РУКОВОДЯЩИЕ УКАЗАНИЯ ПО КОНСТРУИРОВАНИЮ

прикладного программного обеспечения

для операционной системы специального назначения

Astra Linux Special Edition РУСБ.10015-01 (очередное обновление 1.7)

Листов 80

Москва

2022.0427

АННОТАЦИЯ

Настоящие руководящие указания по конструированию (РУК) предназначены для разработчиков прикладных программ, автоматизированных систем (АС) управления в защищенном исполнении и аппаратно-программных средств предназначенных для эксплуатации на платформе операционной системы специального назначения Astra Linux Special Edition РУСБ.10015-01 (далее по тексту — ОС СН).

В документе приводится краткий обзор архитектурных особенностей ОС СН и принципов ее версионности. Описаны применение программного интерфейса средств защиты информации (СЗИ) и основы разработки программ для многофункционального защищенного оконного менеджера Fly. Объясняются способы использования интегрированных в ОС СН прикладных программных средств. Настоящие РУК распространяются на очередное обновление 1.7 ОС СН.

СОДЕРЖАНИЕ

1. СОСТАВ ДИСТРИБУТИВА ОС СН

ОС СН относится к семейству дистрибутивов Linux, и предназначена для применения в составе информационных (автоматизированных) систем в целях обработки и защиты [1)] информации любой категории доступа [2)] — общедоступной информации, а также информации, доступ к которой ограничен федеральными законами (информации ограниченного доступа).

ОС СН сертифицирована в системе сертификации средств защиты информации Министерства обороны Российской Федерации на соответствие требованиям по безопасности информации:

- по 1 уровню контроля отсутствия недекларированных возможностей согласно руководящему документу «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» (Гостехкомиссия России, 1999 г.);
- по соответствию реальных и декларируемых в документации функциональных возможностей;
- документа «Требования безопасности информации к операционным системам» (ФСТЭК России, 2016);
- документа «Профиль защиты операционных систем типа «А» первого класса защиты. ИТ.ОС.А1.ПЗ» (ФСТЭК России, 2017).

ОС СН сертифицирована в системе сертификации средств защиты информации ФСТЭК России на соответствие требованиям по безопасности информации, установленным в документах:

- «Требования безопасности информации к операционным системам» (ФСТЭК России, 2016);
- «Профиль защиты операционных систем типа «А» первого класса защиты. ИТ.ОС.А1.ПЗ» (ФСТЭК России, 2017);
- «Требования по безопасности информации, устанавливающие уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (ФСТЭК России, 2018) по 1 уровню доверия.

Это позволяет применять ОС СН при построении АС в защищенном исполнении, обрабатывающих информацию ограниченного распространения, включая государственную тайну до степени секретности «совершенно секретно».

[1)] От несанкционированного доступа.

[2)] В соответствии с Федеральным законом от 27.07.2006 No 149-ФЗ "Об информации, информационных технологиях и о защите информации"(статья 5, пункт 2).

Архитектурно ОС СН состоит из базовой программной платформы GNU/Linux на основе ядер семейства 5.4/5.10 и интегрированных программных средств (ИПС). При этом в состав ОС СН входит как общее программное обеспечение (ОПО) с открытыми исходными текстами, так и собственные разработки.

За основу построения дистрибутива ОС СН взят открытый репозиторий пакетов для ОС Debian Linux и применяемая в нем система пакетирования `deb`. Таким образом, общие принципы разработки и сборки программ для ОС СН, а также создания своего репозитория пакетов соответствуют дистрибутивам Linux семейства Debian. Поэтому разработчикам программ для ОС СН можно использовать общие руководства по Debian Linux (например, такие как [6], [7]).

В состав ОС СН входят встроенные СЗИ. Они обеспечивают мандатное разграничение доступа в дополнение к стандартному для Linux-систем дискреционному разграничению доступа, а также контроль целостности системы и регистрацию событий при работе с файлами и процессами.

Специальные библиотеки СЗИ предоставляют разработчикам программный интерфейс для использования средств защиты в своих программных продуктах. Защита графической среды пользователя в ОС СН обеспечивается защищенным графическим сервером и оптимизированным рабочим столом Fly (читается «флай»), являющимся единственным вариантом многофункционального графического интерфейса пользователя в составе ОС СН. Программный интерфейс рабочего стола Fly и методы его использования описаны в разделе 2.

1.1. Основные компоненты

Компоненты ОС СН, с которыми интегрированы встроенные СЗИ:

- базовая платформа:
    - ядро Linux семейств 5.4 и 5.10;
    - утилиты управления СЗИ;
    - сетевые службы;
    - графический сервер Xorg;
    - графическая система Fly;
    - система печати CUPS;
- средства организации единого пространства пользователей (ЕПП):
    - система организации домена Astra Linux Directory (ALD);
    - система организации домена FreeIPA;
    - средства для работы с доменами Windows Active Directory:
        - samba/winbind;

- SSSD;

- общее программное обеспечение (ПО):
  - защищенная СУБД PostgreSQL версии 11;
  - HTTP-сервер Apache2;
  - защищенные средства обмена электронными сообщениями Exim4 и Dovecot.

В качестве оконного менеджера в ОС СН возможно использование только оконного менеджера `fly-wm`, адаптированного к требованиям защиты информации;

1.2. Рекомендации по использованию средств для разработки ПО

Разработку ПО рекомендуется проводить с использованием перечисленных в данном подразделе базовых компонент, библиотек и средств разработки.

1.2.1. Базовые библиотеки и утилиты

Базовые библиотеки и утилиты:

1) ядро Linux версий 5.4 и 5.10;
2) `glibc`;
3) `openldap`;
4) `openssl`;
5) `bash`.

1.2.2. Графическая система и многофункциональный рабочий стол

Разработку прикладных программ с графическим интерфейсом для ОС СН рекомендуется проводить с использованием следующих библиотек:

1) `Qt` — основная прикладная библиотека для создания графического интерфейса;
2) `Xorg` — библиотеки графического X-сервера;
3) `Fly` — библиотеки графического интерфейса пользователя.

1.2.3. Средства разработки и отладки

Окружение разработки, к которому привыкли разработчики Linux-систем, доступно и в ОС СН. Оно содержится в отдельном репозитории со средствами разработки.

Разработку прикладного ПО рекомендуется проводить с использованием этих средств (например, для разработки программ на основе библиотек Qt удобно использовать Qt Creator).

Фундаментальную инфраструктуру окружения C/C++ составляют инструменты компиляции кода: библиотеки C/C++ (`glibc`), компиляторы, средства сборки, отладчики и специализированные пакеты для разработки программ (devel-пакеты). Ниже приведен краткий список этих средств:

- GCC — открытый компилятор, соответствующий стандарту ANSI;

- средства сборки:
  - `binutils` — программы компоновки и манипулирования объектными файлами;
  - `make` — средство для автоматизации процесса компиляции;
  - `cmake` — кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода;
- отладчик `gdb` — стандартный отладчик GNU.

1.3. Общее программное обеспечение

В состав ОПО входят программы, которые чаще всего востребованы при построении различных АС: системы управления базами данных (СУБД) реляционного типа, средства работы в сетях, набор офисных программ, система верстки текстов, а также средства работы с мультимедиаданными и графикой.

Для разработчиков АС можно выделить следующие важные компоненты:

1) средства для разработки ПО с использованием СУБД:
   - PostgreSQL версии 11;
2) средства для разработки документации:
   - LaTeX;
   - LibreOffice;
3) средства для работы с мультимедиа:
   - мультиплеер VLC;
   - фреймворк Phonon;
4) средства для разработки web-ориентированного ПО:
   - защищенные средства гипертестовой обработки данных:
     - сервер Apache2;
     - браузер Firefox;
   - язык программирования PHP 7.3;
   - язык программирования Python 3.7;
5) защищенный комплекс программ электронной почты:
   - агент пересылки почтовых сообщений SMTP-сервер Exim4;
   - почтовый IMAP и POP3 сервер Dovecot;
   - почтовый клиент Thunderbird.

1.4. Варианты исполнения ОС СН

Варианты исполнения ОС СН отличаются друг от друга кодом дистрибутива и номером очередного обновления:

- код дистрибутива — определяет специфику дистрибутива: его аппаратную платформу(процессорную архитектуру);

- номер очередного обновления — число, определяющее номер очередного обновления ОС СН.

В установленной ОС СН информация о варианте исполнения хранится в текстовом файле в каталоге `/etc`:

`/etc/astra_version`

Формат записи в этом файле следующий:

`a.b.c`

где:

- a — обозначение аппаратной платформы (процессорной архитектуры):
  - 1 — x86-64;
  - 4 — ARM;
  - 8 — Эльбрус;
  - ...
- b — номер очередного обновления:
  - 6 — базовый репозиторий Stretch;
  - 7 — базовый репозиторий Buster;
  - 8 — базовый репозиторий Bullseye
  - ...ы
- c — номер оперативного обновления:
  - 0 — выпуск нового очередного обновления;
  - 1 — первое оперативное обновление;
  - ...

2. 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.

Выполнение 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:[22]–[27]. Также предполагается знание основных стандартов от `freedesktop.org`.

2.1. 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.

2.2. 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.

2.3. 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
- /usr/share/flybuild/fly_vars.mk

предназначенные 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 пакет flybuild содержит вспомогательные файлы для системы сборки qmake:

- /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly.prf;
- /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/fly_vars.prf.

Файл fly_vars.prf задает пути, используемые всеми приложениями при установке. Файл fly.prf задает 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.

2.4. Указание секции дистрибутива в пакетах

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

2.5. 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.).

2.5.1. 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. (подробнее 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.

2.5.2. Файл desktop entry

Почти каждое fly-приложение должно сопровождаться файлом desktop entry [9] 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 [9]. 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.

Значение поля ‘X-FLY-IconContext‘ указывает в каком контексте надо в первую очередь искать иконку ярлыка. Подробнее см. [10].

Файлы `desktop entry` как правило должны иметь `Type=Application`, но для других случаев возможно использование других типов, разрешенных стандартом [9]: `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».

2.5.3. Файл перевода

Все приложения должны быть русифицированы с использованием стандартных средств Qt.

Не следует статично указывать русские названия элементов графического интерфейса непосредственно в коде программы или в формах, генерируемых с помощью Qt Designer, так как 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`.

2.5.4. Файл для определения иконок приложений

Приложение может установить свою специфичную иконку. Для этого она должна быть в svg-формате, располагаться 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 располагаются. 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.

2.5.5. 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 файле 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
```

2.6. Разработка приложения

2.6.1. 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;

остальные параметры задаются 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 класс ы`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.

2.6.2. 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 методов 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:

```
const std::string &GetFlyDir(FlyDir d);
```

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/Docume
с возможными подкаталогами для файлов с ненулевыми метками безопасности.

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.

Для поддержки каталогов 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 конфиденциальности. Некоторые приложения могут не работать в силу того, что они производят запись в файлы с ненулевыми уровнями конфиденциальности. 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 конфиденциальности, 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 варианта исполнения ОС СН 1.7 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.

2.6.3. 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` [9];
- доступ к темам иконок [11];
- 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 приложений 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.

2.6.4. Локализация

Все заголовки и надписи должны быть выполнены на русском языке. Локализация выполняется средствами и в соответствии с рекомендациями Qt (подробнее см. 2.5.3).

2.6.5. 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).

2.6.6. 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()`).

2.6.7. 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.

Рабочий стол Fly имеет несколько режимов работы (обычный рабочий стол, планшетный, безопасный), выбираемых оператором в одном из меню графического входа. Также доступен мобильный режим работы (не устанавливается по умолчанию). 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`.

2.6.8. 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. В ОС используется тема `fly-astra`, соответствующая как Icon Naming Specification [7], так и некоторым особенностям KDE4. При этом использовать имена иконок, принятые в KDE3, уже нельзя.

Начиная с очередного обновления ОС СН 1.7 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 [11].

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

2.6.9. MIME-типы

В библиотеку `libflycore` включена часть, реализующая API для детектирования MIME-типа любого файла (реализация соответствующего стандарта freedesktop.org [12]). Qt-приложения должны использовать класс `QMimeDatabase`.

2.6.10. 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 [18]. Каждое приложение, желающее пользоваться звуками из этой спецификации, должно использовать текущую звуковую тему, выбранную пользователем в настройках рабочего стола Fly. Для этого предназначен класс `FlySoundTheme`, 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 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 [18] должны поставляться вместе с приложением.

Спецификации [18] 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.

2.7. 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;
}
```

2.7.1. 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.

2.8. Взаимодействие с рабочим столом

2.8.1. 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.

2.8.2. 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 [15], [16]. При установке иконки для `QSystemTrayIcon` рекомендуется использовать `QIcon`, которая может выдавать изображения всех типичных размеров (см., например, `QIcon::fromTheme`).

Начиная с очередного обновления ОС СН 1.7 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 черные и белые иконки, предназначенные для трея, в эти две темы, и переключение будет происходить автоматически при смене цвета панели задач.

2.8.3. 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 со-

здан фиктивный `appname.desktop` с полем `Hidden=true`, что согласно стандарту [19] является пользовательским методом отключения системного автозапуска.

2.8.4. 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 [9]).

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. 2.5.2). 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 очередного обновления ОС СН 1.7 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 [17].

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. 2.5.2). 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 [17]), чтобы облегчить ручную правку меню, как это сделано в MS Windows. С другой стороны, Fly предоставляет пользователю мощную программу `fly-menuedit` для просмотра, редактирования, резерв-

ного 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 очередного обновления ОС СН 1.7 в desktop-файлах можно определять их участие или неучастие в определнных типах сессий (см. `$DESKTOP_SESSION`) путем указания имн сессий в полях `NotShowIn` и `OnlyShowIn`.

2.8.5. 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).

2.8.6. 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 [13].

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.

2.8.7. 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.

2.8.8. 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`. В текущей версии ОС эти уведомления посылать не обязательно, т.к., используя механизм inotofy, fly-fm сам «понимает» момент, когда необходимо сделать обновление.

В случаях, если 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`.

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`.

2.8.9. 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()`.

2.8.10. 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`.

2.8.11. 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 [14] 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 СН 1.7 можно задать область экрана, которую займт рабочий стол, с помощью параметров FlyDesktopX, FlyDesktopY, FlyDesktopWidth, FlyDesktopHeight в файлах $HOME/.fly/theme/*themerc*.

2.8.12. Автозапуск приложений при входе в графическую сессию

Графический вход ОС СН поддерживает автоматический старт приложений с помощью ярлыков. Приложения запускаются:

- Из каталога /usr/share/fly-dm/preload/greeter сразу перед стартом графической части входа;
- Из каталога /usr/share/fly-dm/autostart/greeter сразу после старта графической части входа.

Пример ярлыка для запуска произвольного сценария /usr/bin/script.sh:

```
[Desktop Entry]
Type=Application
Name=No login
Exec=/usr/bin/script.sh
```

В ярлыках автозапуска может быть задано поле ‘X-FLY-Delay‘, определяющее задержку в секундах после запуска ярлыка.

2.8.13. Размещение ярлыков у пользователей

Для централизованного размещения ярлыков на рабочих столах и в меню «Пуск» пользователей предусмотрены каталоги:

- `/usr/share/applications/flydesktop` - каталог для размещения ярлыков на рабочих столах. Может быть переопределен переменной окружения `FLY_SHARED_DESKTOP_DIR`;
- `/usr/share/applications/flystartmenu` - каталог для размещения ярлыков в меню «Пуск». Может быть переопределен переменной окружения `FLY_SHARED_STARTMENU_DIR`.

Если в этих каталогах разместить ярлыки, то эти ярлыки мгновенно появятся у всех (в т.ч. уже работающих) пользователей на рабочем столе или в меню «Пуск» соотвественно. Возможность переопределения этих каталогов с помощью переменных окружения может быть использована для подключения сетевых каталогов.

2.9. Менеджер файлов Fly-fm

В стандартную комплектацию ОС СН входит графический файловый менеджер fly-fm помимо работы с файловыми объектами обеспечивающий работу с сетевыми подключениями (с разделяемыми ресурсами Samba) и предоставлющий широкие возможности для настройки.

2.9.1. Кастомизация меню "Действия"в файловом менеджере fly-fm

В файловом менеджере fly-fm пользователю предоставляется возможность задавать списки действий, вызываемых по нажатию правой кнопки мыши на файловый объект. Эти списки могут привязываться к MIME-типам файлов, т.е. для разных типов файлов могут предлагаться разные типы действий. Списки действий хранятся в файлах с расширением .desktop, расположенных:

- Для действий, определяемых индивидуально пользователем - в домашнем каталоге пользователя в подкаталоге `~/.local/share/flyfm/actions/`;
- Для действий, определяемых системно для всех пользователей - в каталоге `/usr/share/flyfm/actions/`

Пример системного файла, задающего действия с образами ISO (файл `/usr/share/flyfm/actions/fly-admin-iso-action.desktop`):

```
[Desktop Entry]
Type=Application
NoDisplay=false
Actions=WriteISO;
Hidden=false
Terminal=false
```

```
StartupNotify=false
MimeType=application/x-cd-image

[Desktop Action WriteISO]
Name=Fly admin ISO
Name[ru]=Записать на USB носитель
Icon=inode-blockdevice
Exec=su-to-root -X -c "fly-admin-iso -f %F"
```

Подробное описание структуры файлов, определяющих действия, доступно в Википедии Astra Linux по ссылке:[20].

2.9.2. Плагины KDE в файловом менеджере fly-fm

В файловом менеджере fly-fm реализована возможность расширения его функционала с помощью так называемых «плагинов» KDE (см. описание [21]).

2.10. Настройки планшетного режима

В ОС СН 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 scессии и выставлена переменная `$DESKTOP_SESSION`.

2.11. 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`. В ОС СН 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`.

2.12. Межмандатный буфер обмена

В ОС СН имеется возможность копирования-вставки (copy-paste) между программами с различными метками безопасности, даже если эти программы выполняются в разных сессиях. При этом, т.к. весь процесс происходит через файлы, то допустимость копирования-вставки автоматически контролируется на уровне файловой системы. Для включения этой возможности надо в домашнем каталоге пользователя в файлах .fly/theme/*.themerc задать UseClipboardManagerMAC=true. Обратите внимание, что для разных мандатных меток имеются разные домашние каталоги, и настройку нужно выполнить во всех этих каталогах.

2.13. Приложения и мандатная политика

Для разработки графических приложений с использованием библиотек графических интерфейсов Qt 5 в ОС СН реализована библиотека `libparsec-mac-qt5-1`, предоставляющая удобный доступ к функциям подсистемы безопасности PARSEC и содержащая набор классов для работы с мандатными атрибутами (3.1). Для разработки приложений с использованием указанной библиотеки необходимо установить пакет `libparsec-mac-qt5-1-dev`.

Далее приведены примеры использования классов из библиотеки `libparsec-mac-qt5-1`.

Примеры:

1. Создание диалога для установки мандатных атрибутов объектов ФС

```
/* *** Файл main.cpp *** */
#include <QApplication>

#include "filemaclabeldialog.h"

int main(int argc, char *argv[])
{
QApplication a(argc, argv);

FileMacLabelDialog dlg("/usr/bin");
dlg.exec();

return 0;
}

/* *** Файл filemacdialog.h *** */

#ifndef FILEMACLABELDIALOG_H
#define FILEMACLABELDIALOG_H

#include <QDialog>
#include <QVBoxLayout>
#include <QDialogButtonBox>
#include <QLabel>
#include <QPushButton>
#include <QMessageBox>
```

```
#include <parsec-qt5/filemaclabelwidget.h>

class FileMacLabelDialog : public QDialog
{
  Q_OBJECT
  public:
  FileMacLabelDialog(const QString& filePath) : m_filePath(filePath)
  {
    QVBoxLayout *layout = new QVBoxLayout(this);
    QLabel* label = new QLabel(tr("MAC for %1:").arg(m_filePath));
    layout->addWidget(label);

//создание экземпляра класса для работы с мандатными атрибутами
//объектов ФС и получение мандатных атрибутов объекта ФС

    m_fileMacLabelWidget = new ParsecQt5::FileMacLabelWidget;
    layout->addWidget(m_fileMacLabelWidget);
    m_fileMacLabelWidget->getfmac(m_filePath);
    connect(m_fileMacLabelWidget, SIGNAL(changed()), this, SLOT(setChanged()

    m_buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok |
    QDialogButtonBox::Cancel);
    connect(m_buttonBox, SIGNAL(accepted()), this, SLOT(accept()));
    connect(m_buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
    layout->addWidget(m_buttonBox);
    setChanged(false);
  }

~FileMacLabelDialog() {}

  bool isChanged() const
  {
     return m_buttonBox->button(QDialogButtonBox::Ok)->isEnabled();
  }

private slots:

  void setChanged(bool changed = true)
```

```
  {
     m_buttonBox->button(QDialogButtonBox::Ok)->setEnabled(changed);
  }

  void accept()
  {
    if (isChanged()) {
//установка мандатных атрибутов объекта ФС
    if (!m_fileMacLabelWidget->chmac(m_filePath)) {
    QMessageBox::critical(this, tr("Error"), tr("MAC can’t be changed"));
    return;
  }
}

QDialog::accept();
}

protected:

ParsecQt5::FileMacLabelWidget* m_fileMacLabelWidget;
QDialogButtonBox* m_buttonBox;
QString m_filePath;
};

#endif // FILEMACLABELDIALOG_H
```

2. Использование диалога выбора мандатных атрибутов пользователя при входе в систему через графический интерфейс пользователя

```
/* *** Файл main.cpp *** */

#include <QApplication>
#include <QLocale>
#include <QTranslator>
#include <QX11Info>

#include <parsec-qt5/loginmaclabeldialog.h>
```

```
#include <X11/Xlib.h>
#include <stdio.h>
#include <getopt.h>
#include <pwd.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>

#include <fly/flyfileutils.h>

using namespace ParsecQt5;

bool user_id_specified = false;

struct arguments {
  uid_t user_id;
};

static arguments cmd_params;

bool process_user_name( const char * name )
{
  if ( ! name )
  {
    fprintf( stderr,"No name specified\n" );
    return false;
  }

  struct passwd * pwd = getpwnam( name );
  if ( ! pwd )
  {
    fprintf( stderr, "No such user %s\n", name );
    return false;
  }

  cmd_params.user_id = pwd->pw_uid;
  user_id_specified = true;
  return true;
```

```
}

void show_help()
{
  printf( "Options :\n"
  "-u, --user : username\n"
  "-h, --help : this help\n");
}

void print_error()
{
  fprintf( stderr, "Error : invalid arguments\n" );
}

/* returns true, if ok */
bool process_args( int argc, char ** argv )
{
  struct option longopts[] = {
    { "user", 1, NULL, 'u' },
    { "help", 0, NULL, 'h' },
    { 0, 0, 0, 0 }
  };

  int c;

  while( ( c = getopt_long( argc, argv,
  "u:h" /*"u:w:l:c:htea"*/, longopts, NULL ) ) != -1 )
  {
    switch( c ) {
    case 'u':
      if ( ! process_user_name( optarg ) )
      {
        print_error();
        show_help();
        return false;
      } else {
        break;
      }
```

```
    case 'h':
    show_help();
      return false;
    default:
/* Для поддержки опций QApplication,
   игнорируем все неизвестные опции */
     break;
    }
  }
  return true;
}

void print_result( LoginMacLabelDialog * dialog )
{
  if ( ! dialog ) return;

  fprintf( stdout, "%x ", dialog->level() );

//print in hex, new pam accept hex only!
  fprintf( stdout, "%lx ", dialog->categories() );

  std::cout << std::endl;
}

int main( int argc, char ** argv )
{
  if ( ! getenv( "LANG" ) ) setenv( "LANG", "ru_RU.UTF-8", 0 );

  QApplication app( argc, argv );

  if ( ! process_args( argc, argv ) ) exit( 1 );

  QTranslator translator( 0 );
  translator.load( "fly-mac-dialog_" + QLocale::system().name(),
  QString::fromStdString(GetFlyDir(APP_TRANS_DIR)) );
  app.installTranslator( &translator );

//создание диалога выбора мандатных атрибутов пользователя
```

```
//при графическом входе в систему

  LoginMacLabelDialog * dialog = new LoginMacLabelDialog(
  user_id_specified? cmd_params.user_id:getuid(), 0 );

  if ( !dialog->isValid() )
  {
    fprintf( stderr, "Dialog invalid\n" );
    exit( 1 );
  }

  int result = QDialog::Accepted;

  if ( dialog->isSuitable() )  //don't show if not suitable but accept
  {
    dialog->show();
    dialog->activateWindow();
    dialog->setFocus();
    QObject::connect(dialog,SIGNAL(finished(int)),&app,SLOT(quit()));

    app.exec();

    result = dialog->result();
  }

  if ( result == QDialog::Accepted )
  {
    print_result( dialog );
    delete dialog;
    return 0;
  }

  delete dialog;
  return 1;
}
```

3. ПРОГРАММНЫЙ ИНТЕРФЕЙС МАНДАТНОГО РАЗГРАНИЧЕНИЯ ДОСТУПА

3.1. Общие сведения

В ОС СН реализован механизм мандатного разграничения доступа. При этом, принятие решения о запрете или разрешении доступа субъекта к объекту принимается на основе типа операции (чтение/запись/исполнение), мандатного контекста безопасности субъекта и метки безопасности объекта . Кроме того, при принятии решения могут учитываться специальные привилегии субъекта (подробнее см. [2], подраздел 4).

3.2. Работа с метками безопасности файловых объектов

Для работы с метками безопасности файловых объектов в ОС СН используются библиотеки API Parsec, предоставляемые пакетом разработчика приложений parsec-dev. Далее приведен простой пример программы, работающей с метками безопасности файловых объектов. Программа выполняет следующие действия:

1) определяет и выводит на печать максимальную метку безопасности, доступную в системе;

2) определяет и выводит на печать уровень целостности с которым происходит выполнение;

3) получает и выводит на печать метку безопасности указанного файлового объекта (файла или каталога);

4) присваивает указанному файловому объекту новую метку безопасности (в примере в целях предотвращения нарушения работы ОС присваивается прочитанная метка без внесения в нее изменений). Для успешного присвоения метки безопасности пример должен быть запущен с правами суперпользователя с высоким уровнем целостности.

ВНИМАНИЕ! Изменение мандатных атрибутов системных файловых объектов может привести к нарушению работы ОС.

Следует обратить внимание, что для декларации переменных, в которых сохраняются мандатные атрибуты, в примере используются переопределенные в API Parsec типы данных (`PDP_LEV_T`, `PDP_ILEV_T`, `PDP_CAT_T`, `PDP_TYPE_T`). Использование переопределенных типов данных обеспечивает переносимость ПО между различными процессорными архитектурами. Также для обеспечения переносимости применяются переопределенные форматы функции печати printf (`PDP_PRNTF_LEV`, `PDP_PRNTF_ILEV`, `PDP_PRNTF_CAT`).

Порядок сборки и запуска примера описан после исходного текста примера. Исходный текст:

```
#include <inttypes.h>
#include <parsec/pdp.h>
```

```
#include <pdp_common.h>
#include <stdio.h>

static const PDPL_T *max_l = NULL;

int main( int argc, char **argv) {
    if( argc != 2) {
        printf( "Укажите имя файлового объекта в качестве аргумента\n");
        return 1;
    }
    char *path = argv[1];
/*
* Определение максимальной системной метки безопасности
*/
    max_l = pdp_get_sys_max();
    printf( "Максимальная системная метка безопасности:\n"
            "\tИерархический уровень конфиденциальности   : %"PDP_PRNTF_LEV"\n"
            "\tИерархический уровень целостности          : %"PDP_PRNTF_ILEV"\n"
            "\tНеиерахические катагории конфиденциальности: %"PDP_PRNTF_CAT"\n",
            pdpl_lev(max_l), pdpl_ilev(max_l), pdpl_cat( max_l));
/*
* Определение текущего уровня целостности
*/
    printf( "Текущий уровень целостности: %"PDP_PRNTF_ILEV"\n",
        pdp_get_current_ilev());

/*
* Получение метки безопасности файлового объекта
*/

    PDPL_T *file_l = pdp_get_lpath(path);
    if (!file_l) {
        fprintf(stderr, "Ошибка получения метки безопасности объекта %s: %m\n", pa
        return 2;
    }

/*
* Извлечение из полученной метки безопасности атомарных значений
```

```
*/

    PDP_LEV_T lev = pdpl_lev(file_l);    // Иерархический уровень конфиденциальнос
    PDP_ILEV_T ilev = pdpl_ilev(file_l); // Иерархический уровень целостности
    PDP_CAT_T cat = pdpl_cat( file_l);   // Неиерахические катагории конфиденциаль
    PDP_TYPE_T type = pdpl_type(file_l); // Флаги

    printf( "Метка безопасности объекта %s:\n", path);
    printf( "\tИерархический уровень конфиденциальности   : %"PDP_PRNTF_LEV"\n"
            "\tИерархический уровень целостности          : %"PDP_PRNTF_ILEV"\n"
            "\tНеиерахические катагории конфиденциальности: %"PDP_PRNTF_CAT"\n",
            lev, ilev, cat);
    if( type) {
        printf( "\tФлаги                                        : ");
        if( type & PDPT_CCNR) printf( "CCNR ");
        if( type & PDPT_CCNRI) printf( "CCNRI ");
        if( type & PDPT_EHOLE) printf( "EHOLE ");
        if( type & PDPT_EHOLE) printf( "WHOLE ");
        printf( "\n");
    }

/*
* Изменение метки безопасности путем записи в нее новых атомарных значений
*/

    pdpl_set_lev( file_l, lev);
    pdpl_set_ilev( file_l, ilev);
    pdpl_set_cat( file_l, cat);
    pdpl_set_type( file_l, type);

/*
* Присвоение метки безопасности файловому объекту
*/

    int error = pdp_set_path(path, file_l);
    pdpl_put(file_l);
    if (error) {
        printf( "Не могу присвоить метку безопасности: %m\n");
```

```
        return error;
    }
    else printf( "Метка безопасности присвоена успешно\n");

    return 0;
}
```

Порядок сборки и запуска примера:

1) установить пакеты:

```
sudo apt install parsec-dev gcc
```

2) исходный текст примера сохранить в файле с именем `main.c` в текущем каталоге;

3) собрать исполняемый файл командой:

```
gcc main.c \
-I /usr/include/parsec\
-lparsec-base -lparsec-cap -lparsec-mac -lpthread
```

4) запустить собранный исполняемый файл a.out применив его к различным файловым объектам, например:

```
./a.out /tmo
./a.out /dev/null
./a.out /run
sudo ./a.out /dev/null
```

3.3. Сетевое взаимодействие

3.3.1. Передача классификационных меток по сети

В ОС СН сетевые соединения рассматриваются как средство межпроцессного взаимодействия и подвергаются мандатному контролю доступа. Для этого в сетевые пакеты протокола IPv4 внедряются классификационные метки, соответствующие классификационной метке объекта – сетевое соединение (сокет). При этом классификационная метка сокета наследуется от субъекта (процесса).

Порядок присвоения классификационных меток и их формат соответствует национальному стандарту ГОСТ Р 58256-2018 «Защита информации. Управление потоками информации в информационной системе. Формат классификационных меток». Прием сетевых пакетов подчиняется мандатным ПРД. Следует особо отметить, что классификационная метка сокета может иметь специальный тип `ehole`, позволяющий создавать сетевые сервисы, принимающие соединения с любыми классификационными метками.

Классификационные метки передаются в поле опций IPv4 каждого пакета. Для передачи классификационных меток по сети используется вариант опции DoD Security Option IPv4, что обеспечивает совместимость пакетов с международным стандартом RFC

1108 — Security Options for the Internet Protocol — Параметры безопасности для IP-протокола.

В рамках указанных стандартов классификационная метка снабжается классом 0xAB (Unclassified), при этом последующий битовый список (последовательность байт, в которых младший бит указывает на наличие следующего байта в потоке) опции представляет собой упакованную в соответствии со стандартом структуру мандатного контекста, где уровень занимает 8 бит, а категории — 64 бита (порядок байт — от младших к старшим). Последние (старшие) нулевые биты могут быть отброшены.

Для передачи классификационных меток по сети используется формат, представленный в таблице 1.

Таблица 1

Поля, входящие в поле опции	TYPE (8 бит)	LENGTH (8 бит)	CLASSIFICATION LEVEL (8 бит)	PROTECTION AUTHORITY FLAGS (11+байт)
Значение поля	10000010	XXXXXXXX	10101011	AAAAAAA[1|0]AAAAAAA0

Поле TYPE всегда содержит двоичное значение 10000010, т.е. десятичное значение 130.

Поле LENGTH содержит значение длины опции. Минимальная длина опции 3 октета, включая поля TYPE и LENGTH. Поле PROTECTION AUTHORITY FLAGS может отсутствовать. Значение поля LENGTH меньше 3 октетов должно сигнализировать об ошибке.

Поле CLASSIFICATION LEVEL всегда выставляется в 10101011 (Unclassified).

Поле PROTECTION AUTHORITY FLAGS имеет переменную длину, младший бит каждого октета (байта) используется для индикации наличия следующего октета. Если бит равен 1, есть следующий октет, если бит равен 0 — октет последний.

Классификационная метка в подсистеме безопасности PARSEC в ОС СН имеет следующий формат:

```
struct parsec_mac_label {
    uint8_t level;
    uint64_t  categories;
};
```

где: level — иерархический уровень конфиденциальности (беззнаковое целое число, 256 возможных значений), categories — неиерархическая категория конфиденциальности (битовая маска little-endian, всего 64 категории). Структура должна быть упакована (выравнивание полей отключено). В поле PROTECTION AUTHORITY FLAGS данная классификационная метка записывается следующим образом: метка представляется в виде последовательности бит, начиная с младшего байта, вставляется младший бит признака продолжения, со сдвигом остальных бит. Далее анализируются октеты с конца поля PROTECTION AUTHORITY FLAGS. В соответствии со стандартом октеты, полезные 7 бит

которых содержат `0`, отбрасываются с корректировкой поля длины.

Таким образом, при отображении структуры `parsec_mac_label` длиной 9 байт в поле `PROTECTION AUTHORITY FLAGS` длина данного поля может варьироваться от 0 (для пустой метки `{0,0}`) до 11 октетов.

Пример кодирования классификационной метки для протокола IPv4:

```
IPOPT_SEC,5,0xAB,03,12
```

Поле `PROTECTION AUTHORITY FLAGS` в виде битового списка:

```
00000011,00001100
```

Мандатный контекст:

```
уровень 1, категории 3
```

Отсутствие метки безопасности на объекте доступа является синонимом нулевой метки безопасности. Таким образом, ядро ОС, в которой все объекты и субъекты доступа не имеют меток безопасности (имеют нулевые метки безопасности), функционирует аналогично стандартному ядру ОС Linux.

3.3.2. Разработка ПО для обработки информации с различными уровнями конфиденциальности

В ОС СН реализован прикладной программный интерфейс (API) подсистемы безопасности PARSEC для разработки ПО, предназначенного для обработки информации с различными уровнями конфиденциальности.

Как правило, при разработке программного средства, предназначенного для обработки информации, содержащейся в объектах ФС, от имени пользователя, запустившего данное программное средство, или при разработке программного средства, выполняющего отображение информации, обработка которой осуществляется некоторым сервисом, нет необходимости задействовать API подсистемы безопасности PARSEC.

Наиболее распространенной задачей, при решении которой возникает такая необходимость, является разработка сетевого сервиса, предназначенного для обработки и предоставления информации по запросам различных пользователей, имеющих разные полномочия на доступ к информации. В указанном случае рекомендуется использовать следующий обобщенный алгоритм работы сетевого сервиса при взаимодействии с пользователем в условиях мандатного разграничения доступа в ОС СН:

1) основной процесс сетевого сервиса, осуществляющего обработку информации с различными мандатными уровнями, открывает привилегированный слушающий сокет, имеющий возможность принимать соединения с любыми метками безопасности. Основной процесс сетевого сервиса должен обладать определенными привилегиями Linux и привилегиями подсистемы безопасности PARSEC (см. [2], подраздел 4.6):

2) выполняется установка соединения;

3) выполняется идентификация и аутентификация пользователя, от имени которого функционирует клиентский процесс;

4) в случае успешного завершения аутентификации сетевой сервис создает дочерний процесс, в котором будет осуществляться обработка информации в контексте пользователя;

5) в дочернем процессе с сокета, обслуживающего соединение, снимается метка безопасности, передаваемая по сети;

6) в дочернем процессе осуществляется установка необходимых привилегией Linux и привилегий подсистемы безопасности PARSEC;

7) осуществляется переключение дочернего процесса в контекст пользователя (установка мандатного уровня, мандатных категорий, `gid` и `uid` пользователя);

8) в дочернем процессе осуществляется сброс привилегией Linux и подсистемы безопасности PARSEC.

Дальнейшая обработка информации при взаимодействии с пользователем в рамках установленного соединения осуществляется в контексте дочернего процесса.

Пользователь, от имени которого запускается основной процесс сетевого сервиса, должен обладать следующими привилегиями Linux:

- `CAP_SETGID` — для переключения дочернего процесса в работу с правами группы пользователя, являющегося клиентом сервиса;
- `CAP_SETUID` — для переключения дочернего процесса в работу с правами пользователя, являющегося клиентом сервиса.

Примечание. Если разрабатываемый сетевой сервис должен ожидать входящие соединения с использованием значения порта меньшего либо равного `1024`, необходимо использовать привилегию Linux `CAP_NET_BIND_SERVICE`.

Кроме того, пользователь, от имени которого запускается основной процесс сетевого сервиса, должен обладать следующими привилегиями подсистемы безопасности PARSEC (см. [2], подраздел 4.6):

- `PARSEC_CAP_PRIV_SOCK` — для создания привилегированного сокета, имеющего возможность принимать соединения с любыми метками безопасности;
- `PARSEC_CAP_SETMAC` — для переключения дочернего процесса в работу с мандатным контекстом пользователя, являющегося клиентом сервиса.

При разработке сетевого сервиса, предназначенного для обработки и предоставления информации по запросам различных пользователей, имеющих разные полномочия на доступ к информации, рекомендуется использовать принцип наименьших привилегией. Названный принцип означает, что процесс должен обладать только теми привилегиями, которые необходимы для решения его функциональных задач. По завершении решения

функциональных задач, требующих наличия привилегией, необходимо сбросить привилегии и продолжить функционирование в соответствии с общими ПРД.

Далее приведен пример реализации рассмотренного обобщенного алгоритма работы сетевого сервиса на языке С.

Пример

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/prctl.h>
#include <sys/types.h>
#include <sys/capability.h>
#include <unistd.h>
#include <linux/prctl.h>
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
#include <inttypes.h>
#include <netdb.h>
#include <parsec/mac.h>
#include <parsec/parsec_integration.h>
#include <parsec/parsec_mac.h>
#include <pwd.h>
#include <fcntl.h>
#include <errno.h>

#define T(fmt, ...) do { \
    fprintf( stderr, "%s %s %d:", __FILE__, __FUNCTION__, __LINE__); \
    fprintf( stderr, fmt, ##__VA_ARGS__); \
    fprintf( stderr, " erorr: %m\n"); \
    exit(1);\
} while(0)

static volatile int keepRunning = 1;

static void intHandler(int dummy) {
    keepRunning = 0;
}
```

```
static int child_process( int clnt_sock)
{
    char user[128];

    // код, выполняемый в дочернем процессе,
    // предназначенном для обработки информации в контексте пользователя
    // объявление переменной для получения с сокета соединения
    // передаваемой по сети метки безопасности клиентского процесса пользователя ра
    // объявление переменной для получения uid и gid пользователя
    struct passwd* pwd = NULL;

    // объявление переменной, используемой при получении
    // привилегий Linux дочернего процесса
    struct __user_cap_header_struct cur_header;

    printf( "child started\n");
    //объявление переменной для установки привилегий Linux
    // и ее начальная инициализация
    linux_caps_t cur_lcaps={0,0,0};

    // объявление переменной для установки привилегий PARSEC
    // и ее начальная инициализация
    parsec_caps_t cur_pcaps = {0,0,0};

    //инициализация списка разрешенных привилегий Linux
    cur_lcaps.permitted |= CAP_TO_MASK(CAP_SETUID);
    cur_lcaps.permitted |= CAP_TO_MASK(CAP_SETGID);

    //инициализация списка разрешенных привилегий PARSEC
    cur_pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_SETMAC);

    //инициализация списка действующих (эффективных) привилегий Linux
    cur_lcaps.effective |= CAP_TO_MASK(CAP_SETUID);
    cur_lcaps.effective |= CAP_TO_MASK(CAP_SETGID);

    //инициализация списка действующих (эффективных) привилегий PARSEC
    cur_pcaps.cap_effective |= CAP_TO_MASK(PARSEC_CAP_SETMAC);
```

```
//установка привилегий Linux и PARSEC дочернего процесса сетевого сервиса
if(parsec_cur_caps_set(&cur_lcaps,&cur_pcaps) != 0) {
    T("parsec_cur_caps_set");
}

// инициализация переменной, используемой при получении
// привилегий Linux дочернего процесса
cur_header.version = _LINUX_CAPABILITY_VERSION;
cur_header.pid = 0;

// получение привилегий Linux для дочернего процесса
if(capget((cap_user_header_t)(&cur_header), &cur_lcaps ) != 0) {
    T("cap_user_header_t");
}

// проверка наличия необходимых эффективных привилегий Linux
// у дочернего процесса
if((!(cur_lcaps.effective & CAP_TO_MASK(CAP_SETUID)))  ||
    (!(cur_lcaps.effective & CAP_TO_MASK(CAP_SETGID)))) {
    T("cur_lcaps.effective");
}

// получение привилегий PARSEC для дочернего процесса
if(parsec_capget(0, &cur_pcaps ) != 0) {
    T("parsec_capget");
}

// проверка наличия необходимых эффективных привилегий PARSEC у дочернего проце
if(!(cur_pcaps.cap_effective & PARSEC_CAP_TO_MASK(PARSEC_CAP_SETMAC))) {
    T("cur_pcaps.cap_effective");
}

// получение с сокета соединения передаваемой по сети
// метки безопасности клиентского процесса пользователя
if(parsec_fstatmac(clnt_sock,&mac_label)) {
    T("parsec_fstatmac");
}
```

```
printf( "С сокета получена метка безопасности type=%x mac.lev=%d mac.cat=0x%"PI
                              mac_label.type, mac_label.mac.lev, mac_label.mac.ca
// установка на дочерний процесс метки безопасности,
// полученной с сокета соединения
if(parsec_setmac(0,&mac_label.mac) < 0) {
    T("parsec_setmac");
}

// запрос имени пользователя
char prompt[] = "login:";
if( send( clnt_sock, prompt, strlen( prompt), 0) != strlen( prompt)) {
    T("send");
}

// выполнение идентификации и аутентификации пользователя,
// от имени которого функционирует клиентский процесс,
// установка значения переменной user

int rc;
rc = recv( clnt_sock, user, sizeof(user)-1, 0);
if( rc <= 0) {
    T("recv");
}
user[rc-2]=0;
printf( "username is \"%s\"\n", user);

// получение gid и uid для пользователя клиентского процесса,
// прошедшего идентификацию и аутентификацию
pwd=getpwnam(user);
if(!pwd) {
    T("getpwnam");
}

printf( "user = %s UID=%d GID=%d\n", user, pwd->pw_uid, pwd->pw_gid);

// переключение дочернего процесса в работу с правами группы пользователя,
// прошедшего идентификацию и аутентификацию и являющегося клиентом
// сетевого сервиса
```

```
    // переключение выполняется до вызова setuid(...)
    if(setgid(pwd->pw_gid)) {
        T("setgid");
    }
    // переключение дочернего процесса в работу с правами пользователя,
    // прошедшего идентификацию и аутентификацию и являющегося клиентом
    // сетевого сервиса
    if(setuid(pwd->pw_uid)) {
        T("setuid");
    }
    // сброс привилегий Linux и PARSEC дочернего процесса сетевого сервиса
    if(parsec_cur_caps_set(0,0) != 0) {
        T("parsec_cur_caps_set");
    }
    // код, выполняемый в дочернем процессе,
    // для обработки и предоставления информации пользователю
    // клиентского процесса

    shutdown( clnt_sock, SHUT_RDWR);
    close( clnt_sock);
    printf( "child ended\n");
    exit(0);
}

/*
*****************************************************************************
*/

int main( int argc, char **argv)
{
    int sock = 0;
    int clnt_sock = 0;
    int addr_len = 0;

    socklen_t addrlen;
    struct sockaddr_in serv_addr, clnt_addr;
    pid_t child;
```

```
signal(SIGINT, intHandler);

//объявление переменной для установки привилегий Linux и ее начальная инициали
linux_caps_t lcaps={0,0,0};

//объявление переменной для установки привилегий PARSEC и ее начальная инициал
parsec_caps_t pcaps = {0,0,0};

memset(&clnt_addr,0,sizeof(clnt_addr));
addr_len = sizeof(clnt_addr);
serv_addr.sin_port = htons(7777);

//инициализация списка разрешенных привилегий Linux
lcaps.permitted |= CAP_TO_MASK(CAP_SETUID);
lcaps.permitted |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка действующих (эффективных) привилегий Linux
lcaps.effective |= CAP_TO_MASK(CAP_SETUID);
lcaps.effective |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка наследуемых привилегий Linux
lcaps.inheritable |= CAP_TO_MASK(CAP_SETUID);
lcaps.inheritable |= CAP_TO_MASK(CAP_SETGID);

//инициализация списка разрешенных привилегий PARSEC
pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_SETMAC);
pcaps.cap_permitted |= CAP_TO_MASK(PARSEC_CAP_PRIV_SOCK);

//инициализация списка действующих (эффективных) привилегий PARSEC
pcaps.cap_effective|=CAP_TO_MASK(PARSEC_CAP_SETMAC);
pcaps.cap_effective|=CAP_TO_MASK(PARSEC_CAP_PRIV_SOCK);

//инициализация списка наследуемых привилегий PARSEC
pcaps.cap_inheritable|=CAP_TO_MASK(PARSEC_CAP_PRIV_SOCK);
pcaps.cap_inheritable|=CAP_TO_MASK(PARSEC_CAP_SETMAC);

//установка флага наследования привилегий
if(prctl(PR_SET_KEEPCAPS,1)) {
```

```
    T("prctl");
}

//установка привилегий Linux и PARSEC основного процесса сетевого сервиса
if(parsec_cur_caps_set(&lcaps,&pcaps) < 0) {
    T("parsec_cur_caps_set");
}

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);

// создание привилегированного сокета
printf("создание привилегированного сокета\n");
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    T("socket");
}

// привязка привилегированного сокета
printf("привязка привилегированного сокета\n");
if(bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))) {
    T("bind");
}
printf("привязка привилегированного сокета выполнена\n");

// включение ожидания запроса на соединение на привилегированном сокете
if(listen(sock,5) < 0) {
    T("listen");
}

// перевод сокета в неблокирующееся состояние
if( fcntl( sock, F_SETFL, O_NONBLOCK) < 0 ) {;
    T("fcntl");
}

while( keepRunning) {
    // создание соединения на привилегированном сокете
    clnt_sock = accept(sock,(struct sockaddr*)&clnt_addr,&addr_len);
    if( clnt_sock < 0) {
```

```
            if( errno == EAGAIN)
                continue;
            T("accept");
        }
        printf( "Access detected\n");

        // создание дочернего процесса для обработки информации в контексте пользо
        child = fork();
        if(child<0) {
            T("fork");
        }

        if(!child) {
            child_process( clnt_sock);
        }
    }
    shutdown( sock, SHUT_RDWR);
    close( sock);
    printf( "Done\n");
    return 0;
}
```

Рекомендуется сбросить в основном процессе привилегию PARSEC_CAP_PRIV_SOCK, как только исчезла необходимость ее использования, поскольку любые сокеты при наличии эффективной названной привилегии будут создаваться вызовом функции socket(...) как привилегированные.

Для сборки и запуска примера нужно скопировать код в отдельный файл, и использовать следующую команду:

```
gcc имя_файла_примера -lparsec-base -lparsec-mac && sudo ./a.out
```

В примере используется номер порта 7777, и для подключения к запущенному примеру можно использовать команду telnet:

```
telnet <IP_адрес_компьютера> 7777
```

В случае локального компьютера можно использовать IP-адрес 127.0.0.1

Вместо используемой в примере привилегии PARSEC_CAP_PRIV_SOCK для создания привилегированного слушающего сокета, принимающего соединения с любыми метками безопасности, может быть использована следующая комбинация привилегий PARSEC:

- PARSEC_CAP_CHMAC — предоставляет право менять метку безопасности файла;
- PARSEC_CAP_MAC_SOCK — предоставляет право изменять метку безопасности сетевого соединения.

При наличии указанных привилегий после создания сокета необходимо с использованием функции `parsec_fchmac(...)` установить на сокет мандатные атрибуты игнорирования мандатных уровней и категорий при выполнении операция чтения и записи как показано в примере.

Пример

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    // обработка ошибки
}

parsec_mac_label_t m_label;
m_label.type = MAC_ATTR_IGNORE;
m_label.mac.lev=0;
m_label.mac.cat=0;

if(parsec_fchmac(sock,&m_label) != 0)
{
    // обработка ошибки
}

if(bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)))
{
    // обработка ошибки
}
```

4. ОПИСАНИЕ ИСПОЛЬЗОВАНИЯ API СИСТЕМЫ РАСШИРЕННОГО АУДИТА

Для реализации функционала по аудиту рекомендуется использовать интегрированную систему аудита, которая входит в состав ОС СН.

Для получения подробного описания интегрированной системы аудита необходимо получить описание API и прикладных утилит вызовом электронной справки в ОС СН.

Доступ к электронной справке в системе осуществляется либо на рабочем столе Fly, либо вызовом справочной документации `man`.

В частности, для системы аудита вызывать команду `man` по следующему списку функций API и утилит:

```
aud_copy_ext
aud_get_type
aud_set_type
useraud
aud_create_entry
aud_init
aud_to_text
psaud
aud_dup
aud_next_entry
aud_valid
setaudent_r
aud_from_text
aud_set_file
getfaud
setaudent
aud_get_file
aud_set_pid
setfaud
```

5. РАЗРАБОТКА ПО ДЛЯ ВЗАИМОДЕЙСТВИЯ С СУБД POSTGRESQL

СУБД PosgreSQL предоставляет доступные 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. Основная документация по общим вопросам использования СУБД PostgreSQL расположена на официальном сайте разработчиков (https://www.postgresql.org/docs/11/index.html).

5.1. Мандатное разграничение доступа в СУБД PostgreSQL

Расширение команд SQL, используемое при мандатном разграничении в СУБД PostgreSQL, приведено в Руководстве [2] (подраздел 4.14).

6. РАЗРАБОТКА ПО ДЛЯ WEB-СЕРВЕРА

При необходимости обеспечения сквозной аутентификации из скриптов, запускаемых на WEB-сервере, которые должны взаимодействовать со другими службами в режиме ЕПП, например, с защищенным сервером СУБД, в конфигурационном файле виртуального хоста следует дополнительно указать:

```
KrbSaveCredentials on
```

В настройках браузера Mozilla Firefox необходимо задать в качестве значений параметра `network.negotiate-auth.delegation-uris`, маски доменов которым можно передавать данные для сквозной аутентификации. В запускаемых скриптах необхоимо выставить переменную окружения `KRB5CCNAME`. Например, для скрипта на языке PHP установка значения переменной окружения будет иметь следующий вид:

```
putenv("KRB5CCNAME="._$SERVER['KRB5CCNAME']);
```

7. ИНТЕГРАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВ MICROSOFT WINDOWS И LINUX С ОС СН

Анализ возможностей совместного применения ОС СН и других операционных систем в автоматизированных системах в защищенном исполнении показал, что для успешного решения данной задачи необходимо обеспечить:

- совместимость при сетевом взаимодействии в условиях мандатного разграничения доступа;
- совместимость с единым пространством пользователей (ЕПП) в ОС СН;
- совместимость с системой централизованного протоколирования ОС СН.

Для обеспечения совместимости при сетевом взаимодействии в условиях мандатного разграничения доступа в ОС СН передача в поле опций каждого пакета IPv4 классификационной метки сформировавшего его процесса (субъекта доступа) реализована в соответствии с национальным стандартом ГОСТ Р 58256-2018 «Защита информации. Управление потоками информации в информационной системе. Формат классификационных меток», и совместима с международным стандартом RFC 1108. Использование указанных стандартов обеспечивает в том числе совместимость с ОС семейства МСВС. Подробная информация приведена в 3.3.1.

7.1. Интеграция в ЕПП

Интеграция компьютеров под управлением ОС СН в системы ЕПП возможна как в роли клиента ЕПП, так и в роли сервера ЕПП (контроллера домена). ОС СН поддерживает работу в гетерогенных ЕПП, в которых клиентами и серверами могут выступать компьютеры под управлением ОС различных семейств (Microsoft Windows, Linux, Astra Linux Common Edition, Astra Linux Special Edition). При этом важным ограничением является то, что средства мандатного разграничения доступа полностью поддерживаются только в ЕПП состоящей из компьютеров под управлением ОС СН. Компьютеры под управлением других ОС могут входить в ЕПП под управлением ОС СН, но получить доступ к информации имеющей ненулевую классификационную метку они не смогут.

В общем случае в основу ЕПП положен доменный принцип построения сети, подразумевающий объединение в одну сеть логически связанных компьютеров, например, принадлежащих одной автоматизированной системе. При этом пользователи получают возможность работы с разделяемыми сетевыми ресурсами и взаимодействия с другими пользователями.

Организация ЕПП предоставляет следующие возможности:

- сквозная аутентификация в сети;
- централизация хранения информации об окружении пользователей;

- централизация хранения настроек системы защиты информации на сервере.

Сетевая аутентификация и централизация хранения информации об окружении пользователя подразумевает использование двух основных механизмов: NSS и PAM.

Сквозная доверенная аутентификация реализуется технологией Kerberos.

В качестве источника данных для базовых системных служб на базе механизмов NSS и PAM используется служба каталогов, работающая по стандарту LDAP.

Централизация хранения информации об окружении пользователей подразумевает и возможность централизованного хранения домашних каталогов пользователей. Для этого используется сетевая защищенная файловая система (СЗФС) CIFS (известная также как SMB).

В состав ОС СН входят следующие системы управления ЕПП:

- FreeIPA;
- ALD;

Эти системы являются надстройками над технологиями LDAP, Kerberos, CIFS и обеспечивают автоматическую настройку всех необходимых служб, реализующих перечисленные технологии, а так же предоставляют интерфейс управления и администрирования. Также в состав ОС ОН входит клиентское ПО для включения компьютеров под управлением ОС ОН в ЕПП в качестве клиентов:

- клиентское ПО FreeIPA;
- клиентское ПО ALD;
- клиентское ПО Samba (для включения в ЕПП Windows AD);
- клиентское ПО SSSD (для включения в ЕПП Windows AD);

Процедуры настройки серверов и клиентов описаны в соответствующей документации и на web-сайте `wiki.astralinux.ru`.

Ряд специфичных для ОС СН функций ЕПП (ограничение входа в систему, назначение локальных групп пользователям домена, подключение домашних сетевых каталогов и т.п.) для других ОС семейства Linux может быть доступен при условии использования специализированного PAM-модуля или посредством разработки и использования дополнительных PAM-модулей типа `pam_group`, `pam_mount` и т. п.

7.1.1. Служба каталогов

В ALD в качестве службы каталогов используется LDAP-сервер OpenLDAP, во FreeIPA в качестве службы каталогов используется LDAP-сервер 389-ds. В дополнение к серверным службам каталогов в ЕПП могут использоваться клиентские кеширующие службы, позволяющие уменьшить объёмы сетевого трафика (служба SSSD на клиентах FreeIPA). Взаимодействие со службами каталогов может осуществляться как с помощью набора утилит OpenLDAP (как правило, входящих в пакет ldap-utils), так и с помощью API

(libldap).

Для хранения информации в службах каталогов используются форматы, схемы данных и подходы, регламентированные следующими стандартами:

- RFC 4515 – Lightweight Directory Access Protocol (LDAP). Описание структуры запросов к службе каталогов.
- RFC 1274 – The COSINE and Internet X.500 Schema. Описание работы с атрибутами и классами.
- RFC 2307, RFC 2307 bis – An Approach for Using LDAP as a Network Information Service. Схема атрибутов и классов для хранения информации об учетных записях пользователей и другой информации, необходимы для организации домена.
- RFC 2798 – Definition of the inetOrgPerson LDAP Object Class. Схема атрибутов и классов, расширяющих класс пользователя дополнительными сведениями.

Разработчики, решающие задачу интеграции операционных систем или отдельных служб с ЕПП ОС СН, могут получить сведения об используемой схеме LDAP (описание атрибутов, классов и порядок доступа к ним) посредством развертывания стенда, включающего компьютер, под управлением ОС СН с установленным и настроенным сервером ЕПП. Подробная информации о ЕПП приведена документе [**?**] в подразделе 8.

7.1.2. NSS

Механизм NSS предоставляет всем программам и службам, функционирующим на локальном компьютере, системную информацию через соответствующие программные вызовы. Он обращается к конфигурационному файлу `/etc/nsswitch.conf`, в котором указаны источники данных для каждой из системных служб. Для работы в составе ЕПП на компьютерах-клиентах в качестве источника данных должна быть указана соответствующая служба каталогов (LDAP для ALD, или sss (кеширующая служба) для FreeIPA, winbind для Samba, sss для SSSD). В состав ОС ОН входят инструменты, автоматически выполняющие все необходимые настройки (см. конфигурационный файл `/etc/nsswitch.conf`).

7.1.3. Аутентификация

Для сквозной аутентификации используется MIT-реализация Kerberos. Входящие в состав ОС ОН инструменты установки клиентов ЕПП автоматически выполняют все необходимые для работы в ЕПП настройки компьютеров под управлением ОС ОН. На компьютерах под управлением других версий Linux должны быть установлены соответствующие пакеты:

- `krb5-config` – конфигурационные файлы;
- `krb5-user` – клиентские программы аутентификации Kerberos;
- `libpam-krb5` – PAM модуль для аутентификации пользователей при входе в систему;

при этом должна использоваться версия MIT-Kerberos и должен быть соответствующим образом настроен конфигурационный файл /etc/krb5.conf:

```
[libdefaults]
      default_realm = <REALM>
...
[login]
      krb5_get_tickets = true
...
[realms]
      <REALM> = {
      kdc = <сервер ALD>
}
```

Где в качестве <REALM> указывается область Kerberos, соответствующая домену (например, для домена .example.ru таким значением будет EXAMPLE.RU).

Поскольку в конфигурационном файле указывается дополнительная информация по используемым криптографическим преобразованиям и т.п. целесообразно скопировать рассматриваемый конфигурационный файл непосредственно с компьютера под управлением ОС СН, входящего в тот же домен.

Проверка корректности настройки может быть выполнена проведением аутентификации от имени пользователя домена с использованием следующей команды:

```
# kinit <пользователь ЕПП>
```

В случае успеха аутентификации будет получен TGT билет Kerberos, который можно просмотреть с помощью утилиты klist:

```
# klist
```

Для возможности аутентификации пользователя при входе в систему необходимо соответствующим образом настроить PAM сценарий входа с использованием PAM модуля pam_krb5, pam_sss или pam_winbind. Вариант PAM сценария login с использованием PAM модуля pam_krb5.

Вариант PAM сценария login:

```
auth [success=ignore default=die] pam_tally.so per_user deny=10
auth [success=2 default=ignore] pam_unix.so
auth [success=1 default=ignore] pam_krb5.so minimum_uid=<MINUID>
auth requisite pam_deny.so
auth required pam_permit.so
```

В аргументе minimum_uid должен быть указан минимальный идентификатор пользователя домена.

7.1.4. Администрирование

Административные действия по вводу компьютера в домен и настройке локальных служб для работы в ЕПП могут быть выполнены либо прямыми изменениями в ЕПП с помощью утилит Kerberos и ldap с добавляемой системы, либо путем операций в домене средствами ОС СН с последующими дополнительными действиями, выполняемыми на локальной системе.

Например: добавление пользователей, создание учетных записей служб и управление принадлежностью к группам могут выполнятся администратором ЕПП, а выгрузка ключей служб и их настройка для работы с ЕПП должны выполняться на локальной системе.

Порядок введения компьютера в домен:

1) Должно быть настроено разрешение имен для разыменования полного имени сервера - контроллера домена и самого компьютера.

2) Для вводимого компьютера должна быть создана учетная запись Kerberos вида `host/<полное имя компьютера>` и добавлена в соответствующую ветку службы каталогов.

3) Ключи учетной записи должны быть выгружены средствами Kerberos (утилитой `ktutil`) системы в локальный файл `/etc/krb5.keytab`.

Настройка сетевых служб для работы с аутентификацией по Kerberos выполняется согласно документации на эти службы. Примеры создания учетных записей для сетевых служб приведены в документе [3], подраздел 8.

7.2. Интеграция с системой централизованного протоколирования ОС СН

Средства централизованного протоколирования основываются на программном пакете Zabbix. Порядок решения задачи централизованного сбора и анализа журналов регистрации событий в ОС СН описан в документе [3], подраздел 15.2. Таким образом, для взаимодействия с сервером централизованного протоколирования в ОС семейства Linux могут быть использованы стандартные реализации агентов Zabbix.

Кроме того, для ОС семейства Linux существует возможность применения службы `rsyslog` для передачи данных о регистрируемых событиях на удаленный сервер централизованного протоколирования. Пример использования службы `rsyslog` в ОС СН приведен документе [3], подраздел 8.2.6.5.

8. РЕКОМЕНДАЦИИ ПО СПОСОБАМ МИГРАЦИИ ПРИЛОЖЕНИЙ В СРЕДУ ОС СН С ДРУГИХ ПЛАТФОРМ

8.1. Особенности переноса программ с 32-разрядных платформ на 64-разрядную

Задача переноса существующих 32-разрядных приложений в среду 64-разрядной ОС СН может решаться различными методами в зависимости от тех или иных особенностей данных приложений. В некоторых простых случаях для работы таких программ достаточно настройки режима исполнения 32-разрядных приложений (8.2). Но, в общем случае, перенос на 64-разрядную платформу потребует внимательного анализа исходных текстов программ с целью выявления особых участков программного кода, которые необходимо исправить или заново переписать (подробнее об этом в статье [29]).

Объем работ по переводу программных средств на 64-разрядную платформу ОС СН варьируется в зависимости от ответов на следующие ключевые вопросы:

1) В какой мере программные средства используют программный интерфейс СЗИ ОС?
2) Какие используются версии библиотек графического интерфейса?
3) Имеется ли в составе программных средств функционал, работающий в пространстве ядра ОС (например, специализированные модули ядра)?

ОС СН является развитием стандартных Linux-дистрибутивов, поэтому для переноса в ее среду программ, не взаимодействующих в процессе своей работы со встроенными СЗИ, можно использовать весь набор рекомендаций, касающийся разработки, переноса и миграции ПО с других платформ в ОС Linux. При этом надо учесть особенности как 64-разрядной процессорной архитектуры и ее поддержки ядром Linux, так и новых системных и графических библиотек (см. раздел 2).

Основная системная библиотека ОС СН — `glibc` — предполагает некоторую обратную совместимость с предыдущими ее версиями в части программного интерфейса. Таким образом, многие старые программы смогут с ней корректно работать. Напротив, у графических библиотек такая обратная совместимость не наблюдается. В связи с этим в состав ОС СН помимо новой прикладной графической библиотеки Qt 5 входит и библиотека предыдущего поколения Qt 4 с поддержкой Qt 3. Более ранние версии библиотек Qt 1 и Qt 2 в ОС СН не поддерживаются.

Процесс портирования становится намного сложнее в том случае, если в состав программного средства входят специализированные модули, созданные под конкретные версии устаревших ядер операционных систем. В связи с большими различиями в архитектурах построения новых и старых ядер разработчикам потребуется почти полная переработка

этих модулей на базе нового интерфейса (достаточно полное его описание можно найти в руководстве [30]). Советы по переносу программ управления устройствами с платформы Windows в Linux можно найти в статье [**?**].

Если в программном средстве явно используется специфический программный интерфейс СЗИ, то необходимо учитывать особенности реализации мандатного разграничения доступа на конкретной платформе. Следует учесть, что из-за более высокой разрядности в ОС СН диапазоны мандатных уровней и категорий, а также набор специальных системных вызовов шире, чем на ОС предыдущих поколений (см. раздел 3). Для ускорения процесса перевода таких программных средств может потребоваться написание с помощью макросов С-препроцессора «программ-оберток», отображающих старый программный интерфейс СЗИ на новый.

8.2. Настройка среды исполнения 32-разрядных приложений

Процессоры с архитектурой x86-64 (AMD64) способны работать как в 64-разрядном, так и 32-разрядном режимах. Несмотря на то, что репозитории ОС СН не содержат 32-битные пакеты, сама ОС СН поддерживает технологию множественных архитектур (Multiarch), обеспечивающую возможность установки и выполнения 32-битных приложений, которые могут быть установлены из сторонних репозиториев. Для включения возможности работы с 32-битными пакетами и приложениями следует:

1) Включить работу с 32-битными приложениями командой:
   ```
   sudo dpkg --add-architecture i386
   ```
2) Подключить репозиторий с нужными пакетами;
3) Обновить списки пакетов:
   ```
   sudo apt update
   ```
4) Выполнить установку пакетов командой:
   ```
   apt install <имя_пакета>
   ```

Если рассматривать небольшие 32-разрядные Linux-приложения для x86, которые статически связаны со всеми необходимыми библиотеками, то они с большой вероятностью будут успешно работать в среде 64-разрядной ОС СН. Для работы 32-битных приложений использующих динамически подгружаемые библиотеки в ОС СН необходимо настроить среду выполнения 32-разрядных программ. Для этого следует установить пакет `ia32-libs`, содержащий основные системные разделяемые 32-разрядные библиотеки. Для установки пакета выполнить команду:

```
sudo apt install ia32-libs
```

Это создаст в корневом каталоге папку `/emul/ia32-linux` с 32-разрядными библиотеками. По умолчанию они поддерживают только консольные приложения, базовые X- и GTK-приложения. Для Qt-приложений необходимо самостоятельно установить 32-разрядные

библиотеки в каталог /emul/ia32-linux. Также надо поступать и с другими недостающими 32-разрядными пакетами. Сделать это можно, например, так:

```
sudo dpkg -X <имя_пакета>-<версия>_i386.deb /emul/ia32-linux
```

8.3. Перенос программ из среды Windows

При переносе программных средств автоматизированных комплексов из ОС Windows на платформу ОС СН можно руководствоваться общими подходами, разработанными по миграции в среду Linux. Также следует ознакомиться с рекомендациями в книге [28] при переносе программ управления устройствами (драйверов), если таковые имеются в системе.

8.4. Технология компиляции при переносе ПО

При переносе ПО может потребоваться перекомпиляция 32-разрядных программ в том виде, как они написаны, без адаптации к 64-разрядной системе. Далее эти программы можно будет запускать на ОС СН при правильно настроенной 32-разрядной среде исполнения (см. 8.2). Чтобы компиляция в ОС СН проходила корректно и давала на выходе 32-разрядные исполняемые приложения, необходимо использовать для компилятора gcc — ключ -m32, для компоновщика программ ld — ключ -melf_i386. Например, если сборка программы осуществлялась с помощью средств Autotools, то могут применяться следующие опции для configure:

```
./configure --target=i386-linux --cc="gcc -m32" --as="as --32" \\
                                          --with-extralibdir=/usr/lib32
```

В качестве значений параметра --with-extralibdir= необходимо указывать каталоги, где расположены требуемые для сборки приложения 32-разрядные библиотеки. В случае, если перенос осуществляется из среды Windows, компиляция может требовать дополнительные настройки, зависящие от особенностей портируемого исходного кода. В любом случае при компиляции следует включать отладочные предупреждения (опции компилятора -Wall, -Wextra — см. [31]) и корректировать исходный код, добиваясь минимизации количества предупреждений.

9. МОБИЛЬНАЯ СЕССИЯ

При разработке ПО ориентированного на мобильные устройства, рекомендуется применять технологии Qt/QML. Чтобы использовать мобильный стиль Fly нужно взять за основу приложения библиотеку компонентов `fly-qml-components`.

9.1. Пакет fly-qml-components

Библиотека `fly-qml-components` (пакет `fly-qml-components`) представляет собой набор готовых QML компонентов, которые можно использовать в создаваемых приложениях. Полный набор компонентов находится в каталоге: `/usr/lib/x86_64-linux-gnu/qt5/qml/Fly/Components`. Для подключения библиотеки необходимо в `.qml` файле указать: `import Fly.Components 1.0` Базовый компонент — `FlyRoot`, основанный на `QQuickWindow`. Самому создавать окно не нужно, достаточно использовать `QQmlApplicationEngine`. В основе лежит постраничное отображение. Для добавления страницы (FlyPage) используется функция `root.addPage(<pathToPage>, {properties})`. При нажатии на «x», кнопку «Отмена» или кнопку «Назад» страница автоматически закроется и вызовет `cancelFunction`. При нажатии на «ОК» вызовется `okFunction`.В базовом компоненте есть функции для создания и отображения некоторых основных виджетов с заданными параметрами. Можно отобразить виджеты сообщения, списка, выбора числа, календаря, строки ввода, выбора цвета. Например, следующий код отобразит страницу сообщения:

```
root.showMessagePage({
    "title": qsTr("Warning"),
    "message": qsTr("Hello!"),
    "showCancelButton": true,
    "okFunction": function() { /*do something*/ }
})
```

Основные цвета содержатся в компоненте `FlyStyle`, который доступен по ссылке `styles`. Например:

```
Rectangle { color: styles.colors.white }
```

При указании размеров элементов желательно отталкиваться от `root.itemHeight`. Для отображения изображения из темы (например, логотипа) используется следующая запись:

```
Image { source: "image://ThemeImageProvider/astra" }.
```

9.2. Структура приложения

Базовая структура приложения следующая: Файл `main.cpp`:

```
QQmlApplicationEngine engine;
engine.load(QUrl(QStringLiteral("qrc:///main.qml")));
```

Файл main.qml:

```
import QtQuick 2.3
import Fly.Components 1.0

FlyRoot {
    id: root
    minimumWidth: 600; minimumHeight: 600
    title: qsTr("Application")
    icon: "my-application"
    initialPage: Component { MainPage { focus: true } }
}
```

Файл MainPage.qml:

```
import QtQuick 2.3
import Fly.Components 1.0

FlyPage {
    id: mainPage
    frontRect.visible: false
    centerRect.visible: false
    backRect.color: styles.colors.white
    FlyShadow {
        target: toolbar;
        horizontalOffset: 0;
        verticalOffset: 1 * dpiRatio;
        z: 1
    }
    Rectangle {
        id: toolbar
        anchors {
            left: parent.left;
            top: parent.top;
            right: parent.right
        }
        height: root.itemHeight
        color: styles.page.titleColor
        z: 1
        /* toolbar content /
    }
```

```
    Item {
        anchors {
            left: parent.left;
            top: toolbar.bottom;
            right: parent.right; bottom: parent.bottom
        }
        /* main content */
    }
}
```

9.3. Клавиша «Назад»

При нажатии на клавишу «Назад» (левая кнопка в нижней панели) генерируется нажатие Alt+F4. Соответственно чтобы отловить нажатие на эту кнопку нужно обработать событие закрытия окна. При использовании `fly-qml-components`, открытые страницы(FlyPage) будут автоматически закрываться при нажатии на кнопку «Назад». При этом будет вызвана функция `cancelFunc`. Есть возможность добавления своей команды. Например, следующий код выведет в консоль «Hello»:

```
Item {
    function processBackClick() {
        root.popFunctionFromBackStack();
        console.log("Hello");
    }
    Component.onCompleted: root.addFunctionToBackStack(processBackClick);
}
```

При использовании функции `addFunctionToBackStack` в обработчике нужно самостоятельно вызывать `root.popFunctionFromBackStack()` или `root.clearBackStack()`, в зависимости от логики приложения.

9.4. Виртуальная клавиатура

При установленном пакете `qtvirtualkeyboard` виртуальная клавиатура будет автоматически показана при фокусе на текстовом поле в приложениях на базе Qt. Есть 2 способа создания клавиатуры: встроенный в приложение и внешний. При использовании `fly-qml-components` виртуальная клавиатура встраивается в приложение и плавно выезжает при отображении. При этом она поджимает только самую верхнюю страницу. Для доступа к клавиатуре используйте `root.virtualKeyboard`. Например, чтобы установить английскую раскладку можно вызвать `root.virtualKeyboard.setEnglishLayout()`. При использовании горизонтальной ориентации экрана, если высота клавиатуры будет больше 70 % высоты экрана, то на пустом месте будет показано отдельное текстовое поле вво-

да. В приложениях без использования `fly-qml-components`, клавиатура отображается вне приложения поджимая его снизу (без анимации). В коде приложения дополнительно ничего писать не нужно. Обратиться к ней можно только через `Qt.inputMethod`. Используйте `Qt.inputMethod.visible` чтобы узнать отображается ли клавиатура. Используйте флаги inputMethodHints чтобы настроить клавиатуру. Более подробно можно почитать тут: `http://doc.qt.io/QtVirtualKeyboard/deployment-guide.html`

9.5. Основные компоненты fly-qml-components

Описание основных компонентов:

- `FlyButton` — стандартная кнопка. Используется в основном в различных формах;
- `FlyCloseButton` — кнопка закрытия форм;
- `FlyControlButton` — круглая кнопка с иконкой и текстом. Используется в панели управления;
- `FlyMenuButton` — слева иконка, справа текст. Можно указать только иконку или текст. Используется в панелях инструментов (тулбарах);
- `FlyColoredImage` — изображение с возможностью затенения, наложения цвета, обесцвечивания, добавления мини изображения в угол;
- `FlyColorPicker` — инструмент (виджет) выбора цвета;
- `FlyCalendar` — инструмент (виджет) календаря;
- `FlyLabel` — текст;
- `FlyLineEdit` — инструмент (виджет) однострочного поля ввода текста;
- `FlyTextEdit` — инструмент (виджет) многострочного поля ввода текста;
- `FlyListView` — инструмент (виджет) вертикального списка. В качестве модели должен быть использован массив. Поддерживает удаление элементов;
- `FlyListViewRowFlick` — инструмент (виджет) горизонтального списка. Используется в панели уведомлений;
- `FlySlider` — инструмент (виджет) горизонтального слайдера;
- `FlySpinBox` — инструмент (виджет) выбора числа;
- `FlyScrollBar` — используется для отображения полос прокруток;
- `FlySeparator` — вертикальная/горизонтальная линия;
- `FlyInnerShadow` — внутренняя тень. Должна быть поверх элемента;
- `FlyShadow` — внешняя тень. Должна быть под элементом;
- `FlyTouchAnimation` — анимация прикосновения к экрану;
- `FlyWindow` — используется при необходимости создания нескольких внутренних окон в одном приложении. Для каждого окна создается отдельная постраничная структура с поддержкой виртуальной клавиатуры.

9.6. Добавление пользовательского виджета в fly-launcher

Вы можете создать любой QML виджет основанный на `fly-qml-components`. Точка входа это `.qml` файл. Вы можете посмотреть примеры в пакете `fly-phone-widgets` (`/usr/lib/x86_64-linux-gnu/qt5/qml/Fly/Widgets`). Будьте осторожны с `Flickable` и сложной обработкой мыши. На текущий момент MouseArea находится поверх виджета и пропускает только составные события. URI должен быть `Fly.Widgets.MyWidget`, где `MyWidget` имя виджета. Простейший виджет календаря выглядит так:

```
import QtQuick 2.3
import Fly.Components 1.0
FlyCalendar {}
```

Структура виджета (файлы `.pro`, `qmldir`, `_plugin` опущены):

- `MyWidget` — Папка проекта. Последний пункт URI и имя папки должны совпадать;
- `MyWidget.qml` — Точка входа. Имя должно совпадать с последним пунктом в URI или быть указано в config файле;
- `MyWidget.png` — Изображение предпросмотра. Имя должно совпадать с последним пунктом в URI или быть указано в config файле;
- `MyWidget_ru.qm` — Файл перевода. Имя должно совпадать с последним пунктом в URI или быть указано в config файле;
- `config` — Конфигурационный файл. Имя: `'config'`;
- `Settings.qml` — Файл настроек. Видимые пользователю настройки виджета. Любое имя.

9.7. Пакет fly-phone-dbus

Данный пакет предоставляет несколько классов для работы с различными событиями. DBus интерфейсы находятся в `/usr/include/fly-phone/fly-phone-dbus/dbus`. Для подключения используйте:

```
CONFIG += link_pkgconfig
PKGCONFIG += fly-phone-dbus
```

Например, для того чтобы позвонить по номеру 123456789:

```
import QtQuick 2.3
import Fly.Components 1.0
Item {
    FlyCallDbusNotifier { id: callDbusNotifier }
    Component.onCompleted: callDbusNotifier.emitDial(123456789);
}
```

СПИСОК ЛИТЕРАТУРЫ

Общие вопросы построения операционной системы Linux

[1] Руководящие указания по конструированию прикладного программного обеспечения для операционной системы общего назначения «Astra Linux Common Edition», ОАО «НПО РусБИТех», 2010. (https://wiki.astralinux.ru/download/attachments/37290417/RUK-OSON-DEV.pdf)

[2] РУСБ.10015-01 97 01-1 «Операционная система специального назначения «Astra Linux Special Edition». Руководство по КСЗ. Часть 1»

[3] РУСБ.10015-01 95 01-1 «Операционная система специального назначения «Astra Linux Special Edition» Руководство администратора. Часть 1»

[4] ГОСТ Р 58256-2018 «Защита информации. Управление потоками информации в информационной системе. Формат классификационных меток»

[5] RFC 1108 — Security Options for the Internet Protocol — Параметры безопасности для IP-протокола

[6] Осаму Аоки: Справочник по Debian (http://qref.sourceforge.net/Debian/reference/index.ru.html)

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

[8] Д.Бовет, М.Чезати: Ядро Linux (3-е издание); БХВ-Петербург, 2007.

Стандарты открытого рабочего стола Freedesktop.org

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

[10] Icon Theme Specification (https://specifications.freedesktop.org/icon-theme-spec/icon-theme-spec-latest.html)

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

http://standards.freedesktop.org/icon-theme-spec)

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

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

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

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

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

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

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

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

[20] Кастомизация меню "Действия"в файловом менеджере fly-fm
(https://wiki.astralinux.ru/x/owWZAw)

[21] KAbstractFileItemActionPlugin
(https://api.kde.org/frameworks/kio/html/classKAbstractFileItemActionPlugin.html)

Книги по графической библиотеке Qt

[22] Жасмин Бланшет, Марк Саммерфилд: QT4.Программирование GUI на C++. Изд. 2-е. Официальное руководство от Trolltech; КУДИЦ-Пресс, 2008.

[23] Макс Шлее: Qt4. Профессиональное программирование на C++. Наиболее полное руководство; БХВ-Петербург, 2006.

[24] Юрий Земсков: Qt 4 на примерах, БХВ-Петербург, 2008.

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

[26] Johan Thelin: Foundations of Qt Development; APress, 2007.

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

[28] Robert Love: Linux System Programming; O'Reilly Media, 2013.

[29] Анализ уязвимостей при переходе на 64-х разрядные системы (http://www.viva64.com/ru/a/0046/)

[30] Seven Steps of Migrating a Program to a 64-bit System (https://www.codeguru.com/cplusplus/seven-steps-of-migrating-a-program-to-a-64-bit-system/)

[31] Options to Request or Suppress Warnings (https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html)

[32] The Linux driver implementers API guide (https://www.kernel.org/doc/html/v4.11/driver-api/index.html)