Linux Standard Base: история успеха?

А. В. Хорошилов

Институт системного программирования РАН (ИСП РАН),
Б. Коммунистическая, 25, Москва, Россия
E-mail: khoroshilov@ispras.ru

Аннотация

В настоящей статье, посвященной истории развития стандарта Linux Standard Base, не дается ответа на вопрос, вынесенный в заглавие. Окончательный вердикт может вынести только время. Здесь же рассматриваются предпосылки и причины создания стандарта, его уже почти 10-летняя история, настоящее состояние, а также планируемые пути развития и проблемы, стоящие перед его разработчиками.

1. Предпосылки

На первый взгляд кажется, что растущая популярность операционной системы Linux вкупе с возможностью ее свободного распространения и свободного внесения изменений предвещает безоблачное будущее Linux. Однако эти факторы содержат и определенную угрозу дальнейшему развитию операционной системы. Вместе с ростом популярности Linux растет и число различных ее дистрибутивов, каждый из которых имеет определенные особенности, выделяющие его среди конкурентов. Эти особенности предоставляют уникальные возможности конечным пользователям, но в то же время и привносят немало головной боли разработчикам приложений.

Каждое приложение, предназначенное для работы с операционной системой Linux, должно учитывать особенности различных дистрибутивов, чтобы предоставлять своему пользователю высокий уровень сервиса. Это вынуждает поставщиков приложений проводить тестирование своих продуктов на большом числе дистрибутивов операционной системы и поддерживать переносимость между ними, как это проиллюстрировано на Рисунке 1.

Такой подход является весьма затратным и поэтому оказывается применим только для относительно простых приложений. Поставщикам сложных приложений оказывается не под силу следовать этой логике, даже если это IBM, Oracle или SAP. IBM поддерживает только дистрибутивы Linux от RedHat и Novell SuSe, Oracle ограничивается поддержкой RedHat Enterprise Linux, SuSe Linux Enterprise Server и Asianux, а SAP сертифицировал только несколько версий RedHat Enterprise Linux, SuSe Linux Enterprise Server и Red Flag Advanced Server. Таким образом, крупнейшие игроки на рынке программного обеспечения существенно ограничивают число поддерживаемых ими дистрибутивов и оставляют пользователей других систем без доступа к широкому ряду качественных программных продуктов. Этот факт снижает конкурентоспособность других дистрибутивов, но и у них оказывается большая

группа приверженцев и собственный набор важного программного обеспечения, недоступного для пользователей лидирующих дистрибутивов.

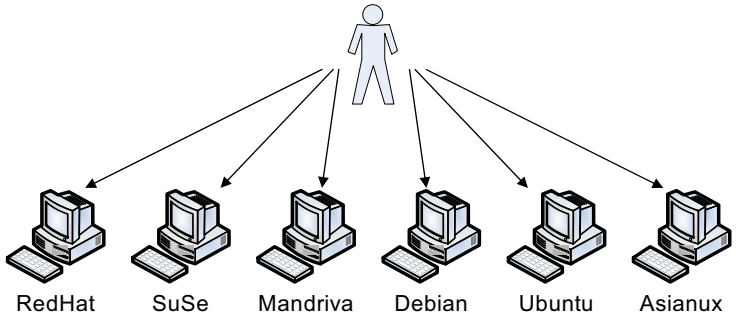

Рисунок 1. Разработчик приложений вынужден поддерживать все дистрибутивы Linux

Такая ситуация ведет к фрагментации рынка приложений для операционной системы Linux, которая, в свою очередь, имеет следующие негативные последствия.

- Она снижает привлекательность рынка для разработчиков программного обеспечения, поскольку
  - либо требует существенных затрат на сопровождение и поддержку программного обеспечения;
  - либо ограничивает рынок только несколькими дистрибутивами.
- Она невыгодна конечным пользователям, так как
  - ограничивает возможности пользователя по выбору операционной системы;
  - сужает круг доступных ему приложений.
- Она ограничивает возможности развития поставщикам дистрибутивов, потому что
  - требует дополнительных усилий для привлечения разработчиков приложений к поддержке своего дистрибутива и для тестирования работоспособности ключевых для данного дистрибутива приложений на каждой новой версии своего продукта;
  - снижает конкурентоспособность дистрибутивов Linux на фоне других систем.

Проблемы назревали, и в 1998 году сообщество Linux объединилось для нахождения путей по их решению. Предложенный подход был не нов и

базировался на опыте преодоления фрагментации рынка UNIX систем. Идея заключалась в стандартизации всех базовых взаимодействий между приложением и операционной системой.

Такой стандарт описывает контракт между операционной системой и приложением. Операционная система обязуется предоставлять определенный набор сервисов в рамках зафиксированных ограничений, а приложение обязуется использовать только эти сервисы и только в соответствии с зафиксированными ограничениями. И если приложение выполняет условия контракта, то оно будет работать одинаковым образом на всех экземплярах операционной системы, удовлетворяющих условиям контракта со своей стороны.

В результате задача разработки приложений, работающих на всех дистрибутивах операционной системы, существенно упрощается. От разработчика требуется только выполнение двух условий:

- использование сервисов операционной системы только в рамках, описанных в стандарте;
- проверка корректности работы своего приложения на одном из дистрибутивов операционной системы.

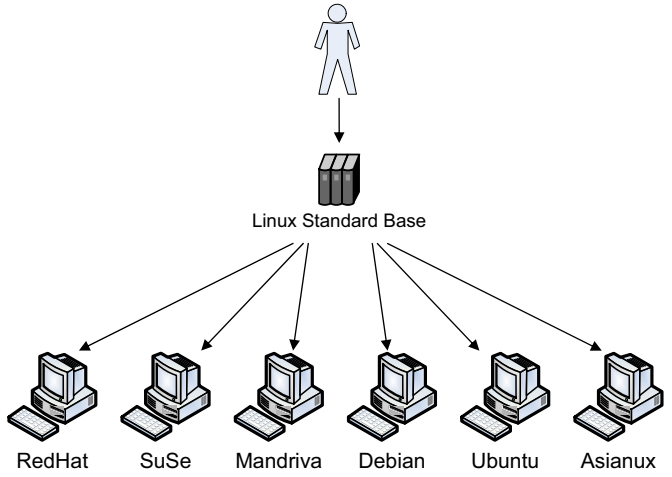

Рисунок 2. Стандарт Linux Standard Base выступает в качестве единой платформы для разработчиков приложений.

Таким образом, разработчик приложения избавляется от необходимости заботиться о проверке работоспособности своего приложения на всем многообразии дистрибутивов операционной системы. Вместо этого выполняется проверка каждого отдельного дистрибутива на соответствие

требованиям стандарта, которая не требует участия поставщиков приложений, как это проиллюстрировано на Рисунке 2.

Но для того, чтобы реализовать рассмотренные выше цели, стандарт должен удовлетворять следующим основным требованиям.

- Он должен очень точно найти ту грань, которая позволит ему предоставлять достаточно богатый набор сервисов для приложений и не слишком сильно ограничить свободу разработчиков операционных систем.
- Стандарт должен быть общепризнанным, чтобы все дистрибутивы операционной системы или хотя бы их основная часть выполняли его требования.
- Он должен быть подкреплен механизмом, надежно удостоверяющим выполнение зафиксированного в нем контракта, как со стороны операционной системы, так и со стороны приложения.

Прежде, чем мы рассмотрим путь, по которому пошло сообщество Linux, обратимся к более ранним попыткам стандартизации контракта между приложениями и операционной системой.

2. Предыстория

История операционной системы UNIX берет свое начало с исследовательских разработок Bell Telephone Laboratories, подразделении AT&T, выполнявшихся в конце 60-х годов прошлого века. Первая версия системы появилась в 1970 году и была написана на ассемблере для машины PDP-7. В 1971 году операционная система была перенесена на машину PDP-11, а уже в 1973 году ядро системы было переписано на языке высокого уровня Си. Это неслыханное ранее решение стало одним из ключевых моментов, оказавших огромное влияние на дальнейшее развитие операционной системы UNIX.

Вторым ключевым моментом в истории UNIX оказались особенности федерального законодательства США того времени, согласно которым компания AT&T не имела права на коммерческое распространение UNIX. AT&T использовала операционную систему внутри компании, а также в 1974 году начала распространять ее среди университетов для образовательных целей.

Использование языка высокого уровня и доступность операционной системы позволяли относительно быстро и дешево портировать UNIX с одной платформы на другую. Все эти факторы в совокупности с многочисленными удачными архитектурными решениями привели к огромной популярности операционной системы UNIX. Но они же стали причиной появления большого числа вариаций системы. Эти вариации были несовместимы по многим параметрам, развивались независимо различными компаниями, в результате чего понятие операционной системы UNIX размылось и образовалось новое понятие семейства UNIX-подобных операционных систем. Для разработчиков приложений такая ситуация привела к тому, что они были вынуждены либо

ориентироваться только на небольшую часть UNIX-систем, либо нести большие затраты на перенос кода и поддержку большого числа их вариаций.

Первой попыткой преодоления разобщенности в мире UNIX систем стало создание в 1980 году комитета по стандартизации в рамках некоммерческой организации /usr/group, объединявшей американских пользователей UNIX-подобных систем и ставившей своей целью дальнейшее их продвижение. Результатом деятельности комитета явилось появление в 1984 году первой спецификации, описывающей требования к интерфейсам, через которые операционная система предоставляет сервисы приложению. Эта спецификация, известная как /usr/group Standard[1], не получила широкой известности, тем не менее усилия разработчиков не пропали даром. Во-первых, они были использованы комитетом ANSI по стандартизации языка Си при описании стандартной библиотеки поддержки. Во-вторых, когда комитет прекратил свою деятельность в рамках /usr/group, большинство его членов присоединилось к Portable Application Standards Committee (PASC), образованному под эгидой IEEE CS в 1984 году.

Основным направлением работы PASC стал Project.1003. Первым публичным результатом проекта был IEEE Trial-Use Standard, опубликованный в марте 1986 года. Тогда он назывался IEEE-IX, как сокращение IEEE's UNIX, но вскоре проект получил новое название Portable Operating System Interface for Computing Environment (сокращенно **POSIX**), которое сохранилось за ним до настоящего времени.

Первая официальная версия стандарта IEEE Std 1003.1 (также известного как POSIX 1003.1) вышла в 1988 году. Она содержала требования к функциям прикладного интерфейса операционной системы на языке программирования С. В течении последующих нескольких лет стандарт немного доработали, устранили выявленные неточности и в 1990 году выпустили вторую редакцию IEEE Std 1003.1-1990, которая была принята в качестве международного стандарта ISO/IEC 9945-1:1990.

Помимо стандартизации функций прикладного программного интерфейса операционной системы PASC разработал стандарт на команды и утилиты IEEE Std 1003.2-1992 (POSIX 1003.2), который получил статус международного стандарта ISO/IEC 9945-2:1993.

После выпуска IEEE Std 1003.1-1990 основная работа PASC проводилась в подкомитетах и заключалась в разработке дополнений к базовому стандарту. Среди наиболее важных дополнений следует перечислить:

IEEE Std 1003.1b-1993 Realtime Extension
IEEE Std 1003.1c-1995 Threads
IEEE Std 1003.1d-1999 Additional Realtime Extension
IEEE Std 1003.1i-1995 Technical Corrigenda to Realtime Extension
IEEE Std 1003.1j-2000 Advanced Realtime Extension

[1] Также этот стандарт известен как UniForum 1984 Draft Standard или UDS 84, где UniForum — это новое название /usr/group.

IEEE Std 1003.1q-2000 Tracing

В 1996 году была выпущена очередная редакция IEEE Std 1003.1-1996, в которую вошли дополнения IEEE Std 1003.1b-1993, IEEE Std 1003.1c-1995 и IEEE Std 1003.1i-1995, а основной текст остался без изменений. Эта редакция также была утверждена в качестве международного стандарта ISO/IEC 9945-1:1996.

Одновременно с началом проекта 1003 IEEE (POSIX), в 1984 году ряд европейских компаний образовали некоммерческую организацию **X/Open**. Ее целью являлась разработка стандарта на набор интерфейсов операционной системы, независимый от конкретных реализаций и позволяющий снизить стоимость переносимости приложений как между операционными системами различных поставщиков, так и между различными версиями одной операционной системы.

И если тактика разработки POSIX заключалась в стандартизации общего подмножества всех UNIX-подобных систем, то X/Open придерживалась другой практики. Она старалась стандартизовать общие решения на передовом крае развития UNIX-подобных систем. Результатом этой деятельности было создание X/Open Portability Guide (XPG).

Первое такое руководство XPG1 появилось в 1985 году на основе интерфейсов System V release 2 и BSD 4.2[2]. Следующий вариант XPG2 был опубликован в 1987 году. Его основным новшеством стала спецификация интерфейсов для взаимодействия приложений с пользователем на основе терминальных устройств (curses). XPG3, выпущенный в 1990 году, включил в себя стандарт на графическую систему X Window System, разработанную с Массачусетском институте технологий. Заключительная версия руководства XPG4 вышла в 1992 году и, в основном, заключалась в интеграции со стандартом ANSI Си 1989 года. Кроме того, при разработке XPG2, XPG3 и XPG4 много внимания уделялось поддержке локализации, работе с кодировками символов и каталогами сообщений.

Одновременно с POSIX и XPG стартовал еще один проект по стандартизации интерфейсов UNIX-подобных операционных систем. На этот раз инициатором выступил правообладатель торговой марки UNIX компания AT&T. В 1985 году она выпустила System V Interface Definition (**SVID**), который содержал спецификацию интерфейсов System V release 2 и основывался на /usr/group Standard. SVID2 и SVID3, появившиеся в дальнейшем, содержали обновленные спецификации System V release 3 и System V release 4 соответственно. SVID4 уже был разработан компанией Novell, выкупившей у AT&T ее подразделение

[2] System V — название реализации UNIX-подобной операционной системы компании AT&T.
BSD (Berkeley Software Distribution) — UNIX-подобная операционная система, распространяемая University of California, Berkeley. Эти две системы являлись ключевыми ветвями в развитии UNIX-подобных систем до начала 90-х годов прошлого столетия.

Unix System Labs вместе с правами торговую марку UNIX и выпустившей собственную UNIX-подобную операционную систему UnixWare.

Используя SVID, компания AT&T попыталась под предлогом стандартизации вновь отождествить UNIX со своей собственной реализацией. Еще одной попыткой унификации UNIX со стороны AT&T было соглашение с Sun Microsystems и Microsoft о совместной работе над System V release 4, в ходе которой планировалось объединить характеристики System V release 3, SunOS (Sun) и Xenix (Microsoft).

Такое усиление позиций AT&T вызвало недовольство у других поставщиков UNIX-подобных систем и они (Apollo Computers, Groupe Bull, Digital Equipment Corporation, Hewlett-Packard, IBM, Nixdorf Computer и Siemens AG) организовали альтернативный консорциум Open Software Foundation (OSF). Под эгидой OSF началась работа по стандартизации интерфейсов между приложениями и операционной системой на уровне двоичного кода, результатом которой стало появление стандарта **OSF/1** (1990). Среди других проектов OSF следует выделить OSF DCE (Distributed Computing Environment) и OSF CDE (Common Desktop Environment).

Завершение воин между поставщиками UNIX-подобных операционных систем привело к объединению X/Open с OSF под новым названием The Open Group в 1993 году. Тогда же The Open Group получила от Novell права на торговую марку UNIX. В результате этих преобразований The Open Group начал работу над единой спецификацией UNIX (Single UNIX Specification, **SUS**). В качестве основы была взята базовая часть XPG4, включавшая в себя IEEE Std 1003.1-1990, а также SVID3 (Level 1) и OSF Application Environment Specification. В дополнение к этому были проведены исследования интерфейсов, используемых 50 различными приложениями, что позволило выявить еще 130 функций, не стандартизованных ранее, но являющихся устоявшейся практикой среди UNIX-подобных систем унаследованных от BSD. В число этих функций вошли функции работы с сокетами и функции управления памятью BSD 4.3.

Во вторую версию Single UNIX Specification, вышедшую в 1997 году, вошли новые функции для работы с файлами большого размера, загрузкой динамических библиотек, а также проведено согласование с IEEE Std 1003.1-1996.

Ключевое событие произошло в сентябре 1998 году, когда была образована The Austin Group – рабочая группа по развитию совместного стандарта IEEE Std 1003, ISO/IEC 9945 и Single UNIX Specification. Результатом работы этой группы стал выпуск новой, единой версии всех этих стандартов IEEE Std 1003.1-2001, ISO/IEC 9945:2001 и базовой части Single UNIX Specification version 3. В этой версии были объединены требования к прикладному программному интерфейсу и к командам и утилитам операционной системы, существовавшие ранее в виде независимых стандартов 1003.1 и 1003.2.

В 2003 и в 2004 годах The Austin Group выпустила два списка исправлений к стандарту, которые были формально приняты всеми заинтересованными сторонами. В настоящее время в рамках The Austin Group ведется работа по

подготовке новой редакции объединенного стандарта, выход которой запланирован на 2008 год.

Подведем итоги усилий по унификации UNIX-подобных систем. Наиболее успешными проектами по стандартизации интерфейсов операционных систем оказались POSIX и Single UNIX Specification, объединившиеся в настоящее время под эгидой The Open Group, IEEE и ISO/IEC. Эти проекты получили наибольшую поддержку как среди поставщиков операционных систем, так и среди разработчиков приложений.

Оба проекта описывали сервисы, предоставляемые операционной системой, на уровне прикладных программных интерфейсов языка программирования Си. Такой подход никак не ограничивал свободу поставщиков операционных систем по выбору архитектуры системы и способам ее реализации. Поэтому практически все операционные системы, в том числе и не являющиеся наследниками UNIX, в той или иной степени стали поддерживать стандарт POSIX.

Для разработчиков приложений POSIX позволил существенно упростить переносимость приложений между UNIX-подобными операционными системами различных поставщиков. Если приложение было написано с использованием только стандартизованных интерфейсов операционной системы, то для переноса приложения на новую платформу разработчикам требуется только перекомпилировать исходный код. Конечно, перекомпиляция требует определенных усилий, но эти усилия не идут ни в какое сравнение с теми проблемами, которые возникали с переносом приложений ранее.

Жизнь разработчиков приложений могла бы быть и еще проще, если бы стандарт описывал интерфейсы операционной системы на двоичном уровне. В этом случае, для переноса приложения между операционными системами в рамках одной аппаратной платформы вообще не требовалось бы никакого участия разработчика. Но попытки внедрения двоичных стандартов, таких как OSF/1, провалились. Основной причиной провала оказалось отсутствие широкой поддержки этих стандартов со стороны поставщиков операционных систем. Так, стандарт OSF/1 был реализован только двумя компаниями, Kendall Square Research и Digital Equipment Corporation, ни одна из которых не существует на настоящий момент.

Отсутствие поддержки со стороны других поставщиков объясняется несколькими факторами. Во-первых, ограничения накладываемые стандартами на двоичном уровне являются чрезмерно жесткими и не допускают достаточной свободы при выборе путей реализации системы. Во-вторых, существующие реализации не являются совместимыми на двоичном уровне, поэтому их унификация требует очень сложных переговоров по изменению существующих решений, принятых в различных реализациях.

3. Стандартизация Linux

С самого начала операционная система Linux являлась POSIX-совместимой операционной системой за исключением небольшого числа отдельных несоответствий. Тем не менее, это не спасло Linux от проблем с переносимостью

приложений и последующей фрагментацией рынка. Недостатки POSIX применительно к рассматриваемой ситуации заключаются в следующем.

Во-первых, интерфейсов специфицированных в POSIX оказывается недостаточно для полноценной разработки всех видов приложений. Если системные приложения и приложения, взаимодействующие с пользователем посредством терминала, могут быть реализованы в рамках интерфейсов POSIX, то приложения с графическим пользовательским интерфейсом и приложения, использующие системную информацию, оказываются вне стандарта POSIX. Попытки стандартизации графического интерфейса существовали. Например, X Window System в рамках X/Open и Common Desktop Environment в рамках OSF, а затем и The Open Group. Но ни один из стандартов не нашел широкой поддержки в индустрии, и, в частности, в сообществе разработчиков Linux. Также очень слабо стандартизована работа с конфигурационными файлами, содержащими системную информацию.

Во-вторых, описание требований на уровне исходного кода требует перекомпиляции приложений. Это не является большим препятствием при распространении приложений с открытыми исходными кодами, но существенно осложняет распространение приложений в двоичном виде. В последнем случае разработчику требуется иметь доступ ко всем поддерживаемым дистрибутивам для сборки своего приложения. Не легко приходится и конечному пользователю. Так как среди поставщиков операционной системы Linux нет устоявшейся традиции поддерживать обратную двоичную совместимость компонентов, то любое обновление системы может привести к потере работоспособности отдельных приложений. И даже в тех случаях, когда приложение поставляется с исходными кодами и пользователю для восстановления его работоспособности требуется только выполнить перекомпиляцию приложения, такая ситуация не приносит пользователю ничего кроме дополнительной головной боли.

Путь, по которому пошло сообщество Linux для преодоления проблем с переносимостью приложений, заключался в стандартизации базовых интерфейсов операционной системы на бинарном уровне. Для этого в 1998 году была образована некоммерческая организация Free Standards Group, под эгидой которой началась разработка стандарта Linux Standard Base (LSB). Деятельность по созданию стандарта поддержали Линус Торвальдс, Брюс Перенс, Эрик Раймонд, несколько поставщиков дистрибутивов Linux, разработчики приложений, члены Linux International, а также Джон Хаббард из проекта FreeBSD [LSB98].

Стандарт LSB изначально не ограничивал себя только операционной системой Linux и допускал возможность реализации альтернативных LSB-совместимых операционных систем. Тем не менее, потенциальная область действия LSB значительно уже области действия POSIX, что позволяло разработчикам надеяться на расширение возможностей по стандартизации интерфейсов.

На первый взгляд не очевидно, что сужение широты охвата реализаций может пойти на пользу стандарту. Тем не менее, это так. Ведь достижение консенсуса

целевой группы крайне необходимо при разработке стандарта, а разнообразие имеющихся реализаций всегда усложняет этот процесс. Например, широкая область действия POSIX неоднократно приводила к невозможности включения в стандарт важных и широко используемых функций из-за того, что разработчики различных реализаций не могли прийти к общему согласию. Это послужило причиной отсутствия в стандарте IEEE Std 1003.2 таких распространенных утилит, как `tar` и `cpio`, а многие другие интерфейсы вошли в стандарт с неполной функциональностью.

Итак, разработчики LSB выбрали для спецификации уровень двоичных интерфейсов и поставили перед собой цель описать все необходимые интерфейсы, чтобы сделать стандарт LSB универсальной платформой для всех разработчиков приложений под операционную систему Linux. Для этого они избрали тактику стандартизации устоявшейся практики, согласно которой в стандарт LSB включаются только требования, которым уже удовлетворяют все основные дистрибутивы Linux. В список таких дистрибутивов не сформулирован четко, но с уверенностью можно сказать, что в него входят RedHat, SuSe, Mandriva, Asianux, Debian, Ubuntu.

Каких результатов удалось добиться разработчикам LSB при помощи выбранной тактики? Рассмотрим это на примере текущей версии стандарта LSB 3.1.

3.1. Стандарт Linux Standard Base 3.1

Любой стандарт, описывающий интерфейсы на уровне двоичного кода, зависит от архитектуры аппаратного обеспечения, на которой должен работать двоичный код. В то же время, только небольшая часть стандарта LSB содержит требования, зависящие от аппаратной платформы. С учетом этого факта, а также того, что стандарт LSB поддерживает несколько аппаратных архитектур, текст стандарта построен по следующему принципу.

Все требования стандарта, которые не зависят от целевой аппаратной платформы, вынесены в отдельный документ, называемый «Общей спецификацией LSB». Кроме того, для каждой поддерживаемой платформы создан дополнительный документ, описывающий требования LSB, специфичные для данной платформы. Таким образом, полные набор требований стандарта LSB для некоторой платформы находится в двух документах: в общей спецификации LSB и в спецификации LSB для этой платформы.

Стандарт LSB 3.1 поддерживает 7 платформ: AMD64, IA32, IA64, PPC32, PPC64, S390 и S390X. AMD64 обозначает AMD64 Architecture [AMD64], IA32 — IA-32 Intel Architecture [IA32], IA64 – Intel Itanium Architecture [IA64], PPC32 — PowerPC Microprocessor Architecture [PPC32], PPC64 — 64-bit PowerPC Microprocessor Architecture [PPC64], S390 – IBM S/390 Architecture [S390] и S390X — IBM z/Architecture [S390X].

Стандарт LSB 3.1 состоит из трех модулей: LSB 3.1 Core, LSB 3.1 C++ и LSB 3.1 Desktop. Кроме того, существует четвертый, необязательный модуль, LSB Qt 4. Рассмотрим обязательные модули подробнее.

Требования стандарта LSB 3.1 Core можно разбить на следующие группы:

- базовые требования;
- требования к формату исполнимых файлов;
- требования к базовому бинарному интерфейсу;
- требования к командам и утилитам;
- требования к устройству файловой системы;
- требования к процессу инициализации;
- требования к пользователям и группам;
- требования к инсталляции программного обеспечения.

Базовые требования включают в себя определения таких ключевых элементов двоичного интерфейса как размеры базовых типов языка C, соглашения о вызове функций и т.д. Большая часть таких требований определяются в платформо-зависимых документах LSB.

Требования к формату исполнимых файлов заключаются в следующем. Операционная система обязана поддерживать выполнение исполнимых файлов в формате Executable And Linking Format (ELF) [ELF] с учетом дополнительных требований, сформулированных в стандарте LSB. LSB требует поддержки следующих элементов:

- дополнительных типов секций ELF файла;
- формата отладочной информации DWARF 2.0;
- версий бинарных символов;
- процесса динамического связывания.

Требования к базовому бинарному интерфейсу операционной системы являются одним из ключевых элементов стандарта. Стандарт предписывает системе содержать 12 разделяемых библиотек:

- интерпретатор программ-редактор динамических связей (ld-lsb.so);
- библиотека поддержки языка Си (libc);
- библиотека математических функций (libm);
- библиотека потоков управления POSIX (libpthread);
- независимая от языка библиотека поддержки компилятора gcc (libgcc_s);
- загрузчик динамических библиотек (libdl);
- библиотека функций реального времени (librt);
- библиотека функций криптографии (libcrypt);
- библиотека настраиваемого модуля идентификации (libpam);
- библиотека функций сжатия данных (libz);
- библиотека функций терминального интерфейса (libncurses);
- библиотека вспомогательных функций (libutil).

Кроме того, для каждой библиотеки стандарт определяет набор общедоступных бинарных символов и описывает требования к каждому отдельному символу. В общем виде требование к бинарному символу звучит следующим образом:

> В разделяемой библиотеке X должен присутствовать бинарный символ Y (версии Z) и при использовании этого символа как функции с заданной сигнатурой его поведение должно соответствовать требованиям стандарта.

В качестве X в утверждении фигурирует имя разделяемой библиотеки, которое может быть общим для всех архитектур (например, libdl.so.2) или зависеть от конкретной архитектуры (например, libc.so.6.1 для IA64 и libc.so.6 для остальных архитектур). Бинарный символ Y (например, `printf`) соответствует функции или глобальной переменной доступным для пользовательских приложений. Версия бинарного символа Z (например, GLIBC_2.2) присутствует только для функций из библиотек, входящих в состав glibc, так как среди разработчиков других библиотек поддержка версий бинарных символов пока не слишком распространена.

Чтобы продемонстрировать отличие стандарта на уровне двоичного кода от стандартов на уровне исходного кода, приведем аналогичное требование для стандарта IEEE Std 1003.1 (POSIX):

> При включении заголовочного файла X программа должна получить доступ к идентификатору Y и при использовании этого идентификатора как функции с заданной сигнатурой ее поведение должно соответствовать требованиям стандарта.

Разработчики LSB предпочли не дублировать описание функций, соответствующих бинарным символам, если в каком-либо из существующих стандартов данная функциональность уже специфицирована. Вместо дублирования текста они указывают ссылку на соответствующий стандарт. И даже если функциональность, реализованная в Linux-системах, отлична от стандартизованной ранее, в LSB обычно стоит ссылка на другой стандарт с дополнительным описанием отличий требований LSB от требований исходного стандарта.

Всего в стандарте LSB 3.1 Core описаны требования к 1532 функциональным бинарным символам. Следующая таблица демонстрирует распределение описаний функциональности этих символов по различным стандартам.

Название стандарта	Количество ссылок	Процент
ISO C99 [ISOC99]	41	2,68%
Large File System [LFS}	24	1,57%
LSB [LSBC31]	229	14,95%
SUS-CURSES [SUSX]	275	17,95%
POSIX 2004 [POSIX]	905	59,07%
SUSv2 [SUSv2]]	10	0,65%
SVID.3 [SVID3]	40	2,61%
SVID.4 [SVID4]	8	0,52%

Таблица 1. Количество ссылок на описания функций из различных стандартов в LSB 3.1.

Важной особенностью стандартов, описывающих интерфейсы на уровне двоичных кодов, является фиксация значений всех констант, а также фиксация размеров всех типов и смещений полей во всех структурах данных. Действительно, если POSIX определяет, что в заголовочном файле должна быть определена константа с заданным идентификатором и структура с заданным именем и набором полей, то этого оказывается достаточно для корректной работы POSIX-совместимого приложения. Конкретные значения констант и смещения полей внутри структур определяются на этапе компиляции программы. Для LSB, который стандартизирует уже скомпилированные приложения, все эти значения должны быть зафиксированы в самом стандарте.

Требования к командам и утилитам в стандарте LSB определяют необходимость наличия и правил функционирования 5 команд и 133 утилит. По своей структуре описания утилит ничем не отличаются от описания утилит в стандарте IEEE Std 1003.1 (POSIX), так как для утилит не существует различия между уровнем исходных кодов и уровнем исходных кодов.

Требования к устройству файловой системы стандарта LSB базируются на стандарте Filesystem Hierarchy Standard (FHS) 2.3 [FHS]. В дополнение к FHS 2.3 LSB требует наличие файлов устройств /dev/null, /dev/zero и /dev/tty, а также семи специальных директорий в каталоге /etc. Также стандарт LSB регламентирует, чтобы приложения использовали для именования конфигурационных файлов в директории /etc либо короткие имена, зарегистрированные в Linux Assigned Names and Numbers Authority (LANANA) [LANANA], либо, так называемые, иерархические имена, включающие в себя DNS имя, зарегистрированное за разработчиком.

Требования к процессу инициализации определяют переносимое окружение для инициализационных скриптов приложения, которые автоматически запускаются при старте и завершении работы системы. Это окружение включает в себя:

- интерфейс, посредством которого приложения могут переносимым образом регистрировать и удалять инициализационные скрипты из системы;
- определение семантики уровней выполнения (run levels);
- определение идентификаторов стандартных устройств для использования при определении зависимостей инициализационных скриптов (например: `$local_fs`, `$network`, `$remote_fs`);
- набор переносимых функций командного процессора для использования в инициализационных скриптах.

Требования к пользователям и группам стандарта LSB заключаются в определении трех обязательных пользователей и групп, которые обязаны присутствовать во всех системах, и набора опциональных системных пользователей и групп.

Требования к инсталляции программного обеспечения предназначены для предоставления возможности реализовать инсталляцию/деинсталляцию приложений переносимым образом. Для достижения этой цели LSB определяет формат файла, содержащего дистрибутив приложения как подмножество формата RPMv3 (Red Hat Package Manager), разработанного компанией Red Hat, Inc [RPM30]. При этом LSB не требует использования утилиты rpm или поддержки базы данных RPM, что позволяет реализовать LSB-совместимую инсталляцию на основе альтернативных механизмов управления программным обеспечением. Кроме того, LSB допускает возможность распространения приложений не только в виде пакетов RPM, но и в других формах, при условии, что используемый инсталлятор распространяется вместе с приложением и является LSB-совместимым приложением.

LSB 3.1 C++ содержит требования следующих видов:

- представление C++ данных;
- правила преобразования идентификаторов;
- бинарный интерфейс базовых библиотек.

Требования к представлению C++ данных описывают правила представления C++ классов в памяти машины. Правила преобразования идентификаторов определяют, как должно происходить преобразование идентификаторов языка программирования C++ в бинарные символы объектного кода. И наконец, описание бинарного интерфейса базовых библиотек C++ содержит требования к составу и функциональности 1942 бинарных символов из библиотеки поддержки языка C++.

LSB 3.1 Desktop определяет требования к бинарному интерфейсу библиотек, предназначенных для работы приложений ориентированных на взаимодействие с пользователем. В число этих библиотек входят:

- графические библиотеки (libX11, libSM, libICE, libXt, libXext, libXi);
- библиотеки OpenGL (libGL);
- библиотеки PNG (libpng12);
- библиотеки JPEG (libjpeg);
- библиотеки конфигурирования шрифтов (libfontconfig);
- библиотеки семейства GTK+ (libglib-2.0, libgobject-2.0, libgmodule-2.0, libatk-1.0, libpango-1.0, libpangoxft-1.0, libpangoft2-1.0, libgdk_pixbuf-2.0, libgdk_pixbuf_xlib-2.0, libgdk-x11-2.0, libgtk-x11-2.0);
- библиотека Qt3 (libqt-mt);
- библиотека XML2 (libxml2).

Всего из перечисленных выше библиотек стандартизовано 19103 бинарных символа. Также в LSB 3.1 Desktop входят 3 утилиты, предназначенные для конфигурирования шрифтов.

Мы рассмотрели текущее состояние стандарта LSB. Тому, как этот стандарт развивался и как он достиг этого состояния, посвящен следующий раздел.

3.2. История Linux Standard Base

Работа по разработке стандарта LSB стартовала в начале 1998 года, и уже в октябре 1999 появился первый вариант стандарта LSB 0.1. После серии предварительных версий 29 июня 2001 года была выпущена первая официальная версия стандарта — LSB 1.0. Эта версия состояла из единого документа, в котором были объединены требования частей LSB Core и небольшого подмножества LSB Desktop. Первоначально стандарт поддерживал только одну архитектуру — IA32.

Прикладной бинарный интерфейс версии 1.0 состоял из требований к 15 библиотекам, в которых было стандартизовано 3040 функций:

Имя библиотеки	Комментарий	Число стандартизованных интерфейсов
libc	Библиотека поддержки языка Си	710
libm	Математические функции	273
libpthread	Потоки управления	75
libdl	Загрузчик динамических библиотек	5
libcrypt	Криптография	3
librt	Функции реального времени	23
libz	Сжатие данных	40
libncurses	Терминальный интерфейс	268
libutil	Вспомогательные функции	6
libX11	Функции X Windows	720
libXt	Функции X Toolkit	304
libGL	Функции OpenGL	451
libXext	Функции расширения X Windows	76
libICE	X Inter-Client Exchange (ICE) Protocol	49
libSM	Управление сессиями	37
Всего:		**3040**

Таблица 2. Библиотеки, входящие в LSB и количество функций в них.

Версия 1.1 была выпущена 22 января 2002 года. Основные изменения затронули библиотеку libc, в которой появилось 68 новых функций. Большую часть из них составили функции поддержки RPC (Remote Procedure Call). Кроме того, в библиотеке libpthread были стандартизованы функции работы с атрибутом `pshared` мьютексов, в библиотеке librt — функции работы с таймерами, а в библиотеке libm — функция `clog`. Из графических библиотек наоборот было удалено 22 функции, 14 из которых вновь появились в версии LSB 1.2, опубликованной 28 июня 2002 года.

Наиболее важной особенностью версии 1.2 стала поддержка второй архитектуры (PPC32). Также в этой версии бинарный интерфейс был расширен 6 новыми функциями. На базе версии 1.2 FSG в августе 2002 года объявила о начале сертификационной программы.

В LSB 1.3, выпущенной 17 декабря 2002 года, была реализована поддержка широкого ряда архитектур (IA32, IA64, PPC32, S390, S390X) и была проведена реорганизация базовых библиотек. Библиотеку librt удалили из стандарта, так как асинхронный ввод-вывод, составляющий основную ее часть, не полностью поддерживался всеми дистрибутивами операционной системы Linux. Зато в стандарте появилось две новые библиотеки: libpam (настраиваемый модуль идентификации) и libgcc_s (независимая от языка библиотека поддержки компилятора gcc).

В LSB 2.0, опубликованном 31 августа 2004 года, впервые был стандартизован бинарный интерфейс библиотеки поддержки языка C++: libstdc++. Другим важным изменением стало то, что до того монолитный текст стандарта был разделен на три модуля: LSB Core, LSB C++ и LSB Graphics. Версия 2.1 от 11 марта 2005 года содержала небольшие изменения, коснувшиеся уточнения интерфейсов libc, libm, libpthread и libz.

Следующая версия стандарта LSB 3.0 появилась 6 июля 2005 года. При ее разработке началась работа по согласованию стандарта LSB с требованиями стандарта IEEE Std 1003.1 (POSIX). Среди ключевых изменений следует отметить переход на новый бинарный интерфейс C++, который был реализован в компиляторе gcc 3.4.4, а также добавление новой библиотеки libXi в состав модуля LSB Graphics и возвращение в ряды LSB Core библиотеки librt.

В составе библиотеки librt вернулись не все функции, а только функции работы с часами, таймерами и разделяемой памятью. А функции асинхронного ввода-вывода по-прежнему остались за бортом LSB 3.0.

В библиотеке libc были стандартизованы функции `duplocale`, `freelocale`, `newlocale`, `uselocale`, `getgrouplist`, `posix_openpt`, `getlogin_r`. Первые четыре из них – функции управления локализацией на уровне потоков управления, которые получили достаточно широкое распространение среди приложений. Остальные функции вошли в стандарт как функции, описанные в POSIX, и уже присутствующие во всех основных дистрибутивах Linux.

Новые функции появились и в библиотеке libpthread. В их число вошли следующие функции из стандарта POSIX:

`pthread_attr_getinheritsched`,
`pthread_attr_getschedpolicy`, `pthread_attr_getscope`,
`pthread_attr_setscope`, `pthread_attr_setschedpolicy`,
`pthread_getschedparam`, `pthread_setschedparam`,
`pthread_attr_setinheritsched`, `pthread_setschedprio`.

В набор команд и утилит стандарта LSB 3.0 были добавлены `ed`, `logger`, `lp`, `mailx`, `pax`, `cd`, `getopts`, `read`, `umask` и `wait`. Эти изменения в стандарте также были вызваны работами по согласованию LSB и POSIX.

Еще одно типичное изменение в стандарте LSB 3.0 было вызвано изменением в размерах структур данных, передаваемых в качестве параметров в функции стандартизованного интерфейса. В glibc была реализована поддержка новых процессоров PowerPC, для чего потребовалось добавить новое поле в структуру

mcontext_t, являющуюся частью стуктуры ucontext_t. В связи с этим у функций бинарного интерфейса (__sigsetjmp, _longjmp, _setjmp, longjmp, siglongjmp, getcontext, setcontext, swapcontext), работающих с ucontext_t, была обновлена версия до GLIBC_2.3.4. Эти изменения были отражены в стандарте LSB 3.0: в спецификациях LSB 3.0 для платформ PPC32 и PPC64 версии бинарных символов, перечисленных выше, также были обновлены до GLIBC_2.3.4.

LSB 3.1 публиковался по частям: LSB 3.1 Core появился 27 октября 2005 года, LSB 3.1 C++ и LSB 3.1 Desktop — 24 апреля 2006 года. Такое решение было связано с переговорами о присвоении стандарту LSB Core статуса международного стандарта ISO. Переговоры на эту тему начались еще во время работы над LSB 2.0, а осенью 2005 года этот процесс успешно завершился и LSB 3.1 Core получил статус международного стандарта ISO/IEC 23360.

С технической точки зрения LSB 3.1 отличается в первую очередь существенным расширением стандартизованных интерфейсов, предназначенных для приложений с графическим пользовательским интерфейсом. Это расширение привело к переименованию модуля LSB Graphics в LSB Desktop. Появление опционального модуля LSB Qt 4 связано с тем, что разработчики стандарта подготовили описание новой версии библиотеки Qt, но так как еще не все основные дистрибутивы Linux перешли на эту версию, то требование на наличие библиотеки Qt 4 было сделано необязательным.

Мы рассмотрели состав стандарта LSB 3.1, изучили историю развития стандарта. Теперь пришло время обсудить, что ожидает стандарт LSB в ближайшем будущем.

3.3. Будущее Linux Standard Base

Дальнейшие планы по развитию стандарта LSB предусматривают появление версии 3.2 весной 2007 года и версии 4.0 в начале 2008 года.

Основными направлениями по развитию LSB 3.2 являются:

- дальнейшая интеграция стандарта с требованиями POSIX;
- включение результатов проекта freedesktop.org в состав LSB Desktop;;
- интеграция результатов рабочих групп по поддержке печати (printing) и доступности (accessibility).

В рамках интеграции LSB с POSIX планируется добавить в стандарт очереди сообщений (mq_*), функции из наборов POSIX Advisory Information и POSIX Spawn Interfaces (pthread_spawn_*). Кроме того, рассматривается возможность включения дополнительных интерфейсов функции ioctl для сокетов. Кандидаты перечислены в следующем списке (жирным шрифтом выделены уже стандартизованные интерфейсы):

SIOCGIFNAME, SIOCSIFNAME, SIOCSIFLINK,

SIOCGIFCONF, **SIOCGIFFLAGS**, SIOCSIFFLAGS,
SIOCGIFADDR, SIOCSIFADDR, SIOCGIFDSTADDR,
SIOCSIFDSTADDR, SIOCGIFBRDADDR, SIOCSIFBRDADDR,
SIOCGIFNETMASK, SIOCSIFNETMASK.

Проект freedesktop.org имеет своей целью разработку универсального интерфейса, который позволит приложениям единообразно работать с двумя наиболее популярными графическими оболочками Linux: KDE и GNOME. В рамках LSB 3.2 планируется стандартизовать только небольшую часть результатов этого проекта, касающуюся информации о содержимом “рабочего стола”. Но в дальнейшем LSB планирует расширять сотрудничество с freedesktop.org.

Большая активность ведется в рамках FSG по обеспечению универсальных механизмов управления печатью и драйверами принтеров. В частности, рабочей группой OpenPrinting был разработан интерфейс PAPI (Printing API) для печати из приложений. Однако, включение этого интерфейса в LSB 3.2 невозможно, так как его реализации пока отсутствуют в основных дистрибутивах Linux.

В настоящее время большинство дистрибутивов Linux, а также MacOS используют для печати CUPS (Common UNIX Print System). Однако, CUPS отсутствует в Solaris, AIX и различных версиях BSD систем. Поэтому стандартизация интерфейсов CUPS также пока находится под вопросом.

Еще одним потенциальным элементом для стандартизации в LSB 3.2 является структура директорий, в которых располагаются драйвера принтеров и ppd файлы, содержащие описания принтеров. Однако, здесь также возникают проблемы с отсутствием устоявшейся практики использования этих директорий в существующих дистрибутивах.

По запросам рабочей группы по доступности Linux планируется добавить в стандарт интерфейсы из следующих графических библиотек: libXcomposite, libXdamage, libXfixes, libXrender, libXft.

Относительно версии 4.0 на настоящий момент нет никаких достоверных планов. Известно, что эта версия должна стандартизовать интерфейсы, которые будут доступны в дистрибутивах Linux следующего поколения (RHEL 6 и SLES 11). Ориентировочные направления, декларированные главой LSB Workgroup Яном Мердоком, следующие:

- интеграция обновлений в компиляторе GCC и базовых библиотеках (glic и др.), влияющих на бинарные интерфейсы;
- требование бинарной совместимости с LSB 3, как шаг на пути бинарной совместимости между различными версиями дистрибутивов;
- стандартизация библиотек поддержки дополнительных языков программирования (Perl, Python, LAMP и др.);
- появление дополнительных модулей LSB на основе результатов работы дополнительных рабочих групп (мультимедийных, безопасности, идентификации и т.д.).

3.4. Позиции Linux Standard Base

Стандарт LSB нашел поддержку в индустрии. Многие дистрибутивы прошли сертификацию на соответствие LSB. Среди них SuSe Linux Enterprise Server, SuSe Linux, RedHat Enterprise Linux, Asianux, Mandrakelinux Corporate Server, SGI ProPack for Linux[3]. Хотя есть и такие дистрибутивы, которые нацелены на работу на переднем крае достижений в мире Linux и не собираются участвовать в коммерческой деятельности. Поэтому они не заинтересованы в сертификации и поддержке LSB. В качестве примера такого дистрибутива можно привести Fedora Core.

Пользователи и поставщики приложений также приняли участие в распространении стандарта. Одним из способов участия стало то, что отдельные игроки на рынке Linux и, в частности, IBM включили сертификацию на соответствие LSB в качестве обязательного требования при заключении контрактов со своими партнерами. Существенную роль в поддержке стандарта также сыграли такие корпорации как Intel и HP.

В то же время нельзя сказать, что разработчики стандарта LSB не испытывали проблем. Основное разочарование, связанное со стандартом, явилось следствием завышенных ожиданий сформировавшихся до его создания. Многие считали, что с появлением LSB можно будет один раз скомпилировать приложение, и оно будет работать на всех дистрибутивах Linux и других LSB-совместимых операционных системах для данной аппаратной платформы. К сожалению, действительность оказалась несколько сложнее. Несмотря на постоянный рост LSB многим приложениям по-прежнему не хватает функциональности, описанной в стандарте, для реализации всех потребностей переносимым образом. LSB развивается в этом направлении, стандартизируя все больше неохваченных ранее интерфейсов между приложениями и операционной системой. Но темпы оказываются недостаточными, чтобы принцип «собрал однажды — работает везде» стал доступен для подавляющего большинства приложений уже в ближайшем будущем.

Еще одна проблема на пути к достижению работоспособности этого принципа заключается в отсутствии обратной совместимости между основными версиями стандарта. Это означает, что приложение должно быть перекомпилировано для каждой такой версии. Конечно, основных версий стандарта значительно меньше, чем различных дистрибутивов Linux, но все равно это нарушение принципа, которое является одним из источников обманутых ожиданий. Причина отсутствия обратной совместимости заключается в постоянном изменении бинарных интерфейсов и проблематичности поддержания их совместимости. На уровне двоичных кодов значительно меньше гибкости по сравнению с уровнем исходных кодов. Хотя поддержка версий бинарных символов в совокупности с другими механизмами

[3] Наиболее полный список сертифицированных дистрибутивов представлен на сайте Free Standards Group (www.freestandards.org). Текущий адрес списка: http://www.freestandards.org/en/Products

и позволяет организовать обратную совместимость, но это особенно сложно сделать стандарту, если разработчики его реализаций не озабочены проблемой обратной совместимости. FSG продекларировало, что попытается обеспечить обратную совместимость LSB 4 относительно LSB 3, но никаких деталей относительно того, как это будет сделано на настоящий момент не известно. Остается надеяться, что разработчикам FSG удастся реализовать свои намерения.

Другая проблема стандарта LSB заключается в разрыве между бинарным уровнем стандарта и исходным кодом приложений. Разработчики LSB предоставляют вместе со стандартом набор инструментов, предназначенных для облегчения разработки LSB-совместимых приложений. Но функциональности инструментов и особенно документации по их использованию не хватает разработчикам приложений для того, чтобы разработка LSB-совместимых приложений стала простым и понятным делом.

Также следует отметить, что информация о стандарте LSB недостаточно распространена среди разработчиков приложений. Данные первого года программы IBM Systems Application Advantage for Linux (также известной как "Chiphopper") [Chi06] показали, что только 25% из разработчиков приложений, участвующих в программе, знали о стандарте LSB до начала участия.

Тактика стандартизации устоявшейся практики основана на достижении консенсуса апостериори. Однако она оказывается неэффективной в тех случаях, когда устоявшегося решения насущной проблемы еще нет. К чести LSB Workgroup во многих случаях она пытается решать такие проблемы, занимая достаточно активную позицию. Но это объясняется скорее личной инициативой ее членов, чем продуманной позицией группы.

И, наконец, наибольшей критики со стороны поставщиков дистрибутивов LSB Workgroup заслужила за низкое качество сертификационного тестового набора LSB. Особенное недовольство было выражено компанией Red Hat, так как компания потратила на сертификацию Red Hat Enterprise Linux 4 на соответствие LSB 3.0 почти 10 недель. Основной причиной задержек стали ошибки в сертификационном тестовом наборе, приводившие к падению тестовой системы или некорректному вынесению вердикта. Причины этих ошибок заключались в том, что тестовый набор разрабатывался на других UNIX-подобных системах с использованием неявных предположений о поведении системы, которые оказались неверны для операционной системы Linux. А LSB Workgroup не обеспечило должного уровня тестирования сертификационного тестового набора, что и привело к недовольству ее пользователей.

В LSB Workgroup также хорошо понимают, насколько важна роль качественного тестового набора для успешного распространения стандарта. Глава LSB Workgroup Ян Мердок сказал по этому поводу следующее:

> The LSB is an interface standard, and an interface standard is only as good as the tests suites that test those interfaces, so we absolutely have to do better here.

«LSB стандартизирует интерфейсы между приложением и операционной системой, а стандарт на интерфейсы настолько хорош, насколько хорош тестовый набор, предназначенный для тестирования этих интерфейсов. Таким образом, мы безусловно должны стать лучше в отношении тестового набора.»

Действительно, так как стандарт претендует на роль надежной платформы для разработчиков приложений и предполагает избавить их от тщательного тестирования приложений на всех возможных дистрибутивах Linux, то сертификационный тестовый набор должен обеспечивать тщательное тестирование дистрибутивов на соответствие стандарту. В противном случае, стандарт не сможет достичь поставленных целей.

5. Заключение

Так сумеет ли Linux Standard Base предотвратить фрагментацию рынка приложений для операционной системы Linux? Сможет ли он обеспечить надежную платформу для разработчиков приложений? На настоящий момент Linux Standard Base не достиг этих целей. Достигнет ли он их в будущем? Будет ли история развития стандарта историей успеха или станет историей одной из многих попыток стандартизации бинарного интерфейса? Ответ на все эти вопросы даст только время.

А мы можем лишь констатировать, что во многом положительный ответ будет зависеть от того, как LSB Workgroup справится со всеми рассмотренными выше ключевыми проблемами:

- предоставлением всех необходимых интерфейсов для разработчиков приложений, в том числе в тех областях, где еще нет сложившихся, проверенных временем решений;
- обеспечением обратной совместимости основных версий стандарта;
- наличием простых, понятных и удобных инструментов для разработчиков приложений;
- разработкой качественного сертификационного тестового набора, обеспечивающего проведение тщательного тестирования соответствия стандарту.

Литература

[AMD64] Linux Standard Base Core Specifications for AMD64 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-AMD64/LSB-Core-AMD64.pdf

[Chi06] Kay A. Tate, Narender Ramireddy. First Year Chiphopper ISV Experience. IBM Chiphopper Team, June 2006.

[ELF] Executable and Linking Format. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic/elf-generic.html

[FHS] Filesystem Hierarchy Standard (FHS) 2.3. http://www.pathname.com/fhs/

[IA32] Linux Standard Base Core Specifications for IA32 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-IA32/LSB-Core-IA32.pdf

[IA64] Linux Standard Base Core Specifications for IA64 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-IA64/LSB-Core-IA64.pdf

[ISOC99] ISO/IEC 9899-1999. Programming Languages — C. Geneve: ISO, 1999.

[LANANA] Linux Assigned Names and Numbers Authority. http://www.lanana.org/

[LFS] Large File Support. http://www.UNIX-systems.org/version2/whatsnew/lfs20mar.html

[LSBC31] ISO/IEC 23360-1-8:2005, Linux Standard Base (LSB) Core Specification 3.1. Geneve: ISO, 2005.

[LSB98] Project Proposal and Call for Participation: The Linux Standard Base (LSB) Project. http://old.lwn.net/1998/0528/a/lsb.html

[Mur06] http://www.freestandards.org/en/LSB_Summit.

[POSIX] IEEE 1003.1-2004. Information Technology — Portable Operating System Interface (POSIX). New York: IEEE, 2004.

[PPC32] Linux Standard Base Core Specifications for PPC32 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-PPC32/LSB-Core-PPC32.pdf

[PPC64] Linux Standard Base Core Specifications for PPC64 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-PPC64/LSB-Core-PPC64.pdf

[RPM30] RPM Package Format V3.0. http://www.rpm.org/

[S390] Linux Standard Base Core Specifications for S390 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-S390/LSB-Core-S390.pdf

[S390X] Linux Standard Base Core Specifications for S390X 3.1. http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-S390X/LSB-Core-S390X.pdf

[SUSv2] CAE Specification, January 1997, System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606).

[SUSX] CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), plus Corrigendum U018.

[SVID3] American Telephone and Telegraph Company, System V Interface Definition, Issue 3; Morristown, NJ, UNIX Press, 1989.(ISBN 0201566524).

[SVID4] System V Interface Definition, Fourth Edition.