DOI: 10.22184/1992-4178.2023.229.8.88.96

Надежность и безопасность операционных систем различной архитектуры

Часть 1

С. Назаров, д. т. н.[1], А. Барсуков, к. т. н.[2]

УДК 621.3.012 | ВАК 2.2.11

Введение широкого пакета санкций привело российскую экономику к переориентации на использование отечественных программных продуктов, в том числе и операционных систем (ОС). Очевидным преимуществом отечественных ОС является то, что они разрабатываются согласно требованиям отечественных регуляторов по безопасности программных продуктов и потенциально могут обеспечить независимость от зарубежного ПО в целом. В данной статье предлагаются простые модели работы ОС различной архитектуры, основанные на теории марковских процессов, описываемых системами дифференциальных уравнений Колмогорова. Большое количество материала, рассмотренного в статье, предопределило ее разбиение на три взаимосвязанных части, публикуемых в последовательности трех номеров журнала.

Министерство цифрового развития, связи и массовых коммуникаций России по итогам опроса определило три отечественные ОС, наиболее перспективные для господдержки, включенные в реестр российского программного обеспечения: Astra Linux (ГК «Астра»), ОС «Альт» («БазАльт СПО») и «Ред ОС» («Ред Софт»). Все они базируются на Linux. Это универсальная система с монолитным ядром, которая хорошо адаптируется под разные применения для серверов, десктопов, мобильных устройств. Тем не менее, в ней много системных вызовов, а модули внутри ядра имеют такую сложную цепочку связей, что нельзя быть уверенным в надежности проверки всех взаимодействий встроенными в ядро средствами безопасности. Главный недостаток Linux в том, что после взлома монолитного ядра злоумышленник получит максимальные привилегии. Этот факт приводит к сомнениям – хороша ли архитектура ОС с монолитным ядром? В каждом ли ее применении будет обеспечена требуемая надежность и безопасность функционирования системы? В связи с этим встает вопрос, насколько надежна и безопасна ОС с монолитным ядром по сравнению с системами другой архитектуры? В статье рассматриваются три модели систем: операционная система с классическим монолитным ядром (Linux), микроядерная операционная система Э. Таненбаума (Minix3) и кибериммунная операционная система KasperskyOS. Оцениваются и сравниваются надежность и безопасность функционирования систем. Предложенный подход к построению моделей архитектур операционных систем может быть использован для начального исследования надежности функционирования конкретных архитектур операционных систем.

СВОЙСТВА, КАЧЕСТВО И ЭФФЕКТИВНОСТЬ ОС

В рамках исследования данной статьи представляется необходимым остановиться на трех основных свойствах операционных систем – надежности, безопасности и эффективности. Под надежностью операционной системы понимается ее свойство сохранять во времени в установленных пределах значения всех параметров, характеризующих способность системы выполнять требуемые функции в заданных режимах и условиях применения (ГОСТ 27.02-2015). Надежность – комплексное свойство, которое в зависимости от назначения системы и условий ее эксплуатации включает в себя свойства безотказности, долговечности, ремонтопригодности и сохраняемости, а также определенное сочетание этих свойств. В практике применения с надежностью ОС тесно связано другое

[1] Московский научно-исследовательский телевизионный институт, главный научный сотрудник, профессор.

[2] Московский научно-исследовательский телевизионный институт, заместитель начальника научно-технического отдела, с. н. с.

ее свойство – безопасность. Это состояние защищенности информационной среды системы, которое достигается наличием в структуре ОС определенных средств защиты информации или специальными архитектурными решениями, принятыми разработчиками операционной системы. Немаловажным фактором обеспечения безопасности является деятельность по предотвращению утечки защищаемой информации и защита от несанкционированных и непреднамеренных воздействий. Под эффективностью ОС и эффективностью вообще любой системы понимается степень соответствия системы своему назначению, ее техническое совершенство и экономическая целесообразность [1, 2]. На показатели эффективности ОС влияет много различных факторов, среди которых основными являются архитектура ОС, многообразие ее функций, качество программного кода, аппаратная платформа (компьютер) и др. Эти вопросы рассмотрены ниже. В работах автора [3, 4] достаточно подробно освещаются вопросы формализации понятия эффективность функционирования применительно к сложным техническим (программно-аппаратным) системам. Определены понятия: параметры системы, свойства, качество системы, показатель эффективности, критерий эффективности. Показана их связь и зависимости. Применительно к рассматриваемой в данной работе задаче эта связь и зависимость могут быть представлены в том виде, как показано на рис. 1.

Качество операционной системы представляется совокупностью свойств, характеризующих ее использование по конкретному назначению. В то же время эффективность, в соответствии с положениями современной теории эффективности, свойством системы не является. Это обусловлено следующими причинами:

- оценка эффективности (уровня эффективности) связана не только со свойствами операционной системы, но и (даже, возможно, в большей степени) со свойствами результатов ее функционирования и ресурсами, затраченными на достижение этих результатов;
- эффективность функционирования операционной системы определяется не только ее свойствами, но и способом использования системы по целевому назначению (например, режимом работы, входящими потоками данных, характеристиками и требованиями функциональных задач к ресурсам и т. п.);
- рассмотрение эффективности как свойства операционной системы вступает в противоречие с понятием ее качества.

Таким образом, понятие эффективность операционной системы следует понимать, как эффективность ее функционирования, то есть свойства ее использования по назначению (свойство операции). Эффективность ОС зависит от ее качества, которое в свою очередь определяется свойствами системы, условиями использования системы, а также свойствами надсистемы, то есть вычислительной системы в целом.

ПРИЧИНЫ СЛОЖНОСТИ, НЕНАДЕЖНОСТИ И НЕБЕЗОПАСНОСТИ ОС

С развитием теории и практики создания операционных систем сложность и объем кода операционных систем только возрастают, чего нельзя сказать о надежности, качестве и безопасности операционных систем. Причины такого положения дел обусловлены как объективными, так и субъективными факторами. Главный объективный фактор – сложность и трудоемкость разработки, желание учесть максимально возможный перечень требований пользователей. Субъективные факторы в основном связаны с конкуренцией на рынке ОС. Основные причины сложности создания надежных безопасных и эффективных операционных систем, по мнению специалистов, достаточно обозначились на практике [5–9]. Прежде всего, надо отметить, что операционные системы – это класс больших программных систем, который характеризуется широкой функциональностью, высокой трудоемкостью и сложностью, рисками и длительностью процесса разработки. Например, создание OS/360 в шестидесятые годы прошлого века потребовало 5000 человеко-лет.

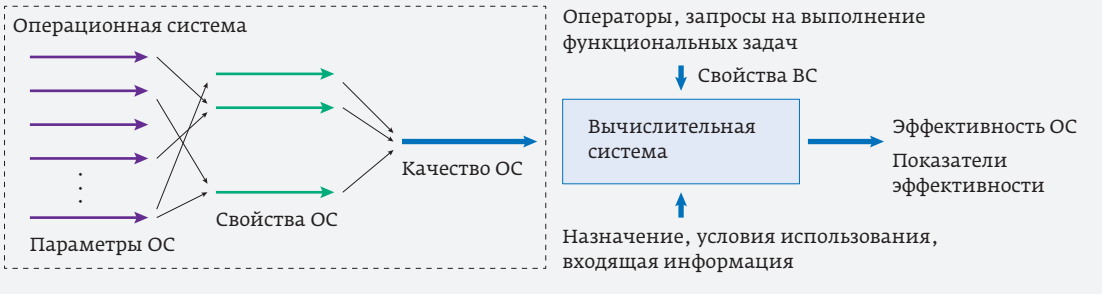


Рис. 1. Связь между параметрами, свойствами и показателями эффективности ОС

Характерная черта больших программных систем – высокая сложность и большой объем программного кода, например ОС Windows 2000 содержит примерно 30 млн строк кода, а следующие версии намного больше. Понятно, что ни один разработчик не может даже надеяться понять целиком эту систему. Следствие гигантского объема – большое количество программных ошибок, которые выявляются долгие годы.

Инерция и желание сохранить «обратную» совместимость требует наличия в новой версии системы интерфейса и возможностей, присутствующих в старой версии, в результате чего ранее разработанные программы и пользователи могут работать с новой версией без переделки (или переучивания). Это усложняет и удорожает систему, а желание сделать систему «более дружественной по отношению к пользователю», то есть такой, которая не предъявляет к пользователю особых требований в части знания компьютера и программирования, как правило, приводит к снижению производительности компьютера, а отсюда необходимо принятие разработчиками специальных мер, компенсирующих этот фактор, как, например, реализация графического интерфейса Windows в режиме ядра ОС.

Негативный эффект привносит стратегия производителей, направленная на повышение продаж. Разработчики зачастую не имеют четкого представления, как будет использоваться их система. Это приводит к тому, что они вынуждены увеличивать ее универсальность, добавляя множество разнообразных функций. К этому надо добавить агрессивную рыночную и рекламную политику производителей-конкурентов, периодически объявляющих о разработке новых, более совершенных, версий программного продукта. В такой ситуации компании-производители снимают поддержку и сопровождение популярных распространенных версий систем, чтобы вынудить потребителей системы перейти на новую версию. При этом необходимо обеспечить максимальную совместимость с предыдущими версиями.

Сложность системы возрастает с течением времени и в связи с необходимым усложнением организации мультипрограммного вычислительного процесса, что связано с прогрессом микроэлектроники и технологии построения вычислительных устройств и с желанием их максимального использования. Отсюда возрастает сложность управления большим числом параллельных заданий, процессов, потоков, волокон в сложных аппаратных многопроцессорных и многоядерных структурах. Большой проблемой является и достижение требуемой надежности и безопасной работы операционной системы. Особенность программных систем, в отличие от аппаратуры, вытекает из факта их неизменности по мере наработки. Если какие-то изменения происходят, особенно на начальных этапах работы, то это связано с установкой «заплаток» и сервисных пакетов. Однако в целом изменения в ОС происходят только в результате устранения программных ошибок. При разработке системы должен быть предусмотрен учет возможных непреднамеренных ошибок легальных пользователей или их незаконных действий, направленных на получение каких-либо преимуществ и даже хищений. Нужно учитывать также действия потенциально враждебных пользователей, желающих вмешаться в работу системы, выполняя незаконные действия.

Дополнительный вклад в возможные причины ненадежности и безопасности операционной системы вносит необходимость обеспечения информационной и программной совместимости с другими системами, требования переносимости на другие платформы и поддержки широкой номенклатуры внешних устройств, которые проектируются независимо друг от друга и часто независимо от какой-либо ОС. Развитие технологии виртуализации несколько сглаживает эту проблему, но возникает другая – разработка эффективных гипервизоров. Современные операционные системы – долгожители (первая версия Unix живет с 1970 года и поныне, Windows – с 1985 года). Проектировщики и разработчики ОС должны представлять, как могут измениться компьютеры и приложения в будущем и как к этому подготовиться. Понятно, что только учет всех перечисленных факторов позволит создавать эффективные операционные системы, однако существенно осложняет решение вопросов надежности и безопасности ОС.

Основная причина аварийных отказов операционных систем кроется в двух принципиальных дефектах разработки, свойственных всем этим системам: наличие слишком большого числа привилегий и отсутствие адекватной изоляции сбоев. Практически все операционные системы состоят из многочисленных модулей, скомпонованных в одном адресном пространстве и образующих единую бинарную программу, которая выполняется в режиме ядра. Ошибка в любом модуле может легко привести к разрушению структур данных в каком-либо другом, не связанном с ним модуле, и к мгновенному выходу системы из строя. Причиной, по которой все модули компонуются в единое адресное пространство без поддержки какой-либо защиты между модулями, является улучшенная производительность ценой бóльшего числа отказов системы. Тесно связанный вопрос относится к первопричине аварийных отказов.

Если бы каждый модуль был безупречным, то не возникала бы потребность в изоляции сбоев между модулями, поскольку не было бы самих сбоев. По утверждению авторов исследования [10], большая часть сбоев возникает из-за ошибок программирования, вследствие чрезмерной сложности и использования чужого кода. Исследования показывают, что в программном обеспечении в среднем

содержится от одной до шестнадцати ошибок на тысячу строк кода [11, 12] и что верхняя граница этого диапазона явно занижена, поскольку учитывались только те ошибки, которые, в конце концов, удавалось обнаружить. Очевидным заключением является то, что в бóльшем объеме кода содержится бóльшее число ошибок. По мере развития программного обеспечения в каждой его новой версии появляется все больше возможностей (и, соответственно, бóльший объем кода), и часто новая версия является менее надежной, чем предыдущая. Известно, что число ошибок на тысячу строк кода стремится к стабилизации по мере роста числа выпущенных версий, но асимптотически этот показатель отличается от нуля. Наличие некоторых из этих ошибок позволяет злоумышленникам применять вирусы и черви для заражения и повреждения системы. Так что некоторые якобы наличествующие проблемы «безопасности» не имеют ничего общего с нарушениями мер безопасности, а вызываются всего лишь ошибками в коде программ (например, переполнение буферов позволяет выполнять внедренный код).

Вторая проблема состоит в привнесении в операционную систему чужого кода. Наиболее искушенные пользователи никогда бы не позволили сторонней организации вставить незнакомый код в ядро операционной системы, хотя, когда они покупают новое периферийное устройство и инсталлируют соответствующий драйвер, они именно это и делают. Драйверы устройств обычно пишутся программистами, работающими на изготовителей периферийных устройств, и контроль качества их продукции обычно ниже, чем у поставщиков. В тех случаях, когда драйвер относится к open-source, его часто пишет благонамеренный, но не обязательно опытный доброволец, и контроль качества обеспечивается на еще более низком уровне. Например, в Linux частота появления ошибок в драйверах устройств от трех до семи раз выше, чем в других частях ядра [10]. Даже компания Microsoft, у которой имеются стимулы и ресурсы для применения более серьезного контроля качества, не может добиться намного лучших результатов: 85% всех аварийных отказов в ранних версиях Windows обусловливаются наличием ошибок в коде драйверов. Имеются публикации, посвященные изоляции драйверов устройств с использованием аппаратуры MMU и виртуальных машин [11]. Эти методы концентрируются на решении проблем в унаследованных операционных системах. В отличие от этого, в работах [12, 13] надежность достигается путем разработки новой облегченной ОС (такой как, например, Minix3) с правильной изоляцией сбоев.

АРХИТЕКТУРЫ ОПЕРАЦИОННЫХ СИСТЕМ

Под архитектурой операционной системы понимают структурную и функциональную организацию ОС на основе некоторой совокупности программных модулей. На архитектуру ранних операционных систем обращалось мало внимания, отсутствовал опыт разработки больших программных систем. Первые ОС разрабатывались как монолитные системы, без четко выраженной структуры. Для построения монолитной системы необходимо было скомпилировать все отдельные процедуры, а затем связать их вместе (примерами могут служить ранние версии ядра UNIX или Novell NetWare). Такой подход был несовместим с расширением программной системы. Уже ОС OS/360 содержала более 1 млн строк кода, а система Maltics содержала к 1975 году 20 млн строк [14]. Стало ясно, что разработка таких систем должна вестись на основе модульного программирования. Большинство современных ОС представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой унифицированной архитектуры ОС не существовало. Принципиально важными универсальными подходами к разработке архитектуры ОС являются: модульная организация, функциональная избыточность, функциональная избирательность, параметрическая универсальность, концепция многоуровневой иерархической организации и др.

Классической стала считаться архитектура ОС, основанная на концепции иерархической многоуровневой машины, привилегированном ядре и пользовательском режиме работы транзитных модулей. Модули ядра выполняют базовые функции ОС: управление процессами, памятью, устройствами ввода-вывода и т. п. Ядро составляет сердцевину ОС, без которой она является полностью неработоспособной и не может выполнить ни одну из своих функций. В ядре решаются внутрисистемные задачи организации вычислительного процесса, недоступные для приложения. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования – API (Application Programming Interface). Для обеспечения высокой скорости работы ОС значительная часть модулей ядра является резидентной и работает в привилегированном режиме. Этот режим, во-первых, должен обезопасить работу самой ОС от вмешательства приложений и, во-вторых, обеспечить возможность работы модулей ядра с полным набором машинных инструкций, позволяющих собственно ядру выполнять управление ресурсами компьютера, в частности переключение процессора с задачи на задачу, управление устройствами ввода-вывода, распределение и защиту памяти и др. Остальные модули ОС выполняют не столь важные функции, нежели ядро, и являются транзитными.

В концепции иерархической машины структура ОС представляется рядом слоев. При такой организации каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют

межслойный интерфейс. На основе этих функций следующий верхний по иерархии слой строит свои функции – более сложные и более мощные и т. д. Такая организация системы существенно упрощает ее разработку, так как позволяет сначала «сверху-вниз» определить функции слоев и межслойные интерфейсы, а при детальной реализации, двигаясь «снизу-вверх», – наращивать мощность функции слоев. Кроме того, модули каждого слоя можно изменять без необходимости изменений в других слоях (но не меняя межслойных интерфейсов!) [9]. Многослойная структура ядра ОС может быть представлена, например, следующими слоями:

- **средства аппаратной поддержки ОС.** К ним относятся система прерываний, средства поддержки привилегированного режима, средства поддержки виртуальной памяти, системный таймер, средства переключения контекстов процессов (информация о состоянии процесса в момент его приостановки), средства защиты памяти и др.;
- **машинно-зависимые модули ОС.** Этот слой образует модули, в которых отражается специфика аппаратной платформы компьютера. Назначение этого слоя – «экранирование» вышележащих слоев ОС от особенностей аппаратуры (например, Windows 2000 – это слой HAL (Hardware Abstraction Layer), уровень аппаратных абстракций);
- **базовые механизмы ядра.** Эта группа модулей выполняет наиболее примитивные операции ядра: программное переключение контекстов процессов, диспетчерскую операцию прерываний, перемещение страниц между основной памятью и диском и т. п.;
- **менеджеры ресурсов.** Модули этого слоя выполняют стратегические задачи по управлению ресурсами вычислительной системы. Это менеджеры (диспетчеры) процессов ввода-вывода, оперативной памяти и файловой системы;
- **интерфейс системных вызовов.** Это верхний слой ядра ОС, взаимодействующий с приложениями и системными утилитами, он образует прикладной программный интерфейс ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной компактной форме, без указания деталей их физического расположения.

Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению – обратное переключение. За счет этого возникает дополнительная задержка в обработке системного вызова. Однако такое решение стало классическим и используется во многих ОС (UNIX, VAX, VMS, IBM OS/390, OS/2, Windows др.). Многослойная классическая многоуровневая архитектура ОС не лишена своих проблем. Дело в том, что значительные изменения одного из уровней могут иметь трудно предвидимое влияние на смежные уровни. Кроме того, многочисленные взаимодействия между соседними уровнями усложняют обеспечение безопасности.

Поэтому, как альтернатива классическому варианту архитектуры ОС, часто используется микроядерная архитектура ОС. Суть этой архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. Микроядро защищено от остальных частей ОС и приложений. В его состав входят машинно-зависимые модули, а также модули, выполняющие базовые механизмы обычного ядра. Все остальные более высокоуровневые функции ядра оформляются как модули, работающие в пользовательском режиме. Таким образом, в архитектуре с микроядром традиционное расположение уровней по вертикали заменяется горизонтальным. Внешние по отношению к микроядру компоненты ОС реализуются как обслуживающие процессы. Между собой они взаимодействуют как равноправные партнеры с помощью обмена сообщениями, которые передаются через микроядро. Поскольку назначением этих компонентов ОС является обслуживание запросов приложений пользователей, утилит и системных обрабатывающих программ, менеджеры ресурсов, вынесенные в пользовательский режим, называются серверами ОС. Схема смены режимов при выполнении системного вызова в микроядерной ОС сопровождается как минимум четырьмя переключениями режимов, в то время как в классической архитектуре – двумя. Следовательно, производительность ОС микроядерной архитектуры, при прочих равных условиях, будет ниже, чем у ОС с классическим ядром (наиболее критикуемый недостаток). В то же время признаны следующие достоинства микроядерной архитектуры [5, 7]:

- единообразные интерфейсы;
- простота расширяемости;
- высокая гибкость;

ГРУППА КОМПАНИЙ

ЭСТО

ЭЛЕКТРОННОЕ СПЕЦИАЛЬНОЕ
ТЕХНОЛОГИЧЕСКОЕ ОБОРУДОВАНИЕ

Акционерное общество
«Научно-производственное
предприятие «Электронное
специальное технологическое
оборудование»

124460, Москва, Зеленоград,
просп. Георгиевский, д. 5, стр. 1
тел.: (499) 729-77-51,
(499) 479-12-39
info@nppesto.ru
www.nppesto.ru

АО НПП ЭСТО (Группа компаний ЭСТО) - объединение ведущих российских предприятий, специализирующихся на разработках, производстве, модернизации, продаже и сервисном обслуживании специального технологического оборудования.

Направления деятельности группы «ЭСТО»

- Разработка и производство технологического оборудования (лазерное, вакуумное, сборочное, нестандартное) и внедрение технологий
- Организация поставок как отдельных единиц зарубежного технологического оборудования, так и комплексных законченных технологий «под ключ»
- Комплексная и частичная модернизация российского и зарубежного технологического оборудования любой сложности
- Сервисное обслуживание российского и зарубежного технологического оборудования
- Проектирование и строительство производств микроэлектроники
- Обучение специалистов заказчика
- Технологический аудит производства

Группа компаний ЭСТО более 20 лет производит оборудование для микроэлектроники в собственном инженерно-производственном комплексе метражом в 5000 кв.м в г. Зеленограде

- возможность переносимости;
- высокая надежность работы (наиболее важное преимущество);
- поддержка распределенных систем;
- поддержка объектно-ориентированных ОС.

По многим источникам вопрос масштабов потери производительности в микроядерных ОС является спорным [5, 11, 12]. Многое зависит от размеров и функциональных возможностей микроядра. Избирательное увеличение функциональности микроядра приводит к снижению количества переключений между режимами системы, а также переключений адресных пространств процессов. Может быть, это покажется парадоксальным, но есть и такой подход к микроядерной ОС, как уменьшение микроядра.

МЕТОД И ОГРАНИЧЕНИЯ, ПРИНЯТЫЕ ПРИ РАЗРАБОТКЕ МОДЕЛЕЙ АРХИТЕКТУР ОПЕРАЦИОННЫХ СИСТЕМ

Разрабатываемые далее модели архитектур операционных систем основаны на предположении, что процесс функционирования компьютерной системы под управлением ОС является марковским. Его особенность заключается в том, что состояния системы изменяются во времени случайным непредсказуемым образом, причем для каждого момента времени t_0 вероятность любого состояния для $t_1 > t_0$ (в будущем) зависит только от вероятности состояния в момент t_0 (в настоящем) и не зависит от вероятностей состояний при $t < t_0$ (в прошлом). Другими словами, свойство марковского процесса заключается в том, что вероятности достижений будущих состояний не зависят от «предыстории» процесса. Если система меняет свое состояние скачкообразно и переходы из одного состояния в другое обладают марковским свойством, то случайный процесс называется марковской цепью [15, 16]. Удобно переходы из одного состояния в другое отображать в виде графа, в котором вершины представляют собой возможные состояния системы, а дуги графа отражают переходы из одного состояния в другое.

Поскольку переход из одного состояния в другое для систем массового обслуживания (СМО) возможен в любой момент времени, определяемый появлением заявки во входном потоке, то для изучения СМО (в нашем случае операционной системы) применяются непрерывные марковские цепи. Одна из важнейших задач теории марковских процессов заключается в нахождении вероятностей состояний цепи. Эти вероятности для непрерывных марковских цепей определяются с помощью дифференциальных уравнений Колмогорова [16]. Так как предельные вероятности состояний системы постоянны, то, заменяя в уравнениях Колмогорова их производные нулевыми значениями, можно перейти к системе линейных алгебраических уравнений, описывающих стационарный режим. Систему этих уравнений можно составить непосредственно по размеченному графу состояний, если руководствоваться правилом, согласно которому слева в уравнениях стоит предельная вероятность данного состояния p_i, умноженная на суммарную интенсивность всех потоков, ведущих из данного состояния, а справа – сумма произведений интенсивностей всех потоков, входящих в i-е состояние, на вероятности тех состояний, из которых эти потоки исходят. Построить граф состояний системы, управляемой ОС некоторой архитектуры, не представляет особой сложности (с некоторыми допущениями, не снижающими адекватность модели цели исследования). Большую сложность представляет собой определение интенсивностей потоков, переводящих систему из одного состояния в другое.

Задачей разработки моделей архитектур ОС в данной работе является оценка надежности работы компьютерной системы, которую обеспечивает операционная система той или иной архитектуры. При этом исходим из предположения, что надежность аппаратуры является абсолютной, и в целом надежность функционирования компьютерной системы определяется надежностью программного обеспечения (ПО), в которое входит операционная система, системные и пользовательские приложения. Если рассматривать отказавшее ПО без учета его восстановления, а также случайный характер и независимость отказов в программах, то основные показатели надежности в этом случае не отличаются от тех, которые характерны для аппаратуры компьютера. Таким образом, основными показателями надежности ПО являются:

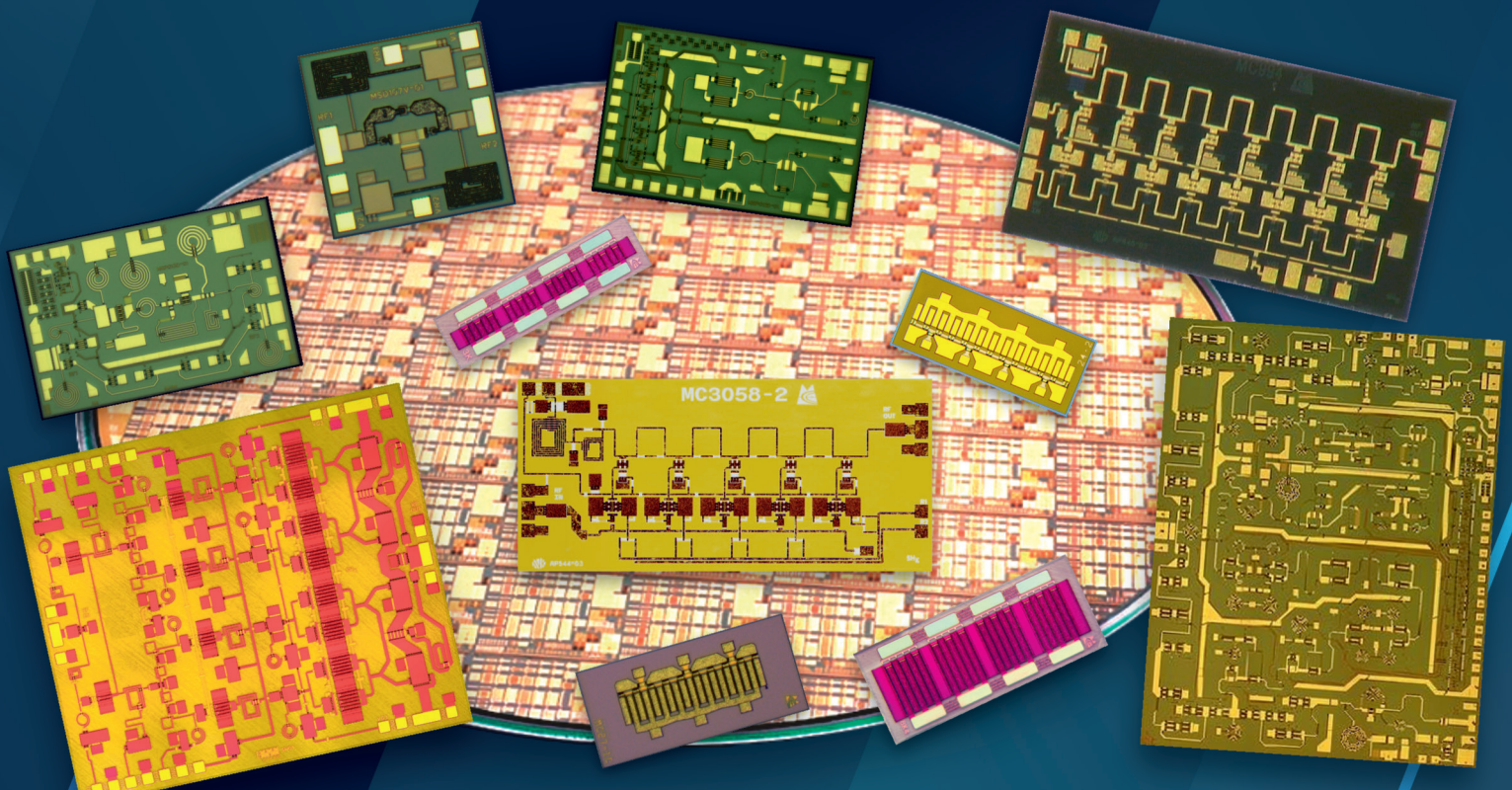
- вероятность безотказной работы программы $p(t)$, представляющая собой вероятность того, что ошибки программы не проявятся в интервале времени $(0, t)$;
- вероятность отказа программы $q(t)$ или вероятность события отказа до момента времени t;
- интенсивность отказов программы $\lambda(t)$;
- средняя наработка программы на отказ T, являющаяся математическим ожиданием временного интервала между последовательными отказами.

МИКРОВОЛНОВЫЕ СИСТЕМЫ

ИНТЕЛЛЕКТ • КАЧЕСТВО НАДЕЖНОСТЬ

- ПРОЕКТИРОВАНИЕ И ПРОИЗВОДСТВО СВЧ GaAs И GaN ТРАНЗИСТОРОВ, МОНОЛИТНЫХ ИНТЕГРАЛЬНЫХ СХЕМ И МИКРОМОДУЛЕЙ
- СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ И ПРОИЗВОДСТВА
- ПРОЕКТИРОВАНИЕ И ПРОИЗВОДСТВО ТВЕРДОТЕЛЬНЫХ СВЧ МОДУЛЕЙ, МНОГОФУНКЦИОНАЛЬНЫХ УСТРОЙСТВ И БЛОКОВ РЭА
- НАИЛУЧШЕЕ СООТНОШЕНИЕ ЦЕНА / КАЧЕСТВО / СРОКИ

СИСТЕМА МЕНЕДЖМЕНТА КАЧЕСТВА СЕРТИФИЦИРОВАНА НА СООТВЕТСТВИЕ ТРЕБОВАНИЯМ ИСО9001



Область применения

- Широкополосная связь и телекоммуникации
- Контрольно-измерительные приборы
- Радиорелейная и спутниковая связь
- Специальная и космическая аппаратура
- Радиолинии «точка-точка», «точка-многоточка»

АО «МИКРОВОЛНОВЫЕ СИСТЕМЫ»
Москва, Щёлковское шоссе, д. 5, стр. 1
Тел.: +7(499) 644-21-03
e-mail: mwsystems@mwsystems.ru
www.mwsystems.ru

Формальные методы оценки надежности программных систем не позволяют получить числовые значения этих показателей в зависимости от числа возможных программных ошибок без испытания программной системы. При этом характер изменения этих показателей во времени будет зависеть от модели надежности ПО. Эти модели учитывают тот факт, что возникающие при работе программ ошибки устраняются, количество ошибок уменьшается и, следовательно, интенсивность их появления понижается, а наработка на отказ программы увеличивается. Так как в нашем случае такой возможности определения наработки программы на отказ нет, примем следующие ограничения и допущения, позволяющие определить исходные данные для моделей функционирования компьютерных систем с ОС различных архитектур. Основное ограничение касается числа состояний исследуемой модели. Оно не должно быть значительным (не более 4–7), в противном случае усложняется поиск решения. Далее будем считать, что 1000 строк программного кода содержат от 4 до 16 ошибок. Это подтверждается практикой и рядом публикаций [7, 10]. Для драйверов число ошибок в 3–7 раз больше. Примем далее, что микроядерная ОС (подобная Minix) содержит 6 000 строк, драйвер – 100 000 строк, основная часть ядра многоуровневой модульной ОС (подобной Linux) содержит 10 млн строк и вспомогательная часть – 20 млн строк. Примем допущение, что микроядерное ядро Minix с числом ошибок, равным 6 (1 ошибка на 1000 строк кода), имеет наработку на отказ T=100 000 ч. Другие ограничения и допущения будут вводиться в конкретных моделях ОС.

Заметим, что любую операционную систему для решения различных задач ее исследования можно представлять системой массового обслуживания. Теория марковских процессов позволяет описывать системы массового обслуживания в форме вероятностных моделей и в установившемся режиме представлять их работу системами линейных алгебраических уравнений. В следующих частях статьи с учетом изложенных ограничений и допущений на основе марковского подхода разрабатываются и исследуются модели трех основных архитектур операционных систем.

ЛИТЕРАТУРА

1. **Назаров С. В** Эффективность и оптимизация компьютерных систем. Монография / 2-е изд. М.: РУСАЙНС, 2021. 294 с.
2. **Назаров С. В.** Эффективность современных операционных систем // Современные информационные технологии и ИТ-образование. 2017. Т. 13 № 2. С. 9–24.
3. **Назаров С. В., Вилкова Н. Н.** Эффективность систем отображения информации коллективного пользования // Электросвязь. 2015. № 9. С. 29–33.
4. **Назаров С. В., Вилкова Н. Н.** Выбор оптимального варианта системы отображения информации коллективного пользования // Электросвязь. 2017. № 1. С. 60–65.
5. **Таненбаум Э., Вудхалл А.** Операционные системы. Разработка и реализация / 3-е изд. СПб: Питер, 2007. 704 с.
6. **Назаров С. В.** Архитектура и проектирование программных систем: монография / 2-е изд., перераб. и доп. М.: ИНФРА-М, 2016. 376 с.
7. **Таненбаум Э., Хердер Дж., Бос Х.** Построение надежных операционных систем, допускающих наличие ненадежных драйверов устройств. [Электронный ресурс]. URL: http://citforum.ru/operating_systems/microkernel_tanenbaum/
8. **Назаров С. В., Широков А. И.** Технологии многопользовательских операционных систем. М.: Изд. дом МИСиС, 2012. 296 с.
9. **Назаров С. В., Вилкова Н. Н.** Структурный рефакторинг многослойных программных систем // Информационные технологии и вычислительные системы. 2016. № 4. С. 13–23.
10. **Tanenbaum Andrew S., Herder Jorrit N., Bos Herbert.** Vrije Universiteit, Amsterdam. Can We Make Operating Systems Reliable and Secure? Computer (IEEE Computer Society, V. 39, No 5, May 2006).
11. **Хердер Й., Бос Х., Таненбаум Э.** Построение надежных операционных систем, допускающих наличие ненадежных драйверов устройств, 2006. [Электронный ресурс]. URL: http://citforum.ru/operating_systems/reliable_os/
12. **Tanenbaum A.** Introduction to MINIX 3 // OS News is Exploring the Future of Computing. 2006. [Электронный ресурс]. URL: http://www.osnews.com/story/15960/
13. **Игнатов Р.** MINIX 3 – реинкарнация? [Электронный ресурс]. URL: http://www.minix3.ru/articles/Ignatov-_minix_reincarnation.pdf
14. **Таненбаум Э., Бос Э.** Современные операционные системы / 4-е изд. СПб: Питер, 2015. 1120 с.
15. **Кельберт М. Я., Сухов Ю. М.** Вероятность и статистика в примерах и задачах. Т. 2. Марковские цепи как отправная точка теории случайных процессов и их приложения. М.: МЦНМО, 2009. 476 с.
16. **Кемени Дж., Снелл Дж.** Конечные цепи Маркова. М.: Наука, 1970. 273 с.
17. **Блинов А.** Лаборатория Касперского – об основах киберимунитета и концепции KasperskyOS. [Электронный ресурс]. URL: https://spbit.ru/news/Laboratoriya-Kasperskogo-ob-osnovakh-kiberimmuniteta
18. Архитектура KasperskyOS[Электронный ресурс]. URL: https://support.kaspersky.ru/help/KCE/1.1/ru-RU/overview_architecture.htm
19. Микроядро KasperskyOS. [Электронный ресурс]. URL: https://os.kaspersky.ru/technologies/microkernel/?ysclid=ll3ego7wh4870216945

ОАО «ИНТЕГРАЛ» – управляющая компания холдинга «ИНТЕГРАЛ»

AMolchanov@integral.by, AVolynets@integral.by
+375 17 342 96 45
ZLozhkina@integral.by, +37517 250 75 13

АДАПТЕРЫ СЕРВИСНЫХ УСТРОЙСТВ «ИВАД»

В рамках научно-технической программы Союзного государства «Автоэлектроника» акционерным обществом «ИНТЕГРАЛ» – управляющая компания холдинга «ИНТЕГРАЛ» разработан набор интеллектуальных высокоинтегрированных адаптеров сервисных устройств (ИВАД), предназначенных для создания интеллектуальных периферийных сервисных мехатронных устройств с электроприводом и сервисных мехатронных устройств управления световыми приборами, управляемых по информационным каналам на базе CAN-шины.

ИВАД имеют два варианта исполнения:

– бескорпусное – в виде печатной платы, оборудованной входными и выходными контактными площадками,
– в корпусе – с входным и выходным жгутами и разъемами в соответствии с требованиями заказчика.

▪ Режим работы ИВАД: продолжительный номинальный S1 по ГОСТ 3940.

– Климатическое исполнение ИВАД: в бескорпусном варианте – УХЛ 5.1, в корпусном исполнении УХЛ1 по ГОСТ 15150.
– Диапазон рабочих температур: от – 40 до +80 °С. – Напряжение питания всех ИВАД: от 8 до 40 В.
– ИВАД имеют 4 модификации: ИВАД-А, ИВАД-В, ИВАД-С и ИВАД-D.

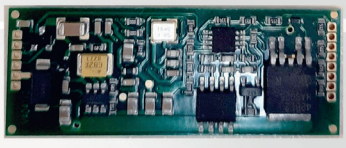
▪ Каждая из модификаций имеет 3 исполнения в бескорпусном и одно в корпусном варианте (суффикс К).

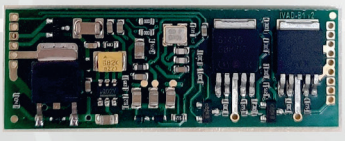
ИВАД-А0 (-А1, -А3, А3-К) и ИВАД-В0 (-В1, -В3, В3-К) могут быть использованы для управления нереверсируемыми сервисными исполнительными устройствами, такими как:

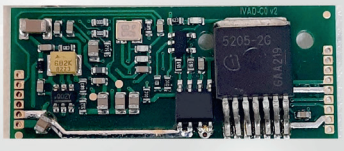
– электропривод стеклоочистителя;
– электропривод насоса омывателя;
– обогрев стекол, зеркал;
– электропривод вентилятора отопителя салона;
– осветительные устройства (фары, фонари);
– звуковая сигнализация.

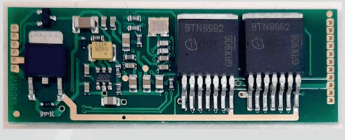
ИВАД-С0 (-С1, -С3, С3-К), ИВАД-D0 (-D1, -D3, D3-К) могут быть использованы для управления реверсируемыми сервисными исполнительными устройствами, такими как:

– электропривод стеклоподъемника;
– электропривод заслонок отопителя салона;
– электропривод подвижного элемента сидения;
– электропривод дверного замка;
– электропривод фар;
– электропривод бокового зеркала заднего вида.


ИВАД-А0


ИВАД-В1


ИВАД-С0


ИВАД-D1

Процессор

Процессор STM8AF526 – восьмиразрядный процессор, ядро Гарвардской архитектуры и с трехступенчатым конвейером. Соответствует стандарту AEC-Q100.

Ядро

– максимальная частота: 24 МГц (в модулях внешний кварц – 16 МГц);
– в среднем 1,6 цикла на инструкцию, что дает 10 MIPS при 16 МГц fCPU для промышленности стандартного теста benchmark;
– программная память: 32 Кбайт Flash-программа;
– память данных EEPROM: 2 Кбайта, выносливость 300 к циклов перезаписи;
– внутренний, настраиваемый пользователем RC 16 МГц;
– внутренний контроллер прерываний с 32 векторами;
– оконные и независимые сторожевые таймеры;
– сохранение данных 20 лет при 55 °С;
– RAM: 6 Кбайт.

Коммуникационные интерфейсы

– высокоскоростной интерфейс CAN 2.0B со скоростью 1 Мбит/с;
– аналого-цифровой преобразователь (АЦП): разрешение 10 бит;
– рабочая температура до 150 °С.
– USART;
– до 6 мультиплексированных каналов;

Разрешение драйвера CAN

Во всех модулях порт PB5 может осуществлять управление разрешением трансивера CAN.
PB5 = 0 – трансивер работает в штатном режиме, PB5 = 1 – трансивер отключен.

Модуль может быть запрограммирован различными способами:

1. В бескорпусном исполнении: через порты PD1/SWIMи NRSTс помощью программатора типа STLinc; через входы шины CAN.
2. В корпусном исполнении: через входной разъем шины CAN.

В состоянии поставки в память контроллера «зашита» программа-загрузчик, позволяющая загружать в модуль пользовательскую программу по CAN-шине. Описание процесса программирования модуля приведено в руководстве пользователя.

ОАО «ИНТЕГРАЛ» – управляющая компания холдинга «ИНТЕГРАЛ»

AMolchanov@integral.by, AVolynets@integral.by
+375 17 342 96 45
ZLozhkina@integral.by, +37517 250 75 13

Таблица 1. Описание и основные характеристики ИВАД

Основное функциональное назначение	Входные и выходные параметры	Модификация	Функциональные особенности
1	2	3	4
Адаптер сервисных устройств для управления нереверсируемыми сервисными исполнительными устройствами	Напряжение питания, UBX – от 8 до 40 В. Цифровых входов – 2. Частотных входов – 1. Один силовой выход с током нагрузки до 1,5 А, в том числе в режиме ШИМ. Выход питания +5 В, от 100 до 500 мА	ИВАД-А0	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания
		ИВАД-А1	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие элементов защиты от ЭМП по каналу CAN и питанию. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
		ИВАД-А3, ИВАД-А3-К	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие полной защиты от ЭМП по питанию и каналу CAN. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
	Напряжение питания, Uвх – от 8 до 40 В. Частотных входов – 1. Два силовых выхода с током нагрузки до 4 А в режиме ШИМ в каждом канале. Выход питания +5 В, 100 мА	ИВАД-В0	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания.
		ИВАД-В1	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие элементов защиты от ЭМП по каналу CAN и питанию. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
		ИВАД-В3, ИВАД-В3-К	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие полной защиты от ЭМП по питанию и каналу CAN. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания

AMolchanov@integral.by, AVolynets@integral.by
+375 17 342 96 45
ZLozhkina@integral.by, +37517 250 75 13

Продолжение таблицы 1.

Основное функциональное назначение	Входные и выходные параметры	Модификация	Функциональные особенности
1	2	3	4
Адаптер сервисных устройств для управления реверсируемыми сервисными исполнительными устройствами	Напряжение питания, Uвх – от 8 до 40 В. Аналоговых входов – 1. Цифровых входов – 1. Частотных входов – 1. Силовой выход с током нагрузки до 1,5 А в режиме ШИМ. Выход питания +5 В, 100 мА	ИВАД-С0	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания
		ИВАД-С1	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие элементов защиты от ЭМП по каналу CAN и питанию. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
		ИВАД-С3, ИВАД-С3-К	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие полной защиты от ЭМП по питанию и каналу CAN. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
	Напряжение питания, Uвх – от 8 до 40 В. Аналоговых входов – 1. Цифровых входов – 1. Частотных входов – 1. Силовой выход с током нагрузки до 7 А в режиме ШИМ. Выход питания +5 В, 100 мА	ИВАД-D0	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания
		ИВАД-D1	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие элементов защиты от ЭМП по каналу CAN и питанию. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания
		ИВАД-D3, ИВАД-D3-К	Сетевой канал связи – CAN. ШИМ – от 0 до 100%. Защита от перенапряжения, обрыва, повышенного тока, короткого замыкания. Наличие полной защиты от ЭМП по питанию и каналу CAN. Наличие защиты от «переполюсовки» по цепям питания. Наличие защиты по информационным входам/выходам от попадания напряжения питания