Санкт-Петербургский политехнический университет

Институт Компьютерных Наук и Технологий

Кафедра компьютерных систем и программных технологий

Лабораторная работа по дисциплине «Проектирование ОС и компонентов» на тему:

Средства тестирования ОС

Выполнил: студент группы № 63501/3

Дедков С.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

Санкт-Петербург

2016

# Оглавление

[Оглавление 2](#_Toc445803706)

[Цель работы 3](#_Toc445803707)

[Описание видов тестирования ОС 4](#_Toc445803708)

[Функциональное тестирование 4](#_Toc445803709)

[Тестирование обратной совместимости 5](#_Toc445803710)

[Обнаружение специфических видов ошибок 5](#_Toc445803711)

[Тестирование производительности ОС 6](#_Toc445803712)

[Средства тестирования ОС 7](#_Toc445803713)

[LAVA (Linaro Automated Validation Architecture) 7](#_Toc445803714)

[Opentest 7](#_Toc445803715)

[Autotest 8](#_Toc445803716)

[Jenkins 8](#_Toc445803717)

[Ktest.pl 9](#_Toc445803718)

[Вывод 10](#_Toc445803719)

[Список литературы 11](#_Toc445803720)

# Цель работы

* Описание видов тестирования ОС, назначения тестирования
* Описание средств тестирования ОС

# Описание видов тестирования ОС

Виды тестирования:

* Функциональное тестирование
* Тестирование обратной совместимости
* Обнаружение специфических видов ошибок
* Тестирование производительности

Рассматривая ОС с точки зрения тестирования, необходимо учитывать следующие особенности.

ОС как программное обеспечение, работающее непосредственно с аппаратурой, обладает:

* значительным внутренним параллелизмом;
* зависимостью от аппаратуры и её конфигураций;
* внутренней активностью.

ОС как основа компьютерной системы в целом и гарант безопасности приложений должна:

* быть устойчивой к различным нестандартным ситуациям, таким как нехватка оперативной памяти, памяти жёсткого диска и т. д.;
* быть устойчивой к атакам и вредоносным действиям со стороны недоверенных приложений, сетевых контрагентов, подключаемых внешних устройств и т. д.
* не допускать утечек ресурсов с целью обеспечения продолжительного функционирования системы без перезагрузки;
* минимизировать накладные расходы на реализацию своих функций.

ОС как среда для работы прикладных программ должна обладать такими свойствами как:

* соответствие стандартам на интерфейсы ОС;
* соответствие документации на интерфейсы ОС;
* совместимость с приложениями как на уровне бинарных интерфейсов, так и на уровне исходного кода.

ОС как платформа, на которой работает тестовая система, является источником дополнительных требований к принципам построения тестовой системы:

* при обнаружении ошибок, ведущих к аварийному прекращению работы ОС, информация, полученная тестом, должна сохраняться;
* тестовая система должна вносить минимальные искажения в поведение тестируемых компонентов ОС, в том числе в их временные характеристики. (Герлиц Е.А., 2015)

## Функциональное тестирование

Одной из ожидаемых характеристик ОС является корректная реализация своей функциональности. Это означает, что реальное поведение компонентов, доступных через публичные интерфейсы ОС, должно соответствовать декларируемым. Среди публичных интерфейсов ОС можно выделить интерфейсы системных библиотек и утилит, а также интерфейсы ядра ОС. Единичным элементом интерфейса библиотеки является функция, например, open(), sin() или fprintf(). Приведённые примеры демонстрируют различные виды реализации библиотечных функций:

* open() является примером функции-обёртки, которая сама по себе мало, что делает, за исключением трансляции запроса к компоненту более низкого уровня, в данном случае, open() обращается к соответствующему системному вызову ядра ОС;
* sin(), напротив, полностью реализуется в математической библиотеке и не требует обращения к другим компонентам;
* fprintf() представляет собой промежуточный вариант, в котором существенная часть функциональности реализуется в библиотеке (форматирование строки), но присутствует также и обращение к внешним компонентам (непосредственный вывод строки).

Функциональное тестирование делится на следующие типы:

* Тестирование обработки внутренних ошибок
* Специализированные тестовые системы

## Тестирование обратной совместимости

Современные операционные системы развиваются независимо от пользовательских приложений. Как следствие, возникает необходимость переноса приложений между различными версиями одной ОС. И так как приложений гораздо больше, чем ОС, то в большинстве случаев задача обеспечения переносимости перекладывается на плечи ОС в виде требования обеспечить обратную совместимость с приложениями. Это означает, что приложение, предназначенное для старой версии ОС, должно беспроблемно работать на новой версии ОС. Обратная совместимость обычно рассматривается на одном из двух уровней: на уровне бинарных файлов или на уровне исходных кодов. В первом случае требуется, чтобы уже скомпилированное приложение можно было установить и использовать на новой версии ОС. Во втором случае речь идёт о возможности перекомпиляции исходного кода приложения под новую версию ОС без каких-либо изменений в коде. Как правило, ОС обеспечивают обратную совместимость при условии, что приложения используют только специфицированные возможности ОС. Также существуют отраслевые и международные стандарты, регламентирующие интерфейс между приложениями и ОС, который позволяет обеспечить переносимость приложений между различными ОС. Примерами таких стандартов являются POSIX и LSB. POSIX описывает интерфейс на уровне исходных кодов для семейства ОС UNIX. LSB нацелен на обеспечение переносимости на бинарном уровне между дистрибутивами ОС Linux. Среди методов тестирования обратной совместимости можно выделить структурные и семантические.

Тестирование обратной совместимости делится на следующие типы:

* Структурное тестирование обратной совместимости
* Семантическое тестирование обратной совместимости

## Обнаружение специфических видов ошибок

Ряд ошибок в программах проявляются не сразу и/или опосредованным образом, поэтому традиционные подходы функционального тестирования не эффективны для их выявления. В этом разделе мы рассмотрим два класса таких ошибок в контексте тестирования ОС: утечки ресурсов и гонки по данным. Также в этом разделе будут рассмотрены ошибки, связанные с некорректной обработкой некорректных входных данных. В качестве интересного факта следует упомянуть, что анализ исправлений ошибок в стабильных версиях ядра ОС Linux за один календарный год показал, что гонки по данным и утечки ресурсов оказались наиболее распространёнными видами исправляемых ошибок: они встречаются в 17% и 16% случаев соответственно.

Этот тип тестирования делится на следующие виды:

* Обнаружение утечек ресурсов
* Обнаружение гонок по данным
* Kernel Strider и OC2000 Data Race Detector
* Race Hound

## Тестирование производительности ОС

Тестирование производительности в инженерии программного обеспечения определяется как тестирование, которое проводится с целью определения, как 98 быстро работает вычислительная система или её часть под определённой нагрузкой. Такое тестирование может также служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов. Заметим, что в отличии от других видов тестирования, тестирование производительности обычно не ставит своей целью выявление нарушений заранее зафиксированных требований. Когда же речь идёт о производительности, такие требования зачастую отсутствуют. Это вызвано тем, что конкретные значения разнообразных показателей производительности для одной и той же программной системы сильно зависят от возможностей аппаратных платформ, на которых может выполняться эта программная система. Поэтому постановка задачи тестирования производительности сложных программных систем, к которым относятся и операционные системы, требует дополнительной конкретизации. Набор задач тестирования определяется как архитектурой программной системы и ее аппаратной платформы, так и целями тестирования.

Тестируются следующие показатели:

* время обработки прерывания;
* время создания потока управления;
* время переключения между потоками управления (переключение контекстов потоков);
* время переключения процессов (переключение контекстов процессов);
* время создания ресурсов (семафоров, очередей, мьютексов, таймеров и т. п.);
* пропускная способность системы обмена сообщениями между потоками управления и/или процессами.

# Средства тестирования ОС

## LAVA (Linaro Automated Validation Architecture)

LAVA это система непрерывной интеграции для развертывания операционных систем на физических и виртуальных аппаратных средств для выполнения тестов. Виды тестов: тестирование загрузки, тестирование начального загрузчика и тестирование на системном уровне, хотя может потребоваться дополнительное оборудование для некоторых системных тестов. Результаты отслеживаются в течение долгого времени, и данные могут быть экспортированы для дальнейшего анализа.

LAVA архитектурно состоит из двух основных компонентов: сервер и диспетчер. Сервер это Django веб приложение, которое содержит центральный планировщик, панель управления, мониторинг рабочих мест, аутентификацию и авторизацию, а также систему визуализации результата теста. Диспетчер - это подсистема, которая управляет связью с физическими устройствами тестирования.

Доступен для скачивания deb пакет. Может быть установлен на Ubuntu 14.04 следующей командой:

$ sudo apt-get install lava

LAVA предназначена для автоматизации тестирования на реальных и виртуальных системах. Linaro выполняет более 800 тестов через ~ 30 типов устройств каждую неделю.

Документация доступна на сайте validation.linaro.org.

Документация включает руководства по установке LAVA, написания тестов, администрирования LAVA инстанса и разработки кода для LAVA. (Linaro Automated Validation Architecture (LAVA), 2015)

## Opentest

OpenTest - это уже проект с открытым исходным кодом, основанный на другом проекте STAF (Software Testing Automation Framework), и являющийся частью Arago Project. Основу для исполнения тестов на целевой платформе составляет TEE (Test Execution Engine), представляющий собой сервис STAF. Сервис STAF - это обыкновенный процесс ОС, который находится в режиме ожидания и ждет команд от планировщика STAF. Планировщик посылает TEE сервису, который запущен на 8 встраиваемой системе, команду выполнить тест, после чего сервис выполняет тест и отсылает обратно сообщение о результатах в виде XML-файла. Cами тесты создаются через web интерфейс системы TestLink, после чего TestLink экспортирует тесты в XML файлы и делает запрос на исполнение теста планировщику STAF. Такой подход является универсальным, так как пользователь может реализовать что угодно в виде сервиса, однако это и создает определенные сложности. Во-первых, TEE должен быть запущен на целевой платформе и, следовательно, иметь все необходимое окружение STAF. Во-вторых, каждый отдельный тест представляет собой набор входов и выходов, являющихся строками, которые задаются через web интерфейс. Но часто выходной результат может описываться более сложным образом, чем просто строка, например, как результат выполнения некоторой команды bash. И проверка этого результата возложена на TEE, который работает на стороне целевой платформы, что подразумевает, как минимум, наличие некоторых основных утилит (sh, grep, awk, и т.д.), то есть предполагается наличие Linux или какой-то другой ОС с достаточно большими требованиями. (Opentest, 2013)

## Autotest

Autotest является основой для полностью автоматизированного тестирования. Он предназначен в первую очередь для тестирования ядра Linux, хотя может быть полезен и для многих других функций, таких как квалифицировать новое оборудование. Autotest – ПО с открытым исходным кодом с лицензией GPL. Используется и разработан рядом организаций, в том числе Google, IBM, Red Hat, и многими другими.

Autotest состоит из нескольких модулей, которые помогут сделать автономные тесты или установки полностью автоматизированной тестовой сетки.

Список модулей:

* Autotest клиент: Средство, которое выполняет тесты (реж клиента). Каждый тест - каталог (директория client/tests), представляется классом питона, который реализует минимальное количество методов. Клиент в основном предназначен для того, чтобы один разработчик локально мог выполнить тесты. Autotest клиент выполняет “клиентские управляемые файлы“, которые являются программами Python, и использовать API-интерфейс клиента.
* Autotest Сервер: Программа, которая копирует клиента к удаленным компьютерам и контролирует их выполнение. Сервер выполняет Autotest ''клиентских управляемых файлов'', которые также являются регулярные программы Python, но с более высокого уровня API, так как сервер Autotest может контролировать выполнение тестов на нескольких машинах.
* Autotest базы данных: Для тестовых сетей, нам нужен способ для хранения результатов тестирования. Для этого существует данный компонент. Эта БД используется Autotest планировщиком и фронтэндом для хранения и визуализации результатов испытаний.
* Autotest планировщик: Для тестовых сетей, нужна утилита, которая может вызвать и планировать выполнение заданий в тестовых машинах, Autotest планировщик является таковой.
* Autotest Веб-интерфейс: Веб-приложение для тестирования сетей, чей Бэкенд написана в Джанго (http://www.djangoproject.com/) и UI, написан в GWT (http://code.google.com/webtoolkit/~~HEAD=pobj), позволяет пользователям вызвать работу и визуализировать результаты испытаний
* Autotest интерфейс командной строки: Кроме того, пользователи могут также использовать Autotest CLI, написанный на Python

(Autotest: Fully automated tests under the linux platform, 2016)

## Jenkins

Jenkins – это инструмент непрерывной интеграции, написанный на Java. Запускается в контейнере сервлетов, таких как Apache Tomcat или GlassFish. Поддерживает инструментарий для работы с разными системами контроля версий, включая CVS, Subversion, Mercurial, Git и Clearcase, может собирать проекты Apache Ant и Apache Maven, а также исполнять shell-скрипты и команды Windows.

Сборка проектов может быть назначена на разные события, например, производиться по расписанию, используя механизм подобный cron, либо стартовать когда другая сборка уже собрана, либо при запросе определённого URL.

Благодаря возможности создания плагинов для Jenkins, его функциональность как инструмента для сборки проектов можно существенно расширить. Примером может быть Serenity, которая представляет собой написанную на Java библиотеку для составления метрик покрытия кода, его сложности, зависимостей в нём, соединённую с плагином Jenkins для отображения отчётов.

(Hudson, 2015)

## Ktest.pl

Ktest.pl - это скрипт на Perl в исходном дереве ядра, который читает простой конфигурационный файл для выполнения тестирования ядра. Может применятся для удаленных целевых машин. Если есть доступ к консоли удаленного сервера, есть возможность использовать ktest.pl для выполнения автоматических тестов. Пример запуска тестов:

\* git bisecting

\* config bisecting

\* randconfig

\* any .config boot/test

\* patch checking

Поддерживаемые виды тестирования – тестирование сборки, загрузки, модульное тестирование. Хотя ktest написан на Perl, не нужно ничего знать о Perl, чтобы использовать ktest. ktest.pl принимает один конфигурационный файл для его ввода и будет запускать тесты в соответствии с этим конфигурационного файла. ktest.pl был включен в основную ветку ядра Linux в директории tools/testing/ktest. Там лежит образец конфигурации, который охватывает все конфигурации, которые могут быть заданы для использования ktest.

(Ktest, 2011)

# Вывод

В данной работе рассмотрены основные виды и средства тестирования ОС. Как видно средств существует достаточно, для тестирования ОС. В освоении они не очень легки, но существует множество документации по каждому из средств. Все проекты активно развиваются и совершенствуются, позволяя проводить все более качественное тестирование ОС.

# Список литературы

*Autotest: Fully automated tests under the linux platform*. (2016). Получено из github.com: https://github.com/autotest/autotest#readme

*Hudson*. (22 апрель 2015 г.). Получено из wikipedia: https://ru.wikipedia.org/wiki/Hudson

*Ktest*. (28 октябрь 2011 г.). Получено из elinux.org: http://elinux.org/Ktest

*Linaro Automated Validation Architecture (LAVA)*. (16 сентябрь 2015 г.). Получено из wiki.linaro.org: https://wiki.linaro.org/LAVA

*Opentest*. (6 сентябрь 2013 г.). Получено из http://arago-project.org/: http://arago-project.org/wiki/index.php/Opentest

Герлиц Е.А., К. В. (2015). *Тестирование операционных систем.*