Об интеграции формальных методов в задачах верификации операционных систем[1]

[1,2,3,4]*А. К. Петренко <petrenko@ispras.ru>*
[1,2,4]*В. В. Кулямин <kuliamin@ispras.ru>*
[1,2,3,4]*А. В. Хорошилов <khoroshilov@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.
[3]*Московский физико-технический институт (государственный университет),*
141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[4] *Национальный исследовательский университет «Высшая школа экономики»*
101000, Россия, Москва, ул. Мясницкая, д.20

Аннотация. В данной работе ставится задача разработки методов качественной верификации операционных систем, перспективным подходом к решению которой является интеграция различных техник верификации. Результатом успешного решения этой задачи может считаться комбинация методов верификации, которая позволяет проверить всю систему в целом и при этом более тщательно верифицировать наиболее важные ее компоненты и функции, используя для этого более строгие и формальные подходы. На основе опыта ИСП РАН, полученного при выполнении многочисленных проектов по верификации операционных систем с помощью различных методов выявлены артефакты разработки, являющиеся удобными кандидатами на роль точек сопряжения методов статической и динамической верификации ядра операционной системы на основе формальных спецификаций.

Выделение общих (разделяемых) артефактов предлагается провести на основе рассмотрения типовых задач верификации. Типовые задач, встречающиеся при использовании различных техник верификации, дают основу для интеграции техник и процессов верификации. К таким типовым задачам относятся: определение программного контракта модулей (функций), построение модели окружения, построение пути, демонстрирующего ошибку, увязка уровней абстракции и оценка полноты верификации. В наибольшей степени на роль артефактов представляющих собой точки интеграции претендуют контрактные спецификации, модели окружения и метрики (измерения) полноты верификации.

[1] Работа поддержана грантом РФФИ 14-01-00484.

Ключевые слова. Дедуктивная верификация, проверка моделей, тестирование на основе моделей, операционная система, интеграция методов верификации, программные контракты.

DOI: 10.15514/ISPRAS-2015-27(5)-10

Для цитирования: Петренко А.К., Кулямин В.В., Хорошилов А.В. Об интеграции формальных методов в задачах верификации операционных систем. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 175-190. DOI: 10.15514/ISPRAS-2015-27(5)-10.

1. Введение

Размеры и сложность современного программного обеспечения (ПО) обычно не позволяют провести верификацию достаточно высокого качества. Это связано как с большим количеством разнородных функций и компонентов, входящих в типичную программную систему, так и с практическими ограничениями самих применяемых технологий верификации. Подобные ограничения в первую очередь касаются формальных методов верификации, дающих самые надежные гарантии корректности верифицированных систем. Причины таких ограничений разнообразны – это могут быть низкая выразительность или, наоборот, чрезмерная усложненность формального аппарата, лежащего в основе метода, ограниченность конкретных инструментов, а также, что является на практике важнейшим фактором, сложность, высокая стоимость и длительность выполнения самих формальных методов верификации.

Интеграция различных формальных методов верификации является перспективным подходом к решению проблемы преодоления таких ограничений. На практике бывает важно совместить результаты верификации различных компонентов сложной системы, проведенные разными командами с помощью различных техник. Другая часто встречающая задача - найти приемлемую комбинацию методов и технологий верификации, которые позволяют проверить всю систему в целом и, одновременно, сфокусировать усилия на отдельных, наиболее критических компонентах и функциях, используя в их верификации наиболее строгие, но и более сложные и дорогостоящие методы. В данной статье мы рассматриваем вторую задачу применительно к ядру операционной системы как к системе с повышенными требованиями по показателям надежности и корректности.

2. Достижения в направлении интеграции методов верификации

При решении задач интеграции методов верификации просматривается как минимум два аспекта. Первый – это повторное использование (reuse) артефактов верификации и возможностей инструментов разных технологий, которые применялись к одним и тем компонентам проверяемой системы. Второй – это стыковка процессов верификации взаимодействующих компонентов, каждый из которых в отдельности верифицировался при

помощи разных методов и инструментов, и увязывание получаемых при этом артефактов.

Проблемы интеграции методов верификации и, в частности, формальных методов рассматриваются исследовательским сообществом достаточно давно - международная конференция по интеграции формальных методов проводится уже с 1999 года [1]. Пожалуй, наибольшим продвижением в этом направлении можно назвать широкое использование инструментов поиска решений систем логических утверждений, так называемых решателей (solvers), таких как Z3 [2,3], CVC4 [4,5] и др., в рамках различных технологий верификации. Они успешно применяются как компоненты многих инструментов динамической верификации (например, SpecExpolrer [6], Pex [7], SAGE [8]), инструментов верификации программ с извлечением их моделей (software model checking, например, BLAST [9], Static Driver Verifier [10], CPAchecker [11]), а также инструментов дедуктивной верификации (например, Frama-C [12], Rodin [13]).

Авторы тоже обращались к теме интеграции методов верификации в нескольких работах, например в [14] и [15]. В данной статье рассматривается еще одно направление интеграции, которому ранее не уделялось должного внимания.

Анализируя публикации на тему интеграции формальных методов, можно прийти к выводу, что наиболее активно разрабатывается тема интеграции подходов, построенных на различных формализмах моделирования поведения. Задача собственно стыковки самих методов рассматривается на примерах частных приложений. Мы считаем, что есть еще одно измерение интеграции, на которое следует обратить внимание, – это комплексная верификация некоторого важного класса систем, обладающих специфическими особенностями архитектуры и имеющих некоторую общую область характеристик функциональности, определяющую их назначение и являющуюся по существу объектом анализа в ходе верификации. В рамках данной статьи в качестве такого класса систем мы рассматриваем операционные системы, в более узком смысле – ядра операционных систем.

3. Терминология и особенности предметной области

В этом разделе мы кратко перечислим особенности операционных систем (ОС) как объекта верификации и так же кратко рассмотрим терминологию, используемую в рамках методов и техник верификации, чтобы используемые далее термины понимались однозначно.

Имеется две основные схемы верификации ядра ОС.

- *Модульная верификация* (unit verification). Все ядро разбивается на модули со строго заданными интерфейсами, при помощи которых осуществляется взаимодействие модулей между собой, с пользователями и устройствами, включая сетевые устройства. Интерфейсы специфицируются, и каждый модуль проверяется (верифицируется) на соответствие заданным спецификациям.

Результат верификации при условии осуществления всех проверок гарантирует формальную корректность модуля в предположении, что спецификации сами по себе корректны и полны.

- *Системная верификация* (system verification). Ядро ОС рассматривается как «черный ящик» (иногда как «серый ящик», т.е., используется определенная информация о его структуре и потоках данных внутри ядра), определяются только видимые извне интерфейсы ядра, в частности, системные вызовы и информационные потоки на уровне аппаратных интерфейсов (процессор, средства хранения информации, сетевые карты и др.). В случае «серого ящика» можно рассматривать интерфейсы на уровне драйверов и/или на уровне аппаратных абстракций (HAL, Hardware Abstraction Layer). Задается спецификация правильного видимого извне поведения ОС. Ставится задача верификации соответствия кода ОС данной спецификации. Можно сказать, что статически такая задача верификации может решаться только для небольших ОС (размер ее кода в 10 тыс. строк – практический предел, более реалистично – 1-2 тыс. строк), также важно условие полной спецификации аппаратуры. Как система в целом, так и отдельные модули могут рассматриваться с большим или меньшим количеством деталей. Как правило, имеет смысл сначала строить модель модуля/системы, верифицировать ее, а потом переходить к верификации либо более детальной модели, получаемой с помощью уточнения абстрактной, либо собственно реализации. При наличии некоторых условий верификация более детальных моделей может опираться на верификационные артефакты, полученные при верификации более абстрактных моделей, что снижает общую трудоемкость работ. Однако это не всегда удается обеспечить.

Основные причины возникновения специфических проблем при верификации операционных систем следующие.

- Выполнение различных функций ОС управляется *внешними событиями* (ОС являются примером event driven system). Этими событиями являются, в первую очередь, обращения к системным вызовам из приложений, а также обращения к обработчикам прерываний со стороны аппаратного обеспечения. По сути, отсутствует некоторый четкий сценарий обращений к различным функциям ядра ОС, хотя обычно существуют некоторые плохо формализованные ограничения на последовательности вызовов и возможные параллельные обращения к ним (а также повторные обращения из обработчиков прерываний, сработавших во время их выполнения, и пр.).
- Большинство ОС должны поддерживать *параллелизм*, т.е., наличие одновременно многих процессов и потоков управления в

приложениях, разделение ресурсов между процессами и параллельные обращения из разных потоков к различным функциям ядра. В том числе, должна учитываться реальная параллельность работы центрального процессора, различных контроллеров и внешних устройств.

- Современные ОС поддерживают *многоядерность*, т.е., должны распределять выполнение процессов и потоков на несколько управляющих устройств процессора, работающих параллельно, с учетом тех ограничений, которые накладываются на корректное распределение задач.
- Многие ОС имеют требования по соблюдению *ограничений реального времени*, связанных как с ограничениями по времени выполнения административных функций самой ОС, так с мониторингом временных ограничений на работу приложений.
- ОС имеют определенные требования к *доступности и надежности* работы, обуславливающие необходимость обработки разнообразных отказов оборудования.

Основной вывод, который важен для нас, состоит в том, провести качественную верификацию операционной системы на практике одним методом невозможно. По этой причине операционные системы являются хорошим полигоном для экспериментов по применению перспективных методов верификации и, в частности, интегрированных подходов к верификации.

Дадим краткие определения используемых терминов. Более полное описание терминов и понятий верификации можно найти в обзоре [16].

- *Верификация* проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов правилам и стандартам разработки. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО.
- *Спецификация* в широком толковании это достаточно подробное, полное и точное описание. В контексте формальных методов под спецификацией понимается формальное описание требований к программной системе, ее поведению, ее функциональных и нефункциональных свойств (например, требований к устойчивости, несмотря на ошибки оборудования, или к производительности и пропускной способности). Поскольку спецификация практически всегда охватывает лишь часть свойств и характеристик реальной системы, часто говорят о спецификации определенной модели, или просто о модели. В таком контексте термины «спецификация» и

«модель» обычно используются как синонимы.

Объектами спецификации могут быть поведение (функциональные свойства), нефункциональные свойства (защищенность, надежность, производительность, переносимость, совместимость и пр.), язык описания чего-либо (в этом случае выделяют синтаксис, статическую и динамическую семантика, прагматические правила и шаблоны), протокол взаимодействия (в телекоммуникационных и интерфейсных стандартах) и др.

- *Реализация* – это проверяемая система, являющаяся объектом верификации. Рассматриваются различные формы представления реализации (например, исходный код, двоичный код, наблюдаемое поведение и внутренняя динамика работы кода).
- *Статический анализ* – анализ свойств системы, выполняемый (обычно специализированными инструментами) без ее реальной работы за счет ее моделирования на основе анализа/интерпретации исходного кода или др. артефактов разработки; *динамический анализ* – анализ свойств системы, выполняемый в ходе ее реальной работы и с использованием наблюдения за этой работой. Во многих случаях эти виды анализа используются совместно и разделяются с трудом (например, для анализа используется симуляция работы кода системы в специализированной виртуальной машине), тогда говорят о *статико-динамическом анализе*.
- Виды динамического анализа.
    - *Тестирование* – оценка соответствия поведения реализации требованиям по ее работе в рамках заранее выбранного (сгенерированного) набора сценариев или ситуаций, который получен для обеспечения определенного критерия полноты (связанного с покрытием всех типичных ситуаций, задаваемых кодом и/или описанных в требованиях).
    - *Мониторинг* (run-time verification или пассивное тестирование, passive testing) – оценка соответствия поведения реализации требованиям по ее работе в рамках реальной эксплуатации или некоторой похожей деятельности (в которой возникающие ситуации выбираются, не для достижения покрытия этих ситуаций, а для достижения каких-то с точки зрения верификации целей). Сам анализ выявленного поведения может выполняться в ходе работы системы (online) или после ее завершения (offline или post-mortem).
- *Средства наблюдения и управления* – доступные средства контроля поведения реализации в ходе ее реальной работы или симуляции, используемые обычно при динамическом анализе, например, внешний интерфейс, обратный интерфейс (набор вызываемых ей

операций в других компонентах и системах), различные журналы (трассы), снимки памяти и др.

- *Дедуктивная верификация* или *дедуктивный анализ* (theorem proving) - проверка того, что набор утверждений, представляющий спецификацию, формально выводится из формального представления реализации и определенного набора гипотез о поведении окружения системы.
- Генератор верификационных задач (verification conditions generator) – инструмент, анализирующий верифицируемую программу и строящий набор утверждений, формально описывающих необходимые и достаточные условия выполнения некоторого заданного свойства программы. Используется в рамках дедуктивной верификации.
- Инструмент автоматизированного доказательства (prover, proof assistant) – инструмент для автоматизированного или полностью автоматического поиска доказательства задаваемых ему на входе утверждений.
- *Решатель* (solver) – инструмент для поиска решений системы утверждений. Такая система может задавать верифицируемое свойство или, наоборот, нарушение проверяемого свойства.
- *Проверка моделей* (model checking, software model checking) – проверка соответствия поведения автоматной формальной модели, описывающей реализацию, формальной модели (обычно декларативной), описывающей требования к системе. Как правило, модель реализации несколько упрощенно представляет проверяемую систему, что облегчает верификацию. В рамках классической проверки моделей построение модели реализации является дополнительной работой, выполняемой вручную, из-за чего модель может оказаться неадекватной целевой системе. Проверка программных моделей (software model checking) предполагает автоматическое построение модели реализации на основе кода системы, при этом используются эвристики, которые также могут привести к нарушению адекватности. По этой причине в некоторых методах в процесс построения модели включаются итеративные проверка адекватности и уточнение моделей, не прошедших проверку (например, в методе CEGAR [17,18] для этого выполняется попытка сгенерировать контрпример, демонстрирующий, что ошибочная ситуация достижима).
- *Предикатная абстракция* – использование в качестве модели состояния системы некоторого набора булевских характеристик (характеризующих как набор текущих внутренних данных, так и состояние выполнения программного кода, включая возможный параллелизм).

- Точность анализа, характеристики и аспекты качества верификации.
  - Важными характеристиками техники верификации являются возможность пропуска ошибок определенного типа или же гарантированность их выявления, а также возможность (или невозможность) ложных сообщений об ошибках определенного типа. Во многих случаях выявление абсолютно всех ошибок без ложных сообщений требует недоступных на практике трудозатрат и времени, приходится, в лучшем случае, выбирать между полнотой обнаружения ошибок и отсутствием ложных сообщений, минимизируя другой показатель.
  - При проведении верификации важно выявить и зафиксировать полный и точный набор предположений, в которых она выполняется. Неадекватность этих условий реальности (также как и наличие неявных предположений) может привести к необнаружению ошибок.
  - В случае сложных систем полная верификация может оказаться недостижимой. В этой ситуации возможно использование метрик полноты верификации, связанных с долей проверенных утверждений по отношению ко всем и проверенной части реализации по отношению ко всему объему ее кода. Для адекватного учета полноты также важно, чтобы наиболее существенные ограничения и элементы кода проходили верификацию как можно раньше.
  - Технические результаты верификации также могут задаваться по-разному, в зависимости от различных предположений об окружении и применяемых методов. Это могут быть либо подозрительные/опасные ситуации, возможные дефекты, которые нужно еще подвергнуть дополнительному анализу на предмет присутствия в них ошибки, либо «настоящие ошибки», для которых в результате верификации становится известен способ демонстрации нарушения требований при работе системы.

4. Возможные сценарии интеграции верификационных техник

В данном исследовании мы основываемся на опыте, полученном в многочисленных проектах по верификации программного и аппаратного обеспечения за последние 20 лет работы в отделе Технологий программирования ИСП РАН. Коротко перечислим эти проекты. Начнем с тех, которые базировались на использовании одного вида формальных методов, а затем перейдем к проектам, где проводились эксперименты с синтезом различных подходов.

Первые проекты полностью опирались на тестирование на основе формальных моделей (model based testing, MBT). В них были использованы следующие формализмы для представления моделей [19].

- Программные контракты интерфейсов (interface contracts) в форме ограничений - пред- и постусловия операций или событий и инварианты типов данных.
- Расширенные автоматы (extended finite state machines, EFSM) в так называемой неявной форме для описания моделей тестовых сценариев.
- Расширенные грамматики для моделирования деревьев разбора, абстрактных синтаксических деревьев с ограничениями, а также для представления частей документов с семантическими зависимостями.

В этих проектах было проведено промышленное тестирование нескольких операционных систем, включая ОС Linux и отечественные операционные системы реального времени, оптимизирующего компилятора компании Intel, некоторых компонентов информационной инфраструктуры компании крупного телекоммуникационного оператора, моделей отечественных микропроцессоров.

В рамках данных проектов решался следующий типовой набор задач, связанных с построением соответствующих компонентов верифицирующей (в данном случае тестовой) системы.

- Построение спецификаций программного контракта.
- Построение отображения реализационных событий и данных в модельные (так называемые адаптеры).
- Построение модели тестового сценария, задающего логику генерации последовательности тестовых воздействий.
- Генерация тестовых данных
- Построение тестовых оракулов – компонентов, которые на основе поведения тестируемой системы дают оценку (вердикт) его корректности.
- Определение метрик полноты тестирования и построение системы сбора и анализа достигнутого тестового покрытия.

Следующая группа проектов использовала технику проверки программных моделей (software model checking). Верифицируемой системой в данном случае являлись компоненты ядра ОС Linux, в первую очередь, драйвера устройств [20]. В рамках этих проектов типовой состав работ был таков.

- Формализация правил взаимодействия компонентов программной системы (их можно рассматривать как некоторый вид программных контрактов) [21].
- Генерация модели окружения, описывающей возможные сценарии вызовов функций проверяемой системы [22].
- Генерация верификационных задач.

- Решение верификационных задач.
- Сбор информации о проанализированных путях в графе потока управления проверяемой системы.

Еще одна группа проектов связана с использованием дедуктивной верификации [23]. Проверяемой системой в данном случае является система защиты информации (СЗИ) ОС AstraLinux. В рамках этой группы проектов проводилась верификация модели требований к СЗИ, в качестве такой модели была использована мандатная сущностно-ролевая модель доступа и потоков данных (МРОСЛ ДП) [24], а затем нужно было провести верификацию кода модуля безопасности ядра ОС Linux (Linux Security Module, LSM).

В рамках этих проектов решались следующие задачи.

- Построение формальных спецификаций операций и инвариантов типов данных для интерфейсов модели и модуля LSM в форме программных контрактов.
- Формализация инвариантов информационной безопасности, определяющих требования обеспечения защищенности проверяемой системы.
- Собственно верификация модели и модуля LSM.

Заметим, что здесь работы с моделью и модулем LSM рассматриваются совместно, так как между этими объектами верификации много концептуально близкого – они описывают механизмы обеспечения контроля доступа к информации. Однако модель и модуль реализованы на разных языках (Event-B и C) и верифицируются разными инструментами (Rodin и Frama-C/Jessie/Why3). Кроме того, в части интерфейсов модель и модуль не совпадают.

Помимо различия в используемых языках и инструментах, задачи верификации модели СЗИ и модуля LSM существенно отличаются друг от друга тем, что модель представляет на соответствующем уровне абстракции целевую систему в целом, а LSM лишь часть ядра ОС, ответственную за выполнение проверок при обращениях к ней других компонентов ядра. Это означает необходимость, помимо решения других задач, описать формально предположения о поведении окружении модуля (т.е., ядра и всех приложений в целом, по отношению к вызовам операций модуля), без таких предположений невозможно верифицировать сам модуль. Отметим, что корректность таких предположений тоже надо проверять, то есть верифицировать.

В выделенных списках работ в каждой из трех рассмотренных групп можно отметить работы, нацеленные на решение одних и тех же задач, хотя пока, в силу технических различий между инструментами верификации, их приходится решать по-разному. Мы разместили схожие задачи в приведенной ниже таблице. Всего получилось пять групп.

Группы задач	Тестирование на основе моделей	Проверка моделей	Дедуктивная верификация
Определение программного контракта	Построение спецификаций программного контракта (тесно связано с построением тестовых оракулов)	Формализация правил взаимодействия компонентов программной системы	Построение формальных спецификаций операций и типов данных в форме программных контрактов
Построение модели окружения	Построение модели тестового сценария	Генерация модели окружения	Формальное описание предположений о поведении окружения проверяемого модуля
Построения пути, демонстрирующего ошибку	Выполнение обхода сценарного автомата, генерация тестовых данных для всех воздействий	Генерация и решение верификационных задач	(эта задача не ставилась)
Увязка уровней абстракции	Построение отображения реализационных событий и данных в модельные	(эта задача не ставилась)	(эта задача не ставилась)
Оценка полноты верификации	Определение метрик полноты тестирования и построение системы сбора и анализа достигнутого тестового покрытия	Сбор информации о проанализированных путях в графе потока управления системы	(эта задача не ставилась)

Таблица 1. Соответствие между задачами в разных методах верификации.

На основании сказанного можно выделить узлы, в которых различные методы и реализующие их инструменты имеют возможные точки сопряжения. Примерами таких точек могут быть следующие артефакты:

- Контрактные спецификации поведения.
- Модель окружения проверяемой системы, состоящего из компонентов, которые вызывают операции системы, вызываются ею или взаимодействуют с ней каким-либо другим способом.
- Метрики полноты верификации.

5. Заключение

В этой статье мы формулируем задачу создания методов верификации ядер операционных систем, дающих достаточно высокие гарантии надежности и пригодных для практического применения. Результирующие методы, скорее всего, будут интегрировать элементы различных техник верификации, и, будучи применимы ко всей системе в целом, позволят тщательнее проверять наиболее важные ее функции и компоненты, используя при этом самые строгие формальные подходы.

Проанализировав имеющийся в ИСП РАН опыт проведения верификации операционных систем с помощью различных методов – тестирования на основе моделей, проверки программных моделей, дедуктивного анализа, мы выделили набор артефактов разработки, которые могут быть наиболее удобными точками интеграции различных верификационных техник. Развитие проведенного исследования будет связано с созданием техники верификации, интегрирующей статический и динамический анализ при помощи использования выявленных артефактов – модели поведения, модели окружения и модели ситуаций.

Литература

[1]. Proceedings of the 1st International Conference on Integrated Formal Methods. Edited by K. Araki, A. Galloway, K. Taguchi, York, 28-29 June 1999. Springer-Verlag, 1999. ISBN:1-85233-107-0.

[2]. L. De Moura, N. Bjørner. Z3: an Efficient SMT Solver. Proceedings of TACAS'2008. LNCS 4963:337-340, Springer-Verlag, 2008.

[3]. http://github.com/Z3Prover/z3

[4]. C. Barret, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli. CVC4. Proceedings of CAV'2011, LNCS 6806:171-177, Springer, 2011.

[5]. http://cvc4.cs.nyuedu/web

[6]. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, L. Nachmanson. Model-based Testing of Object-oriented Reactive Systems with SpecExplorer. Proceedings of FORTEST'2008, LNCS 4949:39-76, Springer-Verlag, 2008.

[7]. N. Tillmann, J. de Halleux. Pex – White Box Test Generation for .NET. Proceedings of TAP'2008, LNCS 4966:134-153, Springer-Verlag, 2008.

[8]. P. Godefroid. Random Testing for Security: Blackbox vs. Whitebox Fuzzing. Proceedings of 2-nd International Workshop on Random Testing, p. 1-1, ACM, 2007.

[9]. T. A. Henzinger, R. Jhala, R. Majumdar, G. Sutre. Software Verification with BLAST. Proceedings of SPIN'2003, Model Checking Software, LNCS 2648:235-239, Springer-Verlag, 2003.

[10]. T. Ball, B. Cook, V. Levin, S. K. rajamani. SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. Proceedings of IFM'2004, LNCS 2999:1-20, Springer-Verlag, 2004.

[11]. D. Beyer, M. E. Keremoglu. CPAchecker: a Tool for Configurable Software Verification. Proceedings of CAV'2011, LNCS 6806:184-190, Springer, 2011.

[12]. G. Canet, P. Cuoq, B. Monate. A Value Analysis for C Programs. Proceedings of SCAM'2009, p. 123-124, IEEE, 2009.

[13]. J.-R. Abrial, M. Butler, S. Hallerstede, L. Voisin. An Open Extensible Tool Environment for Event-B. Proceedings of ICFEM'2006, Formal Methods and Software Engineering, LNCS 4260:588-605, Springer-Verlag, 2006.
[14]. А. К. Петренко. Унификация в автоматизации тестирования. Позиция UniTESK. Труды Института системного программирования РАН 14(1):7-22, 2008.
[15]. В. В. Кулямин. Интеграция методов верификации программных систем. Программирование, 35(4):41-55, 2009.
[16]. В. В. Кулямин. Методы верификации программного обеспечения. Статья-победитель конкурса обзорно-аналитических статей по направлению «Информационно-телекоммуникационные системы», 2008 (http://www.ispras.ru/publications/2008/methods_of_software_verification/).
[17]. E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement. Proceedings of CAV'2000, LNCS 1855:154-169, Springer-Verlag, 2000.
[18]. М. У. Мандрыкин, В. С. Мутилин, А. В. Хорошилов. Введение в метод CEGAR — уточнение абстракции по контрпримерам. Труды Института системного программирования РАН 24: 219-292, 2013. DOI: 10.15514/ISPRAS-2013-24-12
[19]. В. В. Кулямин, А. К. Петренко. Развитие подхода к разработке тестов UniTESK. Труды Института системного программирования РАН 26(1):9-26, 2014. DOI: 10.15514/ISPRAS-2014-26(1)-1
[20]. И.С. Захаров, М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.К. Петренко, А.В. Хорошилов. "Конфигурируемая система статической верификации модулей ядра операционных систем". Труды Института Системного Программирования РАН, Том 26(2):5-42, 2014. DOI: 10.15514/ISPRAS-2014-26(2)-1
[21]. Е. М. Новиков. Развитие метода контрактных спецификаций для верификации модулей ядра операционной системы Linux. Диссертация на соискание ученой степени к.ф.-м.н., Москва, 2013.
[22]. И.С. Захаров, В.С. Мутилин, А.В. Хорошилов. "Моделирование окружения с использованием шаблонов для статической верификации модулей ядра Linux". // Программирование, Том 41, 2015, №3, сс. 3-19.
[23]. P. N. Devyanin, A. V. Khoroshilov, V. V. Kuliamin, A. K. Petrenko, and I. V. Shchepetkov. Using Refinement in Formal Development of OS Security Model. Proceedings of PSI'2015.
[24]. П. Н. Девянин. Ролевая ДП-модель управления доступом и информационными потоками в операционных системах семейства Linux. ПДМ, 2012, № 1, 69–90.

Integration Points of Operating System Verification Techniques[2]

[1,2,3,4]A. K. Petrenko <petrenko@ispras.ru>
[1,2,4]V. V. Kuliamin <kuliamin@ispras.ru>
[1,2,3,4]A. V. Khoroshilov <khoroshilov@ispras.ru>
[1]Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia.

[2] Supported by RFBR grant # 14-01-00484.

[2] Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] Moscow Institute of Physics and Technology (State University)
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] National Research University Higher School of Economics (HSE)
11 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. In this work the problem of high quality verification techniques applicable for operating systems is formulated. A perspective approach to solve this problem is integration of various verification methods. The solution technique can be considered successful if it allows to check the whole operating system and to verify in more accurate way the most important functions and components of the system, using more strict and formal methods for it. Based on the ISP RAS experience in operating system verification projects conducted using various verification techniques we determine development artifacts, that can be suitable integration point candidates for integration of formal specification based static and dynamic verification techniques for operating systems.
The article proposes to define common (shared) artifacts based on the consideration of typical verification tasks. Typical problems encountered in the use of different techniques of verification, provide the basis for integration techniques and verification processes. These types of problems are: the definition of a software contracts of modules (functions); the construction of the model environment; building the path, demonstrating the error; bringing levels of abstraction and assessment of the completeness of the verification. To the greatest extent on the role of artifacts representing the points of integration claim: software contract specifications, environment models and measurement of verification completeness.

Keywords. Theorem proving, software model checking, model-based testing, operating system, integration of verification methods, software contracts.

DOI: 10.15514/ISPRAS-2015-27(5)-10

For citation: Petrenko A. K., Kuliamin V. V., Khoroshilov A. V. Integration Points of Operating System Verification Techniques. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 175-190 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-10.

References

[1]. Proceedings of the 1st International Conference on Integrated Formal Methods. Edited by K. Araki, A. Galloway, K. Taguchi, York, 28-29 June 1999. Springer-Verlag, 1999. ISBN:1-85233-107-0.
[2]. L. De Moura, N. Bjørner. Z3: an Efficient SMT Solver. Proceedings of TACAS'2008. LNCS 4963:337-340, Springer-Verlag, 2008.
[3]. http://github.com/Z3Prover/z3
[4]. C. Barret, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli. CVC4. Proceedings of CAV'2011, LNCS 6806:171-177, Springer, 2011.
[5]. http://cvc4.cs.nyuedu/web
[6]. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, L. Nachmanson. Model-based Testing of Object-oriented Reactive Systems with SpecExplorer. Proceedings of FORTEST'2008, LNCS 4949:39-76, Springer-Verlag, 2008.

[7]. N. Tillmann, J. de Halleux. Pex – White Box Test Generation for .NET. Proceedings of TAP'2008, LNCS 4966:134-153, Springer-Verlag, 2008.
[8]. P. Godefroid. Random Testing for Security: Blackbox vs. Whitebox Fuzzing. Proceedings of 2-nd International Workshop on Random Testing, p. 1-1, ACM, 2007.
[9]. T. A. Henzinger, R. Jhala, R. Majumdar, G. Sutre. Software Verification with BLAST. Proceedings of SPIN'2003, Model Checking Software, LNCS 2648:235-239, Springer-Verlag, 2003.
[10]. T. Ball, B. Cook, V. Levin, S. K. rajamani. SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. Proceedings of IFM'2004, LNCS 2999:1-20, Springer-Verlag, 2004.
[11]. D. Beyer, M. E. Keremoglu. CPAchecker: a Tool for Configurable Software Verification. Proceedings of CAV'2011, LNCS 6806:184-190, Springer, 2011.
[12]. G. Canet, P. Cuoq, B. Monate. A Value Analysis for C Programs. Proceedings of SCAM'2009, p. 123-124, IEEE, 2009.
[13]. J.-R. Abrial, M. Butler, S. Hallerstede, L. Voisin. An Open Extensible Tool Environment for Event-B. Proceedings of ICFEM'2006, Formal Methods and Software Engineering, LNCS 4260:588-605, Springer-Verlag, 2006.
[14]. A. K. Petrenko. Unifikatsia v avtomotizatsii testirovania. Positsia UniTESK [Testing Unification and Automation. UniTESK Viewpoint]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 14(1):7-22, 2008 (in Russian).
[15]. V. V. Kuliamin. Integration of Software Verification Methods. Programming and Computer Software, 35(4):41-55, 2009. DOI 10.1134/S0361768809040057
[16]. V. V. Kuliamin. Software Verification Methods. Paper on Analytical Survey Contest on Information and Communication Systems, 2008 (http://www.ispras.ru/publications/2008/methods_of_software_verification/, in Russian).
[17]. E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement. Proceedings of CAV'2000, LNCS 1855:154-169, Springer-Verlag, 2000.
[18]. Khoroshilov A.V., Mandrykin M. U., MutilinV. S. Vvedenie v metod CEGAR — utochnenie abstrakcii po kontrprimeram [Introduction to CEGAR — Counter-Example Guided Abstraction Refinement]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 24, pp. 219-292, 2013 (in Russian). DOI: 10.15514/ISPRAS-2013-24-12
[19]. V. V. Kuliamin, A. K. Petrenko. Razvitie Podkhoda k Razrabotke Testov UniTESK [Evolution of UniTESK Test Development Approach]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 26(1):9-26, 2014 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-1
[20]. A. V. Khoroshilov, M. U. Mandrykin, V. S. Mutilin, E. M. Novikov, A. K. Petrenko, and I. S. Zakharov. Configurable toolset for static verification of operating systems kernel modules. Programming and Computer Software 41(1):49-64, 2015. DOI 10.1134/S0361768815010065
[21]. E. M. Novikov. Development of Software Contracts for Verification of Linux Operating System Kernel Modules. PhD Thesis, Moscow, 2013 (in Russian).
[22]. A. V. Khoroshilov, V. S. Mutilin, I. S. Zakharov. Pattern-based environment modeling for static verification of Linux kernel modules. Programming and Computer Software 41(3):183-195, 2015. DOI 10.1134/S036176881503007X
[23]. P. N. Devyanin, A. V. Khoroshilov, V. V. Kuliamin, A. K. Petrenko, and I. V. Shchepetkov. Using Refinement in Formal Development of OS Security Model. Proceedings of PSI'2015.

[24]. P. N. Devyanin. The role DP-model of access and information flows control in operating systems of Linux sets. Prikladnaia i Diskretnaia Matematika (Applied and Discrete Mathematics), 2012, № 1, 69–90 (in Russian).