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](#_Toc445394341)

[Цель работы 3](#_Toc445394342)

[Описание видов тестирования ОС 4](#_Toc445394343)

[Функциональное тестирование 5](#_Toc445394344)

[Тестирование обратной совместимости 6](#_Toc445394345)

[Обнаружение специфических видов ошибок 7](#_Toc445394346)

[Тестирование производительности ОС 8](#_Toc445394347)

# Цель работы

* Описание видов тестирования ОС
* Описание способов 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.

# 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.