Открытое акционерное общество
«Научно-производственное объединение
Русские Базовые Информационные Технологии»

РУКОВОДЯЩИЕ УКАЗАНИЯ

по конструированию прикладного программного обеспечения
для операционной системы общего назначения «Astra Linux Common Edition»

Листов 82

Москва
2018

АННОТАЦИЯ

Настоящие руководящие указания по конструированию предназначены для разработчиков прикладных программ, автоматизированных систем управления и аппаратно-программных средств на базе платформы операционной системы общего назначения «Astra Linux Common Edition» Орёл версии 2.12 (далее по тексту — ОС ОН) и будут полезны тем, кто рассматривает вопрос перевода (адаптации) своих разработок с других платформ общего назначения (Windows, Linux и Unix-подобных ОС). Дается краткий обзор состава ОС ОН и принципов ее версионности, устройства репозитория пакетов, основ разработки программ для многофункционального оконного менеджера FLY, использования основных средств разработки.

СОДЕРЖАНИЕ

1. СОСТАВ ДИСТРИБУТИВА ОС ОН

За основу построения дистрибутива ОС ОН взят открытый репозиторий пакетов для ОС Debian Linux и применяемая в нем система пакетирования `deb`.

Архитектурно ОС Astra Linux состоит из базовой программной платформы GNU/Linux на основе ядра семейства 4.15 и интегрированных программных средств (ИПС). При этом в состав ОС входит как общее программное обеспечение (ОПО) с открытыми исходными текстами, так и собственные разработки.

В состав базовой программной платформы ОС ОН входят:

– Базовые библиотеки и утилиты;

– Встроенные средства защиты информации (СЗИ), обеспечивающие дискреционное разграничение доступа[1)] ;

– Сетевые службы (DHCP, DNS, VPN, SSH, NTP, Samba, Apache2, squid);

– Графическая система;

– Средства работы с периферийным оборудованием;

– Средства разработки и отладки (gcc, quilt, subversion, gcc, python, java)

В состав ИПС ОС ОН входят:

– Средства для создания WEB-сервисов и для работы с ними (apache2, squid, firefox, chromium);

– Офисные средства (LibreOffice) и электронная почта (ThunderBird, Evolution);

– Многофункциональный оптимизированный рабочий стол (Fly);

– Системы управления базами данных (СУБД) (PostgreSQL, MySQL);

– Средства виртуализации (KVM, Oracle VirtualBox, FireJail, LXC);

– Средства кластеризации (Pacemaker, corosync , LVS);

– Средства сертификации, идентификации, защиты информации (DogTag, Kerberos, XCA, библиотеки алгоритмов защиты информации ГОСТ);

– Доменные службы (ALD, FreeIPA, Samba);

– Средства работы с мультимедиа и построения мультимедийных решений (Audacity, Blender);

– Система верстки текстов (LaTex);

[1)] Существует защищенный вариант ОС Astra Linux — ОС специального назначения (ОС СН) «Astra Linux Special Edition» Смоленск, предназначенная для построения автоматизированных систем в защищенном исполнении. В ОС СН базовая программная платформа ОС ОН дополнена средствами защиты информации (СЗИ), реализующими мандатное разграничение доступа и контроль целостности. Унификация платформ ОС ОН и ОС СН обеспечивает простую миграцию прикладных программ из общей среды в защищенную. Более подробная информация об особенностях ОС СН доступна в документации ОС СН и на WEB-сайте astralinux.ru.

– Графические инструменты администратора (fly-admin-...);

– Мобильные приложения (SMS, телефон, галерея);

Таким образом, в состав ОС ОН входит полный набор компонент Linux-систем, позволяющий решать любые задачи построения современных информационных систем на всех уровнях от серверного кластера до рабочего места пользователя. Краткий перечень основных пакетов, входящих в состав ОС ОН Орёл с указанием их функционального назначения приведен в 8. Более подробная информация о составе и назначении компонент ОС ОН в справочной системе MAN и на сайте wiki.astralinux.ru.

С точки зрения разработчика наибольший интерес представляют:

– Базовые библиотеки и утилиты;

– Графическая система и многофункциональный оптимизированный рабочий стол;

– Средства разработки и отладки;

– СУБД;

– Отдельные компоненты, входящие в общее программное обеспечение

1.1. Базовые библиотеки и утилиты

Разработку ПО с использованием базовой системы ОС ОН рекомендуется проводить с использованием следующих базовых компонент, библиотек и средств разработки, поставляемых в составе ОС ОН:

1) ядро версии 14.5;
2) `glibc` версии 2.24;
3) `openldap` версии 2.4;
4) `openssl` версии 1.1.0;
5) `gawk` версии 4.14;
6) `bash` версии 4.4.

1.2. Графическая система и многофункциональный оптимизированный рабочий стол

В качестве системы графического интерфейса может использоваться как основной оконный менеджер Fly, так и широко распространенный оконный менеджер KDE4. И тот и другой в качестве базовой графической библиотеки использует Qt 5.x.

Разработку ПО с использованием графической системы ОС ОН рекомендуется проводить с использованием следующих графических компонент, библиотек, поставляемых в составе базовой системы ОС ОН:

1) `qt` версий 5.11;
2) `Xorg` версии 7.7;
3) графический интерфейс пользователя `Fly 2.0`.

1.3. Средства разработки и отладки

Для компиляции и отладки прикладных программ основными рекомендуемыми языками программирования для разработки являются C, C++, Perl, Python, Shell.

Разработку прикладного ПО для ОС ОН рекомендуется проводить с использованием следующих средств разработки, поставляемых в составе базовой системы ОС ОН:

1) `eclipse` версии 3.8;
2) `Qt Creator` версии 4.6;
3) `subversion` версии 1.10;
4) `php` версии 7.0;
5) `python` версии 2.7;
6) `perl` версии 5.24;
7) `build-esential` версии 12.3;
8) `quilt` версии 0.65;

Окружение разработки, к которому привыкли разработчики Linux-систем, доступно и в ОС ОН. Фундаментальную инфраструктуру окружения C/C++ составляют инструменты компиляции кода C/C++: библиотеки C, компиляторы, средства сборки и отладчики. Ниже приведен краткий список рекомендуемых инструментов разработки, существующих в ОС ОН:

C/C++ библиотеки glibc — соответствующие ANSI библиотеки языков C и C++. ОС ОН поставляется с `glibc` версии 2.7 и выше.

C/C++ компилятор GCC — открытый компилятор, также соответствующий стандарту ANSI. ОС ОН поставляется с `GCC` версии 6.3 и выше.

Средства сборки binutils, make, Eclipse — `binutils` — программы компоновки («линковки») и манипулирования объектными файлами;
`Make` — средство для автоматизации процесса компиляции;
`Eclipse` — открытая интегрированная платформа разработчика программ. Позволяет работать со сложными проектами, написанными на объектно-ориентированных языках программирования.

Отладчики gdb, ddd — `gdb` — стандартный отладчик GNU;
`ddd` — улучшенный графический отладчик на

базе gdb.

1.4. СУБД

В состав ОС ОН входят реляционные СУБД:

– Postgresql версии 9.6;

– Mysql версии 5;

1.5. Общее программное обеспечение (ОПО)

В состав ОПО входят программы, которые чаще всего бывают востребованы при построении различных АС. Это средства работы в сетях, набор офисных программ, система верстки текстов, а также средства работы с мультимедиа и графикой.

Из состава ОПО для разработчиков можно выделить следующие компоненты:

1) для разработки документации для ПО:
   – LaTeX (Tex, Lyx);
   – LibreOffice версии 6.0.5;
   – Система контроля версий Subversion (SVN) версии 1.10.x;
2) для разработки ПО для работы в сетях:
   – Apache2 версии 2.4;
   – Thunderbird версии 52.8;
   – Firefox версии 62.0.

2. ПРИНЦИПЫ ОБНОВЛЕНИЯ ВЕРСИЙ ОС ОН

2.1. Общие принципы

ОС ОН — это единая базовая программная платформа для широкого спектра применений: настольная система для дома и офиса, автоматизированное рабочее место предприятия, файловый сервер, интранет-сервер, web-сервер и т. д. Также, она может использоваться как среда для разработки различных прикладных программ и корпоративных приложений.

ОС ОН — это платформа, которую могут поддерживать как сертифицированные специалисты интеграторы, так и независимые разработчики и системные администраторы.

Срок поддержки в рамках одного релиза составляет от 12 до 24 месяцев, что дает независимым разработчикам достаточно времени на адаптацию и распространение своих продуктов. Исправления ошибок и уязвимостей выпускаются по мере надобности, чтобы клиенты всегда имели максимально надежные, стабильные и защищенные системы. Обновления распространяются через Интернет-портал предприятия, что облегчает развертывание обновленного ПО на большом количестве систем. На этом же портале всегда можно найти актуальную версию электронной документации. С помощью приведенных в ней рекомендаций разработчикам будет гораздо легче и быстрее осуществлять перенос своего ПО как между релизами ОС ОН, так и при переходе с других программных платформ.

Дополнительную информацию по особенностям и приемам работы с ОС ОН можно получить на Wiki-портале wiki.astralinux.ru

Обновления в рамках одного релиза проводятся в части устранения ошибок и уязвимостей, выявленных в ходе эксплуатации ОС ОН, и не предполагают изменений версий ядра ОС, базовой библиотеки `glibc` и основного компилятора `gcc`.

Бинарная совместимость разрабатываемого ПО в рамках одного релиза ОС ОН сохраняется на протяжении всего времени производства данного релиза.

2.2. Номер версии дистрибутива

Номер версии дистрибутива можно узнать, набрав команду

```
cat /etc/astra_version
```

Номер версии имеет вид:

```
EDITION V.U.Y (NAME)
```

где `EDITION` — редакция релиза, для ОС ОН всегда «CE»;

`V` — номер версии релиза;

`U` — номер обновления в пределах данного релиза;

`Y` — служебный номер внутреннего обновления. В законченной версии служебный номер должен отсутствовать. Его наличие говорит о том, что данный релиз находится в

стадии разработки;

NAME — название релиза. В качестве названий релизов используются названия городов воинской славы России.

Пример

```
CE 2.12 (orel)
```

3. ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ЛОКАЛЬНОГО РЕПОЗИТОРИЯ

Локальный репозиторий создается при помощи дистрибутивного диска ОС ОН. В первую очередь необходимо выбрать (создать) каталог для расположения подкаталогов репозитория `dists` и `pool` (далее — репозиторий). Для примера будет использован каталог `/opt/repo`.Подкаталог `dists` содержит фалы `Packages` с описаниями пакетов ОС ОН, подкаталог `pool` — сами пакеты.

После определения места расположения репозитрия необходимо скопировать `dists` и `pool` с дистрибутивного диска ОС ОН в каталог `/opt/repo`, таким образом полный путь до каталога с пакетами будет выглядеть так — `/opt/repo/pool`, а полный путь до каталога с файлами описаниями так — `/opt/repo/dists`.

Репозиторий может быть доступен локально или из сети по протоколам FTP и HTTP. Для доступа к репозиторию по FTP или HTTP необходимо настроить сервер FTP или HTTP соответственно. Для доступа к такому репозиторию пользователям необходимо внести следующие строчки в файл `/etc/apt/sources.list`: для доступа по ftp:

```
deb ftp://<ip или имя ftp сервера>/<путь до репозитория> \
<название релиза дистрибутива> main contrib non-free
```

для доступа по http:

```
deb http://<ip или имя http сервера>/<путь до репозитория> \
<название релиза дистрибутива> main contrib non-free
```

для локального доступа к репозиторию

```
deb file:/<полный путь до каталога репозитория> \
<название релиза дистрибутива> main contrib non-free
```

Пример строки из файла `/etc/apt/sources.list`:

```
deb ftp://repo.srv.rbt orel main contrib non-free
```

В данном примере предполагается, что при подключении пользователя по FTP по адресу `repo.srv.rbt` корневым каталогом доступа является каталог, содержащий подкаталоги `dists` и `pool`.

4. РУКОВОДСТВО ПО СБОРКЕ И РАЗРАБОТКЕ ПАКЕТОВ

4.1. Обзор задач

Перед разработчиками пакетов ПО для ОС ОН стоят сразу несколько задач, которые необходимо решить. Прежде всего, это способ хранения разрабатываемых или изменяемых пакетов. Так, кроме разрабатываемых пакетов, разработчики могут модифицировать существующие opensource-проекты, а также поставлять в неизменном виде полностью заимствованные opensource-пакеты, которых нет в составе ОС ОН. Пакеты должны храниться так, чтобы процесс разработки был максимально удобен, результаты работы были надежно сохранены, и чтобы можно было восстановить историю разработки.

4.2. Типы пакетов

Так как источники получения и исходное состояние пакетов ПО для ОС ОН могут быть разными, то разными будут и те действия, которые необходимо произвести над этими пакетами. С точки зрения разработчика ПО, пакеты для ОС ОН можно разделить на четыре типа:

– неизменяемые — пакеты, которые взяты из Debian или другого дистрибутива, использующего систему пакетирования `dpkg`, в неизменном виде;

– наследуемые — пакеты, которые взяты из Debian или другого дистрибутива, использующего систему пакетирования `dpkg`, но в которые необходимо внести собственные изменения;

– собственные — пакеты, разработчик которых является одновременно и сопровождающим пакета;

– дебианизируемые — исходные коды такого пакета берутся из первичного источника в виде авторского архива (upstream source) , и изначально не дебианизированы. Их необходимо дебианизировать и, возможно, внести собственные изменения для включения в состав дистрибутива.

Исходя из специфики каждого из перечисленных типов, хранение и работа с пакетами этих типов будет несколько различаться.

4.3. Форматы хранения пакетов с исходными кодами

Практически во всех дистрибутивах принято хранить изменения, внесенные в исходный код, в виде патчей (patch). Так и система пакетирования `dpkg` использует патчи для хранения пакетов с исходными кодами. Созданием и работой с пакетами исходных кодов занимается утилита `dpkg-source`.

Существует несколько форматов хранения пакетов с исходными кодами. По умолчанию используется формат «1.0». Этот формат подразумевает, что пакет исходных кодов

представляет из себя набор из трех файлов:

<название_пакета>_<версия>.orig.tar.gz

<название_пакета>_<версия>-<релиз>.diff.gz

<название_пакета>_<версия>-<релиз>.dsc

Например, пакет исходных кодов для текстового редактора vim будет выглядеть так:

vim_7.1.314.orig.tar.gz

vim_7.1.314-3.diff.gz

vim_7.1.314-3.dsc

где vim — название пакета, 7.1.314 — авторская версия (версия upstream source), 3 — номер релиза. Номер релиза задается разработчиками дистрибутива и, по сути, означает номер исправления. Файл с именем *.orig.tar.gz содержит первичные авторские исходные коды (upstream source) , и не содержит каких-либо изменений, специфичных для дистрибутива. Файл *.diff.gz содержит все изменения, внесенные разработчиками дистрибутива, относительно первичного дерева исходных кодов. Изменения хранятся в формате diff. В отличие от файла *.diff.gz, файл *.orig.tar.gz не содержит в составе своего имени номер релиза, так как этот файл остается неизменным в течение всего процесса адаптации пакета под конкретный дистрибутив. Номер релиза определяет файл *.diff.gz, содержащий все изменения. Файл *.dsc содержит описания пакетов исходных кодов.

Недостатком такой схемы является то, что все изменения дерева исходных кодов хранятся в одном единственном файле , и невозможно отделить изменения, направленные на решение одной задачи, от изменений, направленных на решение совсем другой задачи. Также, если источником первичных исходных кодов является не один архивный файл, а несколько, то при дебианизации такого пакета все архивы с первичными исходными кодами будут перепакованы в единый архив, что тоже не очень удобно.

Все сказанное выше относится к пакетам, не специфичным для дистрибутива ОС ОН. Пакеты, специально разработанные для дистрибутива, или собственные пакеты (native) дебианизируются на этапе разработки и не содержат дополнительных изменений, так как разработчик первичной версии пакета обычно является одновременно и его сопровождающим. Если необходимо внести изменения в исходные коды, то разработчик делает это прямо в первичных исходных кодах проекта, увеличивая при этом основную версию пакета. Необходимость в номере релиза отпадает. Поэтому пакеты исходных кодов для собственных проектов не включают в себя файлы *.orig.tar.gz и *.diff.gz. Вместо этого используется файл вида: <название_пакета>_<версия>.tar.gz.

Если бы текстовый редактор vim был собственным для какого-либо дистрибутива, то пакет исходных кодов выглядел бы так:

vim_7.1.314.tar.gz

vim_7.1.314.dsc

Кроме формата «1.0» для хранения пакетов с исходными кодами, используется формат «3.0» («3.0 (native)» и «3.0 (quilt)»). Для понимания того, как устроен формат «3.0 (quilt)», необходимо познакомиться с системой управления патчами quilt.

По умолчанию утилита dpkg-source использует формат «1.0». Чтобы явно задать формат пакета с исходным кодом можно использовать три способа:

– поле Format в файле debian/control;

– опция командной строки --format;

– содержимое файла debian/source/format.

Способы задания формата пакета исходных кодов перечислены в порядке приоритета. То есть сначала утилита dpkg-source попытается использовать первый способ, затем второй, и только затем третий. Для разработки пакетов ПО для ОС ОН рекомендуется использовать файл debian/source/format, так как этот способ легче всего автоматизировать. Автоматизация может применяться для преобразования некоторых пакетов формата «1.0» в формат «3.0». Для более сложных пакетов автоматическое преобразование невозможно.

4.4. Система управления патчами quilt

Проект quilt — это система управления множественными патчами. Предположим, существует проект, в который надо внести целый ряд функционально независимых изменений. Каждое такое изменение можно оформить в виде отдельного патча. Таким образом, мы получим целый набор патчей, которые, в общем случае, должны накладываться не в произвольном порядке, а в определенной последовательности.

В корневом каталоге изменяемого проекта должна существовать директория patches/, в которой находятся используемые патчи и файл с именем series. В файле series перечислены патчи, которые необходимо применить.

С помощью утилиты quilt разработчик может накладывать патчи на рабочее дерево проекта, снимать примененные ранее патчи, создавать новые , и выполнять ряд других вспомогательных действий. Так, например, с помощью команды quilt push -a можно применить сразу все патчи, перечисленные в файле series, а с помощью команды quilt pop -a снять все ранее примененные патчи , и получить чистое дерево исходных кодов.

В процессе работы система quilt использует каталог .pc/ для хранения своих временных файлов. Каталоги .pc/ и patches/ могут быть символическими ссылками, а также иметь другие имена, но тогда другие имена каталогов необходимо указать с помощью специальных переменных окружения, либо в конфигурационном файле /.quiltrc. Такая возможность важна для работы с дебианизированными пакетами в формате «3.0 (quilt)».

Система патчей образует стек. Первый файл в списке — дно стека, последний файл — вершина стека. Командами `quilt push` и `quilt pop` можно добавить патч на вершину стека и убрать верхний патч стека, соответственно. Список примененных патчей, то есть текущее состояние стека, хранится в файле `.pc/applied-patches`. Когда дерево исходных кодов чистое и не наложено ни одного патча, директория `.pc/` отсутствует.

С помощью команды `quilt refresh` можно обновить состояние текущего патча. Последний патч в стеке считается рабочим или текущим. Для того чтобы команда обновления отработала правильно, необходимо сообщить системе `quilt` о том, какие именно файлы проекта будут изменяться (`quilt add <имя_файла>`). Причем, сделать это надо строго до того, как будет изменено содержимое самих файлов, так как `quilt` должен сохранить во временной директории `.pc/` прежнее состояние файлов, чтобы потом было с чем сравнивать измененную версию. Такая система довольно неудобна, так как ответственность за добавление файлов в список изменяемых целиком лежит на разработчике. Слишком легко что-нибудь забыть. В системе `quilt` существует специальная команда `quilt edit <имя_файла>`, с помощью которой можно добавить файл в список изменяемых и сразу запустить текстовый редактор для внесения изменений в этот файл. Но и это не решает проблему полностью. Однако, если использовать систему `quilt` вместе с пакетами исходного кода в формате «3.0 (quilt)», проблема может быть решена.

Более подробное описание приведено в:

```
man 1 quilt
/usr/share/doc/quilt/quilt.pdf.gz
```

4.4.1. Пакеты исходного кода формата «3.0 (native)»

Формат «3.0 (native)» используется для создания собственных пакетов , и почти совпадает с форматом «1.0» для собственных пакетов. Отличие в том, что формат «3.0 (native)» поддерживает различные методы сжатия и по умолчанию игнорирует файлы и директории, относящиеся к системам контроля версий, а также множество временных файлов (см. опцию `-I` утилиты `dpkg-source`). При разработке собственных пакетов для ОС ОН рекомендуется использовать формат «3.0 (native)», так как особенности этого формата будут востребованы (при разработке используется система контроля версий, сборка и отладка, порождающая временные файлы, может происходить в дереве исходных кодов).

4.4.2. Пакеты исходного кода формата «3.0 (quilt)»

Формат «3.0 (quilt)» интегрирует преимущества системы управления патчами `quilt`. Пакет исходных кодов может содержать множественные патчи, которыми можно управлять с помощью `quilt`. Пакет исходных кодов в формате «3.0 (quilt)» состоит из следующего набора файлов:

```
<название_пакета>_<версия>.orig.tar.ext
```

<название_пакета>_<версия>-<релиз>.debian.tar.ext
<название_пакета>_<версия>-<релиз>.dsc
<название_пакета>_<версия>.orig-<компонент>.tar.ext

где ext может принимать значения gz, bz2, lzma в зависимости от применяемого алгоритма сжатия. Файл *.orig.tar.ext содержит первичные авторские исходные коды проекта (upstream source) и не содержит каких-либо изменений. Файл *.dsc содержит описание пакета с исходными кодами.

Если исходными кодами для пакета является не один архивный файл, а множество (например основной проект и отдельные дополнения к нему), то можно использовать дополнительные файлы *.orig-<компонент>.tar.ext для хранения исходных кодов отдельных компонентов, относящихся к проекту. Таким образом, если пакет строится на основе нескольких архивных файлов с исходным кодом, нет необходимости перепаковывать их все в один файл, как это делалось при использовании формата «1.0».

Наибольший интерес представляет собой файл *.debian.tar.ext. Он заменяет собой файл *.diff.gz для формата «1.0», но в отличие от него содержит не все внесенные изменения одним файлом *.diff, а архивированный каталог debian/, содержащий управляющие файлы, необходимые для дебианизации, а также множественные патчи, пригодные для использования системой quilt. Множественные патчи и соответствующий файл series хранятся в директории debian/patches/.

Рассмотрим возможную структуру пакета исходных кодов в формате «3.0 (quilt)» на примере. Если бы пакет исходных кодов для редактора vim имел формат «3.0 (quilt)», то он мог бы выглядеть так:

vim_7.1.314.orig.tar.bz2
vim_7.1.314-3.dsc
vim_7.1.314-3.debian.tar.bz2
vim_7.1.314.orig-myextension1.tar.bz2
vim_7.1.314.orig-myextension2.tar.bz2

4.4.3. Дерево исходных кодов

Дерево исходных кодов разворачивается следующим образом: сначала распаковывается основной архивный файл *.orig.tar.ext. Затем все дополнительные архивные файлы *.orig-<компонент>.tar.ext распаковываются в директории с именами <компонент>/. Если директории с такими именами уже существовали, то они будут перезаписаны. После этого распаковывается файл *.debian.tar.ext. Если директория с именем debian/ существовала ранее, то она будет перезаписана. Кроме директории /debian, архив *.debian.tar.ext может содержать двоичные файлы вне этой директории (см. опцию --include-binaries утилиты dpkg-source).

После распаковки архивных файлов к полученному дереву исходных кодов будут применены все патчи, перечисленные в файле debian/patches/series. Утилита dpkg-source всегда использует опцию -p1 при применении патчей (см. утилиту patch) и игнорирует опции, указанные в файле series (после имени патча в каждой строке файла series могут быть указаны дополнительные опции). Если какие-либо патчи были применены к дереву исходных кодов, то создается файл debian/patches/.dpkg-source-applied, в котором перечислены все примененные патчи.

4.4.4. Сборка пакета исходных кодов

Во временной директории разворачивается дерево исходных кодов почти также, как это делалось для основного дерева, в котором происходит разработка. Применяются патчи из файла debian/patches/series, кроме патча с именем debian/patches/debian-changes-<версия>. Полученная таким способом директория сравнивается с основным деревом исходных кодов и все различия сохраняются в файле debian/patches/debian-changes-<версия>. Изменения в двоичных файлах не могут быть представлены в формате «diff» и приведут к ошибке, если только разработчик не включит эти файлы в состав архива *.debian.tar.ext. Это можно сделать, указав нужные двоичные файлы в файле debian/source/include-binaries. Тоже касается и двоичных файлов, находящихся в директории debian/.

Во вновь создаваемый файл *.debian.tar.ext попадет обновленная директория debian/ и заданные двоичные файлы. Автоматически созданный diff-файл не содержит изменения в файлах и директориях, относящихся к системе контроля версий, не содержит изменения во временных файлах (см. опцию -i утилиты dpkg-source), а также игнорируется директория .pc/, используемая системой quilt.

Утилита dpkg-source ожидает, что во время сборки пакета к дереву исходных кодов уже применены все патчи. Если файл debian/patches/.dpkg-source-applied не найден, она попытается применить патчи перед сборкой пакета. Наличие директории .pc/ говорит о том, что какие–то патчи применены , и будет вызвана команда quilt unapplied, чтобы убедиться, что применены все патчи. Все эти проверки можно отключить с помощью опции командной строки --no-preparation.

Пакеты исходного кода должны выполнять требования, касающиеся обязательных полей в файле debian/control и проверять полученный пакет с помощью утилиты lintian. Недоработки, помеченные утилитой lintian как предупреждения «W» допускаются , ошибки «E» не допускаются и должны быть исправлены в обязательном порядке.

В поле Maintainer файла debian/control разработчик указывает свое имя и существующий адрес электронной почты.

4.4.5. Интеграция с quilt

По умолчанию система `quilt` ищет патчи и файл `series` в директории `patches/`, в то время как в развернутом дереве исходных кодов формата «3.0 (quilt)» эти файлы находятся в директории `debian/patches`. Для того чтобы в развернутом дереве исходных кодов заработала утилита `quilt`, необходимо переопределить переменную окружения `QUILT_PATCHES=debian/patches`. Эта переменная определяет путь относительно корня дерева исходных кодов, где `quilt` будет искать файл `series` и патчи. Переменную можно переопределить в файле конфигурации `/.quiltrc`.

Способность утилиты `dpkg-source` автоматически генерировать diff-файл устраняет недостаток системы `quilt`, где для создания diff-файла необходимо было предварительно указывать все файлы (`quilt add <файл>`), в которые планируется вносить изменения. Достаточно выполнить команду `debuild -S` (будет вызвана утилита `dpkg-source` с нужными параметрами), чтобы сгенерировать автоматический diff-файл и одновременно собрать пакет с исходными кодами.

Пока идет работа над текущим патчем, он будет обновляться автоматически и называться `debian/patches/debian-changes-<версия>`. Когда разработчик посчитает, что работа над патчем завершена , что и результат можно зафиксировать, лучше всего переименовать автоматически сгенерированный патч. С помощью команды:

```
quilt rename <новое_имя>
```

будет переименован верхний патч в стеке. Название зафиксированного патча должно быть информативным и отражать задачи, которые он выполняет. После того, как патч переименован, команда `debuild -S` будет генерировать уже новый автоматический патч.

4.5. Использование Subversion

При разработке пакетов ПО для ОС ОН рекомендуется использовать систему контроля версий Subversion.

В одном репозитории Subversion должен храниться один проект. В общем случае, репозиторий не имеет жесткой структуры, и содержимое его может быть произвольным. Однако для удобства ведения проектов рекомендуется следующая структура директорий:

```
trunk
branches
tags
```

Такая структура наилучшим образом отражает различные этапы разработки.

Директория `trunk/` содержит текущее дерево исходных кодов, в котором происходит основной процесс разработки. Здесь в проект добавляются новые возможности, может перерабатываться структура проекта и т. д.

Директория `branches/` содержит поддиректории `branches/<версия>`, в которых

хранятся версии проекта, подготавливаемые к стабилизации и выпуску. Здесь тоже может вестись разработка, но она ограничивается исправлением ошибок и могут находиться и другие произвольные директории, используемые для временного ветвления проекта.

Директория `tags/` содержит поддиректории `tags/<версия>-<релиз>`, которые являются статическими ссылками на некоторые состояния проекта. Эти состояния соответствуют версиям пакета, которые попали в репозиторий дистрибутива (не путать с репозиторием Subversion). В `tags/` разработка не ведется и все поддиректории служат исключительно для того, чтобы была возможность восстановить историю разработки.

Этапы разработки для каждого типа пакетов будут рассмотрены подробнее.

4.5.1. Собственные пакеты

Разработка исходного кода собственных пакетов и их сопровождение происходит на основе одного репозитория Subversion. Проект в виде развернутого дерева исходных кодов находится в директории `trunk/`, в которой ведется основной процесс разработки, добавляются новые возможности, вносятся изменения. Так как пакет собственный, директория `debian/` находится в том же дереве исходных кодов и пакет использует формат «3.0 (native)». Формат «3.0 (quilt)» для собственных проектов можно использовать только тогда, когда это действительно необходимо.

Для собственных пакетов должна использоваться трех-компонентная система нумерации версий, например `2.1.15`. Первый, наиболее весомый, компонент версии меняется только в случае серьезной переработки проекта, при появлении принципиально новых возможностей, внесении изменений, несовместимых с предыдущими версиями. Второй компонент версии меняется регулярно в процессе разработки при добавлении новых возможностей, оптимизациях, небольших переработках кода. Третий компонент отвечает за исправление ошибок в проекте.

Если разработчик планирует вносить в исходный код довольно серьезные изменения, то предварительно он может создать копию проекта в директории `branches/`. Тем самым он породит новую стабильную ветку разработки, в которой исправляются ошибки, но не добавляется новая функциональность. В названии ветки не присутствует третий компонент номера версии, так как он может изменяться. При этом основная разработка продолжается в `trunk/`. Поддерживаться должны те ветки разработки, которые входят в поддерживаемые версии ПО. Устаревшие ветки могут быть удалены.

Предположим, разработчик хочет обновить свой пакет в составе поставляемого ПО. Для этого он фиксирует версию пакета, меняет `debian/changelog` и сохраняет изменения в Subversion. Затем копирует содержимое ветки в директорию `tags/...` Сюда может копироваться как содержимое `trunk/`, так и `branches/`, в зависимости от стабильности. В некоторых случаях, когда разработчик уверен в стабильности ветки `trunk/`, возможно

пропустить этап создания стабилизируемой ветки в branches. Затем разработчик собирает пакет с исходными кодами проекта и передает на тестирование. Теперь, зная полную версию пакета, можно в любой момент восстановить состояние дерева исходных кодов проекта.

Важно, что при копировании в пределах репозитория Subversion целых веток разработки размер репозитория увеличивается незначительно, так как копирования на самом деле не происходит, а вместо этого создаются ссылки на текущее состояние копируемой ветки. То есть пока в порожденную ветку не вносились изменения, это только лишь ссылка.

4.5.2. Дебианизируемые пакеты

Оригинальные архивные файлы перед закладкой в репозиторий Subversion необходимо дебианизировать. Например, с помощью утилиты dh_make или любым другим способом. В дебианизируемых пакетах должен использоваться формат «3.0 (quilt)». Пакеты закладываются в репозиторий Subversion в виде двоичного файла *.orig.tar.ext, дополнительных файлов с исходными кодами *.orig-<компонент>.tar.ext и открытого дерева исходных кодов, в котором будет вестись разработка.

Вне зависимости от планируемого количества изменений будет неправильно сохранять такой проект в trunk/, так как он всегда основан на определенной версии оригинального проекта. Поэтому дебианизируемые пакеты сразу закладываются в директории branches/<версия>/. Для удобства можно создать символическую ссылку trunk/, которая будет указывать на branches/<версия>/ той версии, с которой ведется работа.

Если появляется новая версия оригинального проекта, то в branches/ заводится новая директория — новая ветка разработки. После этого разработчик, по возможности, переносит все необходимые патчи из предыдущей ветки. Поддерживаются только нужные ветки разработки, то есть текущая ветка и более старые ветки, которые входят в состав поддерживаемых версий дистрибутива.

Предположим, разработчик хочет обновить свой пакет в репозитории дистрибутива. Для этого он увеличивает версию релиза, меняет debian/changelog и сохраняет изменения в Subversion. Затем копирует содержимое ветки в директорию tags/<версия>-<релиз>. Собирает пакет с исходными кодами проекта и передает ответственному за дистрибутив. Теперь, зная версию и релиз пакета, можно в любой момент восстановить состояние дерева исходных кодов проекта.

Номер релиза для пакетов дистрибутива ОС ОН представляет собой следующую структуру: <исходный_релиз>astra<внутренний_релиз>. Исходный релиз идентифицирует пакет исходных кодов, взятый из стороннего debian-подобного дистрибутива. Для вновь дебианизируемых пакетов это значение всегда равно 0. Внутренний релиз отражает изменения, специфичные для ОС ОН. Для примера, если бы проект vim был дебианизируемым пакетом, то запись в директории tags/ репозитория Subversion могла выглядеть так:

```
7.1.314-0astra5
```

4.5.3. Наследуемые пакеты

Наследуемые пакеты берутся из существующих debian-based дистрибутивов и уже дебианизированы. Для простых проектов рекомендуется привести формат пакетов с исходными кодами к формату «3.0 (quilt)» для того, чтобы впоследствии их было удобнее поддерживать и накладывать собственные патчи. В некоторых случаях этот процесс можно автоматизировать, используя содержимое файла `*.diff.gz` формата «1.0» в качестве первого патча в стеке `quilt`. Более сложные пакеты используют собственную систему хранения и использования патчей. Привести такую систему к формату «3.0 (quilt)» затруднительно. Поэтому для подобных пакетов может использоваться индивидуальный подход. Разработчик сам может выбрать способ хранения таких пакетов в Subversion.

Однако общая схема ведения репозитория Subversion не должна изменяться. Как и в случае дебианизируемых пакетов, директория `trunk/` может нести только вспомогательную функцию и являться символической ссылкой на другую ветку разработки из директории `branches/`.

Особенностью наследуемых пакетов является то, что они уже имеют номер релиза, присвоенный им разработчиками исходного дистрибутива, откуда взят пакет. Несмотря на это, исходные коды пакета закладываются в директорию `branches/<версия>/` и номер релиза не учитывается. Разработчики исходного дистрибутива могут доработать пакет и увеличить номер релиза. При этом файл `*.orig.tar.ext` останется прежним. Поменяется только файл с накладываемыми изменениями. Поэтому не имеет никакого смысла создавать новую ветку разработки при изменении номера исходного релиза. Вместо этого, новый diff-файл интегрируется в существующее дерево разработки. Система `quilt` упростит процесс интеграции.

Правила ведения директории `tags/` для наследуемых пакетов точно такие же, как и для дебианизируемых. Только в отличие от дебианизируемых, для наследуемых пакетов поле `<исходный_релиз>` в номере релиза не равно нулю. Вместо этого оно определяется релизом исходного пакета, взятого из стороннего debian-подобного дистрибутива. Для примера, если бы проект vim был наследуемым пакетом, то запись в директории `tags/` репозитория Subversion могла выглядеть так:

```
7.1.314-3astra2
```

Примечание. В некоторых наследуемых пакетах в качестве релиза пакета присутствует не число, а слово, например `cdebconf-0.138lenny2` или `firefox-3.0_-3.0.10+nobinonly-0ubuntu1`. В таком случае при наследовании пакета слово с номером релиза заменяется на `astra<номер релиза>`.

После замены пакеты в приведенном выше примере, будут выглядеть следующим

образом:

```
cdebconf-0.138astra1, firefox-3.0_3.0.10+nobinonly-0astra1
```

4.6. Формирование пакетов с исходными текстами

Для формирования пакета с исходными текстами необходимо создать в дереве с исходными текстами программы специальный каталог `debian`, в котором расположены сценарии сборки пакета. Для начального создания каталога `debian` с шаблонами сценариев сборки deb-пакета необходимо выполнить команду, находясь в каталоге с исходными текстами программы:

```
dh_make -s -e builder@build -f ../test-1.1.tar.gz
```

где `test-1.1.tar.gz` ? gzip-архив с исходными текстами программы.

В дальнейшем необходимо изучить структуру каталога `debian` и согласно документу Debian Policy Manual [1] заполнить необходимые параметры конфигурационных файлов. В поле `Maintainer` файла debian/control разработчик указывает свое имя и существующий адрес электронной почты. Особое внимание при разработке пакетов необходимо обратить на правильное описание зависимостей сборки и зависимостей выполнения. Пакеты исходного кода должны выполнять требования, касающиеся обязательных полей в файле `debian/control` и проверять полученный пакет с помощью утилиты `lintian (lintian -c <имя deb пакета или dsc файла>)`. Недоработки, помеченные утилитой lintian как предупреждения «W», допускаются, а ошибки «E» рекомендуется исправить.

4.7. Сборка бинарных пакетов

Сборку пакетов из файлов с исходными текстами нужно выполнять в ОС при помощи команды:

```
dpkg-buildpackage -rfakeroot
```

Сборка должна в обязательном порядке выполняться под учетной записью непривилегированного пользователя. В дальнейшем пакет ПО должен быть протестирован на корректную установку с соблюдением зависимостей в ОС. После проверки установки пакета необходимо провести комплексное тестирование функционала перед поставкой пакета потребителю.

4.8. Разработка пакета с драйверами

Если драйвер использует только открытые исходные тексты, то для разработки пакета с драйверами рекомендуется использовать module-assistant, так как при таком подходе отсутствует зависимость от конкретной версии ядра.

Для использования компонента `module-assistant` необходимо выполнить следующие подготовительные действия:

– создать сценарий сборки модуля ядра (Makefile);

– создать пакет с исходным текстом модуля ядра, сценарием сборки модуля ядра и сценарием сборки пакета;

– собрать пакет с исходным текстом ядра и сценарием сборки и установить его в систему;

– собрать пакет с исполняемым модулем ядра при помощи module-assistant.

Пример сценария сборки (Makefile) модуля ядра представлен в Примере 1.

Для создания пакета с исходными текстами необходимо выполнить операции, описанные выше 4.6, или выполнить его создание вручную. Служебные файлы будут располагаться в каталоге debian внутри каталога с исходными текстами модуля ядра. Файлом сценария сборки пакета будет являться файл `rules` (см. Пример 2.). Для того чтобы `module-assistant` имел возможность собрать исполняемый модуль, в файл `rules` необходимо добавить несколько определений и целей, в заголовке файла должны быть подключены модули `module-assistant` и определены некоторые переменные:

```
PACKAGE=simple-modules
MA_DIR ?= /usr/share/modass
-include $(MA_DIR)/include/generic.make
-include $(MA_DIR)/include/common-rules.make
```

Также должны быть определены цели, необходимые для работы `module-assistant` (цель, в которой выполняется сборка пакета – `binary-modules`):

```
kdist_config: prep-deb-files
kdist_clean: clean
$(MAKE) $(MFLAGS) -f debian/rules clean
rm -f *.o *.ko
binary-modules:
dh_testroot
dh_clean -k
dh_installdirs lib/modules/$(KVERS)/misc

$(MAKE) KERNEL_DIR=$(KSRC) KVERS=$(KVERS)

install -m 0644 simple.$ko debian/$(PKGNAME)/lib/modules/$(KVERS)/misc

dh_installdocs
dh_installchangelogs
dh_compress
dh_fixperms
dh_installdeb
```

```
dh_gencontrol -- -v$(VERSION)
dh_md5sums
dh_builddeb --destdir=$(DEB_DESTDIR)
dh_clean -k
```

Для сборки пакета необходимо наличие файла `control` – файла описания пакета (Пример 3.). Также необходимо создать файл `control.modules.in`. Данный файл необходим для последующей сборки пакета с исполняемым кодом модуля ядра(Пример 4.). После создания всех необходимых служебных файлов выполнить сборку пакета с исходными текстами модуля ядра с помощью команды:

```
dpkg-buildpackage -rfakeroot
```

находясь в каталоге с исходными текстами. По окончании работы программы `dpkg-buildpackage` в вышележащем каталоге появится пакет с исходными текстами ядра. Установить полученный пакет при помощи команды:

```
dpkg -i <имя_пакета>
```

Собрать пакет с исполняемым модулем ядра, выполнив команду:

```
m-a build simple
```

где simple – название собираемого модуля. Пакет с исполняемым модулем ядра будет располагаться в каталоге `/usr/src/`. Исходный текст тестового модуля в Примере 5.

Примеры:

1. `(Makefile)`

```
obj-m := simple.o
mksm_tpm-objs := simple.o

CTAGS_FLAGS := -R
PACKAGE     := $(shell basename $(PWD))

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD       := $(shell pwd)

all: modules

modules:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

tags: $(wildcard *.c) $(wildcard *.h)
ctags $(CTAGS_FLAGS) $^

clean:
```

```
 -@[ -n "`which dh_clean`" -a -d debian ] && { echo Running dh_clean;\\
                                                          dh_clean; }
 rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions
 rm -rf Module.markers Module.symvers modules.order

distclean: clean
rm -rf tags

dist: Makefile $(wildcard *.c) $(wildcard *.h) debian
@echo "Creating distributive source package $(PACKAGE).tar.gz"
@cd ..; tar --exclude=".*" -cvzf $(PACKAGE).tar.gz \\
                                        $(addprefix $(PACKAGE)/,$^)

export: Makefile $(wildcard *.c) $(wildcard *.h) debian
@echo "Creating distributive source package \\
                          $(PACKAGE)_$(shell date +%Y%m%d).tar.gz"
@cd ..; tar --exclude=".*" -cvzf \\
$(PACKAGE)_$(shell date +%Y%m%d).tar.gz $(addprefix $(PACKAGE)/,$^)

install: modules
install -m 0755 -d $(DESTDIR)/lib/modules/$(shell uname -r)/misc
install -m 0644 $(obj-m:.o=.ko) \\
$(DESTDIR)/lib/modules/$(shell uname -r)/misc

deb: clean
dpkg-buildpackage -us -uc -rfakeroot -I".*" -Itags

.PHONY: clean distclean dist export install deb
```

2. (rules)

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37
# of dh-make.
#
```

```
# This version is for a hypothetical package that can build
# a kernel modules
# architecture-dependant package via make-kpkg, as well as an
# architecture-independent module source package, and other packages
# either dep/indep for things like common files or userspace components
# needed for the kernel modules.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

# some default definitions, important!
#
# Name of the source package
psource:=simple-source

# The short upstream name, used for the module source directory
sname:=simple

### KERNEL SETUP
### Setup the stuff needed for making kernel module packages
### taken from /usr/share/kernel-package/sample.module.rules

# prefix of the target package name
PACKAGE=simple-modules
# modifieable for experiments or debugging m-a
MA_DIR ?= /usr/share/modass
# load generic variable handling
-include $(MA_DIR)/include/generic.make
# load default rules, including kdist, kdist_image, ...
-include $(MA_DIR)/include/common-rules.make

# module assistant calculates all needed things for us and sets
# following variables:
# KSRC (kernel source directory), KVERS (kernel version string), KDREV
# (revision of the Debian kernel-image package), CC (the correct
# compiler), VERSION (the final package version string), PKGNAME (full
# package name with KVERS included), DEB_DESTDIR (path to store DEBs)
```

```
# The kdist_config target is called by make-kpkg modules_config and
# by kdist* rules by dependency. It should configure the module so it is
# ready for compilation (mostly useful for calling configure).
# prep-deb-files from module-assistant creates the neccessary debian/ files
kdist_config: prep-deb-files

# the kdist_clean target is called by make-kpkg modules_clean and from
# kdist* rules. It is responsible for cleaning up any changes that have
# been made by the other kdist_commands (except for the .deb files created)
kdist_clean: clean
$(MAKE) $(MFLAGS) -f debian/rules clean
rm -f *.o *.ko
#
### end  KERNEL SETUP

configure: configure-stamp
configure-stamp:
dh_testdir
touch configure-stamp


build-arch: configure-stamp  build-arch-stamp
build-arch-stamp:
dh_testdir

$(MAKE)

touch $@

k = $(shell echo $(KVERS) | grep -q ^2.6 && echo k)

# the binary-modules rule is invoked by module-assistant while processing
# the kdist* targets. It is called by module-assistant or make-kpkg
# and *not* during a normal build
binary-modules:
dh_testroot
dh_clean -k
dh_installdirs lib/modules/$(KVERS)/misc
```

```
$(MAKE) KERNEL_DIR=$(KSRC) KVERS=$(KVERS)

install -m 0644 simple.$ko debian/$(PKGNAME)/lib/modules/$(KVERS)/misc

dh_installdocs
dh_installchangelogs
dh_compress
dh_fixperms
dh_installdeb
dh_gencontrol -- -v$(VERSION)
dh_md5sums
dh_builddeb --destdir=$(DEB_DESTDIR)
dh_clean -k

build-indep:  configure-stamp build-indep-stamp
build-indep-stamp:
dh_testdir

touch $@

build: build-arch build-indep

clean:
#dh_testdir
#dh_testroot
rm -f build-arch-stamp build-indep-stamp configure-stamp

# Add here commands to clean up after the build process.
$(MAKE) clean

dh_clean

install: DH_OPTIONS=
install: build
dh_testdir
dh_testroot
dh_clean -k
```

```
dh_installdirs

dh_installdirs -p$(psource)    \
usr/src/modules/$(sname)/debian

cp *.c debian/$(psource)/usr/src/modules/$(sname)
cp Makefile debian/$(psource)/usr/src/modules/$(sname)
cp debian/*modules.in* \
debian/$(psource)/usr/src/modules/$(sname)/debian
echo "This is dummy file. It contents will be lost." > \\
        debian/$(psource)/usr/src/modules/$(sname)/debian/control
cp debian/rules debian/changelog debian/copyright \
debian/compat debian/$(psource)/usr/src/modules/$(sname)/debian/
cd debian/$(psource)/usr/src && tar c modules | bzip2 -9 > \\
        $(sname).tar.bz2 && rm -rf modules

dh_install

# Build architecture-independent files here.
# Pass -i to all debhelper commands in this target to reduce clutter.
binary-indep: build install
dh_testdir -i
dh_testroot -i
dh_installchangelogs  -i
dh_installdocs -i
dh_installexamples -i
dh_installman -i
dh_link -i
dh_compress -i
dh_fixperms -i
dh_installdeb -i

dh_installdeb -i
dh_shlibdeps -i
dh_gencontrol -i
dh_md5sums -i
dh_builddeb -i
```

```
# Build architecture-dependent files here.
binary-arch: build install
dh_testdir -s
dh_testroot -s
dh_installdocs -s
dh_installexamples -s
dh_installmenu -s
dh_installcron -s
# dh_installman -s
dh_installinfo -s
dh_installchangelogs  -s
dh_strip -s
dh_link -s
dh_compress -s
dh_fixperms -s
# dh_makeshlibs -s
dh_installdeb -s
# dh_perl -s
dh_shlibdeps -s
dh_gencontrol -s
dh_md5sums -s
dh_builddeb -s

binary: binary-indep # binary-arch
.PHONY: build clean binary-indep binary-arch binary install configure \\
binary-modules kdist kdist_configure kdist_image kdist_clean
```

3. (control)

```
Source: simple
Section: admin
Priority: extra
Maintainer: builder <builder@rusbitech.ru>
Build-Depends: debhelper (>= 7), bzip2, coreutils, sed, make
Standards-Version: 3.7.3

Package: simple-source
Architecture: all
Depends: module-assistant, debhelper (>= 7), make, bzip2, coreutils, sed
Description: Source for test driver
```

4. (control.modules.in)

```
Source: simple
Section: admin
Priority: optional
Maintainer: builder <builder@rusbitech.ru>
Build-Depends: debhelper (>= 7), gcc, linux-headers-_KVERS_,\\
coreutils, sed, make
Standards-Version: 3.7.3

Package: simple-modules-_KVERS_
Architecture: any
Depends: linux-image-_KVERS_, module-init-tools
Provides: simple-modules
Description: Simple test driver
```

5. (simple.c)

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void)
{
        printk("Test module hello.Run.\n");
        return 0;
}
void cleanup_module(void)
{
        printk(KERN_ALERT "Test module hello.Exit.\n");
}
```

5. ОСОБЕННОСТИ РАЗРАБОТКИ ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ, ВЗАИМОДЕЙСТВУЮЩИХ С РАБОЧИМ СТОЛОМ FLY

В данном разделе приведены правила для разработчиков приложений, включаемых в состав рабочего стола Fly или взаимодействующих с ним.

Реализация данных правил поможет добиться:

– унификации сборки и установки приложений;

– легкости модификации и сопровождения;

– единообразия внешнего вида, внутренней структуры и поведения.

От разработчика требуется знание языка программирования C++ и библиотеки графического интерфейса Qt 5. Настоятельно рекомендуется изучить, например, следующие руководства по работе с этой библиотекой:[12]–[17]. Также предполагается знание основных стандартов от `freedesktop.org`.

5.1. Среда сборки

Все Fly-приложения должны не только единообразно выглядеть и взаимодействовать с пользователем, но и единообразно собираться и устанавливаться. В этой связи существует ряд требований.

5.2. Группы и имена приложений

Все приложения, кроме особых (например, менеджер окон), создаются с использованием графических библиотек Qt 5.

Создание приложения для Fly начинается с определения его назначения и места в составе рабочего стола.

Основные группы приложений Fly:

1) обычные программы;

2) системные программы, т. е. программы, изменяющие конфигурационные файлы ОС;

3) библиотеки;

4) ключевые или особые программы (графический вход, менеджер окон, менеджер файлов и т. п.).

Имена всех приложений формируются в соответствии с их назначением по единому принципу: `fly[-admin]-имя`.

Все приложения имеют префикс «fly-». Приложения для настройки системы имеют дополнительный префикс «admin-». Имя приложения должно отражать его назначение, быть осмысленным. С учетом имени приложения образуются соответствующие файлы и каталоги, например:

– исполняемый файл (<имя_приложения>);

– файл перевода (<имя приложения>_ru.ts);

– каталог для данных (/usr/share/fly/data/<имя_приложения>);

– desktop entry-файл (<имя_приложения>.desktop).

Таким образом имя приложения — основа идентификации приложения в системе.

5.3. Сборка

Оформление приложений в виде пакетов регулируется правилами сборки пакетов для ОС. Пакет flybuild предоставляет файлы, автоматизирующие некоторые задачи конфигурирования и установки. Для deb-пакетов он предоставляет файлы /usr/share/flybuild/fly.mk и /usr/share/flybuild/fly_vars.mk, которые предназначены для включения в файл debian/rules. Типичный файл debian/rules выглядит следующим образом:

```
#!/usr/bin/make -f

include /usr/share/flybuild/fly.mk
include /usr/share/cdbs/1/class/qmake.mk
```

Также, flybuild содержит вспомогательные файлы для системы сборки qmake: /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly.prf и usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly_vars.prf. fly_vars.prf задает пути, используемые всеми приложениями при установке, fly.prf — основные процедуры и цели для сборки и установки приложений. Обычно их не нужно подключать каким-то особым образом, достаточно добавить строку "CONFIG += fly" в .pro-файл. В этом случае, можно считать, что fly.prf будет добавлен в конец файла проекта. Но если возникает необходимость использовать переменные, заданные в fly_vars.prf, то нужно подключить этот файл командой load(fly_vars) в нужном месте.

При каждой, даже формальной, пересборке пакета для его включения в очередной релиз дистрибутива следует изменять последнюю цифру, т.е. 2.0.1, 2.0.2 и т.д. При изменении функционала программ версия меняется по усмотрению разработчика.

Рекомендуется так же отделять компоненты для разработки (заголовочные файлы, компоненты для Qt Designer) от самих разделяемых библиотек в виде name-dev-пакетов.

5.4. Указание секции дистрибутива в пакетах

В пакетах прикладных программ для Fly следует указывать non-free/fly. Если же программа была основана на opensource-проекте, то следует указывать секцию fly.

5.5. Основные файлы приложения Fly

Получив имя, приложение должно предоставить некоторые файлы:

– pro-файл для сборки;

– desktop entry-файл;

– файлы с переводом;

– возможно также специфичные файлы (иконки и т. п.).

5.5.1. Файл .pro

Требования к pro-файлам приложений:

– имя образуется как `имя программы.pro`;

– краткость и простота, не избыточность;

– единообразие состава и структуры;

– отсутствие каких-либо абсолютных путей;

– обязательное задание параметра конфигурации `CONFIG += fly`.

Выражение `CONFIG += fly` вызывает подключение файла `fly.prf`, который централизует важнейшие особенности сборки и установки приложений, освобождая программистов от необходимости их повторного определения в своих pro-файлах. `fly.prf` использует переменные, заданные в `fly_vars.prf`, для определения путей установки файлов приложения. Ни один pro-файл ни одной программы не должен без необходимости переопределять переменные, заданные в `fly_vars.prf`, а, наоборот, должен их использовать. Таким образом, создание pro-файла приложения начинается с изучения `fly_vars.prf` и `fly.prf`. Их можно найти в каталоге `/usr/lib/x86_64-linux-gnu/qt5/mkspecs/features`. Приведем пример правильного pro-файла, например `fly-commi.pro`:

```
TEMPLATE = app
TARGET = fly-commi

QT += widgets
CONFIG += fly
FLY += core ui

HEADERS += mylineedit.h
SOURCES += main.cpp mylineedit.cpp
FORMS += commimainform.ui optionsform.ui transferform.ui
TRANSLATIONS = fly-commi_ru.ts
```

или fly-admin-cron.pro:

```
TEMPLATE = app
TARGET = fly-admin-cron
```

```
QT += widgets
CONFIG += fly
FLY += core ui ui_extra

SOURCES += main.cpp cconfig.cpp
HEADERS += cconfig.h ccrondata.h ...
FORMS += fly-cron.ui
TRANSLATIONS = fly-admin-cron_ru.ts
```

Примерно в такой последовательности и составе должны следовать строки. Обратите внимание на краткость этих pro-файлов и на переменную `FLY`, которая позволяет подключить такие библиотеки, как: `libflycore` (`FLY += core`), `libflyui` (`FLY += ui`), `libflyuiextra` (`FLY += ui_extra`), необходимые большинству приложений Fly.

Любые отклонения в pro-файлах должны быть обоснованными. Среди часто встречающихся отметим такие:

– необходимость определения наличия каких-либо файлов при сборке;

– необходимость установки дополнительных данных программы.

В первом случае можно использовать конструкции вида:

```
exists( /usr/include/kudzu/isapnp.h ) {
SUBDIRS += fly-admin-snddetect
}
```

Во втором случае допустимо:

```
load(fly_vars)
pics.path = $${FLY_INSTALL_DATA}/fly-parashoot/icons
pics.files = pics/*
sounds.path = $${FLY_INSTALL_DATA}/fly-parashoot/sounds
sounds.files = sounds/*
INSTALLS += pics sounds
```

— для файлов, устанавливаемых в собственный каталог приложения (здесь — `$${FLY_INSTALL_DATA}/fly-parashoot`), или:

```
pam.path = /etc/pam.d
pam.files = ../pam/fly-dm
```

— для файлов, устанавливаемых в разделяемые каталоги (здесь — `/etc/pam.d`).

Еще пример фрагмента pro-файла одной из библиотек:

```
...
load(fly_vars)

headersfly.path = $$FLY_INSTALL_HEADERS
headersfly.files = flyui.h flytranslator.h flyabout.h \
```

```
flyhelp.h flyhelpmenu.h flycmdlineargs.h

uiheadersfly.path = $$FLY_INSTALL_HEADERS
uiheadersfly.extra = $$QMAKE_COPY_FILE $$UI_DIR/ui_flyabout.h \
$(INSTALL_ROOT)/$${uiheadersfly.path}

INSTALLS += headersfly uiheadersfly
...
```

Разработчикам необходимо обратить внимание, что Qt как кросс-платформенное средство предоставляет ряд макросов для часто используемых команд, например `QMAKE_COPY_FILE`, `QMAKE_MKDIR` и т. д. (см. файл `$QTDIR/mkspecs/default/qmake.conf`).

Если возникает необходимость удалять каталоги при выполнении `make clean` или `make distclean`, то можно добавить их к переменным `FLY_CLEAN_DIRS` и `FLY_DISTCLEAN_DIRS` соответственно.

5.5.2. Файл desktop entry

Почти каждое fly-приложение должно сопровождаться файлом `desktop entry` [2] с именем, формируемым по правилу:

`<имя_приложения>.desktop`

Данный файл содержит информацию о том, к какому классу приложений относится поставляемая программа, имя иконки для представления на рабочем столе, способ запуска данной программы и служебную информацию для интеграции приложения в рабочий стол.

Имя иконки приложения должно быть указано в поле `Icon` файла `desktop entry`. В виде исключения в этом поле допускается указывать полный путь к иконке приложения.

Приложения, открывающие специфичные типы файлов, должны объявлять соответствующие MIME-типы данных файлов в поле `MimeType=` своих *.desktop-файлов.

Примеры:

1.
```
[Desktop Entry]
Exec=fly-admin-date
Icon=date
Type=Application
Name=Configure clock
Name[ru]=Дата и время
Comment=Configure date & time
Comment[ru]=Настройка системного времени и даты
Categories=Settings;SystemSetup
X-Fly-AdminOnly=true
```

2. `[Desktop Entry]`

```
Type=Application
Exec=fly-archiver open %f
Icon=package
Name=Archivator
Name[ru]=Архиватор
Comment=Qt RAR, TAR, ZIP, RPM, Gzip, Bzip2 GUI
Comment[ru]=Qt RAR, TAR, ZIP, RPM, Gzip, Bzip2 интерфейс
Categories=Application;Office;
Actions=Open;Decompress;Compress
DocPath=fly-arch/index.html
MimeType=application/x-rar;application/x-bzip-compressed-tar;
 application/x-bzip;application/x-compressed-tar;
 application/x-tar;application/x-gzip;application/zip;
[Desktop Action Compress]
Name=Add file to archive...
Name[ru]=Добавить файл в архив...
Exec=fly-arch compress %F
[Desktop Action Open]
Name=Open with fly-arch...
Name[ru]=Открыть с помощью fly-arch...
Exec=fly-arch open %f
[Desktop Action Decompress]
Name=Extract from archive...
Name[ru]=Извлечь из архива...
Exec=fly-arch decompress %f
```

В принципе, все поля файла имеют очевидное назначение, подробно описанное в [2]. Слово `Settings` в поле `Categories` говорит о принадлежности программы к средствам администрирования. Рабочий стол Fly имеет в своем составе специальную программу «Панель управления», предназначенную для централизованного вызова утилит управления системой. «Панель управления» находит нужные утилиты именно по полю `Categories`.

Категорию можно назначить не только файлу, но и целому каталогу путем заполнения поля `Categories` в файле `.directory` (тип `Directory`), находящемся в этом каталоге. Значение поля ‘X-Fly-AdminOnly‘ определяет, нужно ли показывать данную программу только администратору, или всем пользователям системы.

Файлы `desktop entry` как правило должны иметь `Type=Application`, но для других случаев возможно использование других типов, разрешенных стандартом [2]:

Application, Link, Directory.

Слово Fly следует использовать в поле Name ярлыков приложений (особенно административных), только если настраивается что-то специфичное для самого Fly, например, «горячие» клавиши или меню самого рабочего стола Fly, т. е. то, что вне Fly неприменимо. Если настраиваются какие-то другие общие службы, то слово Fly не следует использовать. Его также можно использовать, чтобы отличать приложения, разработанные специально для Fly, от аналогичных приложений для других рабочих столов, например текстовый редактор может называться «Текстовый редактор Fly», если предполагается присутствие в системе и других текстовых редакторов. В названиях программ, видимых пользователю (в меню или в панели управления), не следует повторять одни и те же слова, например, «Менеджер камер Fly», «Менеджер сканеров Fly», лучше так: «Камеры», «Сканеры». Чем короче названия, тем лучше, например, не «Редактор переменных окружения», а «Переменные окружения».

5.5.3. Файл перевода

Все приложения должны быть русифицированы с использованием стандартных средств Qt.

Не следует статично вбивать русские названия элементов графического интерфейса непосредственно в код программы или в формы, генерируемые с помощью Qt Designer. Это многократно усложняет локализацию приложений для других стран.

Изначально Fly-приложения должны содержать только английские варианты названий элементов своих интерфейсов. Файл с переводами (ts-файл) должен формироваться с помощью программ lupdate и linguist, входящих в состав дистрибутива ОС. При установке ts-файл будет сконвертирован в бинарный qm-файл, подходящий для загрузки qt-приложением. Подключение соответствующего текущей локали qm-файла производится в функции flyInit() библиотеки libflyui, вызываемой в main-функции приложения. Для этого необходимо, чтобы файл перевода был соответствующим образом расположен (обеспечивается скриптами установки fly.prf) и имел имя, сформированное по шаблону имя приложения_локаль.qm, например fly-admin-mouse_ru.qm.

5.5.4. Файл для определения иконок приложений

Приложение может установить свою специфичную иконку. Для этого она должна быть в svg-формате, лежать в том же каталоге, что и pro-файл приложения и название ее файла должно быть производным от переменной TARGET. Например, если TARGET = fly-admin-cron, то файл иконки должен называться fly-admin-cron.svg или fly-admin-cron.svgz. Если необходимо установить несколько svg-иконок, то нужно в переменной FLY_INSTALL_SVG_ICONS_FROM задать каталог относительно pro-файла, в котором они лежат. При этом следует учесть, что иконка, про которую говорилось

в предыдущем способе установки, будет проигнорирована. Названия файлов этих иконок должны соответствовать шаблону <контекст>-<имя_иконки>.svg, где <контекст> — название каталога, представляющего контекст иконки в теме. При сборке приложения они будут сконвертированы в png-файлы и скопированы в поддеревья сборки типа ./debian/tmp/usr/share/icons/hicolor/16x16/apps и т. д., т. е. в подкаталоги <контекст> темы иконок hicolor — родительской темы для всех других тем иконок. Приложение должно обеспечить попадание указанных поддеревьев usr/share/icons/hicolor в свой пакет. Все это позволит приложению в процессе работы операционной системы, даже при смене текущей темы, всегда находить свои иконки, т. к. тема hicolor есть всегда.

5.5.5. Дополнительные параметры

Полезные переменные, задаваемые в fly_vars.prf:

– FLY_INSTALL_PREFIX — корневой каталог для установки (обычно /usr);

– FLY_INSTALL_BINS — каталог, в который попадут исполняемые файлы;

– FLY_INSTALL_LIBS — каталог, в который попадут библиотеки;

– FLY_INSTALL_HEADERS — каталог, в который попадут заголовочные файлы;

– FLY_INSTALL_DATA — корневой каталог для файлов данных;

Их можно определить извне, передав qmake в командной строке:

```
qmake FLY_INSTALL_PREFIX=/usr/local
```

При использовании cdbs (рекомендуемый способ сборки пакетов) для этого достаточно добавить их к DEB_QMAKE_ARGS:

```
DEB_QMAKE_ARGS += FLY_INSTALL_PREFIX=/usr/local
```

Но делать это не рекомендуется. Если возникает потребность использовать эти переменные в debian/rules, то можно включить в него файл /usr/share/flybuild/fly_vars.mk.

Пример файла debian/rules:

```
#!/usr/bin/make -f

include /usr/share/flybuild.mk
include /usr/share/cdbs/1/class/qmake.mk
```

5.6. Разработка приложения

5.6.1. Общие требования

Fly-приложение должно быть интуитивно понятным, ориентированным на неподготовленного пользователя.

Пользователю обязательно должны задаваться вопросы о сохранении изменений, вступлении изменений в силу, перезаписи существующих настроек или файлов, перезапуске системы и т. п. При этом всегда должна даваться возможность осуществить отмену действий

такого рода.

Главной функцией, которую следует вызвать сразу после создания экземпляра `QApplication` является `flyInit()`.

Ее аргументы — версия приложения и описание назначения программы, понятное неподготовленному пользователю, например, в стиле «Данная программа fly-* предназначена для ...». Имя приложения в `flyInit()` извлекается автоматически и используется при поиске переводов, при работе `QSettings` и т. п. Указанные версия и описание используются при создании диалога «О программе...». Пример, настоятельно рекомендуемый к использованию:

```
flyInit("2.0.0",QT_TRANSLATE_NOOP("Fly","Fly hotkeys editor"));
```

где `2.0.0` — строка с номером версии для диалога «О программе»;

`Fly` — фиксированное слово-контекст, остальное задается разработчиком.

Эта функция позволяет потом использовать класс `FlyHelpMenu` без указания каких-либо деталей, использовать `QSettings` с именами параметров без указания имени программы, т. е. в предельно кратком виде типа «`/width`», «`/height`».

Программист может автоматизировать подстановку номера версии в `flyInit()` из версии пакета.

Примеры:

1. В `rules` добавить:

```
SOURCE_VERSION:=$(shell head -1 debian/changelog | \
          cut -d\( -f2 | cut > -d\) -f1)
```

2. Там же передать эту переменную:

```
qmake: $(QTDIR)/bin/qmake SOURCE_VERSION=$(SOURCE_VERSION)
```

3. При использовании cdbs и flybuild вместо предыдущих двух пунктов можно использовать:

```
include /usr/share/flybuild/fly.mk
include /usr/share/cdbs/1/class/qmake.mk
```

4. В pro-файл добавить (только если не используется `CONFIG += fly`):

```
DEFINES += SOURCE_VERSION=\\\"$$SOURCE_VERSION\\\"
```

5. Подставить в `flyInit`:

```
flyInit(SOURCE_VERSION, \
          QT_TRANSLATE_NOOP("Fly", "Fly rich text > editor"))
```

Таким образом, разработчикам останется только исправить `rules` (1 или 2 строки) и pro-файл (1 строка и то, только если не используется `CONFIG += fly`!). При этом, в самой программе перед вызовом `flyInit()` можно, не полагаясь на `rules`, сделать так:

```
#ifndef SOURCE_VERSION
```

```
#define SOURCE_VERSION "2.0.0"
#endif
...
flyInit(SOURCE_VERSION,
        QT_TRANSLATE_NOOP("Fly","... short description..."));
```

Любое Qt-приложение может принимать ряд аргументов командной строки, специфичных для Qt и X11. Например, `-geometry` и `-display` (см. описание класса `QApplication`).

Приложение Fly, если оно самостоятельно (с помощью класса `QCommandLineParser`) анализирует аргументы своей командной строки, должно без препятствий пропускать упомянутые стандартные Qt/X11-аргументы, не останавливая свою работу с сообщениями типа «неверный аргумент».

Также следует обратить внимание на `QtSingleApplication` (см. `libqtsingleapplication-dev`). Этот класс описан в официальной документации от Trolltech. Для его использования нужно добавить опцию qtsingleapplication к переменной CONFIG в pro-файле. Он позволяет обеспечить запуск одного экземпляра приложения в системе. Однако, бывают ситуации, когда надо запустить несколько экземпляров, но только по одному для каждого дисплея.

Различать экземпляры приложений для разных дисплеев можно с помощью переменной окружения `DISPLAY`, например, так

```
QtSingleApplication app("MySingleInstance"+getenv("DISPLAY"), argc, argv)
```

Однако, не стоит забывать, что стандартный X11-аргумент `-display <displayname>` имеет больший приоритет, чем переменная `DISPLAY`. Поэтому сначала надо проверить его наличие и, если он задан, то использовать именно его.

5.6.2. Абсолютные и относительные пути

При разработке приложений следует избегать использования абсолютных и относительных путей для обращения к файлам данных и файлам иконок. Только в случае, если расположение файла заранее предопределено каким-либо стандартом и не меняется от версии к версии системы (например, конфигурационный файл `lilo.conf` загрузчика `LILO` по умолчанию располагается в каталоге `/etc`), можно статично включить абсолютный путь к данному файлу в код программы.

Использование относительных путей, содержащих конструкции типа «.» и «..», может привести к сбою работы программы, т. к. она может быть запущена из любого места.

Если в приложении требуется осуществить доступ к одному из стандартных каталогов пользователя или системы (например, каталогу рабочего стола, который для каждого пользователя свой), то путь к данному каталогу должен быть определен с помощью ме-

тодов класса QStandardPaths (в Qt 5). Для получения путей специфических каталогов, используемых в Fly, можно воспользоваться средствами, предоставляемых библиотекой libflycore. В заголовочном файле flyfileutils.h определена функция:

```
char* GetFlyDir(FlyDir type)
```

где type является перечислением, определяющим требуемый каталог:

```
typedef enum _FlyDir {
WM_DIR, USER_TMP_DIR, USER_BASE_DIR, USER_THEME_DIR,
USER_START_MENU_DIR, USER_CPL_DIR, USER_DESKTOP_DIR,
USER_DOCUMENTS_DIR, USER_TRASH_DIR, FLY_WORK_DIR,
APP_SHARED_DIR, APP_TRANS_DIR, APP_DOCS_DIR,
APP_DOCS_HTML_DIR, WM_SOUNDS_DIR, WM_IMAGES_DIR,
WM_KEYMAPS_DIR, USER_TOOLBAR_DIR, ... USER_CACHE_DIR, ...
} FlyDir
```

где WM_DIR — каталог общих настроек (/usr/share/fly-wm), на основе которых при первом запуске fly-wm создаются начальные индивидуальные настройки для каждого пользователя;

USER_TMP_DIR — временный каталог;

USER_BASE_DIR — каталог настроек для каждого пользователя ($HOME/.fly);

USER_THEME_DIR — каталог тем оформления рабочего стола ($HOME/.fly/theme);

USER_START_MENU_DIR — каталог стартового меню ($HOME/.fly/startmenu);

USER_AUTOSTART_MENU_DIR — каталог автозапуска ($HOME/.fly/startmenu или $HOME/.config/autostart);

USER_DESKTOP_DIR — каталог ярлыков рабочего стола ($HOME/Desktop);

USER_TRASH_DIR — каталог корзины (сейчас $HOME/.local/share/Trash);

APP_SHARED_DIR — каталог данных для приложений (/usr/share/fly);

APP_DOCS_DIR — каталог документации (/usr/share/doc/fly);

APP_DOCS_HTML_DIR — каталог html-документации (/usr/share/doc/fly/html);

APP_TRANS_DIR — каталог файлов переводов (/usr/share/fly/translations);

USER_TOOLBAR_DIR — каталог ярлыков для панели задач ($HOME/.fly/toolbar), т. е. панели быстрого запуска, расположенной на панели задач справа от кнопки «Пуск»;

USER_DOCUMENTS_DIR — каталог для хранения документов пользователя ($HOME/Documents), с возможными подкаталогами для файлов с ненулевыми мандатными метками.

Поддерживаются все каталоги, предусмотренные пакетом xdg-user-dir, а именно: USER_VIDEOS_DIR, USER_MUSIC_DIR, USER_PICTURES_DIR, USER_DOWNLOAD_DIR, USER_PUBLICSHARE_DIR, USER_TEMPLATES_DIR, но в Qt-приложениях следует использовать класс QStandardPaths.

Данная функция возвращает указатель на статическую область памяти, содержащую требуемый абсолютный путь. Так, например, все приложения могут получить доступ к своим файлам переводов из каталога, возвращаемого по вызову `GetFlyDir(APP_TRANS_DIR)`.

Оконный менеджер `fly-wm` может запускать приложения с ненулевыми мандатными уровнями. Некоторые приложения могут не работать в силу того, что они производят запись в файлы с ненулевыми уровнями. Для упорядочивания все приложения, которым требуется временный каталог, должны получить путь к нему из переменной `$TMPDIR`. Гарантируется, что если приложение было запущено с ненулевым уровнем, то `$TMPDIR` для этого приложения выставлена на каталог, позволяющий в него писать/читать. Получить свой временный каталог приложение может с помощью стандартной функции `getenv("TMPDIR")` или более общим вызовом `GetFlyDir(USER_TMP_DIR)` из библиотеки `libflycore`. Аналогичное справедливо и для каталога «Мои документы» – `GetFlyDir(USER_DOCUMENTS_DIR)` и (или) переменной окружения `$USERDOCDIR` и каталога с иконками рабочего стола — `GetFlyDir(USER_DESKTOP_DIR)` и (или) переменной окружения `$DESKTOPDIR` и т. д.

Начиная с версии ОС ОН Орёл 2.12 рекомендуется вместо `$TMPDIR` использовать переменную `$XDG_RUNTIME_DIR`.

Способом установки и получения путей к основным каталогам является пакет `xdg-user-dirs` (см. его описание), кооперация с которым уже реализована в `libflycore`.

5.6.3. Средства разработки

Графический интерфейс должен быть интуитивно понятен пользователю. Диалоги и окна разных приложений должны выглядеть единообразно. Частично это достигается за счет использования графической библиотеки Qt без переопределения параметров по умолчанию, но конечная ответственность лежит на разработчике приложения. Для обеспечения легкости модификации и единообразия при разработке графического интерфейса настоятельно рекомендуется использовать формы Qt Designer.

Для облегчения и унификации разработки предоставляется ряд библиотек, основные из которых `libflycore` и `libflyintegration`. В них содержатся общие для приложений классы (в т. ч. и графические), функции и структуры данных.

Библиотека `libflycore` не зависит от Qt. Ее назначение:

– разбор файлов `desktop entry` [2];

– доступ к темам иконок [3];

– взаимодействие с менеджером окон;

– формирование абсолютных путей к файлам рабочего стола, включая файлы русификации, помощи и т. д.;

– дополнительные задачи, не требующие GUI.

Библиотека `libflyintegration` — библиотека, содержащая общие для всех прило-

жений элементы графического интерфейса, базирующиеся на Qt, такие как меню «Помощь» и диалог «О программе...» и др., а также ряд вспомогательных функций.

Использование указанных библиотек может существенно ускорить разработку приложений и их интеграцию с рабочим столом.

Дополнительно предоставляются библиотеки:

– `libflyuiextra` — для дополнительных элементов графического интерфейса;

– `libflyuinet` — для элементов настройки сети;

– ряд других, включая библиотеки для выполнения файловых операций.

5.6.4. Локализация

Все заголовки и надписи должны быть выполнены на русском языке. Локализация выполняется средствами и в соответствии с рекомендациями Qt (подробнее см. 5.5.3).

5.6.5. Диалоги

Во всех диалогах должен использоваться класс `QDialogButtonBox`, как скрывающий детали перевода и размещения кнопок. Аналогично в диалогах-мастерах должен использоваться набор кнопок по умолчанию. Нестандартный состав кнопок, их расположение и расстояния допускаются только для сложных нестандартных диалогов. Программы, имеющие диалоговый интерфейс, должны поддерживать следующие «горячие» клавиши:

– **<Enter/Return>** — эквивалент нажатия кнопки **[Да]**;

– **<Escape>** — эквивалент нажатия кнопки **[Отмена]**;

– **<F1>** — вызов помощи для диалога (опционально).

Для задания «горячих» клавиш для типичных действий в классе `QKeySequence` существует специальный `Enum`: `QKeySequence::StandardKey`. Его элементы и надо использовать, например, при создании таких типовых действий, как: копирование, вставка, создание, удаление и т. п.

Отметим пару моментов при использовании диалога-мастера (`QWizard`). Не следует без необходимости переопределять стиль этого диалога по умолчанию (`ClassicStyle`). При наличии соответствующих элементов дизайна их надо использовать стандартным способом `QWizardPage::setPixmap()`. При этом первая и последняя страницы, как правило, должны быть с вводными и заключительными пояснениями и иметь только боковые картинки (watermark, слева, не более трети ширины диалога и высотой, равной высоте диалога). Промежуточные страницы, на которых осуществляется основное взаимодействие с пользователем, должны иметь картинку сверху, содержащую, как правило, наряду с картинками (banner, logo) также и пояснения к текущей странице (title, subtitle).

5.6.6. Меню

Типичное меню приложения должно иметь примерно следующую структуру: «Файл», «Правка», «...», «Помощь».

Пункты «Файл» и «Помощь» являются обязательными пунктами меню.

Меню «Файл» может состоять из следующих пунктов:

«Новый» или «Создать»

«Открыть...»

...

–разделитель–

«Закрыть»

«Сохранить»

«Сохранить как...»

...

–разделитель–

«Печать...»

–разделитель–

«Выход»

–разделитель–

«Список последних открывавшихся файлов»

Пункт «Выход» должен иметь либо стандартную иконку из текущей темы «application-exit», «Actions», либо не иметь иконки вообще.

ВНИМАНИЕ! Никакая другая иконка не должна использоваться для этих целей.

Меню «Помощь» имеет фиксированную структуру, которая должна использоваться во всех приложениях:

«Содержание F1»

«О программе...»

без каких-либо сепараторов и иконок. Как исключение у «Содержание F1» допускается иконка «help-contents», «Actions». Рекомендуется использовать класс `FlyHelpMenu` (только при условии правильного использования `flyInit()`).

5.6.7. Сохранение настроек

Каждое приложение, завершая свою работу, должно сохранять набор параметров. Как минимум, приложение должно сохранить свои размеры (высоту и ширину, установленные пользователем), пропорции разделителей (splitter) и колонок (если есть), размеры важных диалогов, положение плавающих панелей инструментов (toolbars) и т. п. В зависимости от специфики решаемой задачи, может потребоваться сохранение каких-либо дополнительных параметров. Например, рекомендуется сохранять/восстанавливать такое состояние окна,

как «maximized» и/или «minimized», но лучше все состояние окна в целом — `windowState` (см. `QWidget` в Qt-документации). Единственный параметр, который приложению нельзя сохранять при завершении, а потом восстанавливать при запуске, — позиция окна на экране, исключение — какие-либо специальные диалоги и сообщения, требующие немедленного внимания оператора.

Для сохранения/восстановления настроек следует использовать возможности класса `QSettings`, который берет на себя задачи определения места расположения конфигурационного файла, сохранения параметров в виде определенной структуры и извлечения данных параметров. При этом следует:

– полностью полагаться на библиотеку Qt в вопросе выбора места сохранения;

– в качестве `key` в `QSettings::value/setValue (key, value)` использовать `[/programName[/sectionName]]/parameterName`. Тогда автоматически в нужном месте библиотекой Qt будет создан (открыт) конфигурационный файл с секцией `sectionName` и параметром `parameterName = value`. При этом, если использовалась функция `flyInit()`, то в качестве имени параметра следует использовать только `parameterName`, т. к. префикс будет автоматически сформирован с учетом имени приложения и названия организации.

Начиная с версии ОС ОН Орёл 2.11 рабочий стол Fly имеет несколько режимов работы (обычный рабочий стол, планшетный, мобильный, безопасный), выбираемых оператором в одном из меню графического входа. Соответственно, некоторым программмам может понадобиться сохранять разные настройки для разных режимов. Имя режима можно определить по переменной `$DESKTOP_SESSION`, и использовать его для сохранения настроек в разных секциях `QSettings`.

5.6.8. Использование тем иконок

Каждое приложение должно использовать тему иконок для общеупотребительных иконок. Для этого в классе `QIcon` есть статический метод `fromTheme`. Также можно воспользоваться методами класса `FlyIcon`: `fromTheme` и `withEmblems`. В отличие от `QIcon::fromTheme`, `FlyIcon::fromTheme` позволяет задавать эмблемы, которые должны быть наложены на иконку. В большинстве ситуаций использование `QIcon::fromTheme` будет наилучшим выбором. Следует отметить, что объекты `QIcon`, возвращаемые этими вызовами, подгружают изображения иконок определенного размера только по мере необходимости.

Пример

```
int main (int argc,char **argv)
{
    QApplication app(argc, argv);
```

```
...
    app.setWindowIcon(QIcon::fromTheme("accessories-text-editor"));
...
}
```

Следует обратить внимание, что нужно задавать только имя иконки без расширения. При неудачном извлечении иконки из темы можно использовать встроенную иконку. Методы `QIcon::fromTheme` и `FlyIcon::fromTheme` имеют для этого параметр `fallback`. Уникальные иконки, которым нет аналогов в теме, могут быть «вшиты» в приложение. Начиная с версии 1.3 в операционной системе используется новая тема `fly-astra`, которая соответсвует как Icon Naming Specification [7], так и некоторым особенностям KDE4. Теперь использовать имена иконок, принятые в KDE3, уже нельзя.

Начиная с версии ОС ОН Орёл 2.12 в соответствии с общим «плоским» дизайном появилась «плоская» тема `fly-astra-flat`, наследующая тему `fly-astra`.

Если возникает редкая необходимость получать пути к файлам иконок, то можно воспользоваться классом `FlyIconTheme` из `libflycore`, который позволяет

- открывать существующие темы иконок;
- создавать новые темы;
- редактировать темы;
- сохранять изменения;
- получать полный путь к иконке по ее короткому имени, размеру и контексту;
- получать полный путь к иконке по названию MIME-типа;
- ряд других возможностей.

Класс `FlyIconTheme` полностью реализует стандарт «Icon Theme Specification» от X Desktop Group, включая механизмы наследования [3].

Для единообразия внешнего вида можно использовать иконки `media-playback-start` и `media-playback-stop` (контекст `Actions`) для запуска и остановки сервисов и служб. Для перемещения элементов между списками подходят иконки `1leftarrow` и `1rightarrow` (контекст `Actions`). В самих списках не следует безосновательно отключать режим множественного выделения.

5.6.9. MIME-типы

В библиотеку `libflycore` включена часть, реализующая API для детектирования MIME-типа любого файла (реализация соответствующего стандарта freedesktop.org [4]). Qt-приложения должны использовать класс `QMimeDatabase`.

5.6.10. Использование звуковой темы

Начиная с версий 2.1.0 пакетов fly-wm, fly-admin-wm, flycore, flyui, fly-data в рабочем столе Fly внедрена поддержка спецификаций для звуковых тем и наименований звуков [10].

Каждое приложение, желающее пользоваться звуками из этой спецификации, должно использовать текущую звуковую тему, выбранную пользователем в настройках рабочего стола Fly. Для этого предназначен класс `FlyIconTheme`, позволяющий:

– открывать существующие темы звуков;

– создавать новые темы;

– редактировать темы;

– сохранять изменения;

– получать полный путь к файлу звука по его короткому имени, локали и профилю;

– ряд других возможностей.

Класс практически полностью реализует стандарты Sound Theme and Naming Specifications [10] от X Desktop Group (freedesktop.org), включая механизмы наследования.

Из текущей для рабочего стола звуковой темы по имени (обязательно), локали (не обязательно) и профилю (не обязательно, но обычно это «stereo») можно получить полный путь к требующемуся звуку, например, следующим образом:

Пример

```
 ...
#include <fly/flysoundtheme.h>
 ...
int main (int argc,char **argv)
{
 ...
char soundPath[PATH_MAX];

//получить полный путь в soundPath по имени ("dialog-warning")
//по локали (пусто "" - необязательный аргумент) и
//профилю ("stereo" - обычно стерео, но необязательный аргумент)

if ( FlySoundTheme::getSound("dialog-warning","", "stereo",soundPath) );
else {//не найден
 ...

}
```

При этом класс сам подберет расширение для звука (обычно «.wav»).

При неудачном извлечении звука из темы должен использоваться либо встроенный звук, либо ничего. Уникальные звуки, которым нет аналогов в спецификации [10] должны поставляться вместе с приложением.

Для удобства программиста для некоторых функций flycore есть соответствующие Qt-функции (wrappers) во `libflyui`, например, `QString flySound(...)`.

Спецификации [10] и их поддержка во `flycore/flyui` (см. flysoundtheme.h и flyui.h) дают возможность получать путь к звуковому файлу. Его дальнейшее использование (проигрывание) полностью лежит на самом приложении.

5.7. Пример простейшего приложения с использованием flybuild

Рассмотрим пример простейшего приложения fly-demo, выводящего при запуске на экран окно с сообщением о наличии или отсутствии прав привилегированого пользователя.

```
#include <QtSingleApplication>
#include <QMessageBox>

#include <flyintegration.h>
#include <unistd.h>

#ifndef SOURCE_VERSION
#define SOURCE_VERSION "2.0.0"
#endif

int main(int argc, char *argv[])
{
QtSingleApplication app(argc, argv);

flyInit(SOURCE_VERSION, QT_TRANSLATE_NOOP("Fly", "Simple sample Fly"));

if (geteuid() != 0)
QMessageBox::critical(nullptr, QObject::tr("Unprivileged user"),
QObject::tr("You run the program as an unprivileged user."
));
else
QMessageBox::critical(nullptr, QObject::tr("Privileged user"),
QObject::tr("You run the program as an privileged user."
));

return 0;
}
```

5.7.1. Перевод на русский язык

При создании приложения строки текста, подлежащего переводу на русский язык должны быть отмечены вызовами `tr()`:

```
QObject::tr("This text is subject to translate"
```

Для создания исходного файла для русского перевода следует воспользоваться командой `lupdate -verbose fly-demo.pro`. В результате работы этой команды будет создан файл `fly-demo_ru.ts`, содержащий строки, отмеченные в исходном коде как подлежащие переводу.

Собственно перевод осуществляется с помощью графического иструмента `linguist`:

```
linguist fly-demo_ru.ts.
```

При этом каждой строке из исходного файла `fly-demo_ru.ts` ставится в соответствие русский перевод, который и будет использован при работе пакета.

5.8. Взаимодействие с рабочим столом

5.8.1. Иконки и заголовки окон

Qt дает достаточно функций для установки заголовков, иконок и подписей к иконкам для высокоуровневых окон приложений. Следует обратить внимание на обязательные к использованию методы класса `QWidget`: `setWindowTitle` и `setWindowIcon`. При этом следует придерживаться следующих рекомендаций. Заголовок окна формируется в таком порядке:

```
[имя открытого файла - ] имя приложения
```

где `имя открытого файла` соответствует имени файла, как правило, без полного пути, открытого в активном окне приложения;

`имя приложения`, как правило, соответствует его названию в меню «Пуск» (панели управления) или детализирует его.

Подпись к иконке можно задавать с помощью `setWindowIconText`. Она, как правило, соответствует заголовку окна или может быть короче его в связи с тем, что пространство для нее крайне ограничено. Например, в подписи к иконке может отсутствовать имя файла, если приложение изменяет только один файл, например, содержащий тему рабочего стола или «горячие» клавиши. В принципе, допускается, чтобы заголовок окна и (или) подпись к иконке содержали только имя приложения.

Иконку по умолчанию для всех окон приложения можно задать с помощью метода `QApplication::setWindowIcon`.

Заметим, что вызов `setWindowIcon(const QIcon & icon)` принимает `QIcon` как аргумент.

5.8.2. Системный трей

Любая программа рабочего стола, призванная постоянно находиться в работе, для экономии экранного пространства должна предоставлять пользователю возможность своего сворачивания в трей и восстановления из него. Самый простой путь для программной реализации этого — использовать Qt-класс `QSystemTrayIcon`, реализующий стандарты [7], [8]. При установке иконки для `QSystemTrayIcon` рекомендуется использовать `QIcon`, которая может выдавать изображения всех типичных размеров (см., например, `QIcon::fromTheme`).

Начиная с версии ОС ОН Орёл 2.12 появилась возможность использовать для панели задач (трея и тулбара) монохромные темы иконок `fly-astra-flat-white` и `fly-astra-flat-black`. Приложение может разместить свои чёрные и белые иконки, предназначенные для трея, в эти две темы, и переключение будет происходить автоматически при смене цвета панели задач.

5.8.3. Автозапуск

Если приложение требуется запускать сразу после запуска оконного менеджера, его можно поместить в категорию «Автозапуск» с помощью функции библиотеки `libflycore` (см. `flydeapi.h`):

```
bool flyAutostartEnabled(const char *appname);
```

проверяет, есть ли в автозапуске (как в системном, так и в пользовательском) ярлык с именем `appname.desktop` и является ли он корректным.

```
bool enableAutostart(const char *appname, bool enable,
            char *newExec=NULL, Display *dpy=NULL);
```

устанавливает в пользовательский автозапуск ярлык `appname.desktop` (можно указать полный путь к файлу типа `desktop`), если `enable=true`, или удаляет, если `enable=false`. При этом, в копии desktop-файла, которая попадет в автозапуск, можно скорректировать строку запуска `Exec`, задав `newExec`. Это может быть полезно, если программе надо передать дополнительный аргумент, свидетельствующий, например, что ее запускают именно из автозапуска, а не из меню и т. п.

Если в системном автозапуске есть `appname.desktop`, а вызывается `flyEnableAutostart(appname,false)`, то в пользовательском автозапуске будет создан фиктивный `appname.desktop` с полем `Hidden=true`, что согласно стандарту [11] является пользовательским методом отключения системного автозапуска.

5.8.4. Меню «Пуск» и рабочий стол

Меню «Пуск» (стартовое меню) создается для каждого пользователя при первом запуске им рабочего стола Fly. В первый раз меню создается автоматически на основе файлов `/usr/share/applications/*.desktop` с `Type=Applications`. Иконки, имена, параметры запуска и т. п. берутся оттуда же (см. «Desktop Entry Standard» от freedesktop.org [2]).

Стартовое меню имеет несколько уровней. Первый уровень обычно формируется на основе `/usr/share/fly-wm/startmenu` и выглядит так:

«Офис»

«Сеть»

«Графика»

«Мультимедиа»

«Научные»

«Игры»

«Мобильные»

«Разработка»

«Утилиты»

«Системные»

–разделитель–

«Последние»

«Панель управления»

«Менеджер файлов»

–разделитель–

«Завершение работы...»

В панель управления попадают только приложения, имеющие определенные поля в *.desktop-файле (см. 5.5.2). При этом, если задано поле `X-Fly-AdminOnly=true`, то соответствующий пункт будет виден только пользователю `root`. Последнее правило распространяется не только на «Панель управления», но и на все пункты меню.

Начиная с версии ОС ОН Орёл 2.12 добавлена новая категория Мобильные, предназначенная для программ, сделанных специально под тачскрины, т.е., под управление пальцами.

Формирование указанных выше подменю происходит автоматически на основе `Categories`, определенных в документе «Desktop Menu Specification» от freedesktop.org [9].

Подменю и категории, по которым приложения в них попадают:

– «Офис» — Office, Spreadsheet,WordProcessor, Presentation, Calendar, Email;

– «Сеть» — Network, Internet, Email;

– «Графика» — Graphics, VectorGraphics, RasterGraphics, Screensaver;

– «Мультимедиа» — AudioVideo, Multimedia;

– «Научные» — Education, Science, Math, Astronomy, Physics, Chemistry;

– «Игры» — Game, *Game;

– «Мобильные» — Mobile;

– «Разработка» — Development, GUIDesigner, IDE, TextEditor;

– «Утилиты» — System, PackageManager;

– «Системные» — SystemSetup, Settings, AdvancedSettings, Accessibility;

– «Прочие» — все, что не удалось разместить в других меню.

Вышеперечисленные категории представлены в порядке своего приоритета. Так, приложение, содержащее ключевое слово «Email», скорее всего, попадет в подменю «Сеть», чем в «Работа с файлами», так как в «Сеть» оно имеет позицию 3, а в «Работа с файлами» — 5. С другой стороны, одно и тоже приложение может задавать в своем *.desktop-файле несколько категорий в порядке приоритета. Тогда приложение со списком категорий «Network»; «Office» попадет в меню «Сеть».

Если у приложения задана только категория «Application», то оно попадет сразу в подменю «Программы», если такое подменю есть. Если указана специальная категория `Core`, то приложение попадет на нулевой уровень, т. е. прямо в меню «Пуск». При категории `None` приложение никуда не попадет. Категории можно присваивать и каталогам (см. 5.5.2). В этом случае каталог помещается сразу целиком в соответствии со своими категориями так же, как это делается для обычных программ. Таким образом, разработчик стороннего приложения может разместить информацию для запуска как отдельным ярлыком, так и целым каталогом в любом из подменю меню «Пуск» без каких-либо ограничений.

Сформированное меню на диске располагается в `$HOME/.fly/startmenu` и является иерархией каталогов (им соответствуют подменю) и *.desktop-файлов (им соответствуют пункты меню). Есть и специальная разновидность *.desktop-файлов — .directory-файлы, имеющие сходную структуру и задающие внешние имена, иконки категорий для самих каталогов (подменю), а также порядок сортировки при показе содержимого каталогов (см. поле `SortOrder` в файле `.directory`).

При старте рабочего стола персональное меню (`$HOME/.fly/startmenu`) и меню для всех пользователей (`/usr/share/fly-wm/startmenu`) сливается воедино в `$HOME/.fly/startmenu`. Меню как иерархия в ФС сделано вопреки документу «Desktop Menu Specification» от freedesktop.org (там рекомендуется XML-файл [9]), чтобы облегчить ручную правку меню, как это сделано в MS Windows. С другой стороны, Fly предоставляет пользователю мощную программу `fly-menuedit` для просмотра, редактирования, резервного копирования и автоматического создания не только меню «Пуск», но и вообще любых иерархических совокупностей каталогов и (или) файлов типа `*.desktop`.

Любое приложение, желающее быть показанным в меню «Пуск» для каждого пользователя, должно в первую очередь разместить в `/usr/share/applications/` свой *.desktop-файл или каталог с такими или иными файлами. Тогда при следующем запуске `fly-wm` или создании меню «с нуля» в `fly-menuedit` приложение автоматически будет включено в меню в соответствии со своей категорией из *.desktop-файла или .directory-

каталога. Однако, если приложение хочет попасть в стартовое меню только одного пользователя, то оно может скопировать свой `*.desktop` или каталог непосредственно в один из подкаталогов `$HOME/.fly/startmenu`. При этом весьма полезная информация не будет доступна никому другому.

Иконки на рабочем столе по сути представляют из себя одноуровневое меню в каталогах `$HOME/Desktop` (персональный набор) и `/usr/share/fly-wm/desktop` (набор для всех пользователей). Все сказанное выше для стартового меню справедливо и для набора иконок на рабочем столе.

При каждом запуске менеджер окон `fly-wm` контролирует наличие обновлений в `/usr/share/applications`. И если там объявилось новое приложение, то его ярлык попадает в меню «Пуск» каждого пользователя (при запуске `fly-wm`).

В дальнейшем пользователь может удалить этот ярлык.

`Fly-wm` контролирует наличие обновлений в `/usr/share/fly-wm/desktop` (общие ярлыки рабочего стола для всех пользователей).

Справа от кнопки **[Пуск]** на панели задач располагается панель с кнопками для быстрого запуска наиболее употребительных приложений. Соответствующие ярлыки располагаются в `$HOME/.fly/toolbar`, вызывать этот каталог следует с помощью:

```
GetFlyDir(USER_TOOLBAR_DIR)
```

Начиная с версии ОС ОН Орёл 2.12 в desktop-файлах можно определять их участие или неучастие в определённых типах сессий (см. `$DESKTOP_SESSION`) путем указания имён сессий в полях `NotShowIn` и `OnlyShowIn`.

5.8.5. Корзина

«Корзина» представляет собой каталог, который индивидуален для каждого пользователя. Он предназначен для временного хранения удаленных пользователем документов с возможностью их восстановления.

Путь к корзине программно можно получить с помощью:

```
GetFlyDir(USER_TRASH_DIR)
```

из библиотеки `libflycore` и обычно он равен `$HOME/.local/share/Trash`.

Оконный менеджер `fly-wm`, файловый менеджер `fly-fm` и другие программы рабочего стола позволяют выполнять файловые операции. Они доступны как через меню, так и с помощью перетаскивания мышью (drag-n-drop). Одной из таких операций является удаление файлов или папок в «Корзину» и их восстановление. Все операции с корзиной (перемещение в нее, восстановление из нее, получение сведений о размещенных там файлах и каталогах) выполняются в соответствии со стандартом от freedesktop.org. При этом не будет иметь значения, где физически располагается корзина и ее реестр (на самом деле это два подкаталога в каталоге `$HOME/.local/share/Trash`), если разработчик использует API корзины, предоставленный классом `FlyTrash` библиотеки `libflycore`:

```
static std::string registerFile(const char *srcPath);
static bool unregisterFile(const char *fileName);
static std::string getFilePathToRestore(const char *fileNameOrPath);

static bool isFileInTrash(const char * path);

static bool cleanRegistry();
static bool cleanFiles();
static bool cleanAll();

static bool isEmpty();

static bool moveToTrash(const char *filePath);
static bool restoreFromTrash(const char *fileNameOrPath);
```

Необходимо отметить, что API не предназначен непосредственно для перемещения/восстановления файлов, а только для получения необходимых для этого путей и регистрации/удаления в реестре корзины. Однако две последние из приведенных выше функций упрощают задачу для небольших файлов. Они применимы только, если работа выполняется в рамках одной ФС (одного раздела диска).

5.8.6. Перетаскивание объектов на рабочем столе

Рабочий стол Fly в прямом смысле — это корневое окно с иконками на нем. Иконки — это либо ярлыки (desktop entry), либо собственно файлы или каталоги. Операция drag-n-drop (`dnd` — перетаскивание мышью) возможна как на сам рабочий стол, так и на иконки на нем. Реализация `dnd` в `fly-wm` и приложениях Qt соответствует стандарту [5].

Иконка рабочего стола может принимать `dnd`, если она представляет собой:

– собственно каталог или ярлык каталога (например, `myhome.desktop`, `mydoc.desktop` и ряд других специальных ярлыков, в том числе `mytrash.desktop` — ярлык корзины);

– ярлык типа `Application`, если в его поле `Exec=` есть один из аргументов: `%F` — список файлов, `%f` — файл, `%u` — URL, `%U` — список URL.

Для поддержки печати с помощью операции `dnd` на рабочем столе приложение, предназначенное для вывода документов на печать, должно:

– принимать аргумент командной строки `--print %F` или `%f`, `%u`, `%U`, при этом передавая файлы на печать;

– описать свою возможность печатать в *.desktop-файле как `Action` с именем «Print» и командной строкой, содержащей `--print %F` или `%f`, `%u`, `%U`.

При перетаскивании мышью пунктов меню «Приложения» или «Панель управления» на рабочий стол на нем создаются соответствующие копии.

5.8.7. Подсистема помощи

Подсистема помощи Fly состоит из:

– программы показа;

– хранилища файлов;

– клиентов.

Основная программа показа помощи — Qt Assistant. Структура хранилища и формат файлов помощи определяются пакетом `fly-doc`. Клиенты программы показа помощи (все приложения Fly) вызывают несколько простых функций (см. в библиотеке `libflyui` класс `FlyHelpMenu` и функции `flyHelpInstall()`, `flyHelpShow()` и т. п.) и не должны заботиться о месте расположения файлов помощи и способах их показа.

5.8.8. Синхронизация с менеджером окон fly-wm

В связи с тем, что любая программа может перемещать файлы в/из рабочего стола (`$HOME/Desktop`), корзины (`$HOME/.local/share/Trash`), стартового меню (`$HOME/.fly/startmenu`), возникает проблема синхронизации содержимого данных каталогов с оконным менеджером `fly-wm`. Требуется сообщать `fly-wm` о том, что произошли изменения и ему требуется перечитать эти каталоги.

Каждая программа, перемещающая файлы в/из каталогов:

– рабочего стола (по `GetFlyDir(USER_DESKTOP_DIR)`);

– корзины (по `GetFlyDir(USER_TRASH_DIR)`);

– стартового меню (по `GetFlyDir(USER_START_MENU_DIR)`).

должна, по окончании операции, уведомить `fly-wm` следующим способом:

– `SendCommandToWM("FLYWM_UPDATE_SHORTCUT\n");`

– `SendCommandToWM("FLYWM_UPDATE_TRASH\n");`

– `SendCommandToWM("FLYWM_UPDATE_STARTMENU\n").`

соответственно.

В этом случае `fly-wm` обновит рабочий стол, корзину, стартовое меню, соответственно. Функция `SendCommandToWM(...)` содержится в библиотеке `libflycore`.

Начиная с версии ОС ОН Орёл 2.11 эти уведомления посылать не обязательно, т.к., используя механизм inotofy, fly-fm сам «понимает» момент, когда необходимо сделать обновление.

Используя тот же механизм, можно отправлять различные команды оконному менеджеру. Например, если некоторому приложению требуется, чтобы были закрыты все текущие приложения и графическая система была перезагружена, оно может использовать функцию

SendCommandToWM из libflycore с параметром FLYWM_RESTART. Практически полный список команд можно увидеть в приложении fly-admin-hotkeys (редактор «горячих» клавиш), большинство из них можно посылать в fly-wm из других приложений.

Полный список команд, принимаемых менеджером окон, можно получить в файле .xsession-errors после выполнения консольной команды fly-wmfunc FLYWM_FUNC_LIST.

5.8.9. Полноэкранный режим

Данная функция может присутствовать как пункт «Полноэкранный режим» в меню «Окна» (для MDI-интерфейсов) или «Вид» (для SDI). Она выполняется с помощью Qt-вызовов showFullScreen() и showNormal().

5.8.10. Смена раскладки клавиатуры

Сменить раскладку клавиатуры можно одной строкой из программы:

```
SendCommandToWM(QX11Info::display(),"FLYWM_ALT_KB\n"
SendCommandToWM(QX11Info::display(),
"FLYWM_BASE_KB\n"
```

или из командной строки:

```
> fly-wmfunc FLYWM_ALT_KB или FLYWM_BASE_KB
```

Аналогично для переключения дополнительной секции клавиатуры на ввод цифр (NumLock) и наоборот можно использовать команды: FLYWM_NUMLOCK_ON, FLYWM_NUMLOCK_OFF, FLYWM_NUMLOCK_TOGGLE.

5.8.11. Дополнительные панели

Fly-wm предоставляет одну панель задач. Окна при максимизации не накрывают ее, иконки рабочего стола не попадают под нее. Таким образом, панель задач образует некую специальную зону экрана. В [6] описаны такие свойства окон приложений: _NET_WM_STRUT и _NET_WM_STRUT_PARTIAL, позволяющие приложению резервировать области экрана подобно панели задач. Fly-wm поддерживает эту возможность. Однако в Qt нет простой поддержки данной возможности. Рекомендуется использовать такой фрагмент:

```
int v[4];
v[0] = w.x(); v[1] = w.x()+w.frameGeometry().width();
v[2] = w.y(); v[3] = w.y()+w.frameGeometry().height();
Atom a=XInternAtom(QX11Info::display(),_NET_WM_STRUT",false);
if (a!=None) XChangeProperty(QX11Info::display(),w.winId(), a,
XA_CARDINAL,32,PropModeReplace, (unsigned char *)&v,4);
```

где w — главное окно приложения.

Через параметр Qt::WindowFlags при создании виджета можно задать его внешний вид (наличие границы, заголовка и т. п.). Однако, есть более высокоуровневый и переносимый способ задания STRUT - использовать функцию KWindowSystem::setStrut() из

библиотеки libKF5WindowSystem.

Начиная с версии ОС ОН Орёл 2.12 можно задать область экрана, которую займёт рабочий стол, с помощью параметров FlyDesktopX, FlyDesktopY, FlyDesktopWidth, FlyDesktopHeight в файлах `$HOME/.fly/theme/*themerc*`.

5.9. Плагины для менеджера файлов Fly-fm

В файловом менеджере реализована возможность расширения его функционала с помощью так называемых «плагинов» (см. электронную справку по файловому менеджеру `fly-fm`).

5.10. Настройки планшетного режима

В версии ОС ОН Орёл 2.12 имеется возможность автоматического конфигурирования графического интерфейса пользователя для работы на мобильных устройствах с сенсорными экранами — т.е. в так называемых «планшетной» и «мобильной» типах сессии. Тип сессии выбирается пользователем при входе в систему, см. меню "Тип сессии"графического логина. При входе в систему будет применен комплект конфигурационных файлов соответствующий типу выбранной сессии и выставлена переменная `\$DESKTOP_SESSION`.

5.11. Приложения и права администратора

Большинство утилит настройки требуют привилегий суперпользователя. Есть ряд способов предоставления привилегий: от `sudo` и членства в группах до PolicyKit. Решение этой задачи возможно с использованием программы `fly-su`. Программа, предполагающая действия администратора, обязательно должна информировать пользователя о невозможности выполнения каких-либо функций без соответствующей авторизации и, по возможности, давать пользователю способ выполнить такую авторизацию, например, с помощью рекомендуемой `fly-su`.

Начиная с версии ОС ОН Орёл 2.12 в системе может быть ограничено использование консоли. В таком случае пользователи, не входящие в группы astra-admin или astra-console, не увидят desktop-файлов, в которых используются терминалы. Пользователи, не входящие в группу astra-admin, не увидят desktop-файлов, в которых есть не только категория `ROOT_ONLY`, но и команды `su-to-root`, `fly-su`.

6. РАЗРАБОТКА ПО ДЛЯ ВЗАИМОДЕЙСТВИЯ С СУБД POSTGRESQL

СУБД PosgreSQL предоставляет доступные в дистрибутиве ОС ОН программные интерфейсы, предназначенные как для разработки клиентского ПО, реализующего функции по вводу, формированию запросов и отображению данных, так и программные интерфейсы, предназначенные для расширения функциональности сервера.

6.1. Клиентские программные интерфейсы

Существует набор клиентских программных интерфейсов, многие из которых поставляются отдельно и имеют свою собственную документацию. В официальной документации на СУБД PostgreSQL версии 9.6, доступной по адресу `https://www.postgresql.org/docs/9.6/static/external-projects.html` , приведен список наиболее популярных из них.

В дистрибутив ОС ОН включаются два клиентских интерфейса:

– библиотека `libpq` — как первичный интерфейс языка C , используемый многими другими клиентскими интерфейсами;

– библиотека `ecpg` — как зависимая от грамматики SQL на стороне сервера и, следовательно, чувствительная к изменениям в самой СУБД PostgreSQL.

Все прочие языковые интерфейсы являются внешними проектами , и поставляются отдельно. В таблице 1 приведен ряд из этих проектов.

Таблица 1 – Клиентские интерфейсы СУБД PostgreSQL, сопровождаемые отдельно

Наименование	Язык	Комментарии	Web-сайт
`DBD::Pg`	`Perl`	Драйвер DBI для Perl	`http://search.cpan.org/dist/DBD-Pg/`
`JDBC`	`Perl`	Драйвер JDBC 4-го типа	`http://jdbc.postgresql.org/`
`psqlODBC`	`Perl`	Наиболее популярный ODBC-драйвер	`https://odbc.postgresql.org/`
`psycopg2`	`Perl`	DB API совместимый с версией 2.0	`http://initd.org/psycopg/`

Следует отметить, что некоторые пакеты могут выпускаться под лицензией, отличной от лицензии PostgreSQL. Таким образом, при разработке клиентских приложений для СУБД PostgreSQL с использованием некоторого языкового интерфейса необходимо руководствоваться лицензионными соглашениями и документацией, приведенными на соответствующих web-сайтах проектов.

Библиотека `libpq` предоставляет прикладной программный интерфейс на языке C к СУБД PostgreSQL. В ней реализован набор функций, позволяющих клиентским программам передавать запросы серверу СУБД PostgreSQL и получать резуль-

таты обработки данных запросов. При разработке клиентских приложений для СУБД PostgreSQL с использованием библиотеки `libpq` необходимо руководствоваться соответствующим разделом официальной документации, который доступен по ссылке `http://www.postgresql.org/docs/9.6/static/libpq.html`.

Кроме того, библиотека `libpq` является основой для ряда прочих прикладных программных интерфейсов, включая предназначенные для разработки приложений на языках C++, Perl, Python, Tcl и ECPG. Таким образом, некоторые аспекты поведения `libpq` будут иметь значение при использовании для разработки ПО одного из указанных языковых пакетов. В разделах, посвященных использованию в библиотеке `libpq` переменных окружения (см. `http://www.postgresql.org/docs/9.6/static/libpq-envars.html`), файла паролей (`http://www.postgresql.org/docs/9.6/static/libpq-pgpass.html`) и поддержке SSL (см. `http://www.postgresql.org/docs/9.6/static/libpq-ssl.html`), описано поведение, видимое пользователем любого приложения, использующего библиотеку `libpq`.

Несколько коротких примеров написания программ, использующих библиотеку `libpq`, приведено в документации на СУБД PostgreSQL (`http://www.postgresql.org/docs/9.6/static/libpq-example.html`). Ряд законченных примеров приложений, использующих библиотеку `libpq`, находится в каталоге `src/test/examples` в дистрибутиве ОС ОН с исходным кодом СУБД PostgreSQL.

Клиентские программы, которые используют библиотеку `libpq`, должны подключать заголовочный файл `libpq-fe.h` и компоноваться с библиотекой `libpq`.

6.2. Программирование сервера

Предоставление соответствующих программных интерфейсов предназначено для расширения функциональных возможностей сервера на основе использования определяемых пользователем функций, типов данных, триггеров и т. д. При разработке серверного ПО необходимо руководствоваться соответствующими разделами официальной документации на СУБД PostgreSQL:

– расширение языка SQL (материалы раздела доступны по ссылке `http://www.postgresql.org/docs/9.6/static/extend.html`);

– триггерные функции (материалы раздела доступны по ссылке `http://www.postgresql.org/docs/9.6/static/triggers.html`);

– система правил (материалы раздела доступны по ссылке `http://www.postgresql.org/docs/9.6/static/rules.html`);

– процедурный язык PL/pgSQL (материалы раздела доступны по ссылке `http://www.postgresql.org/docs/9.6/static/plpgsql.html`);

– процедурный язык PL/Tcl (материалы раздела доступны по ссылке http://www.postgresql.org/docs/9.6/static/pltcl.html);

– процедурный язык PL/Perl (материалы раздела доступны по ссылке http://www.postgresql.org/docs/9.6/static/plperl.html);

– процедурный язык PL/Python (материалы раздела доступны по ссылке http://www.postgresql.org/docs/9.6/static/plpython.html);

– интерфейс программирования сервера (материалы раздела доступны по ссылке http://www.postgresql.org/docs/9.6/static/spi.html).

Интерфейс программирования сервера (Server Programming Interface, SPI) предоставляет разработчикам возможность выполнения команд SQL внутри разрабатываемых ими функций. SPI является набором интерфейсных функций для упрощения доступа к синтаксическому анализатору, планировщику и исполнителю запросов. Кроме того, SPI реализует функции по управлению памятью.

Доступные процедурные языки предоставляют различные способы выполнения команд SQL из процедур. Большинство из данных возможностей основано на SPI, следовательно, разработчикам, использующим процедурные языки, рекомендуется ознакомиться с посвященным SPI разделом документации на СУБД PostgreSQL. Во избежание неоднозначности в документации используется термин «функция» применительно к интерфейсным функциям SPI и термин «процедура» применительно к определяемым пользователем функциям на языке C, использующим SPI.

Следует отметить, в случае возникновения ошибки при выполнении команды посредством SPI возврат управления в процедуру пользователя не осуществляется. Будет осуществлен откат транзакции или вложенной транзакции, в которой выполняется пользовательская процедура. Приведенные в документации для большинства функций SPI соглашения о возвращаемых значениях ошибки применяются только для ошибок, возникающих непосредственно внутри функций SPI. Существует возможность возврата управления после возникновения ошибки посредством задания собственного окружения вложенной транзакции при вызовах SPI, которые могут завершаться с ошибкой.

Функции SPI возвращают неотрицательный результат в случае успеха либо посредством возврата значения типа `integer`, либо в глобальной переменной `SPI_result`. В случае возникновения ошибки возвращается отрицательный результат либо `NULL`.

В файлах исходных текстов, в которых используется SPI, должен подключаться заголовочный файл `executor/spi.h`.

Перечень интерфейсных функций SPI приведен в таблице 2.

Таблица 2

Наименование	Назначение
SPI_connect	Подключение процедуры к менеджеру SPI
SPI_finish	Отключение процедуры к менеджеру SPI
SPI_push	Помещение в стек SPI для рекурсивного использования SPI
SPI_pop	Извлечение из стека SPI для выхода из рекурсивного использования SPI
SPI_execute	Выполнение команды
SPI_exec	Выполнение команды чтения или записи
SPI_execute_with_args	Выполнение параметризованной команды
SPI_prepare	Подготовка плана для выполнения команды без выполнения
SPI_prepare_cursor	Подготовка плана для выполнения команды без выполнения с использованием параметра плана опции курсора
SPI_getargcount	Возвращает число аргументов, необходимое для подготовки плана посредством SPI_prepare
SPI_getargtypeid	Возвращает данные типа OID для аргумента плана, подготавливаемого посредством SPI_prepare
SPI_is_cursor_plan	Возвращает ture, если план, подготовленный посредством SPI_prepare, может быть использован с SPI_cursor_open
SPI_execute_plan	Выполнение плана, подготовленного посредством SPI_prepare
SPI_execp	Выполнение плана в режиме «чтение/запись»
SPI_cursor_open	Установить курсор, используя план, созданный с SPI_prepare
SPI_cursor_open_with_args	Установить курсор, используя запрос и параметры
SPI_cursor_find	Найти существующий курсор по имени
SPI_cursor_fetch	Считать несколько строк от курсора
SPI_cursor_move	Передвинуть курсор
SPI_scroll_cursor_fetch	Считать несколько строк от курсора
SPI_scroll_cursor_move	Передвинуть курсор
SPI_cursor_close	Закрыть курсор
SPI_saveplan	Сохранить план

Перечень вспомогательных интерфейсных функций SPI приведен в таблице 3.

Таблица 3

Наименование	Назначение
SPI_fname	Определить наименование столбца для указанного номера столбца
SPI_fnumber	Определить номер столбца для указанного наименования столбца
SPI_getvalue	Возвращает строковое значение для указанного столбца

Окончание таблицы 3

Наименование	Назначение
SPI_getbinval	Возвращает двоичное значение для указанного столбца
SPI_gettype	Возвращает наименование типа данных для указанного столбца
SPI_gettypeid	Возвращает данные типа OID для указанного столбца
SPI_getrelname	Возвращает наименование указанного отношения
SPI_getnspname	Возвращает пространство имен указанного отношения

Перечень функций управления памятью SPI приведен в таблице 4.

Таблица 4

Наименование	Назначение
SPI_palloc	Выделить память в верхнем исполняемом контексте
SPI_repalloc	Перераспределить память в верхнем исполняемом контексте
SPI_pfree	Освободить память в верхнем исполняемом контексте
SPI_copytuple	Создать копию строки в верхнем исполняемом контексте
SPI_returntuple	Подготовить возврат кортежа как элемента данных
SPI_modifytuple	Создать строку, заменяя выбранные поля строки
SPI_freetuple	Освободить строку, выделенную в верхнем исполняемом контексте
SPI_freetuptable	Освободить набор строк, созданных SPI_execute или аналогичной функцией
SPI_freeplan	Освободить ранее сохраненный план

7. РАЗРАБОТКА ПО ДЛЯ ВЗАИМОДЕЙСТВИЯ С WEB-СЕРВЕРОМ APACHE

Web-сервер Apache является сервером, который принимает HTTP-запросы от клиентов, в общем случае являющихся web-браузерами, осуществляет поиск и формирование содержимого ответов , и передает HTTP-ответы клиентам, включая HTML-страницы, изображения, файлы, медиапотоки и другие данные. Для формирования содержимого ответов клиентам могут быть использованы внешние программы, которые для взаимодействия с web-сервером могут использовать следующие интерфейсы:

- Common Gateway Interaface (CGI);
- Server Side Inludes (SSI);
- Fast Common Gateway Interaface (FastCGI).
- Web Server Gateway Interface (WSGI).

7.1. CGI

CGI определяет способ взаимодействия сервера с внешними генерирующими контент программами, называемыми CGI-программы, CGI-сценарии или CGI-скрипты. CGI позволяет создавать динамическое содержимое страниц на web-сервере. Для обеспечения возможности выполнения CGI-программ должны быть подключены модули mod_cgi и mod_alias. При разработке CGI-программ необходимо руководствоваться спецификацией CGI и прочей дополнительной информацией, которые доступны по ссылке http://www.w3.org/CGI/.

7.1.1. Конфигурирование web-сервера Apache

Настроить web-сервер Apache для выполнения CGI-скриптов возможно различными способами:

1) директива ScriptAlias в конфигурационном файле web-сервера Apache определяет, что в указанной директории находятся CGI-скрипты.

Пример

```
ScriptAlias /cgi-bin/ /usr/local/apache2/cgi-bin/
```

Директива ScriptAlias обычно используется для директорий, которые находятся за пределами DocumentRoot. В случае указанного выше примера, если был запрошен URL http://www.example.com/cgi-bin/test.pl web-сервер Apache попытается запустить файл /usr/local/apache2/cgi-bin/test.pl и отобразить результат его выполнения. Указанный файл должен существовать, быть исполняемым и возвращать данные конкретным образом. В противном случае будет выдано сообщение об ошибке;

2) можно явно использовать директиву Options в конфигурационном файле web-

сервера Apache , чтобы разрешить выполнение скриптов в конкретной директории.

Пример

```
<Directory /usr/local/apache2/htdocs/somedir>
Options +ExecCGI
</Directory>
```

Кроме того, следует указать, какие файлы являются CGI-скриптами, используя директиву AddHandler, которая описывает расширения файлов, обрабатываемых как CGI-скрипты.

Пример

```
AddHandler cgi-script .cgi .pl
```

3) в случае отсутствия доступа к основным конфигурационным файлам web-сервера Apache возможно обеспечить выполнение CGI-скриптов с помощью файлов .htaccess, использование которых должно быть разрешено администратором web-сервера Apache. Наличие файла .htaccess в директории позволяет производить конфигурационные изменения для этой директории и ее поддиректорий. Имя конфигурационных файлов .htaccess может отличаться от .htaccess и определяется администратором сервера в основных конфигурационных файлах следующим образом:

```
AccessFileName .configfilename
```

Файлы .htaccess используют тот же самый синтаксис, что и основные файлы конфигурации, но могут содержать только те директивы, которые разрешены администратором в директиве AllowOverride. Наличие разрешенных директив в файле .htaccess равносильно наличию этих директив для директории, в которой находится файл .htaccess, в основных файлах конфигурации:

```
Содержимое файла htaccess в директории  /www/htdocs/example
AddType text/example .exm

Раздел httpd.conf файла
<Directory /www/htdocs/example>
AddType text/example .exm
</Directory>
```

Использование файлов .htaccess может быть запрещено следующим образом:

```
AllowOverride None
```

При запросе файла сервер осуществляет поиск и обработку всех файлов .htaccess, начиная от корня и до директории, в которой находится запрашиваемый файл. Директивы применяются по мере их нахождения, следовательно,

директивы из поддиректорий могут замещать значения из предшествующих файлов. Список директив для конфигурирования web-сервера Apache и описание, какие из них могут быть доступны из файлов .htaccess, доступен по ссылке http://httpd.apache.org/docs/2.4/mod/core.html.

Описание конфигурирования web-сервера Apache для использования CGI-скриптов доступно по ссылке http://httpd.apache.org/docs/2.4/howto/cgi.html.

7.1.2. Разработка CGI-скриптов

Существует два основных отличия CGI-программы от обычной. Во-первых, возвращаемые данные должны начинаться HTTP-заголовком, в котором обязательно должен быть указан тип возвращаемого содержимого и, возможно, другие дополнительные поля. Простейший заголовок для HTML-документа выглядит следующим образом:

```
Content-type: text/htm
```

Во-вторых, возвращаемые данные должны быть в HTML-формате или каком-либо другом формате, который браузер сможет отобразить.

За исключением двух приведенных различий, создание CGI-скриптов не отличается от разработки других программ. Далее приведен простейший пример CGI-скрипта:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "Hello, World.";
```

Для его выполнения необходимо сохранить его в файл с расширением .pl и поместить в директорию, для которой разрешено выполнение CGI-скриптов одним из описанных выше способов. Первая строка приведенного в примере CGI-скрипта указывает программу, используемую для интерпретации скрипта. Вторая строка выводит определение типа содержимого. В конце заголовка добавляется пустая строка для указания конца HTTP-заголовка и начала данных. Третья строка выводит "Hello, World.".

При выполнении запуска CGI-скрипта сервер выставляет переменные окружения, которые доступны из скрипта. Полный список переменных, обязательных и опциональных, доступен по ссылке https://www.ietf.org/rfc/rfc3875. Кроме того, в зависимости от настроек сервера, могут устанавливаться дополнительные переменные окружения. Подробнее от этом можно посмотреть в документации на web-сервер Apache по ссылке http://httpd.apache.org/docs/2.4/env.html.

Другим способом взаимодействия web-сервера Apache и CGI-скриптов является использование стандартных потоков ввода (STDIN) и вывода (STDOUT). При передаче web-формы CGI-скрипту данные из формы в специальном формате передаются CGI-программе через стандартный поток ввода. Данный формат определяет, что имя поля и его значение объединяются посредством знака равенство (=), пары значений объединяются амперсандом

(&). Для записи ряда символов, таких как пробел, амперсанд, знак равенства используется их шестнадцатеричное представление. Далее приведен пример записи данных.

Пример

```
fieldname1=value1& fieldname2=some%20value&fieldname3=value3
```

Сервер помещает данные в переменную окружения QUERY_STRING. Указанный метод используется при обработке запроса GET. CGI-программа обеспечивает разбор строки и извлечение необходимой информации. Кроме того, в большинстве языков программирования существуют библиотеки и модули, которые обеспечивают разбор данных из запросов и другие аспекты, связанные с использованием CGI.

7.1.3. Типовые ошибки при разработке CGI-скриптов

При разработке CGI-скриптов могут возникнуть следующие ошибки:

1) сообщение, начинающееся с Forbidden, означает, что возникли проблемы с правами доступа. Следует помнить, что web-сервер Apache выполняется от имени специальной учетной записи пользователя. Следовательно, необходимо установить соответствующие права на файлы, содержащие CGI-скрипты. В общем случае устанавливаются следующие права:

```
chmod 0755 first.pl
```

2) выводится сообщение POST Method Not Allowed или исходный код скрипта. Данное сообщение означает, что неправильно сконфигурирован сервер для выполнения CGI-скриптов;

3) выводится сообщение Internal Server Error. В данном случае в журнале ошибок web-сервера Apache возможно появление сообщения Premature end of script headers, которое может содержать дополнительное сообщение об ошибке, генерируемой CGI-программой. В подобном случае необходимо установить причину, по которой CGI-скрипт не может создать правильный HTTP-заголовок. Причиной могут быть неверные права доступа не только для самого файла CGI-скрипта, но и всех прочих файлов, к которым скрипт пытается получить доступ на чтение или запись. В первой строке скрипта должен быть корректно указан полный путь к интерпретатору, выполняющему скрипт, а также все пути к программам, вызываемым из скрипта.

7.2. SSI

SSI — язык для динамического добавления содержимого в существующие HTML-документы. SSI представляет собой набор директив, которые размещаются в HTML-страницах и обрабатываются сервером перед выдачей пользователю. Данные директивы позволяют динамически добавлять содержимое в существующие HTML-страницы без

необходимости полностью генерировать страницу посредством CGI-программы или другую динамическую технологию. SSI рекомендуется использовать для добавления незначительного объема информации, например времени или даты. В случае, если большая часть HTML-страницы генерируется при обработке запроса, следует использовать другие решения. Для работы SSI необходимы модули `mod_includes` и `mod_cgi`.

Рекомендации по использованию SSI приведены в официальной документации на web-сервер `Apache` и доступны по ссылке http://httpd.apache.org/docs/2.4/howto/ssi.html

7.2.1. Конфигурирование web-сервера Apache

В конфигурационных файлах сервера необходимо указать директории, в которых разрешено выполнение директив SSI.

```
<Directory /usr/local/apache2/htdocs/somedir>
Options +Includes
</Directory>
```

Для задания файлов, содержащих директивы SSI можно использовать следующие способы:

1) В конфигурационных файлах указываются расширения, соответствующие файлам с директивами SSI.

Пример

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

Недостаток рассмотренного способа заключается в необходимости изменения в имени существующей HTML-страницы расширения на `.shtml` для добавления SSI директив.

2) Использовать в конфигурационных файлах web-сервера `Apache` директиву `XBitHack`:

```
XBitHack on
```

Использование названной директивы указывает web-серверу `Apache` на необходимость определять наличие директив SSI в файлах, у которых выставлен бит выполнения:

```
chmod +x page.html
```

Не рекомендуется разрешать использование директив SSI для всех файлов `.html`. Подобный подход с высокой вероятностью приведет к потере в производительности вследствие анализа web-сервером `Apache` всех файлов на предмет наличия директив SSI. Использование директивы `XBitHack` позволяет избежать потерь производительности и проблем со сменой расширений файлов.

В случае недоступности основных конфигурационных файлов возможно использование разрешенных администратором конфигурационных файлов `.htaccess` (см. 7.1.1).

7.2.2. Директивы

Директивы SSI имеют следующий синтаксис:

```
<!--#element attribute=value attribute=value ... -->
```

Директивы SSI форматированы подобно комментариям HTML. Таким образом, в случае использования конфигурации web-сервера Apache, не обеспечивающей корректную обработку директив SSI, браузер игнорирует их. При этом в исходном коде страницы директивы SSI будут отображаться. В противном случае директивы SSI будет заменены результатами их выполнения.

Описание возможностей использования директив SSI находится в документации на модуль mod_include и доступно по ссылке http://httpd.apache.org/docs/2.4/mod/mod_include.html.

7.3. FastCGI

FastCGI является дальнейшим развитием CGI и устраняет ряд его недостатков. Проблема CGI в том, что при каждом вызове скрипта запускается выполняющий его интерпретатор. Программы FastCGI являются постоянно запущенными процессами на любом сервере в сети. Общение с web-сервером осуществляется не посредством стандартных потоков ввода/вывода, а через сокеты Unix или сокеты TCP/IP. Кроме того, возможна организация обработки запросов несколькими работающими параллельно FastCGI-процессами.

Существуют два модуля для web-сервера Apache, реализующие поддержку FastCGI: mod_fastcgi и mod_fcgid. Рекомендации по их установке и настройке доступны по ссылке http://httpd.apache.org/mod_fcgid/, соответственно.

7.4. WSGI

WSGI — стандарт взаимодействия между python-программами, выполняющимися на сервере, и веб сервером. Модуль mod_wsgi реализует совместимый с WSGI интерфейс для размещения веб-приложений на базе Python поверх веб-сервера Apache.

Рекомендации по использованию WSGI приведены на github по ссылке: https://github.com/GrahamDumpleton/mod_wsgi/blob/develop/README.rst.

7.4.1. Конфигурирование web-сервера Apache

Для создания типичной конфигурации со сквозной аутентификацией нужно создать файл /etc/apache2/sites-available/wsgiexample.conf, и скопировать туда следующее:

```
WSGIScriptAlias /wsgi/ <project_path>/wsgi/
    <VirtualHost *:80>
        DocumentRoot <project_path>
        <Directory />
```

```
            AuthType Kerberos
            AuthName "KRB"
            KrbAuthRealms TEST.RU
            KrbServiceName HTTP/srv1.test.ru
            KrbMethodNegotiate on
            Krb5Keytab /etc/apache2/keytab
            require valid-user
            KrbSaveCredentials on
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
    </VirtualHost>
```

В данном примере директива `WSGIScriptAlias` указывает, каким образом пользователи будут обращаться к WSGI интерфейсу (`http://srv1.test.ru/wsgi/`), и полный путь расположения скриптов. Скрипты должны располагаться в директории `<project_path>/wsgi/`.

7.4.2. Разработка скриптов для mod_wsgi

Пример python-скрипта может выглядеть следующим образом.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import psycopg2
import os

def application(environ, start_response):
    os.environ["KRB5CCNAME"] = environ["KRB5CCNAME"]
    connection_string = 'host=srv1.test.ru port=5432 dbname=postgres'
    tmp_str = ''
    try:
        db = psycopg2.connect(connection_string)
        cursor = db.cursor()
        cursor.execute('select current_user, session_maclabel;')
        records = cursor.fetchall()
        for record in records:
            tmp_str += "{0}".format(record)
    except Exception as e:
        tmp_str += "{0}".format(e)
    output = bytes(tmp_str, 'utf8')
```

```
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain;charset=utf8')]
    start_response(status, response_headers)

    return [output]
```

Данный пример демонстрирует основные этапы работы приложения, использующего сквозную аутентификацию в базе данных. В данном примере функция `application` выступает в роли главной функции и принимает два параметра `environ` и `start_response`. Эти переменные ей передает веб-сервер. Для сквозной аутентификации в базу данных необходимо получить `environ` от веб-сервера и установить переменную окружения `KRB5CCNAME`. После чего можно выполнить подключение к базе данных и завершить функцию после генерации ответа.

8. ОСНОВНЫЕ ПАКЕТЫ И СЛУЖБЫ ASTRA LINUX

8.1. Инфраструктурные службы

Служба	Описание
Служба автоматического назначения IP-адресов DHCP	Пакет `ics-dhcp-server`. Графический инструмент администрирования `fly-admin-dhcp`.
Служба разрешения имён DNS	Пакет `bind9`. Графический инструмент администрирования `fly-admin-bind`.
Служба каталогов, Lightweight Directory Access Protocol, LDAP	Пакет `389-ds-base` (также используется в контроллерах доменов `ALD` и `FreeIPA`).
Служба кеширования сетевых параметров аутентификации, System Security Services Daemon, SSSD	Пакет `sssd` (также используется в контроллерах доменов `ALD` и `FreeIPA`).
Служба точного времени NTP	Пакет `ntp`. Графический инструмент администрирования `fly-admin-ntp`.
Служба передачи файлов FTP	Пакет `vsftpd` (служба передачи файлов общего назначения). Пакет `tftpd` (упрощенная служба передачи файлов). Пакет `tftpd-hpa` (сервер загрузки (net bootstrap) (бездисковых) рабочих станций по сети). Для пакета `vsftpd` предусмотрен графический инструмент администрирования `fly-admin-ftp`.
Служба организации защищенных соединений SSL	Пакет `openssl`. Входящая в состав дистрибутива ОС ОН версия пакета поддерживает защитное преобразование передаваемых данных по алгоритмам ГОСТ.
Служба организации защищенных виртуальных сетей VPN	Пакет `openvpn`. Входящая в состав дистрибутива ОС ОН версия пакета доработана, и поддерживает защитное преобразование передаваемых данных по алгоритмам ГОСТ. Инструмент командной строки для администрирования сервера `astra-openvpn-server`. Графический инструмент администрирования сервера `fly-admin-openvpn-server`. Администрирование сервера и клиентов через графический инструмент `Network Manager`.
Служба организации защищенного удалённого подключения SSH	Пакет `ssh`. Входящая в состав дистрибутива ОС ОН версия пакета доработана, и поддерживает защитное преобразование передаваемых данных по алгоритмам ГОСТ.
Служба мониторинга Zabbix	Пакет `zabbix-server-mysql` (для работы с СУБД `mysql`). Пакет `zabbix-server-pgsql` (для работы с СУБД `PostgreSQL`).

Служба	Описание
Сетевая служба пакетного удалённого выполнения команд для управления развёртыванием и конфигурированием рабочих станций.	Пакет `ansible`. См. также `vagrant`.
Служба управления печатью	Пакет `cups`.
Службы резервного копирования	Пакет `bacula` (сетевая служба резеравного копирования корпоративного уровня).
Службы мониторинга и управления источниками бесперебойного электропитания	Пакет `nut` (сетевая служба управления ИБП корпоративного уровня). Пакет `apcupsd` (упрощенная система управления ИБП APC для рабочих станций).

8.2. Файловые службы

Служба	Описание
Служба передачи файлов FTP	Пакет `vsftpd` (служба передачи файлов общего назначения). Пакет `tftpd` (упрощенная служба передачи файлов). Пакет `tftpd-hpa` (сервер для автоматической загрузки (бездисковых) рабочих станций по сети). Для пакета `vsftpd` предусмотрен графический инструмент администрирования `fly-admin-ftp`.
Служба сетевого доступа к файловым системам NFS	Пакет `nfs-common`.
Распределённая файловая система GFS2	Пакет `gfs2-utils`.
Распределённая файловая система OCFS2	Пакет `ocfs2-tools`.
Сеть хранения данных Ceph	Пакет `ceph`.
Файловый сервер SMB/CIFS	Пакет `samba`. Может применяться для организации файловых серверов, серверов печати, контроллеров доменов. Графический инструмент для настройки разделяемых ресурсов `fly-admin-samba`.

8.3. Защита информации

Служба	Описание
Сервер инфраструктуры открытых ключей	Пакет `dogtag-pki`.
База данных инфраструктуры открытых ключей	Графический инструмент для организации центра сертификации пакет `xca`.
Сервер авторизации по проколу Kerberos	Пакеты `krb5-kdc`, `krb5-user`, `krb5-admin-server` и пр.

Служба	Описание
Служба организации защищенных соединений SSL	Пакет `openssl`. Входящая в состав дистрибутива ОС ОН версия пакета поддерживает защитное преобразование передаваемых данных по алгоритмам ГОСТ.
Служба управления ключами защиты информации	Пакет `kgpg`.
Библиотеки поддержки алгоритмов ГОСТ	Пакет `gostsum` (контрольные суммы в соответствии с алгоритмами ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-94). Пакет `libgost-astra` (алгоритмы защитного преобразования информации ГОСТ). Пакет `libgost-astra` совеместим с входящими с состав дистрибутива пакетами `openssl`, `openssh` и `openvpn`.

8.4. WEB-сервисы и приложения

Служба	Описание
WEB-сервер	Пакет `apache2`.
Кеширующий прокси WEB-сервер	Пакет `squid`.
WEB-браузеры	Пакеты `chromium`, `firefox`. Пакет fly-phone-webbrowser (для мобильных приложений).
Torrent-клиент	Пакеты `qbittorrent`, `deluge`.

8.5. Доменные службы

Служба	Описание
Astra Linux Directory	Пакет `ald-server-common`, `ald-client-common`. Графический инструмент администрирования контроллера домена ALD `fly-admin-ald-server`. Графический инструмент администрирования клиента домена домена ALD `fly-admin-ald-client`.
Домен Samba FreeIPA	Пакет `freeipa-server`, `freeipa-client`. Инструмент командной строки для быстрого запуска домена FreeIPA пакет `astra-freeipa-server`. Графический инструмент для быстрого запуска домена FreeIPA `fly-admin-freeipa-server`. Инструмент командной строки для ввода компьютера в домен FreeIPA пакет `astra-freeipa-client`. Графический инструмент для ввода компьютера в домен FreeIPA `fly-admin-freeipa-client`.
Домен Windows AD, домен Samba	Пакет `samba`. Графический инструмент администрирования контроллера домена Windows AD `fly-admin-ad-server`. Графический инструмент администрирования клиента домена домена Windows AD `fly-admin-ad-client`.

8.6. СУБД

Служба	Описание
СУБД PostgresQL	Пакет `postgresql`.
СУБД MySQL	Пакет `mysql-common`.

8.7. Кластеризация

Служба	Описание
Linux HA claster	Пакеты `pacemaker`, `corosync`.
Linux Virtual Server	Пакет `keepalived`.

8.8. Виртуализация

Служба	Описание
Виртуализация QEMU	Пакет `qemu-kvm`.
Виртуализация Oracle VM VirtualBox	Пакет `virtualbox` (основной пакет), `virtualbox-ext-pack` (расширения основного пакета), `virtualbox-guest-additions-iso` (дополнения гостевой ОС). Расширенный инструмент командной строки для управления виртуальными образами Vagrant: пакет `vagrant`. См. также ansible.
Сервис исполнения приложений в ограниченной среде FireJail	Пакет `firejail`, `firetools`.
Сервис исполнения приложений в защищенных контейнерах	Пакет `lxc`.

8.9. Почтовые серверы и клиенты

Служба	Описание
Почтовый сервер Exim MTA v4	Пакет `exim`.
Почтовый сервер Dovecot	Пакет `dovecot-core`.
Почтовый клиент Thunderbird	Пакет `thunderbird`, дополнительный пакет для защиты электронных писем: `enigmail`.
Почтовый клиент, календарь, планировщик заданий, органайзер Evolution	Пакет `evolution`.

8.10. Системное ПО

Служба	Описание
Графический менеджер пакетов	Пакет `synaptic`.

Служба	Описание
Межсетевой экран	Пакет `iptables`. Инструмент командной строки для управления межсетевым экраном: пакет `ufw`.
Графический инструмент для мониторинга параметров жестких дисков	Пакет `smartmontools`.

8.11. Прикладное и пользовательское ПО

Служба	Описание
Редактор процессор, электронные таблицы, презентации	Пакет `libreoffice`.
Текстовый редактор	Пакет `kate`.
Инструмент для просмотра файлов PDF	Пакет `qpdfwiev`.
Электронный словарь	Пакет `goldendict`.
Процессор электронной верстки	Пакет `texstudio`.

8.12. Мультимедиа

Служба	Описание
Аудиоредактор	Пакет audacity.
Видеоредактор, редактор 3D-моделей	Пакет `blender`.
Мультимедиа плеер	Пакет `vlc`.
Аудио плеер	Пакет `clementine`.
Аудио плеер	Пакет `qmmc`.
Управление звуком	Пакет `pulseaudio`.
Звуковой миксер	Пакет `kmix`.

8.13. Средства администрирования

Служба	Описание
Домен Samba AD	`fly-admin-ad-client`, `fly-admin-ad-server`
Домен ALD	`fly-admin-ald`, `fly-admin-ald-client`, `fly-admin-ald-server`
Служба DNS	`fly-admin-bind`
Служба DHCP	`fly-admin-dhcp`
Служба FTP	`fly-admin-ftp`
Запись образов на съёмные носители	`fly-admin-iso`
Служба NTP	`fly-admin-ntp`

Служба	Описание
Служба VPN	`fly-admin-openvpn-server`, `astra-openvpn-server`
Менеджер печати	`fly-admin-printer`
Разделяемые ресурсы Samba	`fly-admin-samba`
Контроллер домена FreeIPA	`fly-admin-freeipa-server`, `astra-freeipa-server`, `fly-admin-freeipa-client`, `fly-admin-freeipa-server`
Контроллер домена Samba	`astra-sambadc`, `astra-winbind`

8.14. Средства разработки и отладки

Служба	Описание
Контроль версий (SVN)	Пакет `subversion`
Управление изменениями (патчами)	Пакет `quilt`
Основная среда разработки	Пакет `qtcreator`
Компиляторы	Пакет `gcc`, `python`, `openjdk-8-jdk`

8.15. Мобильные приложения

Служба	Описание
Галерея	`fly-gallery`
Заметки	`fly-notes`
Звукозапись	`fly-record`
Интернет-браузер	`fly-phone-webbrowser`
Калькулятор	`fly-calc`
Контакты	`fly-contacts`
Монитор GPS/Глонасс	`fly-gps`
Музыка	`fly-music`
Погода	`fly-weather`
Почта	`fly-mail`
Просмотр PDF	`fly-pdfview`
Сообщения SMS	`fly-sms`
Телефон	`fly-qml-dialer`
Часы	`fly-date`

8.16. Игры

- `Пасьянс`
- `Сапёр`

– Стратегия Warzone 2012

– Цветные линии

– Шутер Xonotic

– JAG

– SuperTux 2 Платформер

СПИСОК ЛИТЕРАТУРЫ

Общие руководства

[1] Руководство по Debian Policy
(http://www.debian.org/doc/debian-policy/)

Стандарты открытого рабочего стола Freedesktop.org

[2] Desktop Entry Standard
(http://standards.freedesktop.org/desktop-entry-spec)

[3] Icon Theme Specifiations и Icon Naming Specification
(http://standards.freedesktop.org/icon-naming-spec
http://standards.freedesktop.org/icon-theme-spec)

[4] Shared MIME-info database
(http://standards.freedesktop.org/shared-mime-info-spec/)

[5] Drag-and-Drop Protocol for the X Window System
(http://www.freedesktop.org/wiki/Specifications/XDND)

[6] Extended Window Manager Hints
(http://standards.freedesktop.org/wm-spec/wm-spec-latest.html)

[7] XEmbed Protocol Specifications
(http://standards.freedesktop.org/xembed-spec/xembed-spec-latest.html)

[8] System Tray Protocol Specifications
(http://standards.freedesktop.org/systemtray-spec/systemtray-spec-latest.html)

[9] Desktop Menu Specifications
(http://standards.freedesktop.org/menu-spec/menu-spec-latest.html)

[10] Sound Theme and Naming Specifications
(http://www.freedesktop.org/wiki/Specifications/sound-theme-spec)

[11] Desktop Application Autostart Specification
(http://standards.freedesktop.org/autostart-spec/autostart-spec-latest.html)

Книги по графической библиотеке Qt

[12] Марк Саммерфилд: *Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++*. 2018, Символ-Плюс

[13] Макс Шлее: *Qt5.10. Профессиональное программирование на C++. Наиболее полное руководство*, 2018, БХВ-Петербург, ISBN: 978-5-9775-3678-3

[14] Юрий Земсков: *Qt 4 на примерах*, 2008, БХВ-Петербург

[15] Daniel Molkentin: *The Book of Qt4: The Art of Building Qt Applications*, 2007, No Starch Press San Francisco

[16] Johan Thelin: *Foundations of Qt Development*, 2007, APress

[17] Alan Ezust and Paul Ezust: *An introduction to Design Patterns in C++ with Qt4*, 2006, Prentice Hall