Текущее состояние и перспективы развития инфраструктуры LSB

Д. В. Силаков
silakov@ispras.ru

Аннотация. В статье рассказывается о технической стороне разработки стандарта Linux Standard Base и связанной с ним инфраструктуре. Описывается использование базы данных для хранения части информации, входящей в стандарт. Обсуждается процесс генерации на основе этих данных как непосредственно текста стандарта, так и сопутствующих объектов – наборов элементарных тестов, заголовочных файлов, отвечающих стандарту LSB, и пр. Также рассматриваются задачи по развитию существующей инфраструктуры, которые планируется решить в рамках совместного проекта ИСП РАН и организации Free Standards Group, под эгидой которой проводится разработка стандарта LSB.

1. Введение

Стандарты играют существенную роль в современном мире программного обеспечения, внося единообразие в сложные программные комплексы. Наличие стандартов в области операционных систем позволяет разработчикам создавать приложения, взаимодействующие с операционной системой, не изучая при этом детали реализации объекта взаимодействия.

Для написания программы, соответствующей какому-либо стандарту, разработчику необходимо изучить текст этого стандарта. Большинство стандартов попадают к пользователям в виде электронных документов либо напечатанных книг, и объем их достаточно велик. Однако во многих случаях процесс разработки упростится, если некоторые сведения из стандарта будут доступны в какой-то другой форме, более формальной, чем текст, и не требующей изучения человеком. Например, для гарантии того, что в программе будут использоваться только описанные стандартом функции, специфицированные в заданных заголовочных файлах, полезно иметь такие файлы, содержащие описания только стандартных функций. Использование только этих заголовочных файлов при создании программы гарантировало бы отсутствие обращений к функциям, не входящим в стандарт. Но ручное создание таких заголовочных файлов на основе текста стандарта – трудоемкое занятие (например, стандарт LSB [1] содержит описание 18955 функций из 465 заголовочных файлов), и во многих случаях (особенно при разработке

небольших приложений) разработчики предпочитают полагаться на свое знание текста стандарта.

Вполне возможно, что у создателей стандарта имеется необходимая информация в виде, удобном для разработчика, однако процесс создания большинства стандартов скрыт от постороннего взора. Тексты многих стандартов доступны только за плату, а доступа к данным, которые (возможно) используются при создании текста стандарта, нет.

Нетрудно видеть, что такое положение вещей противоречит идеологии open source, согласно которой пользователь должен иметь возможность получить не только готовый продукт, но и все необходимое для его самостоятельной сборки и внесения в него изменений. Конечно, требование открытости относится, прежде всего, к исходным кодам программ, однако многие сторонники open source переносят эту идеологию и на смежные области. И одним из примеров здесь является стандарт LSB, описывающий базовые интерфейсы операционной системы на бинарном уровне. Во Free Standards Group [2], под эгидой которой проводится разработка LSB, используется специфический подход, открывающий всем желающим всю «кухню» по производству стандарта. Каждый может бесплатно получить все необходимое не только для генерации текста стандарта, но и для автоматического создания сопутствующих наборов программ и исходного кода. О том, как устроена инфраструктура, связанная со стандартом LSB, и какие существуют планы по ее развитию, и рассказывается в данной статье.

2. База данных стандарта LSB

Во многих случаях работа со стандартом упростилась бы при наличии описания объектов стандарта и их взаимосвязей в виде, более пригодном для обработки различными программными инструментами, чем обычный текст. Так, в примере из введения генерации заголовочных файлов, содержащих только стандартизованные функции, удобно было бы иметь готовые списки заголовочных файлов и функций, а также знать, какие функции описаны в каждом заголовочном файле.

Free Standards Group дает возможность получить такую информацию для всех сущностей, определенных в стандарте LSB.

2.1. Информация об объектах, описанных в LSB

Для стандарта LSB в качестве хранилища объектов, описываемых стандартом, основной информации о них и связей между ними используется так называемая *база данных стандарта LSB (LSB Specification Database)*. Более точно, в этой базе данных содержится информация о следующих объектах:

- библиотеки (таблица Library);
- классы (таблица ClassInfo);
- интерфейсы – LSB является бинарным стандартом и описывает

интерфейсы, предоставляемые каждой библиотекой. Все интерфейсы хранятся в таблице Interface и могут иметь следующие типы:

  - Function (функции);
  - Data (данные);
  - Common («общие интерфейсы», к которым относятся stdin, stdout и им подобные);
  - Alias (интерфейсы, являющиеся синонимами других интерфейсов);
- заголовочные файлы (таблица Header);
- константы (таблица Constant);
- типы данных (таблица Type);
- команды (таблица Command; под командами здесь понимаются как встроенные команды shell, так и различные утилиты);
- секции исполняемых файлов формата ELF (таблицы ElfSections и SectionTypes);
- теги rpm-файлов (таблица RpmTag).

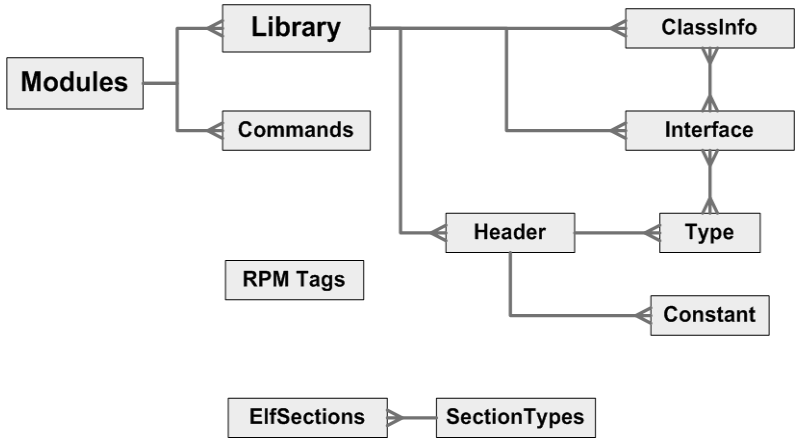

Рис.1. ER-диаграмма сущностей, описываемых стандартом LSB.

Также в базе данных хранятся связи между указанными объектами – каждая константа привязана к заголовочному файлу, в котором она объявляется, интерфейс – к библиотеке, в которой он содержится, либо к классу, если это метод класса, и т.д. ER-диаграмма существующей базы данных приведена на Рис. 1. Все сущности группируются в так называемые *модули* согласно своему назначению (например, модуль LSB_Cpp содержит все, относящееся к стандартной библиотеке C++, LSB_Toolkit_Qt3 – все, относящееся к библиотеке Qt3, и т.д.). Информация о форматах файлов ELF и RPM относится

к модулю LSB_Core, однако база данных этого факта никак не отражает – об этом «знают» только скрипты, генерирующие текст стандарта.

Связи типа «многие ко многим» реализуются посредством использования отдельных таблиц. Самой сложной структурой обладает взаимосвязь таблиц ClassInfo и Interface. Для ее реализации используется несколько вспомогательных таблиц, содержащих информацию о виртуальных таблицах класса (таблица Vtable; каждый класс может иметь одну либо две виртуальные таблицы) и о наследовании (таблица BaseTypes реализует обычное наследование, при описании множественного наследования используется также таблица VMIBaseTypes).

Для удобства интерфейсы и классы, входящие в одну библиотеку, разбиваются на группы согласно их назначению. Информация о таких группах, на которые разбиваются библиотеки, содержится в таблице LibGroup. Так, например, библиотека librt (функции реального времени) разделена на три группы:

- Shared Memory Objects (функции для работы с разделяемой памятью);
- Clock (функции для работы с часами);
- Timers (функции для работы с таймерами).

Аналогично группируются константы и типы данных, описанные в одном заголовочном файле. Информация о таких группах содержится в таблице HeaderGroup. Так, например, файл rpc/rpc_msg.h, который содержит декларации типов и функций для работы с сообщениями, передаваемыми при удаленном вызове процедур (RPC, Remote Procedure Call), делится на следующие группы:

- accepted_reply (типы, которые описывают ответ на rpc-запрос, принятый сервером);
- rejected_reply (типы, которые описывают ответ на rpc-запрос, отвергнутый сервером);
- reply_body (типы, описывающие тело ответа на rpc-запрос);
- call_body (типы, описывающие тело rpc-запроса);
- rpc_msg (типы, описывающие весь rpc-запрос);
- base types (основные типы);
- default HeaderGroup (сюда относится все, не вошедшее в перечисленные выше группы).

Заметим, что типы одной группы могут быть составными типами, определяемыми через типы других групп. Например, типы из группы rpc_msg – это структуры, содержащие тип из call_body либо reply_body и некоторые дополнительные атрибуты.

2.2. Информация об архитектурах

Стандарт LSB специфицирует интерфейсы операционной системы на бинарном уровне. Естественно, это делает стандарт зависимым от архитектуры аппаратного обеспечения – для каждой архитектуры необходима своя версия стандарта. Поэтому стандарт LSB содержит различные документы для различных аппаратных платформ – например, все, что относится к стандартной библиотеке C++, описывается в документах LSB-CXX-IA32 (для архитектуры IA32), LSB-CXX-AMD64 (для архитектуры AMD64) и т.д (в LSB версии 3.1 поддерживаются семь архитектур - AMD64, IA32, IA64, PPC32, PPC64, S390 и S390X). Кроме того, есть так называемая *общая спецификация LSB (LSB Generic Specification)*, описывающая объекты, которые должны присутствовать во всех LSB-совместимых реализациях, независимо от аппаратной платформы.

Для хранения списка аппаратных платформ в схеме базы данных предусмотрена таблица Architecture. Многие объекты, информация о которых содержится в базе данных, включены в спецификации для одних платформ и отсутствуют в спецификациях для других. Кроме того, каждый объект, описываемый стандартом, на каждой платформе может иметь какие-то специфические особенности. Поэтому между таблицами объектов и таблицей архитектур существуют связи «многие ко многим», которые реализуются посредством отдельных таблиц, содержащих пары «идентификатор объекта, идентификатор архитектуры». Эти таблицы также содержат те свойства объектов, которые могут иметь различные значения на различных архитектурах (например, значения констант).

Среди идентификаторов архитектур есть одно выделенное значение – «All». Если объект приписан к архитектуре с этим идентификатором, то он должен присутствовать во всех архитектурах, поддерживаемых LSB, и при этом на всех архитектурах его свойства должны быть одинаковы (допускается одно исключение для версий интерфейсов – для интерфейса, приписанного к архитектуре All, можно завести отдельную запись-привязку к конкретной архитектуре, указав там значение версии, не меньшее, чем для архитектуры All).

Безусловно, можно было бы обойтись и заведением необходимого количества записей для объекта (по одной записи для каждой поддерживаемой архитектуры), но идентификатор «All» имеет дополнительную смысловую нагрузку – именно объекты, привязанные к архитектуре All, попадают в общую спецификацию LSB. При отсутствии такого идентификатора для каждого объекта приходилось бы проверять, что он привязан ко всем поддерживаемым архитектурам и при этом обладает на каждой из них одними и теми же свойствами. Такие проверки существенно усложнили бы создание генераторов текста стандарта и связанных с ним объектов, о которых пойдет речь в следующих двух разделах.

3. Генерация текста стандарта

База данных активно используется при генерации текста стандарта LSB. Ведь основное содержание всех документов LSB – это перечень интерфейсов, которые должны присутствовать в системе, и их описаний. Интерфейсы разбиваются по библиотекам либо по классам; кроме того, внутри каждой библиотеки возможна дополнительная группировка, как было указано в предыдущем разделе. Также указывается, какие заголовочные файлы необходимо подключить для использования того или иного интерфейса, а для интерфейсов-функций приводится их сигнатура.

Здесь снова стоит отметить, что стандарт LSB предназначен для обеспечения переносимости приложений на бинарном уровне, а не на уровне исходного кода. Для бинарной переносимости не важно, в каких заголовочных файлах определяются конкретные объекты – главное, чтобы эти объекты находились в заданных стандартом библиотеках. Поэтому в LSB описывается расположение объектов по заголовочным файлам, принятое в большинстве дистрибутивов Linux. Цель этого описания – помочь разработчикам определить местоположение нужного интерфейса в ходе написания прог-раммы; однако действительное место описания объекта в конкретном дистри-бутиве может отличаться от приведенного в LSB, при этом требования стан-дарта не будут нарушены, если объект входит в предписанную библиотеку.

Перечни интерфейсов с краткой информацией создаются специальными скриптами на основе сведений из базы данных стандарта LSB. Скрипты создают текст стандарта в формате sgml, используя в некоторых случаях заранее заготовленные шаблоны. На основе сгенерированных sgml-файлов создаются файлы в наиболее распространенных форматах – html, rtf, ps, pdf, а также файлы в текстовом формате (естественно, лишенного такого преимущества остальных форматов, как навигация по ссылкам внутри документа, что полезно при чтении стандарта).

На основе соответствующих таблиц также создаются списки констант и опре-деляемых пользователем типов (перечислений, структур и т.п.), сгруппи-рованных по заголовочным файлам, в которых они описаны, список секций исполняемых файлов формата ELF и список тэгов файлов формата RPM.

Что касается более детальных описаний интерфейсов (например, описания того, что делает функция), то здесь стандарт LSB в большинстве случаев ссылается на другие стандарты (такие, как POSIX, стандарт ISO/IEC 9899 для языка C и т.д.). Информация о том, где искать описание каждого конкретного объекта, также содержится в базе данных стандарта – для этого заведена отдельная таблица, содержащая все стандарты, на которые есть ссылки из LSB, и каждая запись в таблицах объектов ссылается на запись в таблице стандартов. Соответственно в тексте стандарта рядом с каждым интерфейсом указано, где искать его подробное описание.

Описания некоторых интерфейсов содержится непосредственно в LSB, но их число очень невелико (на ноябрь 2006 г. – 454 из 32721 интерфейса, включенных в текст стандарта). Эти описания создаются вручную (также в формате sgml) и автоматически включаются в нужные места при генерации стандарта.

4. Генерация зависящих от стандарта объектов

Free Standards Group предоставляет средства для генерации не только самого текста стандарта, но и различных объектов, зависящих от этого стандарта. К таким объектам относятся:

- библиотеки-«заглушки» («stub libraries»);
- заголовочные файлы с описаниями объектов, определенных в стандарте LSB;
- элементарные тесты для проверки соответствия дистрибутива либо приложения стандарту LSB.

Рассмотрим эти объекты подробнее.

4.1. Библиотеки-«заглушки» и заголовочные файлы

Говорить о LSB-совместимости можно применительно не только к дистрибутивам Linux, но и к отдельным программ. Дистрибутив соответствует стандарту LSB, если он содержит все объекты, описываемые LSB, с характеристиками, указанными в стандарте. Программа же является LSB-совместимой, если она в своей работе использует только те объекты, которые описаны в LSB. Таким образом, любая LSB-совместимая программа будет работать в любом дистрибутиве, отвечающим требованиям этого стандарта (поскольку LSB стандартизует интерфейсы на бинарном уровне, а не на уровне исходного кода, то для запуска программы на разных дистрибутивах не требуется перекомпиляция).

Однако, даже если программа пишется в дистрибутиве, отвечающим требованиям LSB, нельзя быть уверенным, что она будет LSB-совместимой – ведь соответствие дистрибутива стандарту не означает, что в нем нет каких-то дополнительных интерфейсов или команд, которые могут оказаться задействованы в разрабатываемой программе.

Для проверки того факта, что программа является LSB-совместимой, рекомендуется производить тестовую сборку программы в так называемом *LSB-совместимом окружении.* Это окружение включает в себя библиотеки-«заглушки» и заголовочные файлы, для генерации которых предоставляются специальные скрипты.

Скрипты создают исходный код для библиотек, описанных в LSB, включая туда все определенные в стандарте LSB интерфейсы с соответствующими сигнатурами. Однако генерируемый исходный код всех интерфейсов-функций

не содержит ничего полезного – если функция не возвращает никаких значений, то тело функции просто пусто, в противном случае генерируется конструкция, возвращающая какое-нибудь значение требуемого типа. Таким образом, весь этот исходный код может быть скомпилирован, в результате чего и будут получены библиотеки-«заглушки» – эти библиотеки формально содержат все интерфейсы из стандарта LSB и не содержат никаких других интерфейсов, однако функции, содержащиеся в них, ничего не делают.

Помимо библиотек-«заглушек» скрипты создают заголовочные файлы, содержащие описания тех и только тех объектов (типов, констант, функций), которые входят в LSB. При этом свойства этих объектов (размеры типов, значения констант, сигнатуры функций) полностью соответствуют стандарту.

Таким образом, если программу удается скомпилировать в таком окружении (т.е. с использованием только библиотек-«заглушек» и сгенерированных заголовочных файлов), то можно гарантировать, что она использует только объекты, описанные в LSB.

Однако проводить функциональное тестирование программы все-таки необходимо с использованием реальных библиотек. Поэтому для получения уверенности в том, что программа будет выполнять поставленные перед ней задачи на любой LSB-совместимой платформе, функциональное тестирование программы необходимо проводить на дистрибутиве, соответствующем стандарту LSB.

Созданию LSB-совместимых приложений с использованием LSB-совместимого окружения посвящена отдельная книга [3], выпущенная разработчиками стандарта и доступная в электронном виде на сайте Free Standards Group.

4.2. Элементарные тесты

Одной из главных задач стандартизации является проведение сертификационного тестирования на соответствие реализации стандарту. LSB, как и большинство промышленных стандартов, имеет большой объем, и написать тесты для проведения сертификации вручную не представляется возможным. Необходима автоматизация процесса создания сертификационного тестового набора.

Наличие базы данных стандарта позволяет сгенерировать на ее основе набор элементарных тестов, определяющих, присутствуют ли в системе все объекты, описанные в стандарте LSB, и соответствуют ли их характеристики (значения констант, сигнатуры функций и т.п.) тем, которые указаны в стандарте. В частности, генерируются следующие наборы тестов:

- cmdchk – проверка того, что в системе присутствуют все описанные в LSB команды, и соответствующие исполняемые файлы находятся в директориях, определенных стандартом (хотя для многих команд имена директорий не указаны);

- devchk – проверка наличия в системе всех заголовочных файлов, описанных в LSB, и наличия в этих файлах описаний всех требуемых стандартом констант и типов (функции данный тест не проверяет). Для констант проверяется, верно ли определены их значений, для типов, определяемых в заголовочных файлах, вычисляется размер и сравнивается с тем, который должен быть согласно LSB;
- libchk – проверка наличия в системе всех библиотек, описанных в LSB, а также проверка содержимого этих библиотек на соответствие стандарту (тестируется наличие в библиотеке всех необходимых интерфейсов с сигнатурами, определенными в стандарте).

Перечисленные выше тесты нацелены на проверку соответствия дистрибутивов Linux стандарту LSB. Кроме этого, существуют скрипты для генерации тестовых наборов, нацеленных на проверку соответствия стандарту отдельного приложения. К таким тестовым наборам относятся:

- appchk – статический анализ файла на предмет использования им библиотек и интерфейсов, не входящих в LSB;
- dynchk – проверка корректности использования приложением интерфейсов, определенных в LSB (разработка этих тестов еще не завершена);
- elfchk – проверка корректности заданного elf-файла (используется в тестах appchk);
- rpmchk – проверка корректности заданного rpm-файла (используется в тестах appchk).

Подробнее о перечисленных выше тестовых наборах и использовании базы данных стандарта при их генерации можно узнать в [4].

Несмотря на то, что перечисленные выше тесты не осуществляют никаких сложных проверок, возможность их автоматической генерации существенно упрощает процесс проверки соответствия дистрибутивов и приложений требованиям LSB. Например, если бы разработчикам тестов было доступно только текстовое описание заголовочных файлов со всем содержимым (константами, типами и функциями), то даже для элементарной проверки, выполняемой тестовым набором devchk, им пришлось бы вручную извлекать из текста всю необходимую информацию. А этот процесс может занять довольно продолжительное время (стандарт LSB описывает более 6000 различных констант и более 8000 типов для 411 заголовочных файлов). Кроме того, при таких объемах информации неизбежны ошибки, связанные с человеческим фактором.

Более сложные тесты на основе базы данных стандарта создать нельзя, поскольку она содержит далеко не всю информацию, присутствующую в тексте LSB. В частности, нельзя создать тестовый набор для проверки соответствия стандарту поведения функций, а именно такие тесты наиболее

важны для сертификационного тестирования. Однако описание поведения функций на данный момент имеется только в виде текста, написанного человеком, и для автоматического создания тестов необходимо проводить формализацию этого текста (пример подхода к формализации приведен в [5], там же имеется краткий обзор других существующих подходов).

О планах по добавлению в базу данных стандарта LSB более детальной информации об интерфейсах и о других направлениях развития инфраструктуры LSB можно узнать в следующем разделе.

5. Планы по развитию инфраструктуры LSB

В настоящее время в ИСП РАН в рамках совместного проекта с Free Standards Group проводится разработка тестовой инфраструктуры, предназначенной для усовершенствования процесса сертификации на соответствие стандарту LSB. Помимо собственно процесса тестирования, проект затрагивает и многие смежные области, в том числе базу данных стандарта и работающие с ней инструменты.

5.1. Атомарные требования и тестовые наборы

В настоящее время достаточно сложно получить информацию о существующих тестовых наборах для проверки соответствия стандарту LSB и о покрытии, которое они обеспечивают. Такую информацию изначально планировалось хранить в базе данных стандарта – для этого в таблице Interface есть поле Itested, показывающее, тестируется ли интерфейс тестовыми наборами, используемыми при сертификационном тестировании. Кроме того, есть таблица TestSuite, содержащая информацию о доступных тестовых наборах, а также таблицы TestInt и TestCmd, показывающие, какие интерфейсы и команды покрываются тестовыми наборами.

Однако сейчас нет никакого способа заполнять эти таблицы автоматически – для их заполнения необходимо проводить анализ отчетов о тестировании либо непосредственно тестовых наборов и определять, на тестирование каких интерфейсов и команд нацелен каждый тест. Последний раз такая работа проводилась в середине 2002 года. С тех пор информация, касающаяся тестов и покрытия, в базе данных не обновлялась, и на текущий момент практически полностью устарела. Так, из 32721 интерфейса, включенных в LSB версии 3.1 (учитывая все архитектуры), только 184 помечены как протестированные (Itested=’Yes’) и еще для 1700 содержатся записи в таблице TestInt (что означает, что для 1700 интерфейсов хотя бы один из тестовых наборов содержит какой-то тест). С командами ситуация несколько лучше – из 141 команды, включенной в стандарт, для 90 существуют тесты.

Кроме того, наличие всего одного поля Itested и записей в таблице TestInt для каждого интерфейса представляется недостаточным. Ведь в случае, если интерфейс является функцией, для него может существовать целый набор

требований, и тестовые наборы могут проверять только часть из них. При существующей структуре определить полноту тестирования каждого конкретного интерфейса невозможно.

Для обеспечения более полной информации о покрытии тестами интерфейсов планируется для каждого интерфейса хранить список атомарных требований, и осуществлять привязку тестов не к самим интерфейсам, а к этим атомарным требованиям. Рассматривается возможность хранить атомарные требования не непосредственно в базе данных, а в отдельных xml-файлах; база данных при этом будет содержать только ссылки на соответствующие xml-файлы. Такая организация представляется более предпочтительной, чем размещение всех требований в БД. Последнее увеличит размер БД в несколько раз, в то время для как большинства инструментов, работающих с БД (генераторы текста стандарта, генераторы библиотек-«заглушек» и т.п.) эти данные не нужны.

Атомарные требования необходимо выделять из спецификации интерфейса. В LSB такие спецификации либо создаются вручную и хранятся в отдельных файлах формата sgml, либо вместо детального описания содержится ссылка на другой стандарт, где такое описание присутствует. Таким образом, для выделения атомарных требований необходимо анализировать описания интерфейсов, написанные человеком. Естественно, непосредственно анализ описаний автоматизировать трудно, однако планируется предоставить инструмент, который позволит упростить процесс занесения атомарных требований в базу данных. Задачу существенно усложняет наличие множества ссылок на другие документы – для выделения атомарных требований для всех интерфейсов необходимо иметь тексты всех этих стандартов (на данный момент в описаниях интерфейсов содержатся ссылки на 39 различных документов).

5.2. Информация о дистрибутивах

Особенностью стандарта LSB является ориентированность на основные дистрибутивы Linux – в стандарт включаются только те объекты, которые присутствуют в последних версиях большинства основных дистрибутивов. (Формально списка «основных дистрибутивов» нет – согласно официальной доктрине, в LSB вносится все то, что «является устоявшейся практикой в мире Linux»; однако в реальной жизни невозможно исследовать все дистрибутивы, и основное внимание уделяется RedHat, SuSe, Mandriva, Asianux, Debian и Ubuntu).

Одним из важных аспектов разработки стандарта является отслеживание списка интерфейсов, команд и пр., используемых в различных версиях дистрибутивов. Если какой-то интерфейс либо команда присутствуют почти везде и востребованы разработчиками приложений, но еще не включены в LSB, то они объявляются кандидатами на включение в следующую версию стандарта. Ручная обработка списка интерфейсов и определение кандидатов на

включение в стандарт – трудоемкая работа, и в настоящее время назрела потребность в ее автоматизации.

Для упрощения работы с дистрибутивами планируется, прежде всего, добавить в базу данных таблицу, содержащую список дистрибутивов (включая различные версии одного и того же дистрибутива). Также планируется добавить таблицу с компонентами, содержащимися в этих дистрибутивах (glibc, zlib, Qt, Gtk и т.д.), и привязку записей из этих таблиц к объектам, описываемых стандартом – библиотекам, заголовочным файлам, интерфейсам и пр.

Информацию о составе дистрибутивов (ассортимент и версии входящих в них компонентов, какие интерфейсы/библиотеки/заголовочные файлы содержатся в компонентах и т.п.) планируется заполнять автоматически на основе анализа установленного дистрибутива. В перспективе для занесения сведений об очередной версии дистрибутива в базу данных разработчику необходимо будет лишь запустить утилиты сбора информации о дистрибутиве (естественно, для обеспечения полноты сведений утилиты необходимо запускать на дистрибутиве, установленном в полной комплектации).

При наличии информации об основных дистрибутивах и их составе можно будет автоматически определять объекты, являющиеся кандидатами на включение в стандарт, а также объекты, которые в ближайших версиях стандарта следует объявить устаревшими, а впоследствии удалить.

5.3. Поддержка версий LSB

В настоящее время база данных содержит информацию только для текущей версии стандарта. Чтобы посмотреть, что входило в какую-либо из предыдущих версий, необходимо взять из CVS-репозитория FSG соответствующую версию базы данных. Также можно сгенерировать и текст стандарта определенной версии, однако для этого необходимо, помимо нужной версии базы данных, взять соответствующие версии скриптов, генерирующих текст стандарта – ведь со временем изменения вносятся как в схему базы данных, так и в сами скрипты.

Аналогично, при необходимости сгенерировать объекты, описанные в разд. 4, необходимо взять нужные версии соответствующих скриптов.

В рамках совместного проекта ИСП РАН и FSG планируется добавить в базу данных поддержку хранения информации для различных версий стандарта LSB, а также доработать все скрипты, чтобы они могли генерировать файлы, соответствующие определенной версии стандарта.

Планируется воссоздать историю изменений стандарта LSB, начиная с версии 3.0. Что касается более ранних версий, то вопрос о предоставлении исторической информации для них остается открытым – в первые годы своего существования стандарт и построенная вокруг него инфраструктура переживали революционные изменения (переход к поддержке нескольких архитектур,

разбиение на модули), и хранить историю в полном объеме было вряд ли целесообразно, а ее воссоздание может потребовать значительных усилий.

5.4. Чистка базы данных стандарта LSB

В процессе развития стандарта схема базы данных стандарта изменялась не один раз. К сожалению, не все изменения схемы сопровождались соответствующими корректными преобразованиями данных и инструментов, работающих с базой данных. В результате во многих таблицах появились поля, устаревшие с точки зрения структуры базы данных, но по-прежнему используемые частью скриптов. При этом были примеры несогласованности значений устаревших полей со значениями структур, пришедших им на смену. В результате скрипты, использовавшие устаревшие поля, генерировали некорректные данные.

Проиллюстрируем сказанное на примере связи между интерфейсами и архитектурами.

Первые версии стандарта LSB описывали бинарные интерфейсы только для одной архитектуры IA32, и схема базы данных не позволяла хранить информацию о различных архитектурах. Однако уже в LSB версии 1.2 появилась поддержка второй архитектуры (PPC32), и соответствующие изменения претерпела схема БД.

Изначально в таблице Interface, хранящей данные об интерфейсах, было введено специальное поле Iarch, ссылающееся на запись в таблице Architecture. Однако связь между интерфейсами и архитектурами в действительности является связью «многие ко многим»: один интерфейс может быть определен для нескольких архитектур. Для таких интерфейсов предлагалось вводить несколько записей в таблице Interface, по одному для каждой архитектуры (соответственно, эти записи различались значением поля Iarch и, возможно, значениями некоторых архитектурно-зависимых свойств).

Практика быстро показала, что решение заводить несколько записей для каждого интерфейса было не очень удачным. При таком подходе во всех таблицах, ссылающихся на интерфейс, приходилось также заводить несколько записей, соответствующих различным записям в таблице интерфейсов. Поскольку большая часть информации об интерфейсе не зависит от архитектуры, во многих таблицах данные просто дублировались, что приводило к неоправданному росту объема данных.

В результате для реализации связей «многие ко многим» между таблицей интерфейсов и таблицей архитектур была создана отдельная таблица ArchInt. После этого все связи между интерфейсами и архитектурами вносились в таблицу ArchInt, однако поле Iarch убрано не было – к этому моменту многие инструменты, работающие с базой данных, использовали это поле, а для их быстрой переработки не хватало ресурсов. Кроме того, в таблице Interface уже было более 5000 записей, созданных исключительно для реализации связи «многие ко многим» между интерфейсами и архитектурами. Они не были

удалены, а в таблицу ArchInt не была занесена соответствующая им информация.

Такой «частичный» переход к использованию таблицы ArchInt привел к тому, что часть информации о связи «многие ко многим» хранилась в этой таблице, а часть задавалась дублированием записей с изменением поля Iarch. Часть инструментов были переписаны с учетом появления новой таблицы; при этом они учитывали как данные из ArchInt, так и поле Iarch (в основном это относится к скриптам, генерирующим текст стандарта – их корректность имеет наивысший приоритет). Вновь написанные инструменты опирались только на таблицу ArchInt, в то время как часть старых инструментов так и не была переписана и использовала только Iarch.

Избавление от проблемы со связями между архитектурами и интерфейсами было произведено в ходе чистки схемы и данных базы данных стандарта LSB на первом этапе разработки новой тестовой инфраструктуры LSB, в четвертом квартале 2006 года. В частности, было удалено поле Iarch, все необходимые данные перенесены в таблицу ArchInt и произведены соответствующие изменения скриптов. Помимо этого, были произведены следующие действия по устранению нестыковок и неоднородностей:

- Унификация названий полей и типов перечислений, имеющих одинаковую семантику в различных таблицах. Например, статус команд в стандарте хранился в поле Command.Cstatus, имевшем тип `enum ('Included','Excluded','Builtin','Unknown')`, а статус констант в стандарте хранился в поле Constant.Cstd, имевшего тип `enum ('Yes','No','SrcOnly')`. Теперь все поля, обозначающие статус какого-либо объекта в стандарте, имеют имена, оканчивающиеся на «stdstatus», и для всех таких полей используется одно и то же перечисление.
- Удаление устаревших либо не используемых полей и таблиц.
- Исправление в таблице Type некорректных значений некоторых полей, полученных при автоматическом заполнении базы. Например, для некоторых типов-перечислений объявление типа не было корректно обработано, и тип не был определен в таблице как перечисление, а ключевое слово «enum» просто присоединялось к имени типа.
- Согласование типов полей-ссылок с типами полей, на которые они ссылаются.
- Удаление дублирующихся индексов и добавление индексов, ускоряющих часто используемые запросы к базе данных (на основе анализа скриптов, работающих с БД). Появление дублирующихся индексов было вызвано особенностями архитектуры СУБД MySQL, которая при создании индекса вида (a,b,c) автоматически создает индексы (a,b) и (a).

6. Заключение

Free Standards Group смогла успешно перенести идеологию open source на процесс разработки стандарта LSB, создав инфраструктуру, предоставляющую каждому желающему возможность самостоятельно генерировать текст стандарта и многие связанные с ним объекты. Наличие базы данных стандарта позволяет предоставлять пользователям более удобные способы изучения сущностей, описанных в LSB, чем простое чтение текста.

Существующий подход уже доказал свою состоятельность – с его применением создавались все версии стандарта, начиная с 1.0. Естественно, за это время как схема базы данных стандарта LSB, так и использующие эту базу данных скрипты претерпели существенные изменения, однако основные идеи, заложенные в основу используемого подхода, остаются неизменными.

Тем не менее, в своем текущем состоянии инфраструктура LSB удовлетворяет не все потребности, возникающие у пользователей, и есть много предложений по ее дальнейшему развитию. Прежде всего, необходимо предоставлять пользователю информацию о дистрибутивах и о наличии в них тех или иных сущностей, описанных в LSB. Также важно предоставить более простые способы получения данных о предыдущих версиях стандарта. Для создания сертификационных наборов, обеспечивающих хорошее покрытие, представляется важным иметь в базе данных более детальную информацию об интерфейсах.

Все эти задачи планируется реализовать в рамках совместного проекта ИСП РАН и Free Standards Group по развитию инфраструктуры LSB.

Литература

[1] Linux Standard Base wiki. https://www.freestandards.org/en/LSB

[2] Free Standards Group wiki. http://www.freestandards.org/en/Main_Page

[3] Linux Standard Base Team. Building Applications with the Linux Standard Base. 2005. http://lsbbook.gforge.freestandards.org/lsbbook.html

[4] Stuart Anderson, Matt Elder. Run-time Testing of LSB Applications. Proceedings of the Linux Symposium, July 2004. http://www.linuxsymposium.org/proceedings/reprints/Reprint-Anderson-OLS2004.pdf

[5] А. И. Гриневич, В. В. Кулямин, Д. А. Марковцев, А. К. Петренко, В. В. Рубанов, А. В. Хорошилов. Использование формальных методов для обеспечения соблюдения программных стандартов. Труды Института Системного Программирования. Т.10, с. 51-68, 2006.