, а не за количество веб-страниц.

Из-за стремительного развития веб-технологий, существует несколько различных способов для получения информации с сервера, чью роль и будет исполнять прибор, сделанный с использованием разрабатываемой библиотеки. Самые известные из них описаны ниже [16], [17]:

– технология AJAX;
– длинные запросы (англ. long polling, COMET);
– веб-сокеты (англ. websocket);
– технология WebRTC;
– события, посылаемые сервером (англ. SSE).

Технология AJAX представляет набор технологий, таких как HTML, CSS, JavaScript, XHR и другие. Для решения данной задачи из этого набора подойдет XHR – запрос XML по протоколу HTTP (англ. XMLHttpRequest). Она позволяет делать синхронные и асинхронные запросы к веб-серверу, что позволяет получать оттуда данные, а также передавать информацию на сервер [18]. Технология проста в реализации со стороны сервера и поддерживается практически всеми браузерами: по данным сайта *caniuse.com* эта технология будет работать у 96,22% пользователей сетью интернет.

Но XHR имеет следующие недостатки:

– для получения каждого нового пакета данных требуется делать новый запрос, что приводит к дополнительной нагрузке на сервер;

– каждое новое установление соединения для получения данных может быть скомпроментировано с помощью атаки посредника (англ. MITM attack);

– из-за расхождений времени, транспортных и иных задержек, частота запроса данных клиентом может быть слишком высокой, что может привести к дополнительным излишним запросам на сервер.

Получение данных с помощью длинных запросов является разновидностью использования XHR, но заслуживает отдельного рассмотрения, поскольку решает ее последний недостаток: ответ на запрос происходит только по готовности данных на сервере путем настройки времени ожидания для сокета, по которому ожидается ответ. Но этот способ не решает оставшиеся проблемы XHR.

Веб-сокеты являются единственным видом сокетов, которые доступны в браузерах для использования их в явном виде. Этот вид является надстройкой над TCP сокетами и используют их в качестве протокола транспортного уровня [19]. Веб-сокеты поддерживаются большинством современных браузеров: по данным сайта *caniuse.com* эта технология будет работать у 95,9% пользователей сетью интернет.

Эта технология имеет следующие особенности [19]:

– при установке соединения требуется рассчитывать SHA-1 хеш-сумму на стороне сервера, что может плохо сказаться на производительности прибора с низкой частотой ядра;

– позволяет производить двустороннюю коммуникацию: передавать данные как от клиента к серверу, так и наоборот;

– значительно меньше подвержен атакам посредника, поскольку соединение устанавливается только по требованию (обычно, один раз при загрузке страницы);

– не требует регулярного опроса сервера для получения от него данных.

В отличие от остальных описанных выше технологий, WebRTC предназначен для потоковой передачи видео- и аудио-трафика. Он разрабатываетя компанией Google и на данный момент подкреплен только черновой версией стандарта [20]. Хоть она и поддерживается у 87,25% пользователей (по данным сайта *caniuse.com*), данная технология имеет проблемы с совместимостью в разных браузерах. К тому же она работает по принципу точка-точка (англ. peer-to-peer), что также не является удобным в данном случае.

Технология SSE также является расширением технологии XHR. Она позволяет использовать как обычные XHR запросы, так и их «длинную» версию. В отличии от оригинального XHR, данная технология предназначена

для однонаправленного обмена информацией: от сервера клиенту. Если же требуется двунаправленный канал связи, то требуется использовать другие технологии [17]. По данным сайта *caniuse.com* эта технология будет работать у 91,3% пользователей сетью интернет. В отличии от оригинальной технологии XHR, в данной соединение с сервером устанавливается однократно, что уменьшает вероятность атаки посредника.

В статье [21] было произведено сравнение производительности технологии XHR и веб-сокетов. Для работы с последними в тесте использовался фреймворк «Socket.io», а для XHR – REST API, которое использовало его для общения с сервером. Результаты показали, что в пакетах веб-сокетов содержится большее количество полезной информации (в тесте размер переданных байтов при использовании веб-сокетов был в пять раз меньше, чем через XHR), и при передаче ответов на 50 запросов дало выигрыш в производительности приблизительно в два раза. При увеличении числа запросов количество обрабатываемых запросов за единицу времени по веб-сокетам было лучше приблизительно в 4 раза. Однако, стоит отметить, что установление соединения является более эффективным в технологии XHR по количеству передаваемых байт, что утверждает об эффективности использования XHR при передаче непотоковой информации. Примером может послужить получение отдельного файла конфигурации для веб-страницы при ее загрузке.

Таким образом, самым оптимальным с точки зрения производительности является использование технологии веб-сокетов. Она поддерживается большинством современных браузеров, требует дополнительные ресурсы процессора только в момент установления соединения и позволяет принимать данные со стороны пользователя, что может быть использовано для конфигурирования системы во время ее работы.

Следовательно, в рамках данного курсового проекта будет реализована утилита для логирования данных и базового профилирования запущенных задач. Она будет выполнена в виде веб-страницы, которая будет получать данные с использованием механизма веб-сокетов, разбирать их и отображать для конечного пользователя.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

ЗАКЛЮЧЕНИЕ

Во время работы над преддипломной практикой была изучена предметная область, детально разобраны внутренние структуры и механизмы ядра Linux, различные сценарии взаимодействия пространства пользователя с ядром, особенности проектирования и разработки кода ядра, инфраструктура поддержки, организация посылки и приема патчей.

Кроме того, начата разработка структур данных, схемы функционирования системных вызовов. Написаны тесты производительности существующих систем сбора системной информации, демострационные программы. Изучены существующие методы сбора системной информации в других системах, а также реализация файловой системы `procfs` в Linux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Heath, Steve. Embedded Systems Design / Steve Heath. — 2nd edition. — ELSEVIER, 2003. — Pp. 1 – 14.

[2] Таненбаум, Э. Современные операционные системы / Э. Таненбаум. — Питер, 2015. — С. 178 – 198.

[3] Valvano, Jonathan. Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers / Jonathan Valvano. — Jonathan Valvano, 2017. — Pp. 19 – 22, 292 – 329.

[4] Real Time Operating System – What is RTOS [Электронный ресурс]. — Режим доступа : https://www.engineersgarage.com/articles/rtos-real-time-operating-system. — Дата доступа : 10.03.2019.

[5] TI-RTOS-PROC TI-RTOS: Real-Time Operating System – Obtain TI-RTOS via Processor SDKs for Sitara Processors & DSPs | TI.com [Электронный ресурс]. — Режим доступа : http://www.ti.com/tool/TI-RTOS-PROC. — Дата доступа : 16.03.2019.

[6] TI-RTOS-MCU TI-RTOS: Real-Time Operating System (RTOS) for Microcontrollers (MCU) | TI.com [Электронный ресурс]. — Режим доступа : http://www.ti.com/tool/TI-RTOS-MCU. — Дата доступа : 16.03.2019.

[7] Embedded Advantage, Texas Instruments. Intro to the TI-RTOS Kernel Workshop. Student Guide / Texas Instruments Embedded Advantage. — 3.2 edition. — 2019. — Pp. 94 – 271.

[8] SYS/BIOS (TI-RTOS Kernel) User's Guide. — T edition. — 2017. — 261 P.

[9] Source code instrumentation overview [Электронный ресурс]. — Режим доступа : https://www.ibm.com/support/knowledgecenter/SSSHUF_8.0.0/com.ibm.rational.testrt.doc/topics/cinstruovw.html. — Дата доступа : 11.03.2019.

[10] ReSharper: The Visual Studio Extension for .NET Developers by JetBrains [Электронный ресурс]. — Режим доступа : https://www.jetbrains.com/resharper/. — Дата доступа : 11.03.2019.

[11] TracingBook – TracingWiki [Электронный ресурс]. — Режим доступа : https://web.archive.org/web/20090224184220/http://ltt.polymtl.ca/tracingwiki/index.php/TracingBook. — Дата доступа : 13.03.2019.

[12] Performance Counters – Windows applications | Microsoft Docs [Электронный ресурс]. — Режим доступа : https://docs.microsoft.com/en-us/windows/desktop/perfctrs/performance-counters-portal. — Дата доступа : 13.03.2019.

[13] Category:UIA – Texas Instruments Wiki [Электронный ресурс]. — Режим доступа : http://processors.wiki.ti.com/index.php/Category:UIA. — Дата доступа : 09.03.2019.

[14] System Analyzer. User's Guide. — 2014. — 135 P.

[15] CCSTUDIO Code Composer Studio (CCS) Integrated Development Environment (IDE) | TI.com [Электронный ресурс]. — Режим доступа : http://www.ti.com/tool/CCSTUDIO. — Дата доступа : 09.03.2019.

[16] javascript – In what situations would AJAX long/short polling be preferred over HTML5 WebSockets? – Stack Overflow [Электронный ресурс]. — Режим доступа : https://stackoverflow.com/a/10029326/6818663. — Дата доступа : 17.03.2019.

[17] Zakas, Nicholas C. Professional JavaScript for Web Developers / Nicholas C. Zakas. — Wiley John + Sons, 2012. — Pp. 701 – 730.

[18] AJAX – Руководство Web-разработчика | MDN [Электронный ресурс]. — Режим доступа : https://developer.mozilla.org/ru/docs/Web/Guide/AJAX. — Дата доступа : 17.03.2019.

[19] Melnikov, Alexey. The WebSocket Protocol / Alexey Melnikov, Ian Fette. Request for Comments no. 6455. — RFC Editor, 2011. — 71 P.

[20] WebRTC 1.0: Real-time Communication Between Browsers [Электронный ресурс]. — Режим доступа : https://w3c.github.io/webrtc-pc/. — Дата доступа : 17.03.2019.

[21] HTTP vs Websockets: A performance comparison – The Feathers Flightpath [Электронный ресурс]. — Режим доступа : https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77. — Дата доступа : 18.03.2019.

ПРИЛОЖЕНИЕ А
(*обязательное)*
Вводный плакат

ПРИЛОЖЕНИЕ Б
(*обязательное)*
Схема структурная