Проблемы совместимости Linux-систем

А. И. Гриневич, Д. А. Марковцев, В. В. Рубанов

Институт системного программирования РАН (ИСП РАН),
Б. Коммунистическая, 25, Москва, Россия
E-mail: {tanur,day,vrub}@ispras.ru

Аннотация

Статья рассказывает о проблемах совместимости и переносимости приложений между различными клонами Linux. Вскрываются причины такого положения дел, и дается краткое описание разнообразия современных Linux-систем. Возможным выходом из сложившейся ситуации является стандартизация интерфейсов между операционной системой и приложениями и строгое следование выработанным стандартам, что Linux-сообщество сегодня и пытается реализовать, разрабатывая и внедряя стандарт Linux Standard Base.

1. Введение

Для решения инфраструктурной проблемы слабой совместимости различных версий Unix, в 1985 году в рамках организации IEEE началась работа над стандартом, обеспечивающим переносимость программного обеспечения (ПО) между ними. В результате к 1990 увидел свет стандарт IEEE 1003, также получивший название POSIX [1]. Это стандарт на программные интерфейсы (API) и команды Unix-клонов.

Для игроков рынка Unix задача унификации привела к сложным политическим проблемам — ведь любое решение, любой выбор между альтернативными вариантами для достижения согласия ведёт к тому, что более стандартным признается решение одного производителя по сравнению с решением другого. В результате стандарт изобилует двусмысленными утверждениями («в данном случае возможен один из двух альтернативных вариантов поведения») и недомолвками («стандарт не специфицирует поведение функции в этом случае»). Различные попытки проанализировать причины поражения Unix в целом приводят к одной основной причине — его фрагментации. Игроки рынка Unix конкурировали не только с другими типами операционных систем (ОС), но и друг с другом, вводя частные расширения и закрытые интерфейсы, тем самым ограничивая круг возможных приложений одним клоном.

Зародившаяся в начале 90-х годов система Linux, вобравшая в себя большинство кода созданного в рамках движения GNU, впитавшая основные идеи Unix, благодаря открытости и независимости, явилась универсальным компромиссом. Код реализовывался с нуля, не опираясь на какую-либо реализацию, а только на текст самого стандарта POSIX. Тем самым система получилась изначально во многом POSIX-совместимой. Независимость позволила объединить усилия различных игроков рынка Unix, в борьбе за

«возврат» упущенного рынка ОС для ПК. Общий код, не являющийся чьей-либо собственностью, оставляет возможность для совместного участия участников сообщества Linux в конкурентной борьбе. Для начала, Linux начал активно занимать ниши в рынках для мэйнфреймов, крупных серверов и кластеров, отобрав эту долю у Unix.

А как обстоят дела у Linux с точки зрения его возможной фрагментации? Наличие конкурирующих дистрибутивов потенциально вызывает опасения в вероятной возможности такого исхода.

На первый взгляд, такая постановка вопроса кажется надуманной, и сама опасность выглядит довольно призрачной для человека знакомого с идеологией и философией открытого и бесплатного ПО. Фактически есть единый код, большинство «дистрибутивов» работают на основе одного и того же ядра, одних и тех же библиотек, и, тем самым во многом являются идентичной и по построению совместимой ОС. Ведь в отличие от Unix-клонов, различные версии Linux объединены не только общими интерфейсами, они имеют общую реализацию, один и тот же код. Из этого, очевидно, следует, что приложение должно сохранять работоспособность и совместимость между различными версиями Linux.

Несмотря на всю очевидность, это утверждение не получило полного подтверждения практикой. Наряду с фрагментацией рынка дистрибутивов Linux по подходам и дополнительной функциональности, наблюдаются существенные перекосы в поддержке различными клонами довольно общих и стандартных приложений. Для того чтобы предотвратить печальный опыт клонов Unix в 1998 году в рамках специально созданной организации FSG [5] (Free Standards Group — консорциум по развитию открытых стандартов), началась работа над стандартом LSB [4] (Linux Standard Base — базовое семейство стандартов Linux). Благодаря усилиям со стороны организаций X/Open, IEEE и ISO, открывших стандарт POSIX и часть тестов для свободного доступа, был заложен начальный фундамент в дело стандартизации Linux.

Что и зачем нужно стандартизовать? Неужели единый открытый код сам по себе не является единым и открытым стандартом? В этой статье мы попытаемся посмотреть на проблему фрагментации Linux с точки зрения архитектурного устройства дистрибутива и связей его остальных компонентов.

В данной статье мы не будем останавливаться на вопросах поддержания архитектурной документации, процесса разработки отдельных компонентов и процедурах взаимодействия между группами разработчиков. Эти вопросы крайне важны и бесспорно оказывают свое влияние на сложившуюся ситуацию, но их более подробное рассмотрение потребует отдельной работы.

2. Проблемы совместимости приложений

Как проявляются различия между дистрибутивами Linux на практике, и является ли эта проблема реальностью? Для ответа на этот вопрос достаточно привести простой пример. IBM является крупнейшим производителем программных решений, в том числе и под Linux. Основу коммерческих

предложений IBM составляют 5 линеек программных продуктов: DB2, WebSphere, Rational, Tivoli и Lotus. Практика показала, что поддержка всех пяти линеек для одного дистрибутива Linux обходится в 20 млн. долл./год. Эта стоимость включает в себя в основном затраты на разработчиков и тестировщиков, ответственных за полноценную работу и поддержку этих приложений под конкретный дистрибутив Linux. Автоматически следует, что поддерживаются только те дистрибутивы, для которых прибыль от продажи продуктов превышает эти 20 млн. Фактически получается, что продукты IBM поддерживают только дистрибутивы SuSE и RedHat. Тем самым возникает ситуация неравноправия — то, что работает на одних дистрибутивах, не хочет запускаться на других, тем самым нарушая ситуацию с открытостью и равноправием. Кроме того, поддержка дистрибутивов удорожает разработку продуктов, автоматически приводя к удорожанию ПО.

Известен рассказ о том, что одной из причин начала участия Intel в работе над стандартом LSB стало то, что в определенный момент времени, когда это понадобилось, не нашлось ни одного дистрибутива, на котором можно было запустить одновременно продукты Oracle и SAP, необходимые для одного из решений. В целом и тот и другой поддерживают Linux, поддерживают различные его дистрибутивы, но в нужный момент ситуация сложилась так, что поддержка пропала для комбинации необходимых версий этих программ. Скорее всего, причина была в том, что в результате наличия постоянно изменяющегося API совместимость между определёнными версиями приложений и версиями ОС Linux была утеряна. То, что работало вчера, перестало работать сегодня.

Мы увидим совсем другую ситуацию, если взглянем на ОС Sun Solaris. Прежде всего, фирма Sun гарантирует, что программа, скомпилированная для Solaris 2.6, будет работать без перекомпиляции и под версией 10 (т.е. будут работать программы более чем десятилетней давности). Более того, если программа компилировалась тогда, то она будет компилироваться и сейчас. Разработчики Sun прилагают усилия, чтобы это было так — при каждом изменении кода прогоняется набор из более 2400 приложений различного назначения и состава для того, чтобы убедиться, что обратная совместимость не нарушилась из-за внесенных изменений. Более того, если кто-то обнаруживает, что приложение перестало работать по причине несовместимости между версиями Solaris, то Sun берёт на себя ответственность за исправление этого несоответствия. В данном случае, разработчик ОС взял на себя ответственность и расходы на поддержание обратной совместимости.

В случае с Linux данная работа долгое время не велась — приложения и дистрибутивы жили своей собственной обособленной жизнью. Как правило, разработчики приложений ведут поддержку пары-тройки конкретных дистрибутивов, или разработчики дистрибутивов поддерживают ограниченный набор необходимых приложений.

Самым печальным в этом случае является отсутствие универсального способа написания программ, гарантированно обеспечивающего их переносимость. На

решение этой проблемы и направлены усилия консорциума FSG, представляющей интересы основных игроков Linux-рынка. В рамках этой организации разрабатывается стандарт Linux Standard Base, направленный на стандартизацию основных бинарных интерфейсов и команд различных дистрибутивов.

3. Структура Linux

Среди некоторых Linux-программистов бытует мнение, что если программа перестала работать, то, имея исходные коды, её очень легко подправить, поэтому проблемы с совместимостью никакой нет. Легко или нет сделать исправление зависит от того, что может измениться. Для того чтобы разобраться в том, что может на это повлиять, давайте подробнее рассмотрим структуру Linux.

Если отвлечься от наличия конкретных дистрибутивов, то «обобщенная» модель системы на базе ОС Linux выглядит примерно так, как это изображено на Рис. 1.

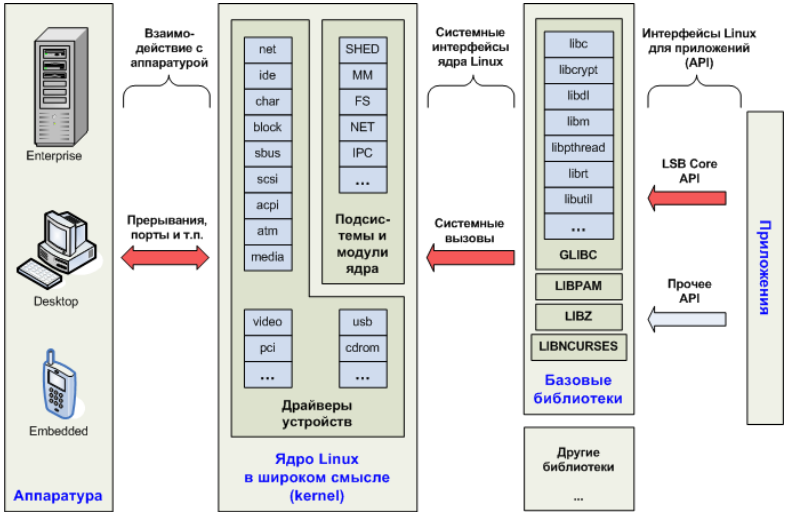

Рисунок 1. Архитектура Linux-системы.

3.1. Роль ядра

ОС Linux имеет *монолитное ядро* [6, 7], т.е. ее ядро представляет собой единую большую программу, внутри которой все функциональные компоненты обладают неограниченным доступом ко всем его внутренним структурам и функциям. Альтернативный подход состоит в том, чтобы иметь *микроядро*, в

котором все функциональные компоненты разделены на различные модули с хорошо определенными интерфейсами и протоколами взаимодействия.

В Linux термин «модуль» имеет иное значение — это динамически подгружаемый вспомогательный элемент ядра. В основном модули ядра это драйверы устройств, драйверы файловой системы и сетевых протоколов.

Итак, ОС Linux содержит монолитное ядро с поддержкой динамически загружаемых модулей. Модули могут загружаться и выгружаться как явно (с помощью команд `insmod` и `rmmod`), так и неявно, когда демон ядра kerneld загружает и выгружает модули автоматически по мере надобности.

Динамическая загрузка/выгрузка, очевидно, позволяет сокращать используемую ядром память. Тем не менее, ничего не бывает бесплатно, и за использование модулей всё-таки придется заплатить — как небольшим падением производительности, так и чуть большим использованием памяти, поскольку модульная организация подразумевает дополнительные структуры с описанием модуля.

После того, как модуль загружен, он эквивалентен в правах с любым другим кодом ядра. Он обладает теми же правами и ограничениями, как и любой другой код ядра, или, если взглянуть с другой стороны, модули ядра могут привести к падению или внештатной ситуации, как и остальной код ядра или драйвер устройства.

Наряду с возможностью «повесить» ОС из-за того, что модуль некачественно реализован, существует ещё одна опасность. Что происходит, если загружаемый модуль создан для более ранней или более поздней версии ядра, чем та, которая его загружает? Это может привести к проблеме, если модуль, обращается к некоторой процедуре ядра и передает неправильные аргументы вызова. Опционально, в ядре присутствует возможность проверки версии загружаемого модуля перед загрузкой.

Почему же все-таки ядро монолитно? Почему не переделать его как микроядро? Опыт показал, что микроядра обладают худшей производительностью по сравнению с монолитными ядрами. В идее микроядра заложена архитектурная проблема — различные компоненты ядра не могут взаимодействовать между собой без преодоления барьеров безопасности, что требует времени и ресурсов.

3.2. Роль системных библиотек

Каждая конкретная Linux система создается для работы одного или нескольких приложений. Однако, кода самого приложения недостаточно, чтобы извлечь необходимый пользователям сервис из аппаратуры. Большинство приложений использует в своей работе обращения к функциям *библиотек*. Стандарт LSB Core 3.1 определяет следующие *системные библиотеки*: libc, libcrypt, libdl, libm, libpthread, librt, libutil, libpam, libz, libncurses. В современных Linux системах интерфейсы этих системных библиотек реализуются библиотеками glibc [9], Linux-PAM [10], zlib [11] и ncurses [12], которые на самом деле

реализуют большее число интерфейсов, чем 1532 функции, определенные в LSB Core.

Функции системных библиотек можно классифицировать следующим образом по степени взаимодействия с ядром Linux.

- Реализация функции полностью содержится в библиотеке и ядро не используется (например `strcpy`, `tsearch`).
- В библиотеке реализуется тривиальная “обертка” (wrapper) для вызова соответствующего интерфейса ядра (например `read`, `write`).
- Реализация функции содержит как вызовы системных интерфейсов ядра (причем возможно нескольких разных), так и часть кода в самой библиотеке (например `pthread_create`, `pthread_cancel`).

Само ядро Linux содержит огромное количество экспортируемых точек входа, однако подавляющее большинство из них составляет внутренний интерфейс для использования модулями и подсистемами самого ядра. Внешний интерфейс содержит порядка 250 функций (версия 2.6). Из них, к примеру, в своей реализации библиотека GLIBC 2.3.5 использует 137.

Таким образом, то, с чем взаимодействует пользовательское приложение в основном реализовано не в ядре ОС, а в библиотеках поддержки – подавляющая часть функциональности обеспечивается внешними библиотеками. Кстати, именно поэтому в свое время Ричард Столлман настоял на том, чтобы дистрибутивы Linux именовались как GNU/Linux. Большая часть исходного кода и функциональности реализовывается не разработчиками ядра, а библиотеками и утилитами, сделанными в рамках проекта GNU.

3.3. Роль этапа сборки

Может показаться, что для надежности системы на уровне поведения интерфейсов системных библиотек все Linux системы одинаковы, и достаточно, если тестирование будет проводиться разработчиками ядра и библиотек. Однако это не совсем верно. Уже на уровне интерфейсов системных библиотек существует масса измерений, которые делают практически каждую Linux систему уникальной с точки зрения проверки качества.

Поведение интерфейсов для приложений определяется комбинацией из библиотек, ядра и аппаратуры. В свою очередь ядро и библиотеки определяются своей версией (включая официальные или неофициальные патчи и модификации) и, что немало важно, *конфигурацией сборки*. Дело в том, что при сборке одной и той же версии ядра или библиотеки можно задавать массу параметров, которые делают понятие версии расплывчатым. Одна и та же версия ядра или библиотеки, собранные при разных значениях конфигурационных параметров, могут работать по-разному. Таким образом, для обеспечения надежности каждой специфической конфигурации Linux системы необходимо ее отдельное тестирование.

Многое может пояснить процедура сборки ядра. Проект Linux From Scratch [8] содержит последовательность шагов необходимых для сборки дистрибутива

Linux «с нуля». Вот упрощённая последовательность шагов для сборки дистрибутива LFS Linux версии 6.0:

Сборка бинарных утилит

1. Binutils-2.15.94.0.2.2 - Pass 1
2. GCC-3.4.3 - Pass 1
3. Linux-Libc-Headers-2.6.11.2
4. Glibc-2.3.4
5. Adjusting the Toolchain
6. Tcl-8.4.9
7. Expect-5.43.0
8. DejaGNU-1.4.4
9. GCC-3.4.3 - Pass 2
10. Binutils-2.15.94.0.2.2 - Pass 2
11. Gawk-3.1.4
12. Coreutils-5.2.1
13. Bzip2-1.0.3
14. Gzip-1.3.5
15. Diffutils-2.8.1
16. Findutils-4.2.23
17. Make-3.80
18. Grep-2.5.1a
19. Sed-4.1.4
20. Gettext-0.14.3
21. Ncurses-5.4
22. Patch-2.5.4
23. Tar-1.15.1
24. Texinfo-4.8
25. Bash-3.0
26. M4-1.4.3
27. Bison-2.0
28. Flex-2.5.31
29. Util-linux-2.12q
30. Perl-5.8.6

Файловая система и доводка утилит

31. Entering the Chroot Environment
32. Changing Ownership
33. Creating Directories
34. Creating Essential Symlinks
35. Creating the passwd, group, and log Files
36. Populating /dev
37. Linux-Libc-Headers-2.6.11.2
38. Man-pages-2.01
39. Glibc-2.3.4
40. Re-adjusting the Toolchain
41. Binutils-2.15.94.0.2.2
42. GCC-3.4.3
43. Coreutils-5.2.1
44. Zlib-1.2.2
45. Mktemp-1.5
46. Iana-Etc-1.04
47. Findutils-4.2.23
48. Gawk-3.1.4
49. Ncurses-5.4
50. Readline-5.0
51. Vim-6.3
52. M4-1.4.3
53. Bison-2.0
54. Less-382
55. Groff-1.19.1
56. Sed-4.1.4
57. Flex-2.5.31
58. Gettext-0.14.3
59. Inetutils-1.4.2
60. IPRoute2-2.6.11-050330
61. Perl-5.8.6
62. Texinfo-4.8
63. Autoconf-2.59
64. Automake-1.9.5
65. Bash-3.0
66. File-4.13
67. Libtool-1.5.14
68. Bzip2-1.0.3
69. Diffutils-2.8.1
70. Kbd-1.12
71. E2fsprogs-1.37
72. Grep-2.5.1a
73. GRUB-0.96
74. Gzip-1.3.5
75. Hotplug-2004_09_23
76. Man-1.5p
77. Make-3.80
78. Module-Init-Tools-3.1
79. Patch-2.5.4
80. Procps-3.2.5
81. Psmisc-21.6
82. Shadow-4.0.9
83. Sysklogd-1.4.1
84. Sysvinit-2.86
85. Tar-1.15.1
86. Udev-056
87. Util-linux-2.12q
88. Конфигурация загрузки
89. ***Linux-2.6.11.12 – Ядро***

Отметим, что сборка ядра осуществляется в самом конце (на самом последнем, 89-м шаге) с помощью собранных перед этим бинарных утилит.

Важно учитывать версии приведённого в каждом элементе списка компонента. Замена одной версии компонента на другую не всегда тривиальна — сборка

системы может оказаться невозможной из-за отсутствия или изменения какой-либо функции, либо усложнена.

Сборка многих компонентов требует дополнительных действий. К примеру, инструкция по сборке flex для данного дистрибутива содержит замечание:

> Flex contains several known bugs. These can be fixed with the following patch:
>
> patch -Np1 -i ../flex-2.5.31-debian_fixes-3.patch

Особо хочется отметить, что в процесс сборки включается сборка средств компиляции, которые также претерпевают существенные изменения во времени. Даже базовые компоненты Linux нередко оказываются устаревшими. Так, например, версия компилятора gcc 4.0.0 не годится для сборки ядра 2.6.11 (хотя они современники) и требует использования специального патча для устранения этой несовместимости.

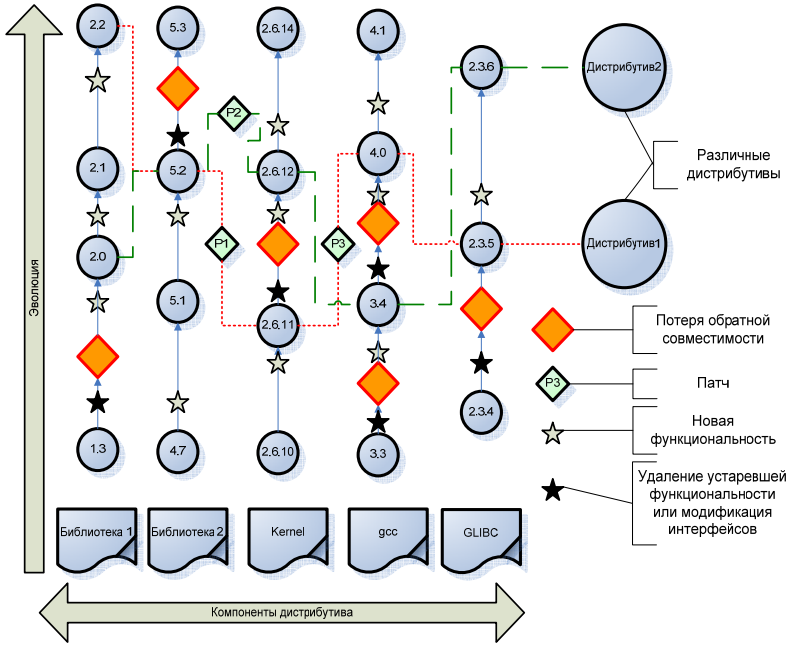

Рисунок 2. Эволюция библиотек и состав дистрибутивов.

3.3.1. Конфигурации

Под *конфигурацией* системной части дистрибутива мы понимаем комбинацию версии ядра (включая отдельные патчи), версий системных библиотек, параметров их сборки и архитектуры, на которой это все работает.

На Рисунке 2 изображена конфигурация сборки двух гипотетических дистрибутивов. Каждый из них есть совокупность версий компонентов и патчей. Между версиями компонентов добавляется новая функциональность, а также убираются морально устаревшие интерфейсы и функции.

Например, на данной диаграмме легко видеть, что поскольку дистрибутивы 1 и 2 используют различные версии GCC, совместимость по исходным кодам между ними отчасти утеряна — не всё что собиралось с помощью gcc 3.4 может быть собрано с помощью gcc 4.0 без доработки.

4. Основные дистрибутивы Linux

С точки зрения прикладного ПО дистрибутив есть конфигурация базовых компонентов Linux. По адресу [13] можно найти перечень известных дистрибутивов Linux. На момент написания статьи их было 508. И это только публично доступные, открытые для широкой публики дистрибутивы. Здесь не учитываются непубличные версии, сделанные для внутреннего применения в различных компаниях, оборонных ведомствах и т.п.

Модифицировать и распространять код позволяет GNU-лицензия. Из этого следует, что можно взять произвольный дистрибутив, внести в него модификации (как минимум в его компоненты, подпадающие под GNU или подобную лицензию) и распространять далее. Это и происходит на практике — различные дистрибутивы «наследуют» основные свойства своих предков, добавляя минимум специфических свойств (локализация, определенный набор прикладного ПО, программа установки, диалоги конфигурации, графические элементы и т.п.).

Дистрибутивы можно разделить по следующим категориям:

1. ***По базовым производителям.*** К примеру, RedHat, Slackware, SuSE, Debian, Asianux, Mandriva являются основными «ветвями» Linux-индустрии. Эти дистрибутивы не являются наследниками других. Их можно считать стратегическими направлениями развития в Linux вообще. Большинство остальных дистрибутивов явно принадлежат к одной из упомянутых ветвей – в основном наследуя исходный код и приложения и добавляя специфическую функциональность.
2. ***По локализации.*** Во многих странах присутствует локальный производитель Linux (скажем, в России всем известны дистрибутивы ASP Linux и ALT Linux).
3. ***Встроенные версии.*** Дистрибутивы для встроенного применения в мобильных устройствах; дистрибутивы, работающие без поддержки файловой системы; облегченные версии для использования в КПК; переносные версии для запуска с ограниченных носителей (Linux на дискете, Linux на CD и т.п.).
4. ***Специализированные версии с поддержкой определенной аппаратной архитектуры*** (AlphaLinux с поддержкой Alpha, ARM Linux с поддержкой ARM и т.п.).

4.1. Red Hat

Компания Red Hat, Inc. [14] представляет, наверное, самые известные дистрибутивы в мире. Это Red Hat Enterprise Linux, коммерческий дистрибутив, стабильный и надёжный, поставляемый с услугами сервиса и поддержки для промышленных потребителей. И Fedora Core (дословный перевод — фетровая шляпа) — спонсируемый Red Hat открытый дистрибутив, разрабатываемый при помощи сообщества. Fedora Core используется также и как тестовый полигон — обкатанные на этом дистрибутиве элементы позже добавляются в Red Had Enterprise.

4.2. Debian

Вот цитата из Wikipedia, коротко характеризующая семейство Debian [15].

«Debian — наиболее строгий из всех дистрибутивов в отношении лицензий программ. Имеет наибольшее хранилище пакетов — готовых к использованию программ, — и если даже не по их числу, то по числу поддерживаемых архитектур: начиная с ARM, используемой во встраиваемых устройствах, наиболее популярных x86 и PowerPC, новых 64-разрядных AMD и заканчивая IBM S/390, используемой в мэйнфреймах. Хранилище разделено на три ветки:

- стабильную (stable), содержащую пакеты, вошедшие в последний официальный дистрибутив (обновление пакетов в нём происходит только для устранения уязвимостей);
- тестируемую (testing), из которой будет формироваться следующий стабильный дистрибутив;
- нестабильную (unstable), в которой пакеты готовятся к помещению в тестируемую ветку.

Существует также ветка, называемая экспериментальной (experimental); в неё помещаются пакеты, претерпевающие особо большие изменения. В Debian серьёзно относятся к любым изменениям, благодаря чему проблемы с обновлением бывают очень редко (ценой этому является не совсем быстрая реакция на выход новых версий программ). Для работы с хранилищем разработаны разные средства, самое популярное из которых — APT.

Debian стал основой целого ряда других дистрибутивов (более 100, см. список дистрибутивов, основанных на Debian). Самые известные из них — Adamantix, Bioknoppix, Clusterix, Gnoppix, Knoppix, Kubuntu, Libranet, Linspire, MEPIS, Ubuntu Linux и Xandros Desktop OS.

4.3. Slackware

Целями, положенными в основу построения этого дистрибутива [16], являются простота и стабильность. Авторы придерживаются базового принципа KISS (Keep it simple, stupid — Не усложняй) — это относится, прежде всего, к простоте построения системы, а не к простоте использования.

Slackware использует стартовые скрипты стиля BSD, в то время как большинство других дистрибутивов использует стиль System V.

4.4. SuSE

Название «S.u.S.E.», позднее сокращённое до «SuSE», является акронимом немецкой фразы «Software- und System-Entwicklung» («Программная и системная разработка») [17]

Дистрибутивы SUSE ориентированы в первую очередь на настольные компьютеры, хотя также доступен ряд продуктов класса предприятия, таких как SUSE Linux Enterprise Server (SLES). Про эти дистрибутивы было написано немало положительных обзоров за их инсталлятор и систему настройки YaST, разработанные программистами SUSE. Документация, идущая с коробочными версиями, постоянно отмечается как наиболее полная, детальная и удобная.

4.5. Asianux

Asianux [18] — объединенный продукт, включивший в себя японский Miracle Linux и китайский Red Flag Linux. Позднее к проекту присоединилась корейская компания Haansoft Inc. Производимый в китайском центре разработок фирмы Oracle в Пекине, Asianux — это совместная попытка основать стандартизованный общий Азиатский Linux, определяющий ядро, набор библиотек и пакетов. Объединение вокруг Asianux также является естественным центром сертификации ПО и оборудования для работы с производными этой системы.

5. Решение проблемы: LSB

LSB (Linux Standard Base) — это основной современный стандарт, определяющий требования совместимости к Linux-системам. Основную часть стандарта (LSB Core) составляют требования на системные интерфейсы, которые должны поддерживаться всеми дистрибутивами Linux. В этой части LSB во многом ссылается на стандарт POSIX. Разработкой и развитием стандарта LSB занимается организация Free Standards Group. Подробные сведения о LSB можно найти на сайте http://www.freestandards.org/en/LSB.

Однако, даже очень хорошие стандарты остаются лишь благими пожеланиями, пока нет удобных и надежных способов проверить формальное соответствие им. Дополнение стандартов формальными описаниями позволяет выявить нечеткие места, устранить ещё встречающиеся противоречия и учесть неявные требования, которые часто упускаются даже разработчиками официальных сертификационных тестов.

Анализ существующих тестов показывает их слишком малый охват — многие функции не тестируются вообще, а многие тестируются только в отношении базовых требований стандарта, не проверяя тонкие аспекты. Подход к тестированию с помощью систематической формализации требований, используемый в Центре верификации ОС Linux, позволяет досконально учесть все принципиально проверяемые требования стандарта. Кроме того, автоматическая генерация тестов обеспечивает полный перебор различных ситуаций в работе тестируемой системы, а также облегчает модификацию

тестов для учета эволюции стандарта или специальных требований пользователя. Эти факторы обеспечивают ценность тестового набора Центра в качестве важного дополнения к существующим тестам Linux.

Проект тестового набора OLVER, разрабатываемого в рамках созданного на базе ИСП РАН Центра верификации Linux [3], является инициативой, нацеленной на повышение качества тестов для Linux и общего покрытия ими интерфейсов стандарта. Главные цели проекта OLVER следующие:

- Проанализировать стандарт Linux Standard Base (LSB) Core 3.1 (ISO/IEC 23360-1) с целью выявления атомарных требований к поведению основных системных библиотек (секции III. Base Libraries и IV. Utility Libraries — всего 1532 функции).
- Разработать каталог требований LSB Core3.1 для 1532 интерфейсных функций Linux. В процессе работы обнаружить неточности и ошибки в тексте стандарта LSB и связанных с ним стандартов и сообщать их оригинальным разработчикам для внесения изменений в будущие версии.
- Разработать открытый тестовый набор для функционального тестирования различных Linux систем на соответствие требованиям стандарта LSB Core в отношении поведения системных интерфейсов прикладного программирования Linux.
- Разработать систему конфигурации и запуска тестов. Задача такой системы заключается в предоставлении удобных средств для настройки тестов по ряду параметров (например, глубина тестирования, опции целевой системы, выбор варианта поведения из нескольких, допустимых стандартом) и визуализации результатов тестирования с указанием на расхождения с конкретными требованиями стандарта.

В рамках проекта OLVER текстовые описания существующих стандартов дополняются формальными описаниями в виде спецификаций на языке SeC (Specification Extention of C) — спецификационном расширении языка C. В процессе формализации выявляются нечеткие места и противоречия, а также учитываются неявные требования. Найденные замечания к текстовой версии стандартов сообщаются соответствующим организациям, отвечающим за их развитие (Free Standards Group по LSB, Austin Group по POSIX).

Результаты работы Центра (готовые тестовые наборы из состава OLVER и инструменты их генерации и эксплуатации) будут публиковаться на сайте Центра и распространяться по свободной лицензии Apache License 2.0.

6. Заключение

Масштабность объекта, коим является ОС Linux и количество параметров сборки любого ее дистрибутива приводят к плохо совместимым между собой продуктам. Общедоступность и открытость кода сами по себе это не являются панацеей — получаемые в результате дистрибутивы обладают существенными

различиями даже на уровне базовых компонентов, рассмотренных в этой статье. Это создает проблемы для разработчиков прикладного ПО, поскольку повышает трудоемкость и стоимость его разработки.

Для того чтобы предотвратить печальные последствия, уже имевшие место для Unix-систем, необходимы меры обеспечения кросс-совместимости дистрибутивов. Переносимость приложений позволит утвердить Linux как единую платформу и заметно снизить стоимость разработки и поддержки приложений, что должно положительно сказаться на их количестве.

На сегодняшний день основной инициативой по обеспечению переносимости и совместимости между Linux-системами является стандарт LSB (Linux Standard Base). Этот стандарт поддерживается основными производителями дистрибутивов (RedHat, SuSe, Mandriva) и уже можно считать, что заложены первые камни в фундамент единой, не фрагментированной платформы Linux, обеспечивающей переносимость приложений, как на основе исходного кода, так и в бинарном виде.

Известно, что интерфейсный стандарт хорош настолько, насколько хороши тестовые набор, проверяющие соответствие ему и позволяющие разработчикам как различных версий ОС, так и приложений убедиться, что у них не будет проблем с совместимостью. К сожалению, в настоящий момент тестовый набор, используемый для проверки соответствия дистрибутива LSB, покрывает менее 20% описываемых стандартом интерфейсов. Разработка тестов, обеспечивающих качественную проверку стандарта, в настоящий момент является одним из ключевых шагов к общему успеху, поэтому такие проекты как OLVER необходимы для гарантирования стабильного развития экосистемы Linux в целом.

Литература

[1] IEEE POSIX® Certification Authority. http://standards.ieee.org/regauth/posix/
[2] Wikipedia: Unix Wars. http://en.wikipedia.org/wiki/Unix_wars
[3] Центр верификации ОС Linux. http://www.linuxtesting.ru
[4] LSB (Linux Standard Base). http://www.freestandards.org/en/LSB
[5] FSG (Free Standards Group). http://www.freestandards.org
[6] Основной сайт разработчиков ядра Linux. http://www.kernel.org/
[7] David A. Rusling. The Linux Kernel book. http://en.tldp.org/LDP/tlk/modules/modules.html
[8] LFS: Linux From Scratch. http://www.linuxfromscratch.org
[9] glibc. http://www.gnu.org/software/libc/libc.html
[10] Linux-PAM. http://sourceforge.net/projects/pam
[11] zlib. http://www.zlib.net
[12] ncurses. http://invisible-island.net/ncurses/
[13] Перечень дистрибутивов Linux. http://lwn.net/Distributions/
[14] RedHat. http://www.redhat.com
[15] Debian. http://www.debian.org
[16] Slackware. http://www.slackware.com
[17] SuSE. http://www.suse.com
[18] Asianux. http://www.asianux.com