# Операционная система Minix 3

Помните, что Minix 3 не является дочерней программой Minix… Minix 1 была написана в качестве учебного пособия… Minix 3 является началом создания высокой надежности, исцеления, без наворотов ОС… Minix 1 и Minix 3 были связаны точно так же, как Windows 3.1 и Windows XP, те одним именем.

Эндрю Таненбаум

Minix – это свободная Unix-подобная микроядерная операционная система, распространяемая по лицензии BSD. Автором операционной системы является Эндрю Таненбаум, который создал первую версию Minix в 1987 году в Амстердамском свободном университете для наглядной демонстрации принципов, описываемых в его книге «Operating Systems: Design and Implementation» (1987, ISBN 0-13-637406-9), где на примере MINIX демонстрируется студентам внутреннее устройство реальной операционной системы. Сокращённые 12000 строк исходного кода ядра, подсистемы управления памятью и файловой системы Minix были напечатаны в книге. Minix написан преимущественно на языке программирования C.

Среди студентов, изучающих теорию операционных систем по исходникам Minix, наибольшей известности достиг Линус Торвальдс, разработчик ОС Linux. Изначально целью Minix являлось обучение студентов строению компьютеров и основ операционных систем. Исходные коды Minix совершенно отличны от кодов ОС Linux, но, как и Linux, Minix является Unix-подобной операционной системой с аналогичным набором команд.

На данное время актуальной является версия Minix 3.3.0, скачать исходный код ОС можно на официальном сайте http://www.minix3.org.

## Об авторе

Эндрю Стюарт Таненбаум (англ. Andrew Stuart Tanenbaum) (родился 16 марта 1944 года) получил степень бакалавра в Массачусетсском технологическом институте и степень доктора наук в Калифорнийском университете в Беркли. На сегодняшний день он является профессором кибернетики в университете Врийе (Vrije) в Амстердаме, где возглавляет группу разработчиков компьютерных систем. Кроме того, вплоть до 2005 года автор в течение 12 лет являлся деканом межуниверситетской школы аспирантов по кибернетике и обработке изображений (Advanced School for Computing and Imaging), занимающейся исследованиями в области современных параллельных систем, распределенных систем и систем обработки изображений.

В прошлом Эндрю Таненбаум занимался исследованиями компиляторов, операционных систем, компьютерных сетей и локальных распределенных систем. В настоящее время его усилия в основном направлены на разработку систем безопасности, особенно для операционных систем, компьютерных сетей и глобальных распределенных систем. Результатом этих исследовательских проектов стали более 100 статей в журналах и отчетах конференций.

Эндрю Таненбаум является автором пяти книг и опубликовал более 120 работ по компьютерной тематике. Его работы известны во всем мире и были переведены на многие языки мира для использования в программе обучения.

Профессор Таненбаум написал множество программ. Под его руководством разрабатывалась архитектура проекта Amsterdam Compiler Kit — инструмента, предназначенного для создания кроссплатформенных компиляторов. Кроме того, он руководил созданием учебной операционной системы Minix — упрощенной версии системы Unix, на базе которой была впоследствии разработана система Linux. Вместе со своими аспирантами и программистами он участвовал в разработке высокопроизводительной локальной распределенной операционной системы Amoeba. Также профессор является одним из разработчиков высокопроизводительной распределенной системы Globe, ориентированной на миллионы пользователей. В настоящее время все эти программные продукты свободно распространяются через Интернет.

1 ноября 2008 года Эндрю Таненбаум получил грант в 2,5 млн. евро от Европейского исследовательского совета для работы над проектом: «Исследование действительно надёжного и безопасного системного программного обеспечения». Благодаря гранту Таненбаум смог продолжить работы по дальнейшему развитию ОС Minix 3 до ноября 2013 года [2].

## История

– Что подтолкнуло вас к разработке Minix?

– Я читал курс с использованием Unix v6. Потом AT&T изменило лицензию, запретив людям изучать их курсы – это было самое глупое решение, которое они могли принять. Согласно их лозунгу; «Чем меньше людей знает Unix, тем лучше», они должны были платить большие деньги людям, изучающим их курс. Из-за этого я решил, если я хочу иметь Unix подобную ОС для обучения, я должен написать ее сам. Так и вышло.

(из интервью с Эндрю Таненбаумом) [3]

Во времена Unix версии 6 ее исходные колы были широко доступны по лицензии AT&T и активно изучались. Джон Лайонс (John Lions) из университета Нового Южного Уэльса в Австралии даже написал небольшую брошюру, шаг за шагом описывающую работу Unix [4]. С разрешения AT&T эта брошюра использовалась во многих университетских курсах по операционным системам.

С выходом версии 7 стало ясно, что система Unix превратилась в дорогостоящий коммерческий продукт, поэтому лицензия, под которой распространялась версия 7, запрещала преподавание исходного кода на учебных курсах, чтобы не подвергать риску его статут коммерческого секрета. Поэтому многие университеты просто прекратили изучение Unix, довольствуясь одной теорией.

К сожалению, изучение одной только теории формирует у студентов однобокий взгляд на то, какой в действительности может быть операционная система. В книгах и курсах, посвященных операционным системам, в подробности рассматривается чисто теоретические вопросы, например, алгоритмы планирования, которые на практике не столь важны. Действительно важные вещи, такие как ввод-вывод и файловые системы, зачастую опускаются, так как им не посвящено достаточно теории. Чтобы исправить ситуацию, Эндрю Таненбаум решил написать собственную операционную систему, которая с точки зрения пользования совместима с Unix, но внутри совершенно самостоятельна. Созданная система не попадает под действие лицензионных ограничений AT&T и может свободно использоваться при обучении.

Название Minix происходит от mini-Unix, так как, по словам авторов, «эта система достаточно мала, чтобы даже начинающий мог понять, как она работает» [1]. У Minix есть еще одно преимущество перед Unix – ее код в большей степени обладает модульной структурой: начиная с первой версии Minix, файловая система и менеджер памяти не являются частью операционной системы, а работают как отдельные пользовательские программы. В версии Minix 3 такая модульность распространилась и на драйверы устройств ввода-вывода – все они, за исключением драйвера часов, выполняются в пользовательском режиме. Еще одно отличие Minix от Unix в том, что что система Unix создавалась, чтобы быть эффективной, а Minix – чтобы быть понятной. Поэтому, например, в коде системы Minix имеются множество комментариев.

В далёком 1987 году Таненбаум относился к Minix только как к учебному пособию. Чтобы оставить код ОС доступным для изучения в течение семестра, Эндрю Таненбаум отвергал предложения усовершенствовать Minix или принять патчи, которые поступали от читателей его учебника. Роль проекта, в котором читатели Таненбаума воплотили своё стремление разрабатывать операционные системы, взял на себя Linux.

Изначально Таненбаум разработал Minix для компьютеров IBM PC и IBM PC/AT, доступных в то время. Minix версии 1.5 был портирован на Motorola 68000, что обеспечило совместимость с популярными платформами Amiga, Atari ST, Apple Macintosh. Кроме того, были сделаны версии для SPARC, National Semiconductor NS32032 и транспьютеров.

### Minix 2

Minix 2.0 снова поддерживал только архитектуру x86. Он прилагался ко второй редакции книги Таненбаума, написанной им в соавторстве с Альбертом Вудхаллом (Albert Woodhull). Minix 1 и 2 были средствами обучения.

За первые десять лет операционная система MINIX претерпела множество изменений. Первоначальный код был рассчитан на IBM PC с процессором 8088 и 256 Кбайт памяти с двумя дисководами, но без жестких дисков. В основе MINIX лежала система UNIX версии 7. С течением времени система MINIX развивалась в различных направлениях: появилась поддержка компьютеров с 32-разрядным защищенным режимом, оснащенных оперативной памятью и жесткими дисками большего объема. Кроме того, система теперь базируется не на UNIX версии 7, а на международном стандарте POSIX (IEEE 1003.1 и ISO 9945-1). В дополнение, система MINIX была перенесена на множество других платформ, включая Macintosh, Amiga, Atari и SPARC.

### Minix 3

К созданию MINIX 3 разработчиков подтолкнули громоздкость, низкое быстродействие и ненадежность существующих операционных систем. Операционные системы выходят из строя значительно чаще, чем электронные устройства – телеприёмники, сотовые телефоны и DVD-плееры, а кроме того, имеют столь огромное количество функций и параметров, что практически ни один человек не способен управлять ими и освоить их полностью. При этом воздействие вредоносных программ (вирусов, червей, шпионских программ, спама и др.) подчас принимает масштабы самой настоящей эпидемии. И многие перечисленные проблемы в значительной степени обусловлены фундаментальным недостатком существующих ОС – отсутствием модульности. Современная операционная система – это одна огромная исполняемая программа, скомпилированная из миллионов строк кода, написанного на языках C и C++, и функционирует в режиме ядра. Ошибка хотя бы в одной строке может стать причиной выхода операционной системы из строя. Обеспечить корректность всего кода невозможно: 70% его объема составляют драйверы устройств, написанные сторонники разработчиками, которые находятся вне поля зрения специалистов, занятых поддержкой ОС.

Minix 3 показывает, что монолитная архитектура операционной системы не является единственно возможной. Ядро Minix 3 включает всего лишь 4000 строк исполняемого кода, в противовес «милионным» ядрам Windows, Linux, Mac OS X и FreeBSD. Остальная часть операционной системы, в том числе драйверы всех устройств (за исключением таймера), представляет собой совокупность компактных модульных процессов, работающих в пользовательском режиме. Деятельность каждого процесса четко ограничена; кроме того, жестко регламентировано и взаимодействие между процессами.

Minix версии 3 был анонсирован 24 октября 2005 года Эндрю Таненбаумом во время его речи на конференции по принципам операционных систем, организованном Association for Computing Machinery (ACM Symposium on Operating Systems Principles). Несмотря на то, что Minix всё ещё служит в качестве учебного примера в новой редакции книги Таненбаума и Вудхалла, он был переработан и доведён до годности к использованию в качестве надёжной операционной системы для микроконтроллеров и других встраиваемых систем. Он доступен на LiveCD, с которым можно работать без необходимости установки системы на жёсткий диск, а также в виде, совместимом со многими эмуляторами, такими, как Bochs, QEMU, VMware, VirtualBox и Virtual PC.

Эта операционная система является компактной. Части, которые работают в пользовательском режиме, разделены на небольшие модули и хорошо изолированы друг от друга. Например, каждый драйвер устройства выполняется как отдельный процесс пользовательского режима, и ошибка в драйвере (безусловно, источник большинства ошибок в любой операционной системе) не может полностью остановить её. В Minix, когда драйвер терпит крах, он автоматически перезапускается, не требуя пользовательского вмешательства, перезагрузки, и не затрагивая выполняющихся программ. Перечисленные особенности, микроядерная архитектура и другие аспекты значительно повышают надёжность этой операционной системы.

# Архитектура

Как можно увидеть на Рисунке 2, нижним уровнем системы является микроядро. Оно обрабатывает прерывания, осуществляет планирование и передачу сообщений. Также он поддерживает API из примерно 30 вызовов ядра, которые сервис или драйвер могут выполнить. Пользовательская программа не может делать такие вызовы. Вместо этого они могут делать системные вызовы стандарта POSIX, которые посылают сообщение сервисам. Вызов ядра выполняет такие функции, как настройка прерываний и копирование данных между адресными пространствами.

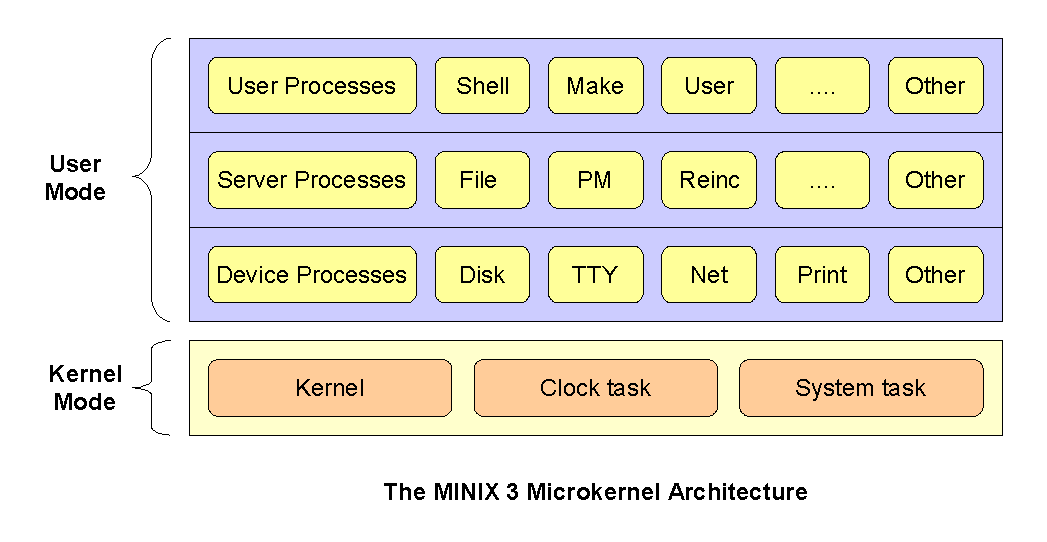


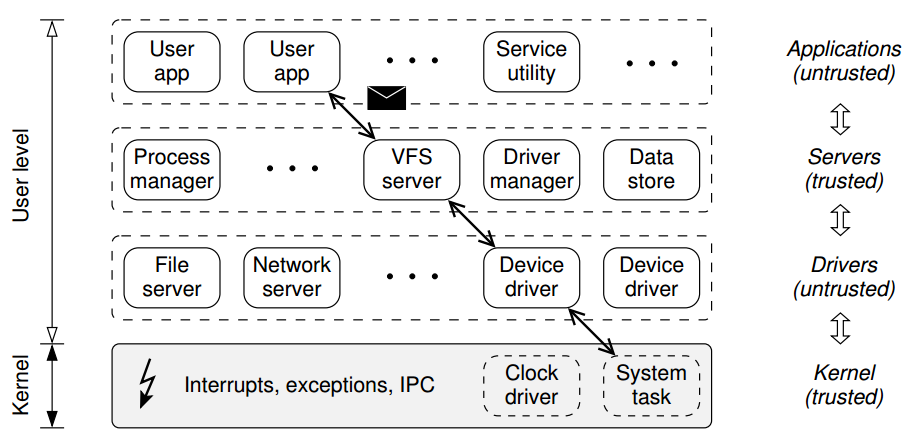
Рисунок 5. Архитектура Minix 3

На следующем уровне находятся драйвера устройств, каждый из которых работает как отдельный пользовательский процесс. Каждый из них управляет определёнными устройствами ввода-вывода, такими как диск или принтер. У драйвера нет доступа к портам ввода-вывода, и он не может давать прямые инструкции. Вместо этого, они должны сделать системный вызов со списком портов ввода-вывода и значениями для записи. Эта схема, доставляя небольшие накладные расходы (около 500 наносекунд), позволяет ядру проверять разрешения, так что, например, аудио-драйвер не сможет записывать данные на диск.

На следующем уровне находится сервер. Там размещены почти все функциональные возможности ОС. Пользовательские процессы работают с файлами, например, путём отправки сообщения на файловый сервер для открытия, закрытия, чтения и записи файлов. В свою очередь, файловый сервер получает ввод-вывод, предназначенный для отправки сообщения на драйвер диска, который фактически управляет диском.

Одним из ключевых серверов является сервер реинкарнации. Его задача состоит в опросе всех серверов и драйверов для периодической проверки их работоспособности. Если компоненты отвечают некорректно, то они попадают в бесконечный цикл, реинкарнационный сервер (который является родительским процессом для драйверов и серверов) уничтожает неисправный компонент и заменяет его на новую копию. В этом случае, система автоматически становится самовосстанавливающейся без вмешательства работающих программ.

В настоящее время сервер реинкарнации, файловый сервер, сервер процессов и микроядро являются частью доверенной вычислительной базы. Если любой из них падает, то вся система выходит из строя. Тем не менее, снижение вычислительной базы с 3-5 миллионов строк кода в Linux или 20000 строк Windows значительно повышает надежность системы.



# Микроядро Minix

В современных операционных системах различают следующие виды ядер.

**Наноядро (НЯ).** Крайне упрощённое 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.

**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. В микроядерной операционной системе можно, не прерывая ее работы, загружать и выгружать новые драйверы, файловые системы и т. д. Микроядерными являются ядра ОС Minix и GNU Hurd и ядро систем семейства BSD. Классическим примером микроядерной системы является Symbian OS. Это пример распространенной и отработанной микроядерной (a начиная c версии Symbian OS v8.1, и наноядерной) операционной системы.

**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 (так называемую 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.

**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. МнЯ является большинство Unix-систем и Linux. 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 объёму оперативной памяти.

**Модульное ядро (Мод. Я)** – современная, усовершенствованная модификация архитектуры МЯ. 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). Мод. Я удобнее для разработки, чем традиционные монолитные ядра. Они предоставляют программный интерфейс (API) для связывания модулей с ядром, для обеспечения динамической подгрузки и выгрузки модулей. Не все части ядра могут быть сделаны модулями. Некоторые части ядра всегда обязаны присутствовать в оперативной памяти и должны быть жёстко "вшиты" в ядро.

**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.

Наиболее тесно элементы микроядерной архитектуры и элементы монолитного ядра переплетены в ядре Windows NT. Хотя Windows NT часто называют микроядерной операционной системой, это не совсем так. Микроядро NT слишком велико (более 1 Мбайт), чтобы носить приставку "микро". Компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах. В то же время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром.

Учитывая природу систем основанных на монолитном ядре, где драйвер (который содержит, по словам самого Таненбаума, примерно в 3-7 раз большее количество ошибок, чем обычная программа) может обрушить всю систему, Minix 3 направлен на создание такой операционной системы, которая была бы «надежным, самовосстанавливающимся и многосерверным UNIX-клоном».

Чтобы достичь этого, код, выполняющийся в режиме ядра, должен быть минимальным, а файловый сервер, сервер процессов и каждый драйвер устройства должны выполняться как отдельные процессы в пользовательском режиме. Каждый драйвер тщательно контролируется частью системы, известной как сервер восстановления. Если драйвер не реагирует на пинги от сервера восстановления, то он закрывается и заменяется новой копией.

В монолитной системе ошибка в драйвере может привести к разрушению всего ядра, что значительно менее вероятно в Minix 3.

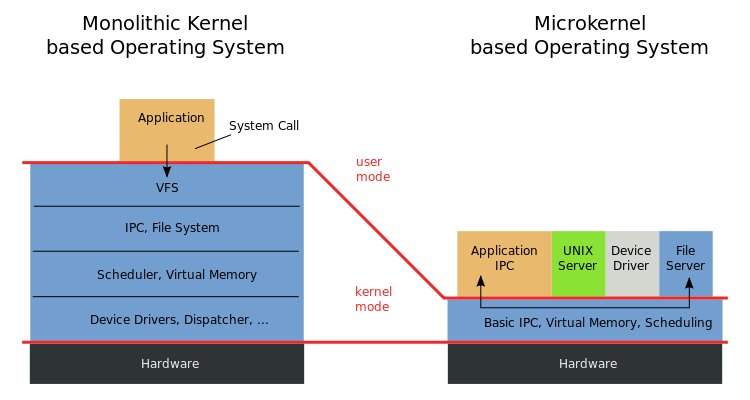


Рисунок 1. Структура операционных систем, основанных на монолитном ядре и микроядре, соответственно

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. Это можно представить, как показано на Рисунке

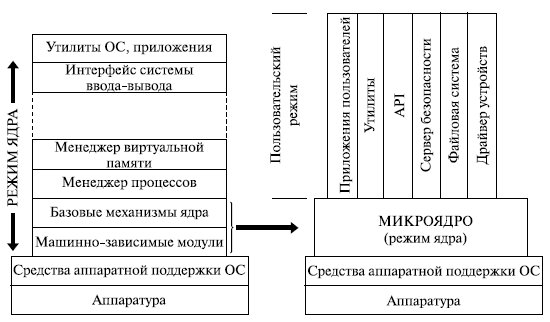


Рисунок 2. Переход к микроядерной архитектуре

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.



Рисунок 3. Клиент-серверная архитектура

Схема смены режимов при выполнении системного вызова в ОС с микроядерной архитектурой выглядит, как показано на рисунке 4. Из рисунка ясно, что выполнение системного вызова сопровождается четырьмя переключениями режимов (4 t), в то время как в классической архитектуре – двумя. Следовательно, производительность ОС с микроядерной архитектурой при прочих равных условиях будет ниже, чем у ОС с классическим ядром.



Рисунок 4. Обработка системного вызова в микроядерной архитектуре

В то же время признаны следующие достоинства микроядерной архитектуры:

* единообразные интерфейсы;
* простота расширяемости;
* высокая гибкость;
* возможность переносимости;
* высокая надежность;
* поддержка распределенных систем;
* поддержка объектно-ориентированных ОС.

По многим источникам вопрос масштабов потери производительности в микроядерных ОС является спорным. 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.

# Надежность Minix

Главной целью Minix 3 является надёжность. Ниже представлены некоторые из наиболее важных принципов повышения надёжности.

## Уменьшение размера ядра

Монолитные операционные системы, такие как Linux и FreeBSD, и такой гибрид как Windows содержат миллионы строк кода ядра. В отличие от Minix 3, имеющего около 6000 строк исполняемого кода ядра, в котором проще отследить проблему в коде.

## Отсечение ошибок

В монолитной ОС драйвера устройств располагаются в ядре. Это означает, что, когда новое периферийное устройство загружается, неизвестный и потенциально ненадежный код вставляется в ядро. Ошибка в коде драйвера может привести к разрушению системы. В MINIX 3 каждый драйвер работает в своём процессе. Драйверы не могут выполнять привилегированные команды: модификацию таблиц страниц, произвольный ввод-вывод или же запись в память по абсолютному адресу. Они вынуждены делать вызовы ядра для этого и оно проверит каждую команду на допустимость.

## Ограничение доступа драйверов к памяти

В монолитной ОС драйвер может написать любое слово в память и таким образом может случайно испортить пользовательскую программу. В MINIX 3, когда пользователь ожидает данные, к примеру, от файловой системы, она создает дескриптор, указывающий, кто имеет доступ и к каким именно адресам. Затем он передает индекс этого дескриптора в файловую систему, которая может передать его драйверу. Файловая система или драйвер запрашивает у ядра запись через этот дескриптор, что делает невозможной для них запись адресов вне буфера.

## Сохранение работоспособности дефектных указателей

Разыменование дефектного указателя в драйвере привело бы к уничтожению процесса драйвера, но никак не повлияло бы на всю систему в целом. Сервер реинкарнации автоматически перезапустит упавший драйвер. Для некоторых драйверов (таких как драйверы диска и сети) восстановление происходит незаметно для пользовательских процессов, для других же (таких как драйверы аудио и принтера), пользователь может получить уведомление. В монолитных системах разыменование дефектного указателя в ядре или драйвере приводит к разрушению системы.

## Ручные бесконечные циклы

Если драйвер попадет в бесконечный цикл, то планировщик будет постепенно понижать его приоритет пока тот не перейдет в режим ожидания. В конечном итоге, сервер реинкарнации увидит, что драйвер не отвечает на запросы статуса, поэтому он будет уничтожать и запускать драйвер циклически. В монолитной системе зацикливание драйвера может привести к зависанию системы.

## Ограничение повреждений от переполненного буфера

В Minix 3 используется фиксированная длина сообщений для внутренней связи, что исключает определенные проблемы переполнения и управления буфером. Также многие эксплойты работают переполняя буфер, для того, чтобы обмануть программу, возвращая из функции вызова, и, перезаписав с помощью стека, адрес возврата, указывающий на перезаполненный буфер. В Minix 3 данная атака не сработает, потому что команда и данные разделены пространством и только код (код для чтения) команды может быть выполнен.

## Ограничение доступа к функциям ядра

Драйверы устройств получают доступ к функциям ядра (такие как копирование данных с пользовательского адресного пространства) через вызовы к ядру. В Minix 3 ядро имеет битовую карту в которой для каждого драйвера указано какие вызовы ему разрешены. В монолитных системах каждый драйвер может вызвать любую функцию ядра, авторизовав её или нет.

## Ограничение доступа к портам ввода-вывода

Ядро также обладает таблицей, в которой для каждого драйвера указано, к какому из портов ввода-вывода он имеет доступ. В результате, драйвер может работать только со своими собственными портами ввода-вывода. В монолитных системах драйвер может получить доступ к портам ввода-вывода, принадлежащим другим устройствам.

## Ограничение связи с компонентами ОС

Не каждый драйвер или сервер должен общаться с другими такими же драйверами или серверами. Соответственно, процесс битовой карты определяет по каким направлениям каждый процесс может обращаться.

## Восстановление убитых или проблемных драйверов

Специальное устройство, называемое реинкарнационным сервером, периодически пингует каждый драйвер. Если драйвер умирает или ему не удается ответить на пинги, то реинкарнационный сервер автоматически заменяет его на новую копию. Обнаружение и замена нефункциональных драйверов происходит автоматически без какого-либо пользовательского вмешательства. Эта функция не работает для драйверов дисков в настоящее время, но в следующей версии система будет в состоянии восстановить каждый драйвер диска, который будет скрыт в RAM. Восстановление драйвера не повлияет на рабочий процесс.

## Встроенные прерывания и сообщения

Когда происходит прерывание, оно преобразуется в уведомление, отправляемое соответствующему драйверу. Если драйвер ожидает сообщения, то он получает прерывание немедленно, в противном случае получит уведомление в следующий раз. Эта схема устраняет вложенные прерывания и делает написание драйвера проще.

# Источники

1. Введение в Minix 3. Эндрю С. Таненбаум (Andrew S. Tanenbaum) Перевод: Павел Макаров

<http://www.iakovlev.org/index.html?p=886>

1. Операционная система Minix 3. Сайт русскоязычного сообщества разработчиков и пользователей операционной системы MINIX 3

<http://minix3.ru/index.html>

1. Minix 3. Общее обозрение

<http://citkit.ru/articles/337/>

1. Архитектура операционной системы

<http://www.intuit.ru/studies/courses/631/487/lecture/11048?page=4>