Современные информационные технологии и ИТ-образование

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

УДК 004.451
DOI: 10.25559/SITITO.14.201801.038-051

АРХИТЕКТУРА И НАДЕЖНОСТЬ ОПЕРАЦИОННЫХ СИСТЕМ

С.В. Назаров[1,2], А.Г. Барсуков[1]

[1] Московский научно-исследовательский телевизионный институт, г. Москва, Россия

[2] Финансовый университет при Правительстве Российской Федерации, г. Москва, Россия

Аннотация

Успехи технологии производства микропроцессоров существенно повысили надежность и производительность аппаратной части компьютерных систем. Этого нельзя сказать о соответствующих характеристиках программного обеспечения и его основы – операционной системы (ОС). Здесь успехи программной инженерии скромнее. Оба направления совершенствования ОС – повышение производительности и надежности – связаны с разработкой эффективных архитектур этих систем. Функциональная сложность ОС приводит к сложности ее архитектуры, которая еще более усиливается специализацией операционной системы в зависимости от сферы применения компьютерной системы (сложные научные расчеты, реальное время, информационно-поисковые системы, системы автоматизированного и автоматического управления и др.). Это привело к большому разнообразию современных ОС. Оценить надежность работы ОС той или иной архитектуры можно только в результате многолетнего натурного эксперимента или имитационным моделированием. Однако это чаще всего неприемлемо по затратам времени и средств на проведение такого исследования. В данной работе делается попытка оценить надежность работы двух основных архитектур ОС – с большим многослойным модульным ядром и микроядерной мультисерверной (клиент-серверной) системы. Разрабатываются модели этих систем, представляемые непрерывными марковскими цепями, которые исследуются в стационарном режиме на основе перехода от систем дифференциальных уравнений Колмогорова к системе линейных алгебраических уравнений.

Ключевые слова

Операционные системы; архитектура; надежность; производительность; микроядро.

Об авторах:

Назаров Станислав Викторович, доктор технических наук, профессор, главный специалист, ЗАО «Московский научно-исследовательский телевизионный институт» (105094, Россия, г. Москва, ул. Гольяновская, д. 7а, стр.1); профессор Департамента анализа данных, принятия решений и финансовых технологий, Финансовый университет при Правительстве Российской Федерации (125993, Россия, г. Москва, Ленинградский проспект, д. 49); действительный член Международной академии информатизации, ORCID: http://orcid.org/0000-0002-7789-1484, nazarov@mniti.ru

Барсуков Алексей Григорьевич, кандидат технических наук, заместитель генерального директора, ЗАО «Московский научно-исследовательский телевизионный институт» (105094, Россия, г. Москва, ул. Гольяновская, д. 7а, стр.1); профессор Академии военных наук, действительный член Международной академии безопасности, Заслуженный изобретатель Российской Федерации, ORCID: http://orcid.org/0000-0002-8787-4987, mniti@mniti.ru

© Назаров С.В., Барсуков А.Г., 2018

Modern Information Technologies and IT-Education

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

ARCHITECTURE AND RELIABILITY OF OPERATING SYSTEMS

Stanislav V. Nazarov[1,2], Alexey G. Barsukov[1]

[1] Moscow Research and Development Television Institute, Moscow, Russia

[2] Financial University under the Government of the Russian Federation, Moscow, Russia

Abstract

Progress in the production technology of microprocessors significantly increased reliability and performance of the computer systems hardware. It cannot be told about the corresponding characteristics of the software and its basis – the operating system (OS). Some achievements of program engineering are more modest in this field. Both directions of OS improvement (increasing of productivity and reliability) are connected with the development of effective structures of these systems. OS functional complexity leads to the multiplicity of the structure, which is further enhanced by the specialization of the operating system depending on scope of computer system (complex scientific calculations, real time, information retrieval systems, systems of the automated and automatic control, etc.) The functional complexity of the OS leads to the complexity of its architecture, which is further enhanced by the specialization of the operating system, depending on the computer system application area (complex scientific calculations, real-time, information retrieval systems, automated and automatic control systems, etc.). That fact led to variety of modern OS. It is possible to estimate reliability of different OS structures only as results of long-term field experiment or simulation modeling. However it is most often unacceptable because of time and funds expenses for carrying out such research. This survey attempts to evaluate the reliability of two main OS architectures: large multi-layered modular core and a multiserver (client-server) system. Represented by continuous Markov chains which are explored in the stationary mode on the basis of transition from systems of the differential equations of Kolmogorov to system of the linear algebraic equations, models of these systems are developed.

Keywords

Operating systems; architecture; reliability; productivity; microkernel.

Введение. 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 и 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). Такое отсутствие структуры было несовместимо с расширением операционных систем. ОС OS/360 содержала более 1 млн строк кода, а система Maltics содержала к 1975 году уже 20 млн строк [1]. 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 существует, но известны универсальные подходы к структурированию ОС. Принципиально важными универсальными подходами к разработке архитектуры ОС являются [2–14]: модульная организация, функциональная избыточность, функциональная избирательность, параметрическая универсальность, концепция многоуровневой иерархической организации и др.

Классической считается архитектура ОС,

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

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 модули ядра (по крайней мере, значительная их часть) являются резидентными и работают в привилегированном режиме (Kernel mode). Этот режим, во-первых, должен обезопасить работу самой ОС от вмешательства приложений, и, во-вторых, должен 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, управлением устройствами ввода-вывода, распределением и защитой памяти и др. Остальные модули ОС выполняют не столь важные функции, как ядро, и являются транзитными. Например, это могут быть программы архивирования данных, дефрагментации диска, сжатия дисков, очистки дисков и т.п.

В концепции многоуровневой (многослойной) 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, т.к. 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!) [1, 2, 12, 13, 15, 16].

Многослойная структура ядра ОС может быть представлена, например, следующими слоями.

1. *Средства аппаратной поддержки О*С. 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. Это собственно средства процессора, но без них современные ОС не работают.

2. *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).

3. *Базовые механизмы ядра*. Эта группа модулей выполняет наиболее примитивные операции ядра: программное переключение контекстов процессов, диспетчерскую прерываний, перемещение страниц между основной памятью и диском и т.п. Модули этого слоя не принимают решений о распределении ресурсов, а только обрабатывают решения, принятые модулями вышележащих уровней. Поэтому их часто называют исполнительными механизмами для модулей верхних слоев ОС.

4. *Менеджеры ресурсов*. Модули этого слоя выполняют стратегические задачи по управлению ресурсами вычислительной системы. Это менеджеры (диспетчеры) процессов ввода-вывода, оперативной памяти и файловой системы. Каждый менеджер ведет учет свободных и используемых ресурсов и планирует их распределение в соответствии запросами приложений.

5. *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.

Повышение устойчивости ОС обеспечивается переходом ядра в привилегированный режим.

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

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 [1,18 - 22]. 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. Так, менеджеры ресурсов, являющиеся неотъемлемой частью обычного ядра, становятся «периферийными» модулями, работающими 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, т.е. модулями, основным назначением которых является обслуживание запросов локальных приложений и других модулей 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 [1, 18-20]:

- единообразные интерфейсы;
- простота расширяемости;
- высокая гибкость;
- 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 [18- 20]. 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. В современных операционных системах различают следующие виды ядер [2, 12 -14].

1. *Наноядро* (НЯ). Крайне упрощенное и минимальное ядро, выполняет лишь одну задачу – обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки посылает информацию о результатах обработки вышележащему программному обеспечению. НЯ используются для виртуализации аппаратного обеспечения реальных компьютеров или для реализации механизма гипервизора.

2. *Микроядро* (МЯ) предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием. Большая часть работы осуществляется с помощью специальных пользовательских процессов, называемых сервисами. Микроядерными являются ядра ОС Minix и GNU Hurd и ядро систем семейства BSD.

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

Классическим примером микроядерной системы является Symbian OS. Это пример распространенной и отработанной микроядерной (а начиная с версии Symbian OS v8.1, и наноядерной) операционной системы.

3. *Экзоядро* (ЭЯ) предоставляет лишь набор сервисов для взаимодействия между приложениями, а также необходимый минимум функций, связанных с защитой: выделение и высвобождение ресурсов, контроль прав доступа и т. д. ЭЯ не занимается предоставлением абстракций для физических ресурсов – эти функции выносятся в библиотеку пользовательского уровня (libOS). В отличие от микроядра ОС, базирующиеся на ЭЯ, обеспечивают большую эффективность за счет отсутствия необходимости в переключении между процессами при каждом обращении к оборудованию.

4. *Монолитное ядро* (МнЯ) предоставляет широкий набор абстракций оборудования. Все части ядра работают в одном адресном пространстве. МнЯ требуют перекомпиляции при изменении состава оборудования. Компоненты операционной системы являются не самостоятельными модулями, а составными частями одной программы. МнЯ более производительно, чем микроядро, поскольку работает как один большой процесс. Большинство Unix-систем и Linux имеют МнЯ. Монолитность ядер усложняет отладку, понимание кода ядра, добавление новых функций и возможностей, удаление ненужного, унаследованного от предыдущих версий кода. «Разбухание» кода монолитных ядер также повышает требования к объему оперативной памяти.

5. *Модульное ядро* (МдЯ) – современная, усовершенствованная модификация архитектуры МнЯ. В отличие от «классических» МнЯ, модульные ядра не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера. Вместо этого они предоставляют тот или иной механизм подгрузки модулей, поддерживающих то или иное аппаратное обеспечение (например, драйверов). Подгрузка модулей может быть, как динамической, так и статической (при перезагрузке ОС после переконфигурирования системы). МдЯ удобнее для разработки, чем традиционные монолитные ядра. Они предоставляют программный интерфейс (API) для связывания модулей с ядром, для обеспечения динамической подгрузки и выгрузки модулей. Некоторые части ядра всегда обязаны присутствовать в оперативной памяти и должны быть жестко «вшиты» в ядро.

6. *Гибридное ядро* (ГЯ) – модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра. Имеют «гибридные» достоинства и недостатки. Примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляются монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.

Несмотря на достаточно большое количество возможных вариантов, основными архитектурами современных операционных систем следует считать модульную многослойную архитектуру и микроядерную архитектуру. Первая обеспечивает более высокую производительность компьютерной системы, вторая – большую надежность. Учитывая, что в настоящее время вопросы производительности операционных систем теряют актуальность в связи быстрым ростом производительности компьютеров, представляет интерес, насколько микроядерные операционные системы надежнее больших модульных многослойных операционных систем. Признанным популяризатором микроядерных системстал Э. Таненбаум, автор и разработчик ОС Minix. Именно он первым провел исследования производительности микроядерной ОС, которые показали, что уменьшение производительности этой ОС по сравнению с монолитной ОС не превышает 10% [18, 21]. Однако, что касается надежности работы микроядерной ОС по сравнению с монолитной ОС, то здесь Э. Таненбаум дал только качественный анализ, не подкрепленный числовыми показателями.

Метод и ограничения, принятые при разработке моделей архитектур операционных систем

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

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 [10, 11]. 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 СМО возможен в любой момент времени, определяемый появлением заявки во входном потоке, то для изучения систем массового обслуживания (в нашем случае компьютерной 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 [23 - 27]. 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 [28].

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 учёта его восстановления, а также случайный характер и независимость отказов в программах, то основные показатели надёжности в этом случае не отличаются от тех, которые характерны для аппаратуры компьютера.

Таким образом, основными показателями надёжности ПО являются [29]:

- 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 λ(t);
- средняя наработка программы на отказ T, являющаяся математическим ожиданием временного интервала между последовательными отказами.

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 надёжности ПО [30, 31]. Эти модели учитывают тот факт, что возникающие при работе программ ошибки устраняются, количество

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

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 [1, 2, 17, 18]. Для драйверов число ошибок в 3 – 7 раз больше. Примем далее, что микроядерная ОС (подобная Minix) содержит 6000 строк, драйвер – 100000 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 25, имеет наработку на отказ T=100000 час. Другие ограничения и допущения будут вводиться в конкретных моделях компьютерных систем.

Модель операционной системы с многоуровневым модульным ядром

На рис.1 представлен граф состояний и переходов компьютерной системы с многоуровневой модульной операционной системой. Перечислим состояния и соответственно вероятности нахождения системы в этих состояниях:

P_1 – вероятность работы ядра операционной системы $ОС_Я$ в привилегированном режиме;

P_2 – вероятность работы модулей операционной системы $ОС_П$ в пользовательском режиме;

P_3 – вероятность работы пользовательских приложений ПП;

P_4 – вероятность отказа системы.

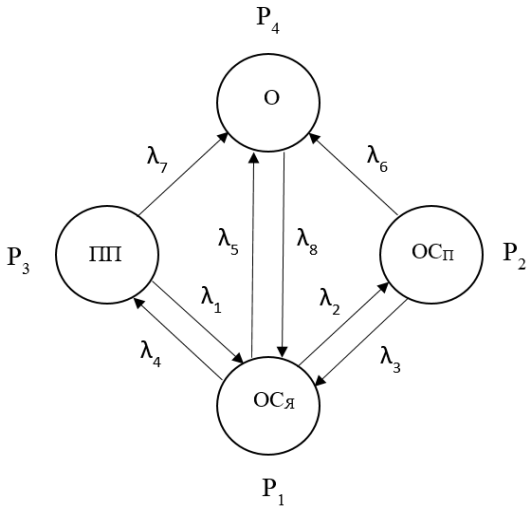

Рис. 1. Граф состояний и переходов системы с многослойным модульным ядром

Определим интенсивности переходов системы из одного состояния в другое (здесь и далее параметры моделей выбраны на основе измерений в реальных операционных системах [28]):

λ1 – интенсивность системных вызовов со стороны прикладных программ, примем значение λ1 = 10000 1/с;

λ2 – интенсивность системных вызовов со стороны прикладных программ, в выполнении которых участвуют устройсва системы с соответствующими драйверами; пусть каждый 200-й системный вызов требует участия драйверов устройств, тогда значение λ2 = 50 1/с;

λ3 – интенсивность выполнения системных вызовов драйверами устройств; пусть на выполнение одного вызова требуется 5 мс, тогда λ3 = 200 1/с;

λ4 – интенсивность передачи выполненных системных вызовов в прикладные программы. Эти вызовы выполняются только ОСЯ (назовём их короткими) или совместно с ОСП или ОСЯ передает результат работы ОСП (назовем их длинными). Примем среднее время передачи выполненных системных вызовов 0,01 мс. Тогда значение λ4 =100000 1/с;

λ5 – интенсивность отказов ОСЯ . Выше было принята интенсивность отказов ядра Minix равной 100000 часов при 25 программных ошибках. Размер ОСЯ был приннят равным 10 млн. строк. Число программных ошибок в данном случае следует принять равным не менее 25 на 6000 строк кода. Таким образом ОСЯ содержит не менее42000 ошибок. Считая,

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

что наработка на отказ обратно пропорциональна числу ошибок, получаем наработку на отказ для ОСЯ равную 60 час. При таких допущениях λ 5 = 0,000005 1/с.

λ6 – интенсивность отказов ОСП . Размер ОСП был принят равным 20 млн. строк. Число программных ошибок в данном случае следует принять равным не менее 75 на 6000 строк кода (как отмечено выше число ошибок в драйверах не чем в 3 раза больше, чем в программах ядра). Число ошибок в ОСП припринятых допущениях не менее 250000. Однако следует заметить, реально используется р аботе не более 1/3 возможностей ОСП. С учетом этого допущения можно принять Λ6 = 0,00001 1/с.

λ7–интенсивность отказов ПП. Считая средний размер пользовательского прило-жения 20000 строк, можно считать, что в нем (при 50 ошибках на 1000 строк) порядка 1000 ошибок. Отсюда λ7 = 0,00000012 1/с.

λ8 – интенсивность перезагрузки операционной системы. Примем время перезаг-рузки равное трем минутам, тогда λ8 = 0,0055 1/с.

По графу состояний и переходов системы, представленному на рис. 1, можно составить систему уравнений, руководствуясьследующим правилом: слева в уравнениях стоит предельная вероятность данного состояния Pi, 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 [12].

Полученная система уравнений имеет следующий вид:

$$\left.\begin{aligned}(\lambda_2+\lambda_4+\lambda_5)P_1=\lambda_1P_3+\lambda_3P_2+\lambda_8P_4;\\(\lambda_3+\lambda_6)P_2=\lambda_2P_1;\\(\lambda_1+\lambda_7)P_3=\lambda_4P_1;\\\lambda_8P_4=\lambda_5P_1+\lambda_6P_2+\lambda_7P_3;\end{aligned}\right\}\quad(1)$$

В данном случае имеем четыре уравнения в системе (1) при четырех неизвестных. Уравнения однородны (не имеют свободного члена) и определяют неизвестные только с точностью до произвольного множителя. Но можно воспользоваться так называемым нормировочным условием:

$$P_1+P_2+P_3+P_4=1.$$

и с его помощью решить систему. При этом одно (любое) из уравнений можно отбросить (оно вытекает как следствие из остальных). Отбросим первое уравнение и, таким образом, получаем следующую систему уравнений:

$$\left.\begin{aligned}(\lambda_3+\lambda_6)P_2=\lambda_2P_1;\\(\lambda_1+\lambda_7)P_3=\lambda_4P_1;\\\lambda_8P_4=\lambda_5P_1+\lambda_6P_2+\lambda_7P_3;\\P_1+P_2+P_3+P_4=1.\end{aligned}\right\}\quad(2)$$

Подставив определенные выше значения коэффициентов при переменных и записав систему уравнений в матричной форме, удобной для решения средствами Excel, получим систему (2) в следующем виде:

$$\left.\begin{aligned}50\,P_1-200{,}00001\,P_2+0\,P_3+0\,P_4=0;\\100000\,P_1+0\,P_2-10000{,}00000012\,P_3+0\,P_4=0;\\0{,}000005\,P_1+0{,}00001\,P_2+0{,}00000012\,P_3-0{,}0055\,P_4=0;\\1\,P_1+1\,P_2+1\,P_3+1\,P_4=1.\end{aligned}\right\}\quad(3)$$

Модель мультисерверной микроядерной операционной системы

В качестве примера такой операционной системы возьмем структуру MINIX 3, предложенную Э. Таненбаумом около 30 лет назад. В данном случае предлагается иметь несколько небольших модулей (микроядро – МЯ и сервер реинкарнации – СР), работающих в режиме ядра, остальная часть операционной системы представляет собой набор полностью изолированных серверов (серверных процессы – СП) и драйверов (Д), работающих в режиме пользователя. В операционной системе Minix 3 микроядро обрабатывает прерывания, обеспечивает основные механизмы для управления процессами, реализует межпроцессные взаимодействия и производит планирование процессов. Оно также предоставляет небольшой набор вызовов ядра для авторизованных драйверов и серверов, например, для чтения части заданного пользовательского адресного пространства или записи в авторизованные порты ввода-вывода. В адресном пространстве микроядра работает драйвер таймера, но он планируется как отдельный процесс. Никакие другие драйверы в режиме ядра не работают.

Над уровнем микроядра находится уровень

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

драйверов устройств. Для каждого устройства ввода-вывода имеется собственный драйвер, выполняемый в виде отдельного процесса в собственном адресном пространстве, защищенном аппаратурой устройства управления памятью. Драйверы работают в пользовательском режиме и не могут исполнять привилегированные команды, а также читать из портов компьютера или писать в них. Для получения последней возможности они должны производить вызовы ядра. Такая конструкция повышает надежность, хотя и порождает небольшие дополнительные расходы.

Поверх уровня драйверов устройств располагается уровень серверов. Файловый сервер является небольшой (4500 строк исполняемого кода) программой, которая принимает запросы от пользовательских процессов по обработке Posix-совместимых вызовов, относящихся к файлам (read, write, lseek и stat) и выполняет их. На этом уровне находится и менеджер процессов, который поддерживает управление процессами и памятью и выполняет Posix-совместимые и другие системные вызовы, такие как fork, exec и brk. Сервер реинкарнации является родительским процессом всех других серверов и всех драйверов. Если драйвер или сервер аварийно или по собственной инициативе завершается, либо не отвечает на периодические запросы отклика, то сервер реинкарнации принудительно завершает его, если это требуется, и перезапускает из копии на диске или в основной памяти. В число других серверов входит сервер сети, поддерживающий весь стек TCP/IP; простой сервер имен, используемый всеми остальными серверами; и информационный сервер, способствующий отладке.

Наконец, над уровнем серверов находятся пользовательские процессы (ПП). Для пользователей единственным отличием мультисерверной системы от других Unix-систем является то, что библиотечные процедуры для системных вызовов выполняют свою работу путем посылки сообщений серверам. Во всем остальном это обычные пользовательские процессы, в которых может использоваться API Posix.

Межпроцессные взаимодействия (IPC) в MINIX 3 поддерживаются на основе передачи сообщений фиксированной длины с использованием принципа рандеву: система копирует сообщение напрямую от отправителя к получателю, когда оба они к этому готовы. Кроме того, поддерживается механизм асинхронного уведомления о событиях. С системой передачи сообщений интегрирована обработка прерываний. Обработчики прерываний используют механизм уведомлений для сигнализации о завершении ввода-вывода. Этот механизм позволяет обработчику установить бит в битовой шкале "необработанных прерываний" драйвера и продолжить выполнение без блокировки. Когда драйвер становится готовым к получению прерывания, ядро преобразует его в обычное сообщение.

Среди других особенностей, способствующих повышению надежности, наиболее важным является свойство самовосстановления. Если драйвер производит запись по неверному указателю, впадает в бесконечный цикл или неправильно ведет себя каким-либо другим образом, то сервер реинкарнации автоматически заменит его, часто без влияния на другие процессы.

В соответствии с рассмотренной архитектурой микроядерной мультисерверной операционной системой ее граф состояний и переходов может быть представлен, как показано на рис. 2.

Перечислим состояния и соответственно вероятности нахождения системы в этих состояниях:

P_1 – вероятность работы микроядра операционной системы МЯ в привилегированном режиме;

P_2 – вероятность работы модулей сервисных процессов (СП) в пользовательском режиме;

P_3 – вероятность работы драйверов ДР в пользовательском режиме;

P_4 – вероятность работы сервера реинкарнации (СР) в пользовательском режиме;

P_5 – вероятность работы пользовательских приложений ПП;

P_6 – вероятность отказа системы.

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

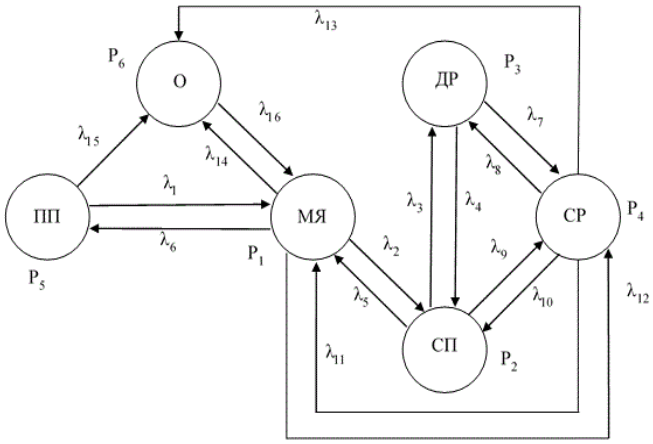

Рис. 2. Граф состояний и переходов мультисерверной микроядерной операционной системы

Аналогично тому, как это было сделано в предыдущей модели, определим интенсивности переходов системы из одного состояния в другое, сохраняя при этом характеристики выполняемых в системе программ пользователей ПП:

λ_1 – интенсивность системных вызовов со стороны прикладных программ, примем значение λ_1 = 10000 1/с;

λ2 – интенсивность системных вызовов со стороны прикладных программ, в выполнении которых участвуют сервисные процессы (СП) без устройств компьютерной системы и, следовательно без драйверов. Учитывая (как принято в модели 1), что каждый 200-й системный вызов требует участия драйверов устройств, тогда значение λ2 = 950 1/с;

λ3 – интенсивность системных вызовов со стороны прикладных программ, требующих участия в их выполнении драйверов устройств; λ3 = 50 1/с;

λ_4 – интенсивность выполнения системных вызовов драйверами устройств; пусть на выполнение одного вызова требуется 5 мс, тогда λ_4 = 200 1/с;

λ_5 – интенсивность передачи результатов системных вызовов, выполненных самим микроядром МЯ и драйверами совместно с сервисными процессами, в прикладную программу ПП. Примем среднее время передачи 0,01 мс. Тогда значение λ_5 =100000 1/с;

λ6 – интенсивность передачи результатов завершения выполнения системныхвызовов драйверами совместно с сервисными процессами, в пользовательские процессыПП (по сути это – разблокировка процессов). Примем среднее время передачи 0,001 мс. Тогда значение λ6 =1000000 1/с;

λ_7 – интенсивность отказов драйверов. Выше было принята интенсивность отказов ядра Minix равной 100000 часов при 25 программных ошибках. Примем размер драйверов 1 млн. строк. Число программных ошибок в данном случае следует принять равным не менее 50 на 1000 строк кода. Таким образом,драйверы содержит не менее 50000 ошибок. Считая, что наработка на отказ обратно пропорциональна числу ошибок, получаем наработку на отказ для Д равную 50 час. При таких допущениях λ_7 = 0,0000055 1/с.

λ8 – интенсивность перезапуска драйверов сервером реинкарнации. Перезапуск может происходить из оперативной памяти или (если в данный момент там есть его копия) или с диска (в противном случае). Примем среднее значение времени перезапуска драйвера 0,01 мс. Таким образом, λ8 = 100000 1/с;

λ9 – интенсивность отказов серверных процессов (СП). Будем считатать, надежность работы СП примерно такая же, как и драйверов. В этом случае λ9 = 0,0000055 1/с.

λ 10 – интенсивность перезапуска серверных процессов сервером реинкарнации. Можно считать это действие аналогичным действию перезапуска драйверов. Тогда λ10 = 100000 1/с;

λ_{11}–интенсивность отказов сервера реинкарнации (СР). Следует считать, что СР также надежен, как и микроядро. В противном случае пропадает сама суть микроядерной операционной системы. Следовательно, наработка на отказ СР равна 100000 час. и λ_{11} = 0,0000000027 1/с;

λ_{12} – интенсивность презапуска сервера реинкарнации. Предполагая, что этот процесс аналогичен перезапуску Д и СП, будем считать, что λ_{12} = 100000 1/с;

λ_{13} – интенсивность фатальных отказов СР (восстановление СР с помощью МЯ невозможно). Примем, что интенсивность таких отказов на порядок ниже интенсивности отказов СР, когда восстановление возможно, тогда λ_{13} = 0,00000000027 1/с;

λ_{14} – интенсивность фатальных отказов МЯ. Считая, что надежности СР и МЯ эквивалентны, λ_{14} = 0,00000000027 1/с;

λ_{15} – интесивность фатальных отказов пользовательских приложений.Примем это

Том 14, № 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

значение, равным соответствующему значению в модели 1, т.е. λ_{15} = 0,0000012 1/с.

λ_{16} – интенсивность перезагрузки операционной системы. Примем время перезагрузки равное трем минутам, тогда λ_8 = 0,0055 1/с.

Аналогично тому, как это сделано в модели по рис. 1, составляем систему уравнений для модели 2:

$$\left.\begin{array}{c}(\lambda_2+\lambda_6+\lambda_{12}+\lambda_{14})P_1=\lambda_5P_2+\lambda_{11}P_4+\lambda_1P_5+\lambda_{16}P_6;\\(\lambda_3+\lambda_5+\lambda_9)P_2=\lambda_2P_1+\lambda_4P_3+\lambda_{10}P_4;\\(\lambda_4+\lambda_7)P_3=\lambda_3P_2+\lambda_8P_4;\\(\lambda_8+\lambda_{10}+\lambda_{11}+\lambda_{13})P_4=\lambda_{12}P_1+\lambda_9P_2+\lambda_7P_3;\\(\lambda_1+\lambda_{15})P_5=\lambda_6P_1;\\\lambda_{16}P_6=\lambda_{14}P_1+\lambda_{13}P_4+\lambda_{15}P_5;\\P_1+P_2+P_3+P_4+P_5=1.\end{array}\right\} \quad (4)$$

Исключаем из системы (4) первое уравнение. Учитывая установленные выше значения коэффициентов при переменных и записав систему уравнений в матричной форме, удобной для решения средствами Excel, получим систему (4) в следующем виде:

$$\left.\begin{array}{c}950\,P_1-100050\,P_2+200\,P_3+100000\,P_4+0\,P_5+0\,P_6=0;\\0\,P_1+50\,P_2-200\,P_3+100000\,P_4+0\,P_5+0\,P_6=0;\\100000\,P_1+0{,}0000055\,P_2+0{,}0000055P_3-200000\,P_4-0\,P_5+0\,P_6=0;\\1000000\,P_1+0\,P_2+0\,P_3+0\,P_4-10000\,P_5+0\,P_6=0;\\0{,}00000000027P_1+0\,P_2+0\,P_3+0{,}00000000027\,P_4+0{,}0000012P_5-0{,}0055\,P_6=0;\\1\,P_1+1\,P_2+1\,P_3+1\,P_4+1\,P_5+1\,P_6=1.\end{array}\right\} \quad (5)$$

Обсуждение результатов эксперимента

Решение системы уравнений (3) дает следующие значения переменных: P_1 = 0,088876; P_2 = 0,022219; P_3 = 0,888764; P_4 =0,000141. Таким образом, вероятность того, что компьютерная система в установившемся состоянии исправно функционирует, равна сумме $P_1+P_2+P_3$ = 0,999685. Вероятность нахождения системы в состоянии перезагрузки равна 0,000141.

Решение системы уравнений (5) дает следующие значения переменных: P_1 = 0,002834; P_2 = 0,002861; P_3 = 0,70940; P_4 =0,001417; P_5 =0,28347; P_6 =0,000006. Таким образом, вероятность того, что компьютерная система в установившемся состоянии исправно функционирует, равна сумме $P_1+P_2+P_3+P_4+P_5$ = 0,999994. Вероятность того, что система в установившемся состоянии будет находиться на этапе перезагрузки, равна 0,000006.

Таким образом, результаты проведенных экспериментов позволяют сделать вывод о значительно более высокой надежности работы микроядерной операционной системы по сравнению с операционной системой с большим многоуровневым модульным ядром.

Заключение

Натурный эксперимент для определения надёжности функционирования операционных систем различных архитектур практически невозможен в силу неприемлемых временных и финансовых затрат. Имитационное моделирование ситуацию не облегчает. Поэтому единственный путь решения этой проблемы – разработка простых и достаточно адекватных моделей.

Предложенный подход к построению моделей архитектур операционных систем в условиях принятия ограничений об абсолютной надёжности аппаратуры, случайном характере и независимости отказов в программах без учёта их восстановления и отсутствии параллельных процессов в работе модулей ОС может быть использован для начального исследования надежности функционирования конкретных архитектур операционных систем.

Повышение точности результатов моделирования в подобных моделях связано с уточнением исходных данных, которые определяют значения наработки на отказ модулей исследуемой операционной системы и интенсивности переходов исследуемой системы из одного состояния в другое. Для уточнения характеристик надежности модулей системы необходимо построение моделей изменения надежности при выявлении и устранении программных ошибок и соответствующая статистика разработчика операционных систем.

Уточнение некоторых значений интенсивностей переходов системы из одного состояния в другое связано с определением временных характеристик работы модулей операционной системы. Для этого можно использовать средства профилирования и трассировки или специально разработанные тесты, например, как показано в [13].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] *Таненбаум Э., Бос Э.* Современные операционные системы. 4-е изд. СПб: Питер, 2015. 1120 с.
[2] *Таненбаум Э., Вудхалл А.* Операционные системы. Разработка и реализация. 3-е изд. СПб: Питер, 2007. 704 с.
[3] *Руссинович М., Соломон Д.* Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Мастер-класс. 4-е издание. М.: Издательство «Русская Редакция»; СПб.: Питер, 2006. 992 с.
[4] *Эбен М., Таймэн Б.* FreeBSD: Администрирование: искусство достижения равновесия – Энциклопедия пользователя. – 3-е изд., перераб. и дополн.: СПб. ДиаСофтЮП. 2005. 768 с.
[5] *Келли-Бутл С.* Введение в UNIX: Пер. с англ. М.: Лори, 1997. 341с.
[6] *Власенко О., Иевлев С., Ионов А.* и др. ALT Linux снаружи. М.: ALTLinux: ДМК-пресс. 2009. 195 с.
[7] *Аленичев Д., Боковой А., Бояршинов А.* и др. ALT Linux изнутри. М.: ALT Linux: ДМК-пресс. 2006. 214 с.
[8] *McGrath M.* Windows 10 in easy steps - Special Edition, 2nd Edition: Covers the Creators Updat. Publishing house: In Easy Steps Limited. In Easy Steps. 2017. 480 p.
[9] *Asadi A.* Linux & Open Source Genius Guide. Volume 7th. Revised Edition Publishing house: Imagine Publishing Ltd. 2015. 180 p.
[10] *Asadi A.* Ubuntu. The Complete Manual 2016. Imagine Publishing Ltd. 2016. 194 p.
[11] *Назаров С.В., Широков А.И.* Технологии многопользовательских операционных систем. М.: Изд. Дом МИСиС, 2012. 296 с.
[12] *Назаров С.В., Широков А.И.* Многопользовательские операционные системы. М.: Изд. дом МИСиС, 2010. 193 с.
[13] Экзоядро: архитектура операционной системы управления ресурсами на уровне приложений, 2011. URL: https://geektimes.ru/post/127207/ (дата обращения: 5.01.2018).
[14] *Engler D.R., Kaashoek M.F., J. O'Toole Jr.* Exokernel: an operating system architecture for application-level resource management // ACM SIGOPS Operating Systems Review. 1995. Vol. 29, issue 5. Pp. 251-266. DOI: http://dx.doi.org/10.1145/224057.224076
[15] *Назаров С.В., Вилкова Н.Н.* Структурный рефакторинг многослойных программных систем // Информационные технологии и вычислительные системы. 2016. №4. С. 13-23. URL: https://elibrary.ru/item.asp?id=27656660 (дата обращения: 5.01.2018).
[16] *Назаров С.В.* Архитектура и проектирование программных систем. 2-е изд., перераб. и доп. М.: ИНФРА-М, 2016. 376 с.
[17] *Хердер Й., Бос Х., Таненбаум Э.* Построение надежных операционных систем, допускающих наличие ненадежных драйверов устройств, 2006. URL: http://citforum.ru/operating_systems/reliable_os/ (дата обращения: 5.01.2018).
[18] *Tanenbaum A.* Introduction to MINIX 3 // OSNews is Exploring the Future of Computing. 2006. URL: http://www.osnews.com/story/15960/Introduction-to-MINIX-3 (дата обращения: 5.01.2018).
[19] *Herder J. N., Bos H., Gras B., Homburg P., Tanenbaum A.S.* MINIX 3: A Highly Reliable, Self-Repairing Operating System // ACM SIGOPS Operating Systems Review. 2006. Vol. 40, issue 3. Pp. 80-89. DOI: http://dx.doi.org/10.1145/1151374.1151391
[20] *Herder J. N., Bos H., Tanenbaum A.S.* A Lightweight Method for Building Reliable Operating Systems Despite Unreliable Device Drivers, Technical Report IR-CS-018, January 2006. 14 p. URL: http://www.minix3.org/doc/reliable-os.pdf (дата обращения: 5.01.2018).
[21] *Игнатов Р.* MINIX 3 - реинкарнация? URL: http://www.minix3.ru/articles/Ignatov-_minix_reincarnation.pdf (дата обращения: 5.01.2018).
[22] *Кельберт М.Я., Сухов Ю.М.* Вероятность и статистика в примерах и задачах. Том 2. Марковские цепи как отправная точка теории случайных процессов и их приложения. М: МЦНМО. 2009. 476 с.
[23] *Гихман И.И., Скороход А.В.* Теория случайных процессов. В 3-х томах. Том 2. Теории случайных процессов. М.: Наука: 1971. 641 с.
[24] *Kolmogorov A.N.* Anfangsgruende der Theorie der Markoffshen Ketten mit unendlichen vielen moeglichen Zustaenden // Sbornik: Mathematics. 1936. Vol. 1, no. 4. Pp. 607-610.
[25] *Lahres H.* Einfuerung in die diskreten Markoff-Prozesse und ihre Anwendungen. Braunschweig, Germany: Friedr. Viewigund Sohn, 1964.
[26] *Кемени Дж., Снелл Дж. Конечные* цепи Маркова. М: Наука. 1970. 273 с.
[27] *Назаров С.В.* Эффективность современных операционных систем // Современные информационные технологии и ИТ-образование. 2017. Т. 13, № 2. С. 9-24. DOI: https://doi.org/10.25559/SITITO.2017.2.229
[28] *Брауде Э.* Технология разработки программного обеспечения. СПб.: Питер, 2004. 655 с.
[29] *Василенко Н.В., Макаров В.А.* Модели оценки надежности программного обеспечения // Вестник Новгородского государственного университета. Серия: Технические науки. 2004. № 28. С. 126-132. URL: https://elibrary.ru/item.asp?id=18184720 (дата обращения: 5.01.2018).
[30] *Павловская О.О.* Статические методы оценки надежности программного обеспечения // Вестник Южно-Уральского государственного университета. Серия: Компьютерные технологии, управление, радиоэлектроника. 2009. № 26 (159). С. 35 – 37. URL: https://elibrary.ru/item.asp?id=12925824 (дата обращения: 5.01.2018).

Поступила 10.11.2017; принята к публикации 01.03.2018; опубликована онлайн 30.03.2018.

REFERENCES

[1] Tanenbaum A., Bos E. Modern Operating Systems. Prentice Hall, 4th edition, 2015. 1136 p. (In Russian)
[2] Tanenbaum A., Vudkhall A. Operating Systems. Design and Implementation. Pearson, 3rd edition, 2008. 1080 p. (In Russian)
[3] Russinovich M., Solomon D. Internal organization of Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Master-klass. 4th edition. M.: Izdatel'stvo «Russkaia Redaktsiia»; SPb.: Piter, 2006. 992 p. (In Russian)
[4] Eben M., Taimen B. FreeBSD. Entsiklopediia pol'zovatelia Administrirovanie: iskusstvo dostizheniia ravnovesiia. 2nd Edition. M.: DIASOFT, 2002. 736 p. (In Russian)
[5] Kelli-Butl S. Introduction to UNIX. M.: Lori, 1997. 341 p. (In Russian)
[6] Vlasenko O., Ievlev S., Ionov A., ads. ALT Linux outside. M.: ALTLinux: DMK-press. 2009. 195 p. (In Russian)
[7] Alenichev D., Bokovoi A., Boiarshinov A., ads. ALT Linux from within. M.: ALT Linux: DMK-press 2006. 214 p. (In Russian)
[8] McGrath M. Windows 10 in easy steps - Special Edition, 2nd Edition: Covers the Creators Updat. Publishing house: In Easy Steps Limited. In Easy Steps. 2017. 480 p.
[9] Asadi A. Linux & Open Source Genius Guide. Volume 7th. Revised Edition Publishing house: Imagine Publishing Ltd. 2015. 180 p.
[10] Asadi A. Ubuntu. The Complete Manual 2016. Imagine Publishing Ltd. 2016. 194 p.
[11] Nazarov S.V., Shirokov A.I. Multi-user operating system technologies. M.: Izd. Dom MISiS, 2012. 296 p. (In Russian)
[12] Nazarov S.V., Shirokov A.I. Multi-user operating systems. M.: Izd. dom MI-SiS, 2010. 193 p. (In Russian)
[13] ExoJadro: architecture of the operating system of resource management at the application level, 2011. Available at: https://geektimes.ru/post/127207/ (accessed 5.01.2018). (In Russian)
[14] Engler D.R., Kaashoek M.F., J. O'Toole Jr. Exokernel: an operating system architecture for application-level resource management. *ACM SIGOPS Operating Systems Review*. 1995; 29(5):251-266. DOI: http://dx.doi.org/10.1145/224057.224076
[15] Nazarov S.V., Vilkova N.N. Structural refactoring of multilayered program systems. *Journal of Information Technologies and Computing Systems*. 2016; 4:13-23. Available at: https://elibrary.ru/item.asp?id=27656660 (accessed 5.01.2018). (In Russian)
[16] Nazarov S.V. Architecture and design of software systems. 2nd Edition. M.: INFRA-M, 2016. 376 p. (In Russian)
[17] Kherder I., Bos Kh., Tanenbaum A. Building reliable operating systems that allow unreliable device drivers, 2006. Available at: http://citforum.ru/operating_systems/reliable_os/ (accessed 5.01.2018). (In Russian)
[18] Tanenbaum A. Introduction to MINIX 3 // OSNews is Exploring the Future of Computing. 2006. Available at: http://www.osnews.com/story/15960/Introduction-to-MINIX-3 (accessed 5.01.2018).
[19] Herder J. N., Bos H., Gras B., Homburg P., Tanenbaum A.S. MINIX 3: A Highly Reliable, Self-Repairing Operating System. *ACM SIGOPS Operating Systems Review*. 2006; 40(3):80-89. DOI: https://doi.org/10.1145/1151374.1151391
[20] Herder J. N., Bos H., Tanenbaum A.S. A Lightweight Method for Building Reliable Operating Systems Despite Unreliable Device Drivers, Technical Report IR-CS-018, January 2006. 14 p. Available at: http://www.minix3.org/doc/reliable-os.pdf (accessed 5.01.2018).
[21] Ignatov R. MINIX 3 – reincarnation? Available at: http://www.minix3.ru/articles/Ignatov-_minix_reincarnation.pdf (accessed 5.01.2018). (In Russian)
[22] Kelbert M.Ya., Sukhov Yu.M. Probability and statistics in examples and problems. Vol. 2. Markov chains as a starting point of the theory of random processes and their applications. M: MCCME. 2009. 476 p. (In Russian)
[23] Gikhman II., Skorokhod A.V. The theory of random processes. In 3 volumes. Vol. 2. Theories of random processes. M.: Nauka: 1971. 641 p. (In Russian)
[24] Kolmogorov A.N. Anfangsgruende der Theorie der Markoffshen Ketten mit unendlichen vielen moeglichen Zustaenden. *Sbornik: Mathematics*. 1936; 1(4):607-610.
[25] Lahres H. Einfuerung in die diskreten Markoff-Prozesse und ihre Anwendungen. Braunschweig, Germany: Friedr. Viewigund Sohn, 1964.
[26] Kemeni J., Snell J. Finite Markov chains. M: Science. 1970. 273 p. (In Russian)
[27] Nazarov S.V. Efficiency of modern operating systems. *Modern Information Technology and IT-education*. 2017; 13(2):9-24. (In Russian) DOI: https://doi.org/10.25559/SITITO.2017.2.229
[28] Braude E. Software Development Technology. SPb.: Piter, 2004. 655 p. (In Russian)
[29] Vasilenko N.V., Makarov V.A. Software reliability assessment models. *Vestnik NovSU. Issue: Engineering sciences*. 2004; 28:126-132. Available at: https://elibrary.ru/item.asp?id=18184720 (accessed 5.01.2018). (In Russian)
[30] Pavlovskaya O.O. Static methods of assessment of software. *Bulletin of the South Ural State University. Series "Computer Technologies, Automatic Control & Radioelectronics*. 2009; 26(159):35–37. Available at: https://elibrary.ru/item.asp?id=12925824 (accessed 5.01.2018). (In Russian)

Submitted 10.11.2017; Revised 01.03.2018; Published 30.03.2018.

About the authors:

Stanislav V. Nazarov, doctor of technical sciences, professor, Chief specialist, Moscow Research and Development Television Institute (7a, build. 1 Golyanovskaya St., Moscow 105094, Russia); Professor of the Department of Data Analysis, Decision Making and Financial Technologies, Financial University under the Government of the Russian Federation (49 Leningradsky prospect, Moscow 125993, Russia); Member of the International Academy of Informatization, ORCID: http://orcid.org/0000-0002-7789-1484, nazarov@mniti.ru

Vol. 14, no. 1. 2018 ISSN 2411-1473 sitito.cs.msu.ru

Alexey G. Barsukov, Candidate of technical sciences, Deputy CEO, Moscow Research and Development Television Institute (7a, build. 1 Golyanovskaya St., Moscow 105094, Russia); professor of Academy of military sciences, Full member of the International academy of safety, Honored inventor of the Russian Federation, ORCID: http://orcid.org/0000-0002-8787-4987, mniti@mniti.ru

This is an open access article distributed under the Creative Commons Attribution License which unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).