Развитие ядра операционной системы Linux[1]

Е.М. Новиков <novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Существующие исследования, которые посвящены анализу развития ядра операционной системы Linux, рассматривают ядро вместе с поставляемыми с ним загружаемыми модулями или некоторые конкретные подсистемы ядра. Целью данной работы является оценка развития ядра без загружаемых модулей, для чего предлагается метод определения границы между ними. Оценка развития дается для всех версий ядра операционной системы Linux, выпущенных за последние 7,5 лет. Также приводятся результаты классификации и распределение типовых ошибок, исправленных в ядре, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. Полученные результаты могут быть использованы при оценке актуальности и применимости различных методов и инструментов обеспечения качества программных систем.

Ключевые слова: операционная система; монолитное ядро; качество программной системы; анализ изменений.

DOI: 10.15514/ISPRAS-2017-29(2)-3

Для цитирования: Новиков Е.М. Развитие ядра операционной системы Linux. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 77-96. DOI: 10.15514/ISPRAS-2017-29(2)-3

1. Введение

В настоящее время ядро операционной системы (далее – ОС) Linux используется повсеместно огромным количеством пользователей на большом количестве аппаратных платформ от встроенных систем до суперкомпьютеров. По своей архитектуре ядро ОС Linux является монолитным – оно целиком загружается в оперативную память во время старта ОС и затем работает в одном адресном пространстве [1]. Как правило, в ядре реализуется основная функциональность, например, планирование процессорного времени, управление памятью и межпроцессным взаимодействием, обработка прерываний, поддержка разнообразных примитивов синхронизации. Изменить

[1] Исследование выполнено при финансовой поддержке РФФИ, проект «Инкрементальная статическая верификация подсистем монолитного ядра операционных систем» № 16-31-60097.

набор функций ядра можно путем динамической загрузки модулей или его статической перекомпиляции с другой конфигурацией и/или для другой целевой архитектуры. В обоих случаях модули становятся частью монолитного ядра.

Для ядра ОС Linux в виде загружаемых модулей могут быть собраны большинство драйверов устройств, файловых систем, сетевых протоколов, аудиокодеков и т.д. Необходимость в конкретных загружаемых модулях определяется аппаратным обеспечением, а также потребностями конкретных пользователей. На типовых персональных компьютерах может быть загружено несколько десятков или сотен модулей. Общее количество загружаемых модулей, которые поставляются вместе с ядром ОС Linux последних версий, составляет порядка 5 тысяч. В данной работе будет показано, что их совокупный размер во много раз превосходит размер ядра без загружаемых модулей.

Для оценки различных аспектов развития ядра ОС Linux постоянно проводятся исследования [2-7]. В данных исследованиях описывается, что происходило на протяжении нескольких лет в прошлом, и отмечается актуальность задач, которые предстоит решить в будущем. Анализируются общие аспекты разработки, такие как время выхода новых версий, размер кода, изменения между различными версиями и авторство данных изменений. В большинстве работ основное внимание уделяется качеству ядра ОС Linux. Изучаются как те ошибки, которые уже были обнаружены и исправлены до публикации соответствующей статьи, так и те ошибки, которые находили инструменты во время ее подготовки.

Насколько известно автору, до сих пор не было предпринято ни одной попытки оценить развитие ядра ОС Linux без загружаемых модулей – рассматривается ядро вместе с поставляемыми с ним загружаемыми модулями или некоторые конкретные подсистемы ядра, например файловые системы (если не оговорено иное, то далее в работе ядро без загружаемых модулей будет называться ядром, как это принято в англоязычной литературе). Это оправдано, например, с точки зрения критичности ошибок, поскольку архитектура ядра является монолитной, а значит, любая ошибка или сбой в одном из загруженных модулей может привести к серьезным последствиям для ядра, других загруженных модулей и пользовательских приложений. Однако на большинстве систем используется относительно малая часть загружаемых модулей. Как следствие, ядро ОС Linux является наиболее критичной частью ядра в совокупности со всеми загружаемыми модулями – ошибки в нем вероятнее всего затронут наибольшее количество пользователей, как на уровне ядра и различных загружаемых модулей, так и на уровне пользовательских приложений.

В данной работе предлагается метод определения границы между ядром и загружаемыми модулями, позволяющий оценить развитие ядра ОС Linux. В разделе 2 приведены общие сведения о развитии ядра вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет с целью последующей оценки развития ядра ОС Linux. Раздел 3 описывает предлагаемый метод и

оценивает развитие ядра за последние 7,5 лет. В разделе 4 приведены результаты классификации и распределение типовых ошибок, исправленных в ядре ОС Linux, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. В заключении делаются выводы на основе проведенного исследования.

В качестве базы для анализа развития ядра ОС Linux был выбран официальный Git-репозиторий[1], в котором собраны все версии, изменения и стабильные ветки для оригинального ядра и поставляемых с ним загружаемых модулей, начиная с версии 2.6.12, выпущенной 17 июня 2005 года. В рамках данной работы не рассматриваются ни сторонние модификации ядра, например, поддержка выполнения в реальном времени, ни сторонние загружаемые модули, такие как проприетарные драйверы видеокарт.

2. Развитие ядра ОС Linux вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет

В табл. 1 указаны общие сведения для всех версий ядра, начиная с 2.6.30 и заканчивая 4.8 – последняя версия ядра на момент написания данной статьи (всего 39 версий). В среднем на разработку одной версии уходил 71[2] день (около 10 недель). Каждая версия в среднем включала 12 тысяч изменений в ядре и загружаемых модулях, что соответствует в среднем 7,5 изменениям в час.

Табл. 1. Общие сведения о развитии ядра ОС Linux вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет
Table 1. General information about the development of the Linux kernel, along with the loadable modules for the last 7.5 years

Версия ядра	Дата выпуска	Количество дней на разработку	Количество изменений, тысяч	Среднее количество изменений в час
4.8	2 октября 2016	70	15[3]	8,7
4.7	24 июля 2016	70	13	8,0
4.6	15 мая 2016	63	15	9,7
4.5	13 марта 2016	63	13	8,7
4.4	10 января 2016	70	14	8,4
4.3	1 ноября 2015	63	13	8,8
4.2	30 августа 2015	70	15	8,8
4.1	21 июня 2015	70	13	7,7
4.0	12 апреля 2015	63	11	7,5
3.19	8 февраля 2015	63	14	9,0
3.18	7 декабря 2014	63	12	8,2
3.17	5 октября 2014	63	13	8,8
3.16	3 августа 2014	56	14	10
3.15	8 июня 2014	70	15	8,9
3.14	30 марта 2014	70	13	8,0
3.13	19 января 2014	77	13	7,2
3.12	3 ноября 2013	62	12	8,0
3.11	2 сентября 2013	64	12	7,7
3.10	30 июня 2013	63	15	9,7
3.9	28 апреля 2013	69	13	7,8
3.8	18 февраля 2013	70	14	8,1
3.7	10 декабря 2012	71	13	7,6
3.6	30 сентября 2012	71	11	6,5
3.5	21 июля 2012	62	12	7,9
3.4	20 мая 2012	63	12	7,8
3.3	18 марта 2012	74	11	6,4
3.2	4 января 2012	72	13	7,3
3.1	24 октября 2011	95	9,4	4,1
3.0	21 июля 2011	64	9,8	6,4
2.6.39	18 мая 2011	65	11	7,1
2.6.38	14 марта 2011	69	10	6,3
2.6.37	4 января 2011	76	12	6,7
2.6.36	20 октября 2010	80	10	5,3
2.6.35	1 августа 2010	77	10	5,7
2.6.34	16 мая 2010	81	10	5,2
2.6.33	24 февраля 2010	84	12	5,8
2.6.32	2 декабря 2009	84	12	5,9
2.6.31	9 сентября 2009	92	12	5,3
2.6.30	9 июня 2009	78	13	6,9

[1] git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git.
[2] Здесь и далее выполнено округление до 2 значащих цифр.
[3] Количество изменений в данной ячейке посчитано с помощью команды «git rev-list --count v4.7..v4.8». В других ячейках данного столбца – аналогично для соответствующих версий.

Для всех версий ядра ОС Linux, выпущенных за последние 7,5 лет, с помощью утилиты cloc 1.70[1] для ядра вместе с поставляемыми с ним загружаемыми модулями было посчитано количество файлов (рис. 1) и количество строк кода (рис. 2), включая комментарии и пустые строки, на различных языках программирования. Общее количество файлов увеличилось с 25 тысяч до 45 тысяч – на 83%, а количество строк кода – с 10 миллионов до 20 миллионов – на 95%, что в среднем соответствует приросту на 7,5 файлов и 3,6 тысяч строк кода в день.

Графики наглядно демонстрируют, что основной вклад как в количество файлов, так и в количество строк кода вносили файлы на языке программирования Си: в среднем 33 тысячи (91%) и 15 миллионов (97%) соответственно. Также за счет них происходил и основной рост ядра ОС Linux – на 91% и на 97% соответственно. Файлы на языке ассемблера в среднем вносили вклад 1,3 тысячи файлов (4%) и 0,36 миллионов строк кода (2%). Их

[1] https://github.com/AlDanial/cloc.

вклад в рост – 3% и 2% соответственно. Файлы на других языках программирования, в основном сценарных, описывающих сборку и представляющих различные вспомогательные утилиты, в среднем вносили вклад 2 тысячи файлов (5%) и 0,14 миллионов строк кода (1%) соответственно. Их влияние на рост составило 6% и 1% соответственно.

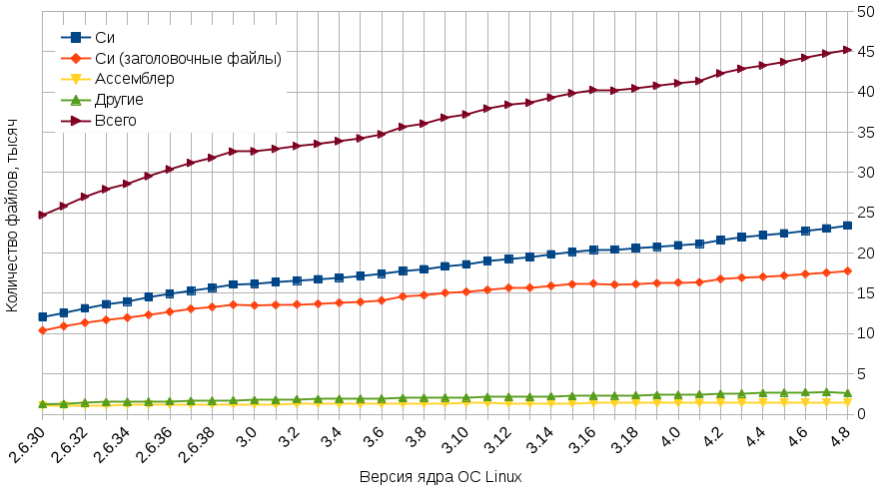

Рис. 1. Количество файлов на различных языках программирования в ядре ОС Linux вместе с поставляемыми с ним загружаемыми модулями
Fig. 1. The number of files in different programming languages in the Linux kernel, along with the loadable modules

3. Развитие ядра ОС Linux за последние 7,5 лет

В данном разделе предлагается метод, позволяющий оценить развитие ядра ОС Linux. Представляются результаты данной оценки с разбиением по основным подсистемам и без него для всех версий ядра, которые были выпущены за последние 7,5 лет.

3.1 Определение границы между ядром ОС Linux и загружаемыми модулями

Основной сложностью при оценке развития ядра ОС Linux является определение границы между ядром и загружаемыми модулями. Данная граница зависит от версии ядра, а также от целевой архитектуры и конфигурационных опций, с которыми собирается ядро.

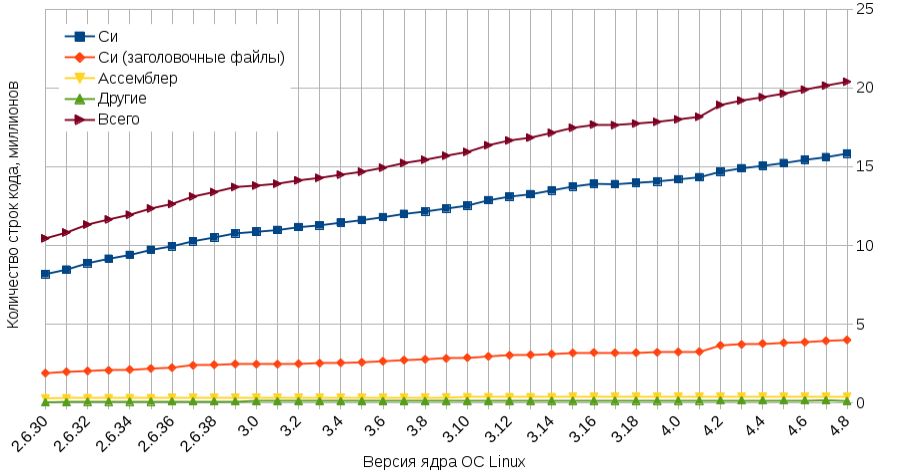

Рис. 2. Количество строк кода на различных языках программирования в ядре ОС Linux вместе с поставляемыми с ним загружаемыми модулями
Fig. 2. Number of lines of code in different programming languages in the Linux kernel, along with the loadable modules

Проводить границу для каждой исследуемой версии ядра предлагается независимо, поскольку между ними могут быть достаточно большие различия, примеры чего приведены в предыдущем разделе.

Для конфигурирования предлагается использовать набор опций, получаемых для стандартной конфигурационной цели *allmodconfig*. В этом случае все, что можно собрать в виде загружаемых модулей, собирается в таком виде. При использовании другого набора опций можно получить другое подмножество исходного кода ядра, например, можно отключить или включить поддержку определенной функциональности, использовать альтернативные реализации или даже включить некоторые загружаемые модули в состав ядра.

Проводить границу предлагается по файлам, а не по строкам кода. Это позволяет включить в ядро хотя бы частично исходный код альтернативных реализаций (активируемых при выборе соответствующих конфигурационных опций), который помещен в файлы с исходным кодом основных реализаций. При этом в состав ядра не будет включен исходный код альтернативных реализаций, который полностью помещен в отдельные файлы. Например, это относится к различным механизмам распределения памяти. Для *allmodconfig* в состав ядра включается файл *mm/slub.c*, соответствующий механизму распределения памяти *SLUB*, но не включаются файлы *mm/slab.c* и *mm/slob.c*, соответствующие механизмам распределения памяти *SLAB* и *SLOB*[1]. Также

[1] Подробнее о различных механизмах распределения памяти в ядре ОС Linux 1)2)можно узнать, например, из слайдов следующей презентации: http://events.linuxfoundation.org/sites/events/files/slides/slaballocators.pdf.

благодаря такому способу проведения границы при подсчете количества строк кода включаются комментарии и пустые строки, которые традиционно учитываются при оценке развития ядра вместе с загружаемыми модулями.

Предлагается выбирать все файлы с исходным кодом на языке программирования Си и языке ассемблера, которые после компиляции и компоновки включаются в состав файла *vmlinux.o*, представляющего образ ядра ОС Linux без загружаемых модулей. Заголовочные файлы на языке программирования Си, а также файлы на других языках программирования предлагается не учитывать. Заголовочные файлы преимущественно используются для описания программного интерфейса, а не его реализации. Более того, заголовочные файлы включаются как в файлы с исходным кодом ядра, так и в файлы с исходным кодом загружаемых модулей. Аналогично файлы на всех других языках программирования достаточно сложно отнести к ядру и/или загружаемым модулям. Однако в предыдущем разделе показано, что их совокупный вклад в общее количество строк кода незначительный, около 1%, поэтому допустимо исключить их при сравнении.

Собирать ядро ОС Linux предлагается для архитектуры *x86_64*. В следующем подразделе будет показано, что это является наиболее существенным недостатком предложенного метода определения границы между ядром и загружаемыми модулями.

Результаты оценки развития ядра ОС Linux за последние 7,5 лет с использованием предложенного метода представлены в следующих двух подразделах. Данный метод также был использован при определении границы ядра для изменений в стабильных ветках ядра, однако для этого было сделано дополнительное предположение (раздел 4).

3.2 Развитие ядра ОС Linux

На рис. 3 и рис. 4 для ядра ОС Linux представлены количество файлов и количество строк кода, включая комментарии и пустые строки, на языке программирования Си и языке ассемблера для всех версий ядра, которые были выпущены за последние 7,5 лет.

Общее количество файлов увеличилось с 1,1 тысячи до 2,2 тысяч – на 90%, а количество строк кода – с 0,7 миллионов до 1,4 миллионов – на 105%. Это показывает, что ядро ОС Linux развивалось немного интенсивнее ядра вместе с загружаемыми модулями.

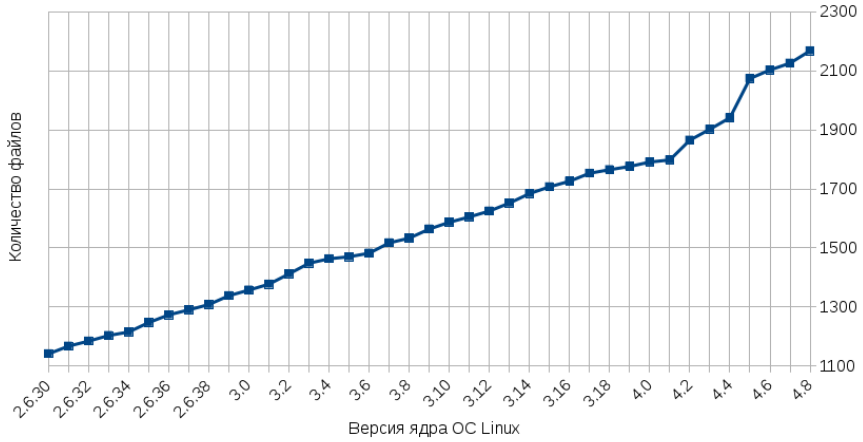

Рис. 3. Количество файлов на языке программирования Си и языке ассемблера в ядре ОС Linux

Fig. 3. The number of files in the C programming language and the assembler language in the Linux kernel

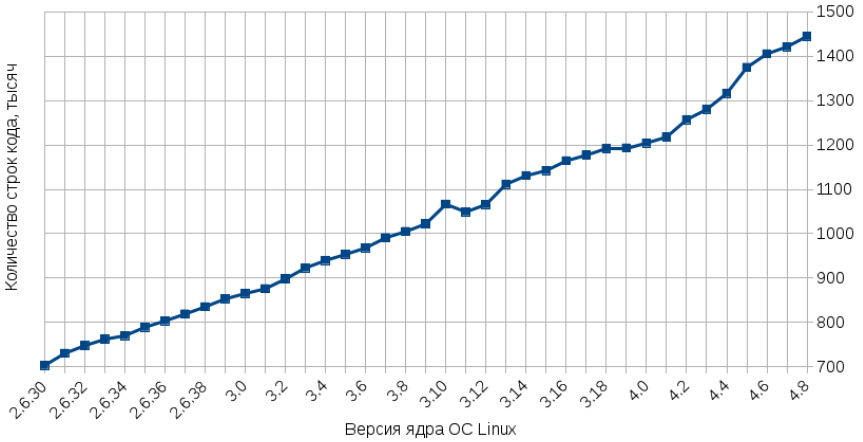

Рис. 4. Количество строк кода на языке программирования Си и языке ассемблера в ядре ОС Linux

Fig. 4. Number of lines of code in C programming language and assembler language in Linux kernel

Среднее количество файлов на языке программирования Си для ядра ОС Linux составило 1,5 тысяч (97%), на языке ассемблера – 43 (3%). Среднее количество строк кода – 1 миллион (99%) и 6,3 тысячи (1%) соответственно. Таким образом, при использовании предложенного метода определения границы ядра ОС Linux получилось, что количество файлов и количество строк кода на языке

ассемблера в ядре по сравнению с ядром вместе с загружаемыми модулями меньше в 30 и 58 раз соответственно, в то время как для языка программирования Си данные соотношения составляют 21 и 15. Однако следует учесть, что при определении границы ядра не учитывался исходный код для других архитектур, кроме *x86_64*, притом, что файлы на языке ассемблера достаточно редко используются в загружаемых модулях – большинство из них попадает в ядро для соответствующей архитектуры и/или конфигурации.

В среднем по количеству файлов и строк кода на языке программирования Си и языке ассемблера ядро ОС Linux составило по 8,4% относительно ядра вместе с загружаемыми модулями. Таким образом, по размеру загружаемые модули, которые поставляются вместе с ядром, в 11 раз больше ядра.

3.3 Развитие основных подсистем ядра ОС Linux

Помимо оценки развития ядра ОС Linux в целом также полезно оценить развитие его основных подсистем. Основными подсистемами ядра в данной работе считаются следующие 12 подсистем в соответствии со структурой директорий с исходным кодом[1]:

- *arch* – аппаратные платформы;
- *block* – блочный уровень;
- *crypto* – криптографический интерфейс;
- *drivers* – поддержка различных типов устройств;
- *fs* – поддержка файловых систем;
- *init* – общие настройки ядра;
- *ipc* – межпроцессное взаимодействие;
- *kernel* – основная часть ядра;
- *lib* – вспомогательные библиотеки;
- *mm* – управление памятью;
- *net* – сеть;
- *security* – модели безопасности.

Единственной основной подсистемой ядра ОС Linux, в которой содержится достаточно много исходного кода на языке ассемблера, является *arch*. Поскольку в предыдущем подразделе было показано, что в ядре файлов и строк кода на языке ассемблера всего 3% и 1% соответственно, далее они не рассматриваются отдельно от файлов и строк кода на языке программирования Си.

Аналогично рис. 3 и рис. 4 на рис. 5 и рис. 6 представлены количество файлов и количество строк кода, включая комментарии и пустые строки, на языке

[1] Подсистемы *certs*, *firmware* и *usr* не рассматриваются, поскольку они содержат менее 250 строк кода на языке ассемблера для всех версий ядра ОС Linux.

программирования Си и языке ассемблера для всех основных подсистем и версий ядра ОС Linux, которые были выпущены за последние 7,5 лет. На графиках не представлены основные подсистемы *init* и *ipc*, поскольку они включают до 13 файлов и до 8,9 тысяч строк кода на языке программирования Си.

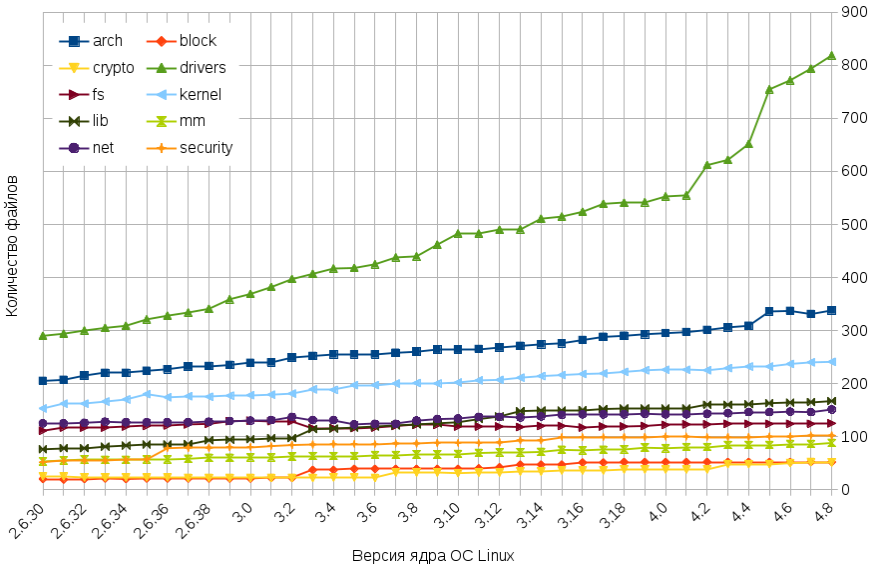

Рис. 5. Количество файлов на языке программирования Си и языке ассемблера в основных подсистемах ядра ОС Linux
Fig. 5. The number of files in the programming language C and assembly language in the core subsystems of the Linux kernel

Графики показывают, что основной вклад в ядро ОС Linux вносила основная подсистема *drivers*. Также преимущественно за счет нее происходил рост ядра. По количеству файлов лидерами также являлись основные подсистемы *arch* и *kernel*, по количеству строк кода – *kernel*, *arch* и *net*.

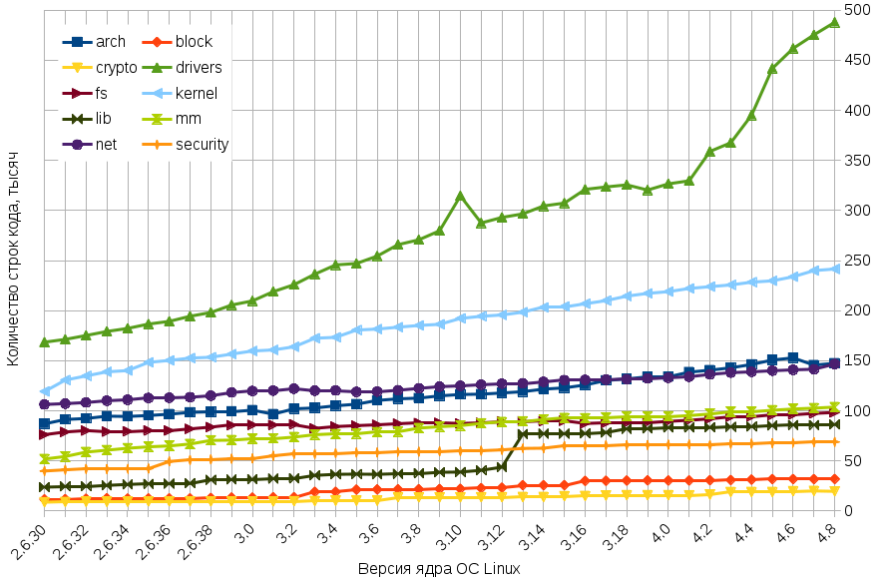

Рис. 6. Количество строк кода на языке программирования Си и языке ассемблера в основных подсистемах ядра ОС Linux
Fig. 6. Number of lines of code in the C programming language and assembler language in the core subsystems of the Linux kernel

4. Классификация и распределение типовых ошибок, исправленных в ядре ОС Linux

В соответствии с доработанной методикой выявления и классификации типовых ошибок [4] в данной работе был проведен анализ изменений, сделанных в стабильных ветках ядра за последние 2 месяца 2015 года.

Оказалось, что в указанный период времени изменения были сделаны только в 11 из 58 стабильных веток ядра. Это объясняется тем, что большая часть стабильных веток ядра более не поддерживается, а также тем, что несколько новых версий ядра еще не были выпущены, а значит, не было создано соответствующих стабильных веток ядра.

Вообще говоря, для каждого изменения может быть своя граница ядра ОС Linux. Для того чтобы не вычислять ее 3210 раз (общее количество изменений в стабильных ветках ядра ОС Linux за последние 2 месяца 2015 года), что требует достаточно много времени, было использовано дополнительное предположение, что для каждой стабильной ветки ядра граница не изменяется. Это предположение оправдано, поскольку в стабильных ветках ядра достаточно редко делают существенные изменения, в том числе переименования, перемещения и удаления файлов. Соответственно, для каждой стабильной

ветки граница ядра ОС Linux определялась только для последнего изменения в ней, а затем использовалась для всех остальных изменений в данной ветке.

Табл. 2. Количество изменений с уникальными заголовками в стабильных ветках ядра ОС Linux
Table 2. Number of changes with unique headers in stable Linux kernel branches

Название стабильной ветки ядра ОС Linux	Общее количество изменений	Количество изменений в ядре ОС Linux	Количество изменений с уникальными заголовками
linux-4.3.y	201	31 (15%)	31 (100%)
linux-4.2.y	647	121 (19%)	94 (78%)
linux-4.1.y	510	98 (19%)	10 (10%)
linux-3.18.y	370	81 (22%)	31 (38%)
linux-3.16.y	412	70 (17%)	20 (29%)
linux-3.14.y	266	60 (23%)	12 (20%)
linux-3.12.y	212	43 (20%)	7 (16%)
linux-3.10.y	184	42 (23%)	2 (5%)
linux-3.4.y	67	11 (16%)	9 (82%)
linux-3.2.y	300	70 (23%)	16 (23%)
linux-2.6.32.y	41	22 (54%)	1 (5%)
Всего	**3210**	**649 (20%)**	**233 (36%)**

Некоторые изменения, которые сделаны в нескольких стабильных ветках, могут быть достаточно сильно похожи. Например, к таким изменениям относятся исправления одних и тех же ошибок. Сами изменения могут отличаться, однако предполагается, что их заголовки одни и те же. В табл. 2 представлено количество изменений с уникальными заголовками в стабильных ветках ядра ОС Linux за последние 2 месяца 2015 года. В среднем таких изменений было 36%, что намного меньше, чем в драйверах – 85% [4]. Это показывает то, что для ядра существенно больше изменений переносится в различные стабильные ветки ядра, что косвенно служит показателем большей значимости ядра, в том числе с точки зрения исправления ошибок.

При построении табл. 3 все изменения с уникальными заголовками были автоматически отфильтрованы с использованием доработанных подходов из [4]. Затем 170 изменений, в которых исправляются типовые ошибки, были проанализированы вручную. Результаты данного анализа представлены в табл. 4.

Табл. 3. Количество изменений, в которых исправляются типовые и нетиповые ошибки, а также в которых не исправляются ошибки, в стабильных ветках ядра ОС Linux (отфильтровано автоматически)
Table 3. The number of changes that fix typical and non-standard errors, as well as in which errors are not fixed, in stable Linux kernel branches (automatically filtered)

Название стабильной ветки ядра ОС Linux	Количество изменений, в которых исправляются типовые ошибки	Количество изменений, в которых исправляются нетиповые ошибки	Количество изменений, в которых не исправляются ошибки
linux-4.3.y	21 (68%)	7 (23%)	3 (10%)
linux-4.2.y	70 (74%)	21 (22%)	3 (3%)
linux-4.1.y	9 (90%)	1 (10%)	–
linux-3.18.y	21 (68%)	6 (19%)	4 (13%)
linux-3.16.y	18 (90%)	1 (5%)	1 (5%)
linux-3.14.y	9 (75%)	2 (17%)	1 (8%)
linux-3.12.y	3 (43%)	1 (14%)	3 (43%)
linux-3.10.y	2 (100%)	–	–
linux-3.4.y	6 (67%)	3 (33%)	–
linux-3.2.y	11 (69%)	3 (19%)	2 (13%)
linux-2.6.32.y	–	–	1 (100%)
Всего	**170 (73%)**	**45 (19%)**	**18 (8%)**

Табл. 4. Количество изменений, в которых исправляются типовые и нетиповые ошибки, а также в которых не исправляются ошибки, в стабильных ветках ядра ОС Linux (проанализировано вручную)
Table 4. The number of changes that fix typical and non-standard errors, as well as in which errors are not fixed, in stable Linux kernel branches (analyzed manually)

Название стабильной ветки ядра ОС Linux	Количество изменений, в которых исправляются типовые ошибки	Количество изменений, в которых исправляются нетиповые ошибки	Количество изменений, в которых не исправляются ошибки
linux-4.3.y	12 (57%)	9 (43%)	–
linux-4.2.y	42 (60%)	24 (34%)	4 (6%)
linux-4.1.y	3 (33%)	3 (33%)	3 (33%)
linux-3.18.y	15 (71%)	5 (24%)	1 (5%)
linux-3.16.y	5 (28%)	13 (72%)	–
linux-3.14.y	6 (67%)	3 (33%)	–
linux-3.12.y	–	3 (100%)	–
linux-3.10.y	–	2 (100%)	–
linux-3.4.y	3 (50%)	3 (50%)	–
linux-3.2.y	7 (64%)	4 (36%)	–
linux-2.6.32.y	–	–	–
Всего	**93 (55%)**	**69 (41%)**	**8 (5%)**

В итоге получилось, что среди всех 233 изменений с уникальными заголовками в стабильных ветках ядра в ядре ОС Linux за последние 2 месяца 2015 года:

- 26 (11%) изменений соответствуют расширению функциональности;
- 114 (49%) – исправлению нетиповых ошибок;
- 93 (40%) – исправлению типовых ошибок.

По сравнению с драйверами [4] количество изменений, в которых:

- расширяется функциональность намного меньше: 11% вместо 21%;
- исправляются нетиповые ошибки – практически столько же: 49% вместо 52%;
- исправляются типовые ошибки – намного больше: 40% вместо 27%.

Таким образом, по сравнению с драйверами для ядра ОС Linux в стабильных ветках ядра исправляется больше ошибок, причем среди них больше типовых ошибок.

Все изменения, в которых исправлялись типовые ошибки, были классифицированы на основе методики из [4]. Предложенная в [4] классификация была изменена и дополнена. Класс *specific* был переименован в *linux* для того, чтобы подчеркнуть, что данные ошибки являются специфичными для ядра ОС Linux. Были изменены несколько существующих подклассов:

- *generic:buffer_overflow* – на *generic:buf overflow*;
- *generic:resource* – на *generic:memory*;
- *generic:null_ptr_deref* – на *generic:null ptr dereference*;
- *generic:int_overflow* – на *generic:int*, дополнительно к этому подклассу помимо переполнений целых чисел были отнесены такие ошибки, как преобразования знаковых целых чисел к беззнаковым;
- *specific:check_params* – на *specific:param*;
- *specific:resource* – на *specific:res*;
- *specific:check_ret_val* – на *specific:ret*;
- *specific:lock* – на *specific:one thread*.

Также было добавлено несколько новых подклассов:

- *generic:incorrect read/write* – общие ошибки чтения/записи в том случае, если конкретный подкласс ошибки, например, *generic:buf overflow* или *generic:null ptr dereference*, не удается точно

определить;

- *generic:infinite loop* – по сути данные типовые ошибки представляют собой непреднамеренное зависание программы;
- *generic:invalid cast* – некорректное преобразование переменных к типу другого размера;
- *generic:info leak* – ошибки данного подкласса происходят, когда значения в памяти не затираются и могут быть прочитаны кем-то, кто не имеет на это прав;
- *generic:always true/false condition* – использование условия, которое истинно или ложно на всех возможных путях выполнения;
- *generic:dead code* – недостижимый код;
- *generic:err ptr dereference* – некорректное разыменование указателей из определенного диапазона;
- *linux:info leak* – аналогично *generic:info leak*, но ошибки данного подкласса связаны с конкретным программным интерфейсом, например, с *copy_to_user()*.

В табл. 5 представлены результаты классификации и распределение типовых ошибок, исправленных в ядре ОС Linux, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. Общее количество представителей всех подклассов типовых ошибок больше на 2 общего количества типовых ошибок из табл. 4, поскольку два изменения исправляли сразу две типовые ошибки.

По сравнению с драйверами для ядра ОС Linux в стабильных ветках исправлялось существенно больше общих ошибок: 47% вместо 29%, а также таких ошибок, как состояния гонки и взаимные блокировки: 32% вместо 20%. При этом ошибок, связанных с использованием специфичного программного интерфейса ядра ОС Linux, наоборот намного меньше: 21% вместо 50%. Также заметны следующие существенные отличия в распределении типовых ошибок по пересекающемуся списку подклассов:

- *generic:buf overflow* – 20% в ядре ОС Linux против 8% в драйверах (данное различие может быть даже больше с учетом того, что ошибки из подкласса *generic:incorrect read/write* могут относиться к данному подклассу);
- *generic:memory* – 13% и 24%;
- *generic:null ptr dereference* – 13% и 30%;
- *generic:syntax* – 4% и 14%;
- *linux:param* – 40% в ядре ОС Linux против 14% в драйверах;
- *linux:context* – 20% и 11%.

Табл. 5. Классификация и распределение типовых ошибок, исправленных в ядре ОС Linux
Table 5. Classification and distribution of typical errors fixed in the Linux kernel

Класс типовых ошибок	Подкласс типовых ошибок	Количество исправленных типовых ошибок	Суммарный процент от общего
generic (45 – 47%)	*buf overflow*	9 (20%)	20%
	memory	6 (13%)	33%
	null ptr dereference	6 (13%)	47%
	incorrect read/write	5 (11%)	58%
	int	5 (11%)	69%
	uninit	2 (4%)	73%
	infinite loop	2 (4%)	78%
	invalid cast	2 (4%)	82%
	info leak	2 (4%)	87%
	syntax	2 (4%)	91%
	always true/false condition	2 (4%)	96%
	dead code	1 (2%)	98%
	err ptr dereference	1 (2%)	100%
sync (30 – 32%)	*race*	28 (93%)	93%
	deadlock	2 (7%)	100%
linux (20 – 21%)	*param*	8 (40%)	40%
	context	4 (20%)	60%
	res	4 (20%)	80%
	info leak	2 (10%)	90%
	ret	1 (5%)	95%
	one thread	1 (5%)	100%

В табл. 6 представлено количество типовых ошибок, исправленных в основных подсистемах ядра ОС Linux. Общее количество исправленных типовых ошибок в данной таблице больше на 1 общего количества типовых ошибок из табл. 4, поскольку одно изменение затронуло сразу две подсистемы.

Табл. 6. Количество типовых ошибок, исправленных в основных подсистемах ядра ОС Linux
Table 6. Number of typical errors fixed in the core subsystems of the Linux kernel

Основная подсистема	Количество исправленных типовых ошибок	Суммарный процент от общего
net	27 (29%)	29%
drivers	22 (23%)	52%
kernel	12 (13%)	65%

arch	8 (9%)	73%
mm	6 (6%)	80%
fs	6 (6%)	86%
block	5 (5%)	91%
lib	3 (3%)	95%
security	2 (2%)	97%
crypto	2 (2%)	99%
ipc	1 (1%)	100%

5. Заключение

Метод определения границы между ядром ОС Linux и загружаемыми модулями, предложенный в данной работе, позволил оценить развитие ядра для всех версий, выпущенных за последние 7,5 лет. В результате были получены следующие наиболее существенные выводы:

- Ядро ОС Linux развивалось немного интенсивнее ядра вместе с поставляемыми с ним загружаемыми модулями. За 7,5 лет общее количество файлов увеличилось на 90% и составило 2,2 тысячи, а количество строк кода – на 105%, до 1,4 миллионов. Для ядра вместе с загружаемыми модулями данные показатели следующие: на 83% (45 тысяч) и на 95% (20 миллионов) соответственно. По размеру загружаемые модули, которые поставляются вместе с ядром, в 11 раз больше ядра.
- Основной вклад в состав и рост ядра ОС Linux вносила основная подсистема *drivers*. За ней следовали основные подсистемы *kernel*, *arch* и *net*.

Также данный метод позволил проанализировать типовые ошибки, исправленные в ядре ОС Linux, на основе изменений, сделанных в стабильных ветках ядра за последние 2 месяца 2015 года. Основные выводы следующие:

- По сравнению с драйверами для ядра ОС Linux существенно больше изменений переносится в различные стабильные ветки ядра, что косвенно служит показателем большей значимости ядра, в том числе с точки зрения исправления ошибок.
- По сравнению с драйверами для ядра ОС Linux количество изменений, в которых расширяется функциональность намного меньше (11% вместо 21%), в которых исправляются нетиповые ошибки – практически столько же (49% вместо 52%), в которых исправляются типовые ошибки – намного больше (40% вместо 27%). Таким образом, по сравнению с драйверами для ядра ОС Linux в стабильных ветках ядра исправляется больше ошибок, причем среди них больше типовых ошибок.
- По сравнению с драйверами для ядра ОС Linux в стабильных ветках исправлялось существенно больше общих ошибок: 47% вместо 29%, а

также таких ошибок, как состояния гонки и взаимные блокировки: 32% вместо 20%. При этом ошибок, связанных с использованием специфичного программного интерфейса ядра ОС Linux, наоборот намного меньше: 21% вместо 50%. Таким образом, для ядра ОС Linux в первую очередь важно искать ошибки первых двух классов, особенно с учетом того, что их последствия могут быть существенно хуже, чем последствия ошибок третьего класса. Для первого класса наибольшее количество ошибок связаны с некорректными чтением/записью и управлением памяти, для второго – с состояниями гонки.

- На половину основных подсистем, *net*, *drivers*, *kernel*, *arch*, *mm* и *fs*, пришлось 86% от общего количества типовых ошибок. В целом это коррелирует с их размером, хотя, например, по количеству строк кода основная подсистема *net* меньше основной подсистемы *drivers* в 2,3 раза в среднем за 7,5 лет в то время, как по количеству исправлений типовых ошибок в 1,2 раз больше.

Данные выводы демонстрируют развитие ядра ОС Linux, а также указывают на наиболее актуальные направления в области обеспечения качества данного проекта. Аналогичные исследования возможно провести и для ядра других ОС, особенно для тех, для которых репозитории с исходным кодом находятся в открытом доступе.

Что касается основного недостатка данного исследования, в первую очередь следует предложить подход к определению исходного кода, который входит в ядро ОС Linux для различных целевых архитектур. Скорее всего, это не повлияет существенно на классификацию и распределение типовых ошибок, но может сильно сказаться на размере некоторых основных подсистем ядра, в первую очередь, *arch*. Также следует оценить размер неучтенного исходного кода альтернативных реализаций, активируемых при выборе соответствующих конфигурационных опций.

Список литературы

[1]. В.Е. Карпов, К.А. Коньков. Основы операционных систем. Курс лекций. Учебное пособие. М.: Интернет-университет информационных технологий, 536 стр., 2005.

[2]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.

[3]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.

[4]. В.С. Мутилин, Е.М. Новиков, А.В. Хорошилов. Анализ типовых ошибок в драйверах операционной системы Linux. Труды ИСП РАН, т. 22, стр. 349-374, 2012. DOI: 10.15514/ISPRAS-2012-22-19

[5]. L. Lu, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, S. Lu. A study of Linux file system evolution. In Proceedings of the 11th USENIX conference on File and Storage Technologies (FAST'13), pp. 31-44, 2013.
[6]. N. Palix, G. Thomas, S. Saha, C. Calvès, G. Muller, J. Lawall. Faults in Linux 2.6. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 2, 2014.
[7]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.

Evolution of the Linux kernel

E.M. Novikov <novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Existing research analyzing evolution of the Linux kernel considers the kernel together with loadable modules delivered with it or some specific subsystems of the kernel. The aim of this paper is to evaluate evolution of the kernel without loadable modules. It proposes a method for determining boundaries between them and evaluates evolution for all versions of the Linux kernel, released over the past 7.5 years. Also the paper presents a classification and a distribution of typical bugs that were fixed in the kernel, based on analysis of changes that have been made to stable branches of the kernel during the last 2 months of 2015. One can use the obtained results for evaluation of applicability of various methods and tools for software quality assurance.

Keywords: operating system; monolithic kernel; software quality; changes analysis.

DOI: 10.15514/ISPRAS-2017-29(2)-3

For citation: Novikov E.M. Evolution of the Linux kernel. Trudy ISP RAN/Proc. ISP RAS, volume 29, issue 2, 2017, pp. 77-96 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-3

References

[1]. V.E. Karpov, K.A. Kon'kov. Fundamentals of operating systems. Kurs lekcij. Uchebnoe posobie [Lectures. Study material]. M.: Internet-universitet informacionnyh tehnologij, 536 p., 2005 (in Russian).
[2]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.
[3]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.
[4]. V.S. Mutilin, E.M. Novikov, A.V. Horoshilov. Analysis of typical faults in Linux operating system drivers. Trudy ISP RAN/Proc. ISP RAS, volume 22, pp. 349-374, 2012 (in Russian). DOI: 10.15514/ISPRAS-2012-22-19
[5]. L. Lu, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, S. Lu. A study of Linux file system evolution. In Proceedings of the 11th USENIX conference on File and Storage Technologies (FAST'13), pp. 31-44, 2013.

[6]. N. Palix, G. Thomas, S. Saha, C. Calvès, G. Muller, J. Lawall. Faults in Linux 2.6. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 2, 2014.
[7]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.