Федеральное государственное бюджетное учреждение науки
Институт системного программирования Российской академии наук

УДК 519.685

На правах рукописи

Мутилин Вадим Сергеевич

Верификация драйверов операционной системы Linux при помощи предикатных абстракций

Специальность 05.13.11 –
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата физико-математических наук

Москва

2012

Работа выполнена в Федеральном государственном бюджетном учреждении науки Институте системного программирования РАН.

Научный руководитель: профессор, доктор физико-математических наук
Петренко Александр Константинович

Официальные оппоненты: профессор, доктор физико-математических наук
Ломазова Ирина Александровна
кандидат физико-математических наук
Романенко Сергей Анатольевич

Ведущая организация: **Федеральное государственное бюджетное учреждение науки Научно-исследовательский институт системных исследований Российской академии наук (НИИСИ РАН)**

Защита диссертации состоится « 15 » _ноября_ 2012 г. в 16 часов на заседании диссертационного совета Д.002.087.01 при Федеральном государственном бюджетном учреждении науки Институте системного программирования Российской академии наук по адресу: 109004, Москва, ул. Александра Солженицына, д.25.

С диссертацией можно ознакомиться в библиотеке Федерального государственного бюджетного учреждения науки Институте системного программирования Российской академии наук.

Автореферат разослан « » ______________ 20__ г.

Ученый секретарь
диссертационного совета /Прохоров С.П./
канд.физ.-мат.наук

Актуальность темы

В работе ставится задача верификации драйверов операционной системы (ОС) Linux. Обеспечение надежности и, в частности, информационной безопасности программных и программно-аппаратных систем является одной из главных задач современных информационных технологий. Особый вклад в решение этой задачи вносят работы по верификации системного программного обеспечения, в первую очередь операционных систем. Верификация драйверов ОС Linux является критически важной задачей, так как:

- Корректность драйверов является необходимым условием обеспечения надежности и безопасности систем, поскольку драйверы работают с тем же уровнем привилегий, что и остальное ядро.
- Драйверы ОС Linux, входящие в ядро, это большой, стремительно растущий класс программных систем. На данный момент суммарный объем драйверов составляет более 9 миллионов строк исходного кода. За последний год он увеличился на полмиллиона строк.

При разработке драйверов ОС Linux применяются различные подходы к обеспечению качества, в частности, тестирование (динамический анализ) и статический анализ. Важным преимуществом статического анализа по сравнению с динамическим является то, что его можно проводить даже в условиях, когда в распоряжении нет оборудования, которым должен управлять тот или иной драйвер.

Для драйверов, которые входят в состав ядра, обеспечение качества проводится силами разработчиков ядра. По данным Linux Foundation, на сегодняшний день в разработке каждой версии ядра, выходящей раз в 2-3 месяца, участвуют более 1000 человек, представляющих более 200 различных организаций. Несмотря на такое большое количество разработчиков, значительное число изменений в уже выпущенных и новых версиях ядра связано с исправлением ошибок в драйверах. Так, например, анализ изменений в драйверах стабильных версий ядра за год разработки с 26.10.10 по 26.10.11

показал, что порядка 80% изменений являются исправлениями ошибок. Происходит это в силу того, что обеспечивать надежность драйверов ОС Linux затруднительно ввиду огромного количества достаточно сложного исходного кода, который должен удовлетворять большому числу разнообразных правил корректности, начиная от общих правил, которым должны подчиняться все программы на Си, и заканчивая специфичными правилами, которые говорят о том, как драйверы должны использовать интерфейс ядра.

Методы общецелевого статического анализа позволяют обнаруживать нарушения общих правил, таких как разыменование нулевых указателей, выход за границу массива. Однако, помимо них остается значительная часть специфичных для ОС Linux ошибок взаимодействия драйвера с интерфейсом ядра. По данным анализа изменений в драйверах стабильных версий ядра за год разработки с 26.10.10 по 26.10.11 количество специфичных ошибок составляет более половины от всех ошибок, являющихся нарушениями правил корректности.

Перспективным направлением является разработка высокоточных инструментов статической верификации при помощи предикатных абстракций. Примерами являются инструменты BLAST, CPAchecker, SLAM. За счет построения предикатных абстракций они позволяют показать не только наличие ошибок, но и их отсутствие для заданного правила.
Ключевым свойством высокоточных инструментов для проверки специфических правил является возможность итеративного уточнения абстракции по методу CEGAR (Counter Example Guided Abstraction-Refinement), что позволяет подстраивать абстракцию как под произвольное правило корректности, так и для конкретного драйвера. Это выгодно отличает такие инструменты от общецелевых, в которых настройка под правило корректности требует реализации соответствующей функциональности в инструменте статического анализа.

Применение инструментов высокоточного статического анализа к драйверам ОС Linux имеет хорошие предпосылки. Высокоточные методы в

настоящее время имеют ограничения по размеру анализируемого кода (до 50-100 тыс. строк). В случае драйверов это ограничение приемлемо. Размер драйверов, входящих в состав ядра ОС Linux, не превышает 50 тысяч строк, а в среднем составляет порядка 2-3 тыс. строк кода. Кроме того, важно, что большинство драйверов публикуется вместе с исходным кодом, который является необходимым для большинства инструментов статического анализа.

Для применения высокоточных инструментов требуется подготовка драйвера к верификации. Во-первых, требуется подготовить специальное окружение драйвера, которое должно описывать возможности воздействия на него с учетом ограничений, накладываемых на взаимодействия сердцевины ядра ОС с драйверами данного типа. Во-вторых, требуется формально описать правила корректности в виде, удобном для сведения задачи верификации к доказательству достижимости точки программе. При решении этих задач необходимо учитывать специфику ядра ОС Linux. Одна из главных особенностей состоит в том, что интерфейсы между драйвером и ядром ОС Linux постоянно меняются. Меняются правила взаимодействия, появляются новые, меняется окружение драйвера. Поэтому правила, модели окружения и другие части системы верификации должны быть устроены так, чтобы обеспечивать простоту развития, адаптации к текущему состоянию ядра ОС, правилам взаимодействия между драйверами и ОС. На сегодняшний день не существует методов верификации, учитывающих данные особенности.

Таким образом, задача верификации драйверов ОС Linux является актуальной, для ее решения требуется разработка нового метода верификации.

Цель и задачи работы

Цель работы – разработка метода верификации драйверов устройств операционных систем для проверки выполнения правил корректного взаимодействия драйверов с ядром операционной системы.

Для достижения цели работы были поставлены следующие задачи:

1. Провести анализ существующих методов верификации драйверов;

2. Провести анализ ошибок в драйверах ОС Linux, приводящих к некорректному взаимодействую с ядром ОС;
3. Разработать метод верификации и архитектуру системы верификации драйверов ОС Linux при помощи предикатных абстракций, обеспечивающий:
   - верификацию драйверов в условиях непрерывного развития ядра;
   - возможности расширения (конфигурируемости) системы верификации драйверов за счет пополнения набора правил корректности и набора инструментов верификации.
4. Разработать систему верификации, реализующую метод;
5. Оценить реализацию метода на практике, дать оценку области применимости метода.

Научная новизна работы

Научной новизной обладают следующие результаты работы:

1. Метод построения моделей окружения драйверов устройств ОС Linux;
2. Метод построения конфигурируемой системы верификации;
3. Математическое доказательство адекватности предложенного метода формализации правил корректности в рамках заданного класса правил работы со структурами обработчиков событий;
4. Методы оптимизации предикатной абстракции в BLAST.

Практическая значимость

На основе разработанного метода была создана система верификации драйверов ОС Linux LDV (Linux Driver Verification). Система LDV позволяет находить нарушения правил корректности взаимодействия с ядром ОС для драйверов, входящих в поставку ядер ОС Linux с версии 2.6.31 по 3.4. По состоянию на 25.09.2012 было найдено более 60 ошибок, которые признаны

разработчиками ядра и уже исправлены или будут исправлены в последующих версиях.

Результаты работы будут полезны для автоматизации сертификационных процессов для компьютерного обеспечения, которые существуют у многих поставщиков дистрибутивов ОС Linux и ОС, построенных на основе ядра Linux. Также результаты могут быть использованы для проверки качества драйверов ОС Linux, для создания инструментов верификации других видов программного обеспечения, критичного по безопасности. Результаты работы могут быть использованы для сравнения характеристик различных инструментов статического анализа кода.

Доклады и публикации

По теме диссертации автором опубликовано 14 работ (из них 5 в изданиях из перечня ВАК), список которых приведен в конце автореферата.

Основные положения работы докладывались на следующих конференциях и семинарах:

- Весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Санкт-Петербург, 2008 г.);
- Общероссийская научно-техническая конференция «Методы и технические средства обеспечения безопасности информации» (г. Санкт-Петербург, 2008 г.);
- Международный семинар «Принципы и технические средства сертификации свободного программного обеспечения» (OpenCert: International Workshop on Foundations and Techniques for Open Source Software Certification, г. Милан, 2008 г.);
- Конференция разработчиков свободных программ на Протве (г. Обнинск, 2009 г.);

- Международная конференция памяти академика А.П. Ершова «Перспективы систем информатики» (PSI: Perspectives of System informatics, г. Новосибирск, 2011 г.);
- Международная конференция по инструментам и алгоритмам создания и анализа систем (TACAS: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, г. Таллин, 2012 г.);
- Семинар «День свободного программного и аппаратного обеспечения» (г. Москва, 2012 г.);
- Семинар Института системного программирования РАН (г. Москва, 2012 г.).

Структура и объем диссертации

Работа состоит из введения, семи глав, заключения и списка литературы (55 наименований). Основной текст диссертации (без приложений и списка литературы) занимает 115 страниц.

Краткое содержание диссертации

Во введении обосновывается актуальность темы работы, определяются ее цели и задачи, раскрывается практическая значимость.

В первой главе приводится анализ работ в области верификации драйверов операционных систем.

Описывается специфика задачи верификации драйверов операционных систем. Драйвер – это программа, которая работает в привилегированном режиме (в ОС Windows, Linux и многих других) и имеет событийный интерфейс, т.е. не только она обращается к функциям-интерфейсам ядра ОС, но и само ядро при наступлении определенных событий обращается к драйверам для выполнения соответствующей обработки. Дисциплина взаимодействия

драйверов и ядра операционной системы задается общей логикой функционирования операционной системы. Учет этой логики важен при верификации драйверов. Например, существуют ограничения на порядок прихода событий и, соответственно, на порядок вызова функций-обработчиков драйверов. В частности, без учета данного порядка зачастую нельзя знать, в какой момент необходимые ресурсы уже инициализированы и их можно использовать, так как ресурсы могут инициализироваться в одном из обработчиков, а использоваться в другом.

Анализируются техники и инструменты, которые используются для верификации драйверов устройств, и указывается их область применимости. Рассматриваются техники тестирования (динамической верификации), общецелевые техники статического анализа, специализированные системы верификации драйверов.

Инструменты тестирования (динамической верификации). Данные, необходимые для анализа, в этих инструментах собираются во время исполнения кода драйвера, а поэтому инструменты требуют наличия оборудования, для которого предназначен драйвер, или эмуляции этого оборудования. С другой стороны, исходный код анализируемых драйверов обычно не требуется. Это позволяет анализировать драйверы с закрытым исходным кодом. Инструменты динамической верификации обычно дают меньше ложных срабатываний, чем системы статического анализа, однако проверяют только один путь исполнения в данный момент времени. Инструменты статического анализа, напротив, одновременно проверяют многие (или даже все) пути исполнения в коде драйвера, но ложные срабатывания в них более вероятны, чем при использовании динамического анализа.

Общецелевые инструменты статического анализа. Эти инструменты крайне полезны для выявления типовых ошибок, общих для языка программирования, для которого они предназначены.

Инструмент Sparse развивается внутри ядра Linux и находит такие ошибки как, например, использование указателей из неподходящего адресного

пространства и другие. В проверках инструмент опирается на аннотации в исходном коде ядра с использованием расширений GCC. Известно, что с помощью инструмента Sparse было найдено не менее 140 ошибок в ядрах версий 2.6.25, 2.6.29, 2.6.30.

Инструмент Coverity находит такие типы ошибок, как наличие недостижимого кода, разыменование нулевых указателей, использование до сравнения, переполнение буфера, утечки ресурсов, небезопасное использование возвращаемых значений, несовпадение размеров типа и выделяемой памяти, чтение не инициализированных переменных и использование памяти после освобождения. Существуют адаптации данных типов ошибок под различные интерфейсы, например, в ядре Linux в качестве функции выделения ресурсов рассматривается функция выделения памяти под структуру urb подсистемы usb.

Инструмент применялся к ядру ОС Linux в промежутке между 2006 и 2009 годами, в рамках контракта с Департаментом безопасности США. В работах приводятся сведения о том, что из всех предупреждений было отобрано 2,125. Все эти предупреждения были отправлены разработчикам ядра. Однако лишь 61% из них было принято в обработку. Сведения о количестве ложных предупреждений и исправленных ошибках не приводятся.

Описываются также и другие общецелевые инструменты статического анализа, такие как Klocwork Insight, PREfast, Svace.

Все общецелевые статические анализаторы направлены на обнаружение общих ошибок, типичных для большинства Си программ. Однако они не учитывают специфику анализа драйверов, так как не обладают знанием о логике взаимодействия драйвера и ядра ОС. По этой причине они не могут выявить ошибки нарушения такой логики или правил взаимодействия драйверов с ядром ОС.

Системы верификации драйверов операционных систем. Им присущи схожие черты. Все они предназначены для поиска ошибок связанных с нарушением правил взаимодействия драйверов с ядром ОС. В связи с этим, требуется подготовка специального окружения драйвера, что позволяет, в

частности, учитывать ограничения на порядок вызовов обработчика драйвера. Для верификации используются инструменты статической верификации, позволяющие проверять достижимость произвольной точки программы.

Наиболее развитой системой верификации драйверов операционных систем является система Microsoft SDV, позволяющая проводить статическую верификацию драйверов операционной системы Microsoft Windows. Она используется в процессе сертификации драйверов и включена в состав Microsoft Windows Driver Developer Kit, начиная с 2006 года. Система Microsoft SDV наглядно демонстрирует возможность применения верификации реальных программ.

В системе SDV для создания окружения от пользователя требуется вручную аннотировать исходный код драйверов, указав в нем роли каждой из функций обработчиков. Проверяемые правила корректности формализуются с помощью языка SLIC, в котором связь с исходным кодом драйвера задается с помощью аспектно-ориентированных конструкций, перехватывающих вызовы функций ядра. В настоящее время уже выделен набор из примерно 200 правил, а в исследовательской версии была добавлена возможность добавления новых правил. В систему интегрированы инструменты статической верификации SLAM и Yogi. В виду того, что интерфейс между драйвером и ОС Windows изменяется крайне редко, при разработке архитектуры системы не требовалось учитывать возможность постоянного развития интерфейса между драйверами и ядром ОС.

Существуют также системы верификации драйверов ядра ОС Linux: Avinux, разработка университета города Тюбинген (Германия) и DDVerify, разработка университета Оксфорд (Англия). Разработка систем была нацелена во многом на апробацию новых возможностей инструментов верификации и не учитывала ряд особенностей, необходимых для промышленного использования, в настоящее их разработка не ведется. Например, в системе Avinux требуется ручное задание последовательности вызовов обработчиков драйвера, и не поддерживаются драйверы, состоящие из нескольких файлов. DDVerify требует

серьезной переработки собственного процесса сборки, заголовочных файлов и модели ядра для каждой новой версии ядра, что заметно усложняет сопровождение системы в условиях большого количества изменений в ядре ОС Linux.

Верификация драйверов устройств ОС Linux обладает рядом особенностей, которые необходимо учитывать. Одна из главных особенностей состоит в том, что интерфейсы между драйвером и ядром ОС Linux постоянно меняются. Это является принципиальной позицией разработчиков ядра ОС Linux, отмеченной в документации к ядру. Вызвано это тем, что разработчики не ограничивают себя в возможностях усовершенствования интерфейсов взаимодействия, например для ускорения работы системы. Таким образом, ядро Linux постоянно развивается, меняются правила взаимодействия, появляются новые, меняется окружение драйвера. Поэтому правила, модели окружения и другие части системы верификации должны быть устроены так, чтобы обеспечивать простоту развития, адаптации к текущему состоянию ядра ОС, правилам взаимодействия между драйверами и ОС. Эти особенности не позволяют использовать архитектуру системы Microsoft SDV для верификации драйверов ОС Linux.

Время жизни программных инструментов в мире свободного программного обеспечения прогнозировать трудно. Поэтому важно, чтобы система была открыта к изменениям, в частности, была готова интегрировать в себя новые инструменты верификации взамен устаревших или лишившихся поддержки.

Делается вывод, что требуется разработка нового метода и архитектуры системы верификации, учитывающих особенности верификации драйверов ОС Linux.

Вторая глава описывает новый метод верификации драйверов.

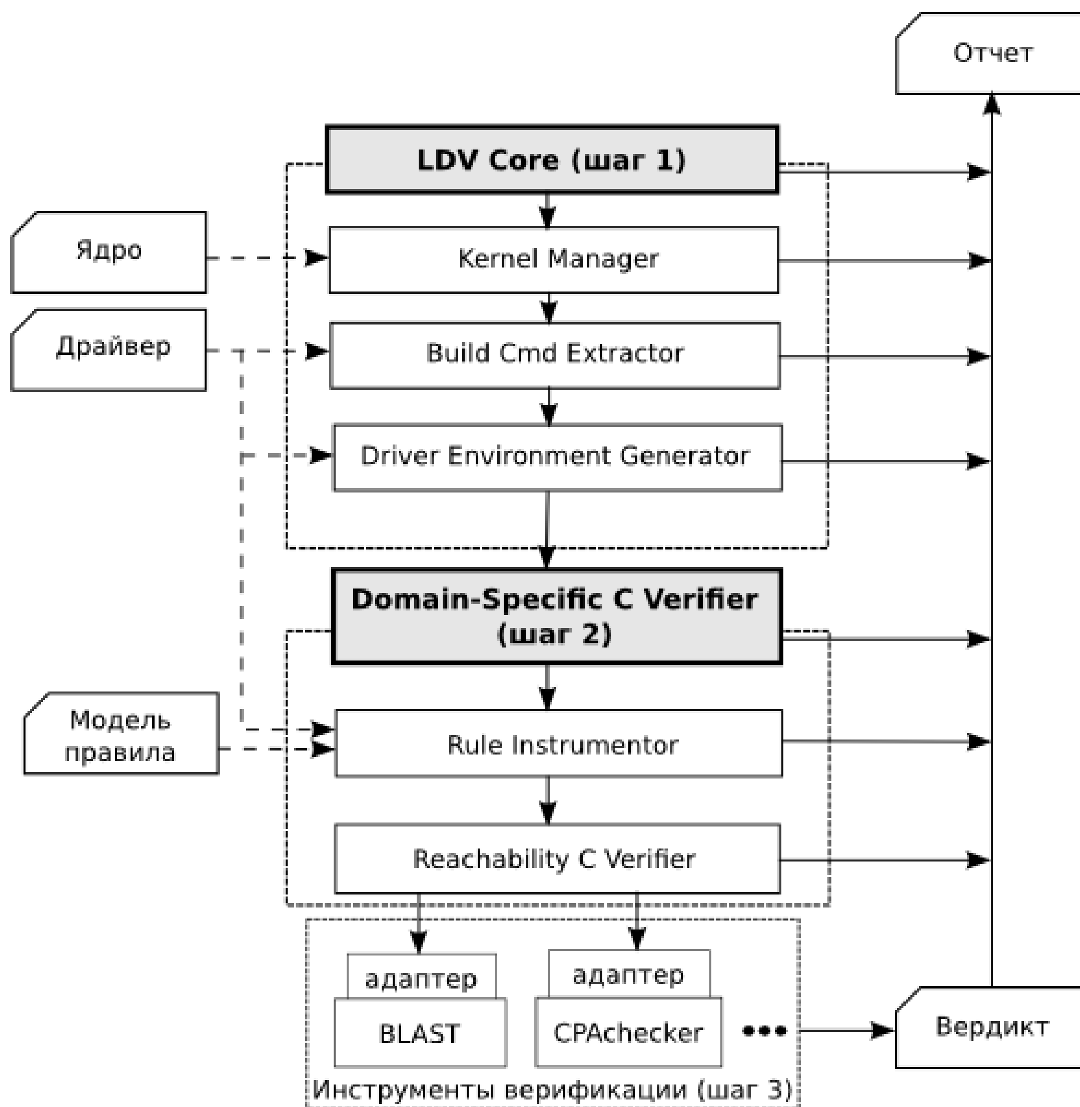

Рис. 1. Компоненты архитектуры и шаги метода верификации

Предлагаемая архитектура системы верификации разработана с учетом особенностей задачи верификации драйверов ОС Linux, рассмотренных выше. Компоненты архитектуры осуществляют действия, соответствующие шагам метода.

Архитектура схематично изображена на Рис. 1. Компоненты архитектуры изображены в центре. Стрелки указывают порядок, в котором соответствующие компоненты обрабатывают входные данные. Слева изображены входные данные. Справа показан порядок формирования отчета по результатам верификации.

Компонент LDV-Core (шаг 1) осуществляет подготовку ядра ОС Linux

(компонент Kernel Manager), извлечение исходного кода драйверов (Build Cmd Extractor) и генерацию модели окружения драйвера (Driver Environment Generator).

Компонент Domain Specific C Verifier (шаг 2) уже не должен учитывать специфику конструкции драйвера, поданный ему на вход программный код может быть и произвольной программой на языке программирования Си. Данный компонент принимает программу на языке программирования Си и передает ее компоненту Rule Instrumentor, чтобы тот встроил в нее код проверок указанных правил корректности. Затем Domain Specific C Verifier, используя Reachibility C Verifier, передает код конкретному инструменту верификации.

Инструмент статической верификации (шаг 3) выносит вердикт. По ходу этого процесса вердикт дополняется отчетами о работе других компонентов, после чего формируется финальный отчет о проверке всего задания, который затем анализируется.

На первом шаге требуется извлечь информацию об исходных кодах драйвера, которые необходимо анализировать. Метод позволяет анализировать как драйверы поставляющиеся отдельно от ядра, так и драйверы, входящие в состав ядра ОС Linux.

Для извлечения исходного кода драйвера осуществляется встраивание в процесс сборки драйвера. Команды компиляции и линковки, используемые при сборке драйвера, подменяются на специальные заглушки, собирающие данную информацию в файл *потока команд*. Далее происходит запуск сборки драйвера или драйверов входящих в состав ядра. В процессе сборки в поток команд для каждой команды сохраняются пути к входным и выходным файлам, а также многочисленные опции этой команды.

В случае, когда анализируются драйверы, входящие в состав ядра, дополнительно к извлечению команд осуществляется их разбиение на отдельные файлы потока команд по одному для каждого драйвера, так чтобы их можно было анализировать независимо.

Затем осуществляется генерация окружения драйвера, моделирующего

воздействия на драйвер, осуществляемые операционной системой с учетом ограничений, накладываемых на данные взаимодействия. Создание синтезируемого окружения позволяет, с одной стороны, построить замыкание программного кода драйвера (инструменты верификации требуют наличия замкнутого кода) и, другой стороны, нацелить инструменты верификации на поиск ошибок характерных для данного вида драйверов. *Метод генерации окружения целевого драйвера* подробно описан в третьей главе. Метод позволяет осуществлять генерацию моделей окружения полностью автоматически на основе конфигурации типов драйверов, которая описывает ограничения на взаимодействия между драйвером и сердцевиной ядра. Таким образом, конфигурацию требуется изменять, только если меняются ограничения на взаимодействия. Причем, данное изменение осуществляется сразу для всех драйверов данного типа. Это позволяет обеспечивать актуальность системы верификации в условиях непрерывного развития ядра.

В результате этого шага файл потока команд пополняется командами сборки модели окружения, а также данными о точках входа для каждой из сгенерированных моделей окружения.

На втором шаге осуществляется связывание формализованных в виде моделей правил корректности с исходным кодом проверяемого драйвера для последующей верификации с помощью одного из инструментов статической верификации.

Для построения моделей правил корректности используется подход аспектно-ориентированного программирования (АОП). Правило формализуется в аспекте, специальном отдельном модуле. *Аспект* состоит из набора так называемых *рекомендаций*, каждая из которых состоит из множества *точек соединения* – точек в программе, к которым осуществляется привязка, и *тела рекомендации*. В теле указываются действия, которые должны быть выполнены для точек соединения. Также в рекомендации указывается, должны ли эти действия быть выполнены *до* (англ. before), *после* (англ. after) или *вместо* (англ. around) выполнения точки соединения программы. Тело рекомендации

записывается с помощью инструкций языка программирования Си, на котором пишутся драйверы. АОП предоставляет средства, позволяющие использовать в теле рекомендации информацию о соответствующей точке соединения, например, имя вызываемой функции, типы ее аргументов и т.п.

Для упрощения сопровождения в условиях непрерывного развития ядра написание аспектов осуществляется так, что точки соединения отделены от модели, проверяющей правило, таким образом, что при изменении названий функций, атрибутов, параметров, не влияющих на модель, требуется изменять только точки соединения, но модель при этом остается неизменной. Если же меняется семантика функций, к которым осуществляется привязка, то изменять требуется только соответствующие модели.

Инструментирование исходного кода драйвера на основе аспектов осуществляется специальным инструментом автоматически. Так как инструментированный исходный код должен обрабатываться инструментами статического анализа кода, на выходе после инструментирования создается программа на Си, которая представляет собой препроцессированный исходный код драйвера, дополненный описанием модели соответствующего правила.

На третьем шаге запускаются инструменты статической верификации Собственно инструменты и анализ результатов верификации описаны в четвертой главе.

Сначала осуществляется преобразование из внутреннего представления в виде потока команд в представление конкретного верификатора. Подающийся на вход поток команд уже содержит файлы для проверки со сгенерированными моделями окружения и описаниями моделей правил.

Подключение инструментов статической верификации выполняется при помощи специальных *адаптеров*. Каждый адаптер состоит из следующих блоков: получение исходного кода драйверов, запуск инструмента статического анализа кода повышенной точности и обработка получаемых результатов. В работе описываются средства получения исходного кода драйвера, подготовки аргументов и обработки результатов.

Описывается гибкий и расширяемый *общий формат трасс ошибок*, который похож на формат трасс ошибок инструмента статической верификации CPAchecker. Данный формат предполагает наличие ряда строк, каждая из которых содержит набор опциональных атрибутов, например, положение соответствующей конструкции в исходном коде, тип конструкции, непосредственно сама конструкция и т.д. Атрибуты должны записываться в формальном виде. Важно заметить, что для удобства анализа трасс ошибок и определения, является ли срабатывание инструмента ложным или нет, было предложено показывать соответствующий исходный код драйверов и ядра одновременно при просмотре трассы ошибок. Для определения взаимосвязи между трассой и соответствующим исходным кодом используются ссылки на положение в коде конструкций, которые встречаются в трассе.

Третья глава описывает метод генерации окружения целевого драйвера.

Как уже упоминалось, для работы статических верификаторов драйвер нужно заключить в некоторое "окружение" с тем, чтобы программный код был замкнутым. Такое окружение будем называть "синтезируемым окружением". К синтезируемому окружению предъявляются две группы требований. Первая — связана с тем, что оно должно воспроизводить те же сценарии взаимодействия с драйвером, что и реальное окружение драйвера в операционной системе. Вторая группа требований определяет ограничения на конструкцию (структуру) синтезируемого окружения, обусловленную возможностями современных средств статической верификации.

Предлагаемый метод построения состоит из двух шагов. Сначала строится логическая модель реального окружения (с учетом специфики каждого вида драйверов). Эта модель учитывает многопоточность окружения и асинхронность взаимодействия драйвера с окружением. Для описания модели используется π-исчисление (The Polyadic π -Calculus: a Tutorial, Robin Milner, October 1991). Затем строится Си-программа, которая соответствует данной модели, и при этом является однопоточной и отвечает ограничениям

инструментов верификации.

Поскольку трансформация многопоточной модели в однопоточную программу не является тривиальной, возникает необходимость доказать корректность построения синтезируемого окружения.

Дается определение эквивалентности между моделью в π-исчислении и программой на языке Си, в котором учитываются трассы, содержащие только общие действия для процессов драйвера и окружения с ограничением на переключения при выполнении функций-обработчиков драйвера.

В диссертации сформулирована и доказана теорема о том, что для каждого драйвера логическая модель окружения эквивалентна синтезированному окружению для того же самого драйвера.

Четвертая глава описывает методы оптимизации анализа при помощи предикатных абстракций.

Описываются предложенные диссертантом методы оптимизации, использованные в инструменте BLAST 2.7, в частности:

- Настройка автоматического управления памятью в языке OCaml позволила уменьшить время, затрачиваемое на операции с памятью, что дало 20% прироста в количестве посещённых точек проверяемой программы на затраченную единицу памяти.
- В синтаксическом анализаторе CIL, используемым инструментом BLAST, были внесены исправления, которые позволили успешно анализировать до 98% драйверов ядра (измерено на версии 2.6.37), и лишь около 2% приводят к ошибкам разбора. До этого BLAST 2.5 не мог разобрать ни одного драйвера в ядре 2.6.31.
- Была улучшена работа с решателями, получающими на вход формулы в формате SMTLIB, за счет снижения накладных расходов при конвертации формул из внутреннего представления.
- Оптимизирован алгоритм фильтрации формулы пути, который осуществляет удаление частей формулы, не влияющих на

получение интерполянта. Время работы алгоритма стало O(log N), вместо O(N).

- Был реализован новый алгоритм анализа алиасов, который позволяет анализировать программы, использующие указатели на базовые типы и структуры.

Данные улучшения позволили инструменту BLAST 2.7 занять первое место в категории DeviceDrivers64 и третье в категории DeviceDrivers32 на международных соревнованиях по верификации программ SV-COMP 2012.

Пятая глава описывает систему верификации драйверов ОС Linux (Linux Driver Verification), являющуюся реализацией метода верификации.

Разработанная система верификации обладает следующими важными особенностями:

- Система интегрирована с процессом сборки ядра, поэтому вся необходимая информация о составе и настройках сборки драйверов извлекается автоматически.
- Генерация окружения осуществляется полностью автоматически на основе иерархии шаблонов, покрывающей все типы драйверов.
- Система позволяет добавлять новые правила корректности с помощью аспектно-ориентированного расширения языка программирования Си.
- Система поддерживает встраивание внешних инструментов статической верификации с помощью написания адаптеров.
- Для анализа трасс ошибок и сравнительного анализа результатов система предоставляет специальные компоненты с Веб-интерфейсом.

Шестая глава описывает методику выявления и классификации правил корректности.

В первой части описывается способ выбора источника. Описываются различные источники. Достаточно много информации можно найти в

документации, в том числе, входящей в состав ядра и в исходный код ядра и драйверов, а также в литературе по разработке драйверов и ядра ОС Linux.

Однако, в этих источниках документированы далеко не все особенности различных подсистем ядра и типов драйверов. Поскольку ядро развивается очень стремительно, данные источники не всегда поддерживаются в актуальном состоянии.

В качестве еще одного источника для выявления правил можно рассмотреть список рассылки ядра Linux (от англ. Linux Kernel Mailing List, LKML). В данном списке рассылки обсуждаются различные актуальные вопросы разработки ядра, в том числе вопросы, связанные с исправлением ошибок. На основании этих обсуждений можно выявить множество общих и специфичных правил, но анализ сообщений в LKML достаточно трудоемок, поскольку сообщений много и они содержат большое количество информации, причем не только технической. Кроме того, много ошибок обсуждается в других списках рассылок, на форумах, системах отслеживания ошибок и т.д.

Правила корректности можно выявлять на основе анализа **журнала изменений**, содержащего изменения, вносимые в драйверы ОС Linux. Данный источник является достаточно актуальным, поскольку рано или поздно среди изменений появляются все важные исправления ошибок, которые обсуждаются в различных списках рассылок, на конференциях, форумах и т.д. Изменения в драйверах ОС Linux проходят достаточно тщательную предварительную процедуру согласования и проверки, в том числе, у экспертов в соответствующих подсистемах ядра.

Каждое изменение содержит в себе достаточно подробную информацию, включающую сведения об авторе, краткое название изменения, подробное описание (возможно, со ссылками на соответствующие обсуждения), изменение исходного кода драйверов и/или ядра ОС Linux и различную вспомогательную информацию.

В заключение в качестве основного источника для выявления правил предлагается использовать журнал изменений.

Во второй части описываются шаги методики анализа журнала изменений.

Первый шаг методики анализа изменения в драйвере ОС Linux заключается в том, чтобы определить, является ли оно исправлением ошибки или оно каким-либо образом расширяет функциональность драйвера (например, добавляет поддержку новых устройств).

Далее из рассмотрения исключаются ошибки, исправления которых связано с требованиями к функциональности конкретного драйвера или ошибки взаимодействия драйвера с конкретным оборудованием. В том случае, если ошибка связана с неправильным использованием специализированных для конкретного драйвера или группы драйверов констант, условий и вычислений, ее следует исключить из дальнейшего рассмотрения, так как для нее нельзя сформулировать правило корректности.

Основным шагом предлагаемой методики анализа изменений в драйверах ОС Linux является классификация ошибок, для которых можно сформулировать общие или специфические правила.

Данный шаг требует осмысления причины, которая привела к ошибке. Дело в том, что с первого взгляда может показаться, что ошибка проявилась вследствие нарушения некого общего правила. На самом деле, причина ошибки может крыться в особенностях параллельного выполнения либо в некорректном использовании интерфейса сердцевины ядра. Кроме того, не всегда легко определить, какую именно общую или специфичную ошибку исправляет рассматриваемое изменение. На данном шаге помимо анализа описания и изменения в исходном коде предполагается более тщательный анализ взаимодействий различных подсистем ядра и драйвера.

Важно отметить, что список классов не известен заранее и может быть определен только непосредственно по ходу самого анализа. Предполагается, что, в первую очередь, классы позволят различить общие и специфичные ошибки в драйверах.

В третьей части описывается разработанная классификация ошибок

взаимодействия драйверов с ядром ОС Linux, разработанная в рамках исследований. Были проанализированы изменения в драйверах стабильных версий ядра с 2.6.35 по 3.0 за время с 26.10.10 по 26.10.11. Среди исправлений выявлялись ошибки, для которых можно сформулировать правило корректности взаимодействия драйвера с сердцевиной ядра. Таких ошибок было выделено 176, что составляет 14,9% от общего количества ошибок (1182).

Список классов ошибок представлен в Табл. 2.

Табл. 2. Описание классов ошибок взаимодействия в драйверах ОС Linux

№	**Класс**	**Краткое описание класса**
1	specific:resource	Утечки ресурсов и использование после освобождения
2	specific:check_params	Нарушение ограничений на входные параметры
3	specific:context	Вызовы функций в недопустимых контекстах
4	specific:uninit	Некорректная инициализация специфичных объектов
5	specific:lock	Некорректное использование механизмов синхронизации
6	specific:style	Нарушение стиля оформления драйверов ядра ОС Linux
7	specific:net	Некорректное использование интерфейса сетевой подсистемы
8	specific:usb	Некорректное использование интерфейса USB подсистемы
9	specific:check_ret_val	Отсутствие проверок возвращаемых значений
10	specific:dma	Некорректное использование интерфейса подсистемы прямого доступа к памяти
11	specific:device	Некорректное использование интерфейса общей модели драйвера
12	specific:misc	Разное

В диссертационной работе показано, что метод позволяет обнаруживать нарушения для всех классов правил взаимодействия драйверов с ядром операционной системы.

Седьмая глава описывает практические результаты.

Система верификации выносит один из трех вердиктов: Safe, Unsafe и Unknown. Вердикт Safe означает отсутствие нарушений проверяемого правила корректности. Вердикт Unsafe говорит об обнаружении нарушения правила и сопровождается трассой ошибки, которая показывает путь выполнения программы, приводящий к ошибочному состоянию. Вердикт Unknown означает, что система по тем или иным причинам (например, нехватка памяти или времени) не смогла найти однозначного ответа на поставленный вопрос.

Следует отметить, что количество истинных вердиктов Unsafe по мере того, как разработчики их исправляют, в каждой версии неизменно снижается в результате остаются только неисправленные истинные вердикты Unsafe, например, ошибки в неподдерживаемых драйверах или ошибки, для которых на данный момент не имеется приемлемого исправления.

Система верификации предоставляет возможность сохранять результаты анализа вердиктов Unsafe в базе знаний. База знаний позволяет автоматически отображать количество истинных вердиктов Unsafe (True unsafe), ложных вердиктов Unsafe (False unsafe) и непроанализированных или не до конца изученных вердиктов Unsafe (Unknown unsafe).

При верификации новых версий ложные и истинные вердикты Unsafe, проанализированные в предыдущих версиях и сохраненные в базе знаний, автоматически отображаются в новой версии. Совпадение трасс ошибок определяется с точностью до заданного критерия (трасса ошибки целиком, дерево вызовов, стек вызовов).

В качестве иллюстрации приводится фрагмент из отчета по результатам верификации драйверов ядра ОС Linux linux-3.4 в мае 2012 года на моделях правил 32_7, 39_7, 43_1a, при ограничении на память 6Gb, ограничении на анализ одного драйвера 15 мин (Рис. 2).

Rule	Total	Safe	Unsafe	Unknown	Verdicts		
					True	False	?
32_7	3441	2998	44	399	4	40	-
39_7	3441	3002	40	399	11	29	-
43_1a	3441	2965	10	466	2	7	1

Рис. 2. Результаты верификации драйверов ядра ОС Linux версии 3.4.

В данном запуске системы верификации было проверено 3441 модуля драйверов. Количество положительных вердиктов Safe или Unsafe от 86,5% до 88,4% (в среднем 87,5%). В среднем получено от 11,6% до 13,5% вердиктов Unknown (12,5% в среднем). Количество ложных вердиктов Unsafe относительно количества проанализированных драйверов от 0,2% до 1,1%, что составляет (в среднем 0,7%). Для примера, по сравнению с версией 3.2 ядра ОС Linux, в ядре 3.4 для правила 32_7 появилось лишь 3 новых вердикта Unsafe.

Делается вывод, что количество ложных вердиктов Unsafe невелико - при анализе очередных новых версий ядра появляются лишь единицы новых ложных Unsafe вердиктов.

В **заключении** перечисляются основные результаты работы.

Основные результаты работы

Основные научные и практические результаты, полученные в диссертационной работе и выносимые на защиту, состоят в следующем:

1. Разработан метод верификации драйверов устройств операционных систем для проверки выполнения правил корректного взаимодействия драйверов с ядром операционной системы;
2. Разработан метод построения моделей окружения драйверов устройств ОС Linux;
3. Разработан метод построения конфигурируемой системы верификации,

обеспечивающий возможность расширения системы за счет пополнения набора правил корректности и набора инструментов верификации;

4. Разработаны методы оптимизации предикатной абстракции в инструменте BLAST;
5. На основе предложенных методов разработана система верификации драйверов ОС Linux.

Система верификации драйверов ОС Linux разработана в рамках проектов отдела Технологий программирования Института системного программирования РАН при непосредственном участии автора в качестве руководителя и участника разработки основных компонентов системы верификации.

Работы автора по теме диссертации

1. Мутилин В.С., Новиков Е.М., Хорошилов А.В. Анализ типовых ошибок в драйверах операционной системы Linux //Сборник трудов ИСП РАН, том 22, 2012 г.
2. М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.В. Хорошилов. Обзор инструментов статической верификации Си программ в применении к драйверам устройств операционной системы Linux //Сборник трудов ИСП РАН, том 22, 2012 г.
3. В.С. Мутилин, М.У. Мандрыкин. Интерполяция формул с кванторами в CSIsat на основе инстанцирования //Сборник трудов ИСП РАН, том 22, 2012 г.
4. М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.В. Хорошилов, П.Е. Швед. Использование драйверов устройств операционной системы Linux для сравнения инструментов статической верификации, Программирование №5, 2012.
5. П.Е. Швед, В.С. Мутилин, М.У. Мандрыкин . Опыт развития инструмента статической верификации BLAST. //ПРОГРАММИРОВАНИЕ, №3, с. 24–

35, 2012.

6. V. Mutilin, M. Mandrykin. Instantiation-Based Interpolation for Quantified Formulae in CSIsat. //SYRCOSE 2012
7. P. Shved, M. Mandrykin, V. Mutilin. Predicate Analysis with Blast 2.7. In Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 525–527. Springer, Heidelberg, 2012.
8. Shved P., Mutilin V., Mandrykin M. Static verfication “under the hood”: Implementation details and improvements of BLAST // Proceedings of SYRCoSE. — Vol. 1. — 2011. — Pp. 54–60.
9. Khoroshilov A., Mutilin V., Novikov E., Shved P., Strakh A. Towards an open framework for C verification tools benchmarking //Proceedings of PSI. — 2011. — Pp. 82–91.
10. A.V.Khoroshilov, V.V.Mutilin, A.K.Petrenko, V.A.Zakharov. Establishing Linux Driver Verification Process PSI 2009 Perspectives of System informatics 2009 LNCS, Vol. 5947 165-176.
11. В.Мутилин, А.Хорошилов. База правил для верификации драйверов Linux // Протва 2009 Тезисы докладов VI Конференции разработчиков свободных программ на Протве, Обнинск, 2009.
12. В.С. Мутилин, А.В. Хорошилов, В.А. Захаров. Верификация безопасности драйверов ОС Linux. Материалы XVII Общероссийской научно-технической конференции «Методы и технические средства обеспечения безопасности информации» 106-112 2008 .
13. A. Khoroshilov, V. Mutilin, V. Shcherbina, O. Strikov, S. Vinogradov, and V. Zakharov. How to Cook an Automated System for Linux Driver Verification. SYRCoSE'2008 2nd Spring Young Researchers' Colloquium on Software Engineering, Volume 2 11-14.
14. A. Khoroshilov, V. Mutilin. Formal Methods for Open Source Components Certification OpenCert 2008 2nd International Workshop on Foundations and Techniques for Open Source Software Certification 52-63 Milan 2008.