

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

РУКОВОДЯЩИЕ УКАЗАНИЯ ПО КОНСТРУИРОВАНИЮ
прикладного программного обеспечения
для операционной системы специального назначения
«Astra Linux Special Edition» РУСБ.10015-01
Листов 63

Москва
2011

АННОТАЦИЯ

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

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 версии 1.2, 1.3 ОС СН.

СОДЕРЖАНИЕ

1. Состав дистрибутива ОС СН	4
1.1. Основные компоненты	5
1.2. Рекомендации по использованию средств для разработки ПО	5
1.2.1. Базовые библиотеки и утилиты	5
1.2.2. Графическая система и многофункциональный рабочий стол	5
1.2.3. Средства разработки и отладки	6
1.3. Общее программное обеспечение	6
1.4. Релизы дистрибутива и их версии	7
2. Особенности разработки приложений с графическим интерфейсом, взаимодействующих с рабочим столом Fly	8
2.1. Среда сборки	8
2.2. Группы и имена приложений	8
2.3. Сборка	9
2.4. Секция в пакетах	9
2.5. Основные файлы приложения	9
2.5.1. Файл .pro	9
2.5.2. Файл desktop entry	13
2.5.3. Файл перевода	15
2.6. Разработка приложения	16
2.6.1. Общие требования	16
2.6.2. Абсолютные и относительные пути	18
2.6.3. Средства разработки	20
2.6.4. Локализация	20
2.6.5. Диалоги	20
2.6.6. Меню	21
2.6.7. Сохранение настроек	22
2.6.8. Использование тем иконок	22
2.6.9. MIME-типы	24
2.7. Взаимодействие с рабочим столом	24
2.7.1. Иконки и заголовки окон	24
2.7.2. Системный трей	25
2.7.3. Автозапуск	25

2.7.4. Меню «Пуск» и рабочий стол	26
2.7.5. Корзина	28
2.7.6. Перетаскивание объектов на рабочем столе	29
2.7.7. Подсистема помощи	30
2.7.8. Синхронизация с менеджером окон fly-wm	30
2.7.9. Полноэкранный режим	31
2.7.10. Смена раскладки клавиатуры	31
2.7.11. Дополнительные панели	31
2.8. Плагины для менеджера файлов Fly-fm	32
2.9. Настройки планшетного режима	32
2.10. Приложения и права администратора	38
2.11. Приложения и мандатная политика	38
3. Программный интерфейс мандатного разграничения доступа	45
3.1. Общие сведения	45
3.2. Сетевое взаимодействие	45
3.2.1. Передача мандатных меток по сети	45
3.2.2. Разработка ПО для обработки информации с различными уровнями конфиденциальности	47
4. Описание использования API системы расширенного аудита	56
5. Разработка ПО для взаимодействия с СУБД PostgreSQL	57
5.1. Мандатное разграничение доступа в СУБД PostgreSQL	57
6. Разработка ПО для WEB-сервера	58
7. Рекомендации по способам миграции приложений в среду ОС СН с других платформ	59
7.1. Особенности переноса программ с 32-разрядных платформ на 64-разрядную	59
7.2. Настройка среды исполнения 32-разрядных приложений	60
7.3. Перенос программ из среды Windows	60
7.4. Технология компиляции при переносе ПО	61
Список литературы	62

1. СОСТАВ ДИСТРИБУТИВА ОС СН

ОС СН относится к семейству дистрибутивов Linux и имеет сертификат соответствия требованиям руководящих документов Министерства обороны Российской Федерации:

- по 3 классу защищенности информации от несанкционированного доступа согласно руководящего документа «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации» (Гостехкомиссия России, 1992 г.);
- по 2 уровню контроля отсутствия недеklarированных возможностей согласно руководящего документа «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недеklarированных возможностей» (Гостехкомиссия России, 1999 г.);
- по соответствию реальных и декларируемых в документации функциональных возможностей.

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

ОС СН разработана на основе открытой операционной системы «Astra Linux Common Edition» (далее по тексту — ОС ОН). Особенности архитектуры ОС ОН и ее состав описаны в документе [1].

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»), который является единственным вариантом многофункционального графического интерфейса пользователя в составе ОС СН. Программный интерфейс рабочего стола Fly и методы его использования описаны в разделе 2.

ОС СН унаследовала из ОС ОН основу построения дистрибутива — открытый репозиторий пакетов для ОС Debian Linux с применяемой в нем системой пакетирования. Таким образом, общие принципы разработки и сборки программ для ОС СН, а также создания своего репозитория пакетов соответствуют дистрибутивам Linux семейства Debian.

Поэтому, кроме документа [1], разработчикам программ для ОС СН можно использовать общие руководства по Debian Linux (например, такие как [3], [4]).

1.1. Основные компоненты

В целях оптимизации состава в дистрибутив ОС СН не включен ряд дублирующих компонентов, входящих в ОС ОН и решающих сходные целевые задачи (например, различные текстовые редакторы, клиенты электронной почты, средства обработки мультимедиа).

Компоненты ОС СН, с которыми интегрированы встроенные СЗИ:

- базовая платформа:
 - ядро Linux 2.6;
 - базовая библиотека `glibc`;
 - утилиты управления СЗИ;
 - сетевые службы;
 - графический сервер Xorg;
 - графическая система Fly;
 - система печати CUPS;
 - система организации домена Astra Linux Directory (ALD).

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

- общее программное обеспечение (ПО):
 - защищенная СУБД PostgreSQL версии 8.4;
 - HTTP-сервер Apache2;
 - защищенные средства обмена электронными сообщениями Exim4 и Dovecot;

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 версии 2.6.34;
- 2) `glibc` версии 2.7;
- 3) `openldap` версии 2.4;
- 4) `openssl` версии 0.9;
- 5) `bash` версии 3.2.

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 версии 4.6.3 (в версии 1.3 — 4.8.2) — основная прикладная библиотека для создания графического интерфейса;
- 2) Xorg версии 7.7 — библиотеки графического X-сервера;
- 3) Fly 2.0 — 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 на отдельном DVD-диске 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 версии 2.7), компиляторы, средства сборки, отладчики и специализированные пакеты для разработки программ (devel-пакеты). Ниже приведен краткий список этих средств:

- GCC — открытый компилятор, соответствующий стандарту ANSI, версия 4.3 (4.7);
- средства сборки:
 - binutils — программы линковки и манипулирования объектными файлами, версия 1.18;
 - make — средство для автоматизации процесса компиляции, версия 3.81;
- отладчик gdb — стандартный отладчик GNU, версия 6.8.

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) для разработки ПО с использованием СУБД:
 - PostgreSQL версии 8.4;
- 2) для разработки документации:
 - L^AT_EX;
 - Openoffice версии 3.2.0;
- 3) для работы с мультимедиа:
 - мультимедиаплеер VLC версии 1.0.1;
 - фреймворк Phonon версии 4.6.0;
- 4) для разработки web-ориентированного ПО:
 - Apache2 версии 2.2.16;

- Thunderbird версии 3.1.4;
- Firefox версии 3.6.9;
- Exim4 версии 4.70;
- Dovecot версии 1.2.10;
- PHP5 версии 5.2.

1.4. Релизы дистрибутива и их версии

Дистрибутивы ОС СН отличаются друг от друга именем релиза и номером версии:

- имя релиза — определяет специфику дистрибутива: его аппаратную платформу и область применения;
- номер версии — 2 или 3 цифры, определяющие версию дистрибутива ОС СН.

В инсталлированном дистрибутиве ОС СН версию его релиза можно посмотреть в текстовом файле в каталоге /etc:

```
/etc/astra_version
```

Формат записи в нем следующий:

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

где <EDITION> — редакция дистрибутива Astra Linux (для Special Edition — «SE», для Common Edition — «CE»);

V — первая цифра номера релиза, связанная с именем релиза;

U — версия релиза;

Y — номер обновления в пределах данной версии релиза (если таких обновлений не было, отсутствует);

NAME — имя релиза (привязано к первой цифре номера релиза). Для ОС СН в качестве имен релизов используются названия городов-героев России.

Пример

```
SE 1.1.2 (Smolensk)
```

```
SE 1.2 (Smolensk)
```

Настоящие РУК относятся к релизу «Смоленск» ОС СН, предназначенному для функционирования на средствах вычислительной техники с процессорной архитектурой x86-64.

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.

Реализация данных правил поможет добиться:

- унификации сборки и установки приложений;
- легкости модификации и сопровождения;
- единообразия внешнего вида, внутренней структуры и поведения.

От разработчика требуется знание языка программирования C++ и библиотеки графического интерфейса Qt 4. Настоятельно рекомендуется изучить, например, следующие руководства по работе с этой библиотекой:[16]–[21]. 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 4.

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 многие другие файлы и каталоги, например:

- каталог для сборки (<имя приложения>-версия);

- файл перевода (имя приложения_ru.tsi или имя приложения_ru.qm);
- исполняемый файл (имя приложения);
- 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. Для сборки пакет `flycore` предоставляет два `pri`-файла, автоматизирующих некоторые задачи конфигурирования и установки. Так, `flyconf.pri` задает основные переменные, используемые всеми приложениями при сборке, `flyinst.pri` — основные переменные и процедуры для установки приложений.

При каждой, даже формальной, пересборке пакета для его включения в очередной релиз дистрибутива следует изменять последнюю цифру — номер релиза, т. е. 2.0.0-1, 2.0.0-2 и т. д. Номера версий (первые три цифры) изменяются при существенном изменении функционала программ 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. А отладочные версии — в виде `name-dbg`-пакетов.

2.4. Секция в пакетах

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

2.5. Основные файлы приложения

Получив имя и место в дереве сборки, 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 использование `flyconf.pri` и `flyinst.pri`.

Файлы `flyconf.pri` и `flyinst.pri` очень важны, они централизуют важнейшие особенности сборки и установки приложений, освобождая программистов от необходимости их повторного определения в своих `pro`-файлах. Ни один `pro`-файл ни одной программы не должен без необходимости переопределять переменные, заданные в этих файлах, а, наоборот, должен их использовать. Таким образом, создание `pro`-файла приложения начинается с изучения `flyconf.pri` и `flyinst.pri`. Приведем пример правильного `pro`-файла, например `fly-commi.pro`:

```
TEMPLATE = app
CONFIG += qt fly
include(/usr/include/fly/flyconf.pri)
TARGET = fly-commi
HEADERS += mylineedit.h
SOURCES += main.cpp mylineedit.cpp
FORMS += commimainform.ui optionsform.ui transferform.ui
TRANSLATIONS = fly-commi_ru.ts
DESKTOPDIR=
include(/usr/include/fly/flyinst.pri)
или fly-admin-cron.pro:
TEMPLATE = app
CONFIG += qt fly
include(/usr/include/fly/flyconf.pri)
TARGET = fly-admin-cron
SOURCES += main.cpp cconfig.cpp
HEADERS += cconfig.h ccrondata.h ...
FORMS += fly-cron.ui
TRANSLATIONS = fly-admin-cron_ru.ts
ICON16 = ./images/fly-admin-cron16.png
ICON22 = ./images/fly-admin-cron22.png
ICON32 = ./images/fly-admin-cron32.png
ICON48 = ./images/fly-admin-cron48.png
ICON64 = ./images/fly-admin-cron64.png
include(/usr/include/fly/flyinst.pri)
```

Примерно в такой последовательности и составе должны следовать строки. Обратите внимание на краткость этих `pro`-файлов и что `CONFIG`-параметр содержит слово `fly`, что позволяет во включаемом затем файле `flyconf.pri` подключить такие библиотеки, как: `flycore`, `flyui`, необходимые большинству приложений `Fly`.

Если есть переменные `ICON16`, `ICON22` и прочие, то они задают специфичные иконки нескольких размеров для данного приложения. Эти иконки в `flyinst.pri` с помощью утилиты `xdg-icon-resource` копируются при сборке приложения в поддеревья

сборки типа `./debian/tmp/usr/share/icons/hicolor/16x16/apps` и т.д., т.е. в подкаталоги `apps` будут скопированы 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.

При использовании файлов `flyconf.pri/flyinst.pri` переменными, которые можно определить вне их, а точнее в `rules (spec)`-файле являются:

- `FLY_MANDIR` — корневой каталог для `man`-файлов;
- `FLY_BINDIR` — по умолчанию `/usr/bin`;
- `FLY_SBINDIR` — по умолчанию `/usr/sbin`;
- `FLY_LIBDIR` — по умолчанию `/usr/lib`;
- `FLY_INCDIR` — по умолчанию `/usr/include`;
- `FLY_DATDIR` — по умолчанию `/usr/share`;

Эти переменные нужно задавать в командной строке `qmake`:

```
qmake FLY_MANDIR=/usr/share/man
```

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

```
DEB_QMAKE_ARGS += FLY_MANDIR=/usr/share/man
```

Возможно также использование переменных окружения с теми же названиями, но без префикса `FLY_`, т.е. `MANDIR`, `BINDIR` и т.д., но этот способ не рекомендуется и существует для совместимости со старыми пакетами.

Пример файла `debian/rules`:

```
#!/usr/bin/make -f
DEB_BUILD_PARALLEL ?= 1
include /usr/share/cdb/1/rules/debhelper.mk
DEB_QMAKE_ARGS += SOURCE_VERSION=$(DEB_VERSION) \
                  FLY_LIBDIR=/usr/lib/$(DEB_HOST_MULTIARCH)
include /usr/share/cdb/1/class/qmake.mk
```

Из примеров `pro`-файлов видно, что основная работа по конфигурированию приложения для сборки возложена на единый для всех

```
include(/usr/include/fly/flyconf.pri)
```

а основная работа по установке приложения — на единый для всех

```
include(/usr/include/fly/flyinst.pri)
```

что избавляет большинство приложений от ненужного дублирования одних и тех же сборочных и установочных процедур. В этом примере `DESKTOPDIR` указывает на каталог, откуда при установке будут взяты `desktop entry`-файл и две иконки данного приложения. Если они лежат в том же каталоге, что и исходные тексты, то `DESKTOPDIR` должна быть определена

как DESKTOPDIR=.

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 ) {
    !contains(QT_VER, 3.0.x) {
        SUBDIRS += fly-admin-snddetect
    }
}
```

Во втором случае допустимо:

```
pics.path = $$${FLYDATADIR}/fly-parashoot/icons
pics.files = pics/*
sounds.path = $$${FLYDATADIR}/fly-parashoot/sounds
sounds.files = sounds/*
INSTALLS += pics sounds
```

— для файлов, устанавливаемых в собственный каталог приложения (здесь — `$$${FLYDATADIR}/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:

```
...
headersfly.path = $$FLY_INCDIR/fly
headersfly.files = flyui.h flytranslator.h flyabout.h \
                  flyhelp.h flyhelpmenu.h flycmdlineargs.h

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

INSTALLS += headersfly uiheadersfly
...
```

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

Если возникает необходимость удалять каталоги при выполнении `make clean` или `make distclean`, то можно добавить их к переменным `FLY_CLEAN_DIRS` и `FLY_DISTCLEAN_DIRS` соответственно. Также нужно, чтобы в `CONFIG` была задана опция `fly_clean_dirs`. При подключении `flyconf.pri` это происходит автоматически.

Выполняется ли сборка для Linux-подобной среды можно определить так:

```
contains( DEFINES, FLY_LINUX) {
...
}
```

2.5.2. Файл desktop entry

Почти каждое fly-приложение должно сопровождаться файлом `desktop entry` [6] 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. Также приложение должно поставлять иконки размеров 16, 22, 32, 48, 64 и 128 точек, символизирующие приложение. В этом случае имена файлов данных иконок должны содержаться в переменных от `ICON16`, `ICON22`...`ICON128` в `pro`-файле. Файлы `desktop entry` и иконок могут находиться в каталоге с именем `desktop` в поддереве сборки приложения.

В случае, если иконки, подходящие для приложения, уже содержатся в иконной теме, файлы иконок могут не содержаться в дереве сборки и переменные `ICON16` и т.д. могут быть опущены, но файл `desktop entry` все равно в поле `Icon` должен содержать имя используемой иконки, а в поле `X-FLY-IconContext` ее контекст. В виде исключения в поле `Icon` файла `*.desktop` 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=ROOT-ONLY;Settings;SystemSetup`

`X-SuSE-YaST-Group=System`

```

X-SuSE-YaST-RootOnly=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 [6]. Слово `Settings` в поле `Categories` говорит о принадлежности программы к средствам администрирования. Слово `FLY-CONTROL-PANEL` позволило бы программе попасть в меню «Пуск – Панель управления». Однако рабочий стол Fly имеет в своем составе и специальную программу «Панель управления», предназначенную для централизованного вызова утилит управления системой. Если требуется, чтобы программа могла быть вызвана через эту панель, desktop entry-приложения должны содержать следующие поля:

`X-SuSE-YaST-Group=Hardware` или др.

`X-SuSE-YaST-RootOnly=false` или `true`

где группы могут быть: `Hardware`, `Software`, `System`, `Desktop`, `Network`, `Security`, `Misc`.

Значение поля `X-SuSE-YaST-RootOnly` определяет: показывать данную программу только администратору или всем пользователям системы. Категорию можно придать не только файлу, но и целому каталогу путем заполнения поля `Categories` в файле `.directory` (тип `Directory`), находящемся в этом каталоге.

Файлы `desktop entry` как правило должны иметь `Type=Application`, но для других случаев возможно использование других типов, разрешенных стандартом [6]: `Application`, `FSDevice`, `Link`, `Directory`, `MimeType` и ряд других.

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».

Для существенного ускорения поиска иконок в `desktop`-файлах предусмотрено задание контекста иконки (раньше это было поле типа `Reserved=IconContext=...`). Специфичный для `Fly` параметр называется `X-FLY-IconContext` (`X-FLY-IconContext[ru]` для локализованных иконок) и должен быть задан как для основной иконки так и для иконок в секциях `actions` (если они есть).

2.5.3. Файл перевода

Все приложения должны быть русифицированы с использованием стандартных средств `Qt`.

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

Изначально `Fly`-приложения должны содержать только английские варианты названий элементов своих интерфейсов. Файл с переводами (`qm`-файл) должен формироваться с помощью программ `lupdate` и `linguist`, входящих в состав дистрибутива ОС `СН`. Подключение соответствующего текущей локали `qm`-файла производится в функции `flyInit()` библиотеки `flyui`, вызываемой в `main`-функции приложения. Для этого необходимо, чтобы файл перевода был соответствующим образом расположен (обеспечивается скриптами установки `flyinst.pri`) и имел имя, сформированное по шаблону

имя приложения_локаль.qm, например fly-admin-mouse_ru.qm.

Есть возможность не генерировать бинарный файл перевода вручную заранее, а делать это при сборке пакета автоматически. Для этого надо определить переменную TRANSLATIONS, содержащую ts-файлы. Они будут автоматически переведены при сборке, если flyconf.pri включить в свой pro-файл после задания переменной TRANSLATIONS.

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 отмены действий такого рода.

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

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

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

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

Fly — фиксированное слово-контекст, остальное задается разработчиком.

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

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

Примеры:

1. В rules добавить:

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

2. Там же передать эту переменную:

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

3. При использовании cdbс вместо предыдущих двух пунктов можно использовать:

```
include /usr/share/cdbс/1/rules/debhelper.mk
```

```
DEB_QMAKE_ARGS += SOURCE_VERSION=$(DEB_VERSION)
```

```
include /usr/share/cdb/1/class/qmake.mk
```

4. В pro-файл добавить (только если не используется flyconf.pri):

```
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 подправить rules (1 или 2 строки) и pro-файл (1 строка и то, только если не используется flyconf.pri!). В 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 FlyCmdLineArgs, см. flyui) анализирует аргументы своей командной строки, должно без препятствий пропускать упомянутые стандартные Qt/X11-аргументы, не останавливая свою работу с сообщениями типа «неверный аргумент».

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

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

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

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

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.

Для работы с файлами иконок следует использовать библиотеку `flycore`, которая берет на себя задачу поиска иконки необходимого размера из текущей темы (2.6.8). Только в случае, если используемая иконка уникальна для данного приложения, она может быть статично включена в код программы.

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 средств, предоставляемых библиотекой `flycore`. В заголовочном файле `flyfileutils.h` определена функция:

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

где `type` является перечислением, определяющим требуемый каталог:

```
typedef enum _FlyDir {
WM_DIR, USER_TMP_DIR, USER_BASE_DIR, USER_THEME_DIR,
USER_START_MENU_DIR, USER_CPL_DIR, USER_DESKTOP_DIR,
USER_DOCUMENTS_DIR, USER_TRASH_DIR, FLY_WORK_DIR,
APP_SHARED_DIR, APP_TRANS_DIR, APP_DOCS_DIR,
APP_DOCS_HTML_DIR, WM_SOUNDS_DIR, WM_IMAGES_DIR,
WM_KEYMAPS_DIR, USER_TOOLBAR_DIR, ...
} 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/.fly/trash);

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

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

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

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

USER_TOOLBAR_DIR — каталог ярлыков для панели задач (\$HOME/.fly/toolbar),

т.е. панели быстрого запуска, расположенной на панели задач справа от кнопки «Пуск»;

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

Например, 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`.

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

Оконный менеджер `fly-wm` может запускать приложения с ненулевыми мандатными уровнями. Некоторые приложения могут не работать в силу того, что они производят запись в файлы с ненулевыми уровнями. Для упорядочивания все приложения, которым требуется временный каталог, должны получить путь к нему из переменной `$TMPDIR`. Гарантируется, что если приложение было запущено с ненулевым уровнем, то `$TMPDIR` для этого приложения выставлена на каталог, позволяющий в него писать/читать. Получить свой временный каталог приложение может с помощью стандартной функции `getenv("TMPDIR")` или более общим вызовом `GetFlyDir(USER_TMP_DIR)` из библиотеки `flycore`. 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.

Каталог `/etc/profile.d` может содержать скрипт `fly.sh`, устанавливающий в виде переменных окружения информацию об основных путях, используемых рабочим столом Fly, т.е. примерно ту же информацию, что программно можно получить по `GetFlyDir(...)`. Переменную `FLYHELPPDIR`, указывающую на каталог с html-документацией по Fly, может устанавливать скрипт из пакета `fly-doc`.

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

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 Qt Designer.

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

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

- разбор файлов `desktop entry` [6];
- доступ к темам иконок [7];
- взаимодействие с менеджером окон;
- формирование абсолютных путей к файлам рабочего стола, включая файлы русификации, помощи и т.д.;
- дополнительные задачи, не требующие GUI.

Библиотека `flyui` — 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:

- `flyuiextra` — для дополнительных элементов графического интерфейса;
- `flynet` — для элементов настройки сети;
- `flyscan` — для работы со сканерами;
- 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 «exit», 16x16, «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», 16x16, «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.

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 `forMimeType`. В отличие от

`QIcon::fromTheme`, `FlyIcon::fromTheme` позволяет задавать контекст иконки и эмблемы, которые должны быть на нее наложены. В большинстве ситуаций, когда не нужно подгружать огромное количество иконок и/или накладывать на них эмблемы, 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"));
    ...
    // получение иконки для MIME-типа
    QIcon gzipIcon = FlyIcon::forMimeType("application/x-gzip");
    ...
}
```

Следует обратить внимание, что нужно задавать только имя иконки без расширения. При неудачном извлечении иконки из темы можно использовать встроенную иконку. Методы `QIcon::fromTheme`, `FlyIcon::fromTheme` и `FlyIcon::forMimeType` имеют для этого параметр `fallback`. Уникальные иконки, которым нет аналогов в теме, могут быть «вшиты» в приложение. Тема иконок `fly-nuvola` по именам иконок на 100 % совместима с KDE3. Совместимость с KDE4, `gnome` и `Icon Naming Specification` [7] в версии 1.2 (релиз «Смоленск») - неполная. Но начиная с версии 1.3 в операционной системе используется новая тема `fly-astra`, которая соответствует как `Icon Naming Specification` [7], так и некоторым особенностям KDE4. Теперь использовать имена иконок, принятые в KDE3, уже нельзя.

Если возникает редкая необходимость получать пути к файлам иконок, то можно воспользоваться функциями из `flyui: flyIcon()`, `flyMimeTypeIcon()`, `flyURLIcon()` и т.п. Они являются qt-обертками над методами класса `FlyIconTheme` из `flycore`, 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 [7].

Для единообразия внешнего вида можно использовать иконки `media-playback-start` и `media-playback-stop` (контекст `Actions`) для запуска и остановки сервисов и служб. Для перемещения элементов между списками подходят иконки `lleftarrow` и `lrightarrow` (контекст `Actions`, например размером 32). В самих списках не следует безосновательно отключать режим множественного выделения.

2.6.9. MIME-типы

В библиотеку `flycore` включена часть, реализующая API для детектирования MIME-типа любого файла (реализация соответствующего стандарта `freedesktop.org` [8]). При использовании `flyui` можно воспользоваться оберточной функцией `flyFileMimeType`.

2.7. Взаимодействие с рабочим столом

2.7.1. 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.

2.7.2. 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-класса `QSystemTrayIcon`, который реализует стандарты [11], [12]. При установке иконки для `QSystemTrayIcon` рекомендуется использовать `QIcon`, которая может выдавать изображения всех типичных размеров, а не только 16x16 (см., например, `QIcon::fromTheme`).

2.7.3. Автозапуск

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

```
int installInAutostart(const char *appname, bool install);
```

где `appname` — имя приложения (например, `fly-mailmon`);

`install` — установить или удалить приложение из автозапуска.

Эта функция копирует файл `/usr/share/applications/appname.desktop` в каталог для автозапуска приложений при старте рабочего стола (обычно `~/.fly/startmenu/applications/autostart` или `~/.config/autostart`) или удаляет его из этого каталога. Возвращаемое значение равно 0, если операция завершилась успешно.

Таким образом, приложения могут в своих настройках иметь флаг (checkbox) «Автозапуск», при выборе которого вызывается функция `installInAutostart(..., true)`, а при снятии — `installInAutostart(..., false)`.

Начиная с версии 2.1.0, в библиотеке `flycore` появилась более удобная и полная реализация стандарта системы автозапуска [15]. Несмотря на то, что для совместимости старые функции остаются, теперь рекомендуется использовать следующие:

```
bool autostartEnabled(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 `enableAutostart(appname,false)`, то в пользовательском автозапуске будет создан фиктивный `appname.desktop` с полем `Hidden=true`, что согласно стандарту [15] является пользовательским методом отключения системного автозапуска.

2.7.4. 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 [6]).

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

«Завершение работы...»

«Новая сессия...»

—разделитель—

«Выполнить...»

«Последние»

—разделитель—

«Помощь»

«Найти...»

«Менеджер файлов»

«Настройка»

«Программы»

В меню «Настройка» находятся избранные системные приложения и панель управления. В панель управления попадают только приложения, имеющие определенные поля в `*.desktop`-файле (`FLY-CONTROL-PANEL` см. 2.5.2). При этом, если в поле `Categories` есть ключевое слово `ROOT-ONLY`, то соответствующий пункт меню будет виден только пользователю `root`. Последнее правило распространяется не только на «Панель управления», но и на все пункты меню.

В многоуровневом меню «Программы» формирование подменю происходит автоматически на основе `Categories`, определенных в документе «Desktop Menu Specification» от freedesktop.org [13].

Подменю и категории, по которым приложения в них попадают:

- «Автозапуск» — аналогично соответствующему подменю в системе MS Windows;
- «Графика» — `Graphics`, `VectorGraphics`, `RasterGraphics`, `Screensaver`;
- «Игры» — `Game`, `*Game`;
- «Мультимедиа» — `AudioVideo`, `Multimedia`;

- «Научные и инженерные» — Education, Science, Math, Astronomy, Physics, Chemistry;
- «Работа с файлами» — Office, Spreadsheet, WordProcessor, Presentation, Calendar, Email;
- «Разработка» — Development, GUIDesigner, IDE, TextEditor;
- «Сеть» — Network, Internet, Email;
- «Системные» — SystemSetup, Settings, AdvancedSettings, Accessibility;
- «Утилиты» — System, PackageManager;
- «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 то же 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 указано поле FLY-CONTROL-PANEL, то приложение попадет в подменю «Настройки». 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) сливаются воедино в \$HOME/.fly/startmenu. Меню как иерархия в ФС сделано вопреки документу «Desktop Menu Specification» от freedesktop.org (там рекомендуется XML-файл [13]), чтобы облегчить ручную правку меню, как это сделано в MS Windows. С другой стороны, Fly предоставляет пользователю мощную программу fly-menedit для просмотра, редактирования, резервного копирования и автоматического создания не только меню «Пуск», но и вообще любых

иерархических совокупностей каталогов и (или) файлов типа *.desktop.

Любое приложение, желающее быть показанным в меню «Пуск» для каждого пользователя, должно в первую очередь разместить в /usr/share/applications/ свой *.desktop-файл или каталог с такими или иными файлами. Тогда при следующем запуске fly-wm или создании меню «с нуля» в fly-menedit 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)
```

2.7.5. 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 flycore и обычно он равен \$HOME/.fly/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 `flycore`:

```
static bool registerFile(const char *srcPath, char *dstPath);
static bool unregisterFile(const char *fileName);
static bool getFilePathToRestore(const char *fileNameOrPath, char *dstPath);

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).

2.7.6. 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 [9].

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 `FSDevice` — устройство с ФС (например, `cdrom` и `floppy`);
- 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.

2.7.7. Подсистема помощи

Подсистема помощи Fly состоит из:

- программы показа;
- хранилища файлов;
- клиентов.

Основная программа показа помощи — Qt Assistant. Никаких изменений в программу не вносилось, кроме исправления ошибок и добавления файла с русским переводом. Структура хранилища и формат файлов помощи определяются пакетом `fly-doc`. Клиенты программы показа помощи (все приложения Fly) вызывают несколько простых функций (см. в библиотеке `flyui` класс `FlyHelpMenu` и функции `flyHelpInstall()`, `flyHelpShow()` и т.п.) и не должны заботиться о месте расположения файлов помощи и способах их показа.

2.7.8. Синхронизация с менеджером окон fly-wm

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

Каждая программа, перемещающая файлы в/из каталогов:

- рабочего стола (по `GetFlyDir(USER_DESKTOP_DIR)`);
- корзины (по `GetFlyDir(USER_TRASH_DIR)`);
- стартового меню (по `GetFlyDir(USER_START_MENU_DIR)`).

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

- `SendCommandToWM("FLYWM_UPDATE_SHORTCUT\n");`
- `SendCommandToWM("FLYWM_UPDATE_TRASH\n");`
- `SendCommandToWM("FLYWM_UPDATE_STARTMENU\n").`

соответственно.

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

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 `flycore` с параметром `FLYWM_RESTART`. Практически пол-

ный список команд можно увидеть в приложении `fly-hotkeys` (редактор «горячих» клавиш), большинство из них можно посылать в `fly-wm` из других приложений.

2.7.9. 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()`. Однако из-за недоработки в реализации Qt 3.3.3 от программиста требуется при возврате к `showNormal()` дополнительно установить иконку приложения с помощью `setIcon()`. Qt 3.3.3 для высокоуровневого окна с MDI-интерфейсом принудительно формирует заголовок определенного формата, однако это можно и нужно переопределить с помощью установки `eventFilter`, в котором обрабатывается событие `CaptionChange`, выставляя свой заголовок, формат которого см. в 2.7.1.

2.7.10. 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`.

2.7.11. 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 [10] 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 3 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.)

2.8. Плагины для менеджера файлов Fly-fm

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

2.9. Настройки планшетного режима

В версии 1.3 (релиз «Смоленск») появилась возможность конфигурирования графического интерфейса пользователя для работы на мобильных устройствах с сенсорными экранами — т.е. в так называемом «планшетном режиме». Ниже приведены необходимые для этого режима настройки.

1) Автозапуск и/или настройка необходимых приложений:

– включить полноэкранный режим панели запуска, для чего скопировать (из `/usr/share/applications`) или отредактировать указанные ниже файлы:

`/etc/xdg/autostart/fly-start-panel.desktop` (для всех)

`~/.config/autostart/fly-start-panel.desktop` (для конкретного пользователя),

с добавлением `--maxi` в поле `Exec`:

`Exec=fly-start-panel --maxi --hide`

Можно уменьшить количество иконок стартовой панели как слева так и справа, удалив ненужные файлы из каталогов:

`/usr/share/fly-start-panel/actions_left`

`/usr/share/fly-start-panel/actions_right`

Для вступления данных изменений в силу `fly-start-panel` надо перезапустить выполнив 2 команды:

`fly-start-panel --quit`

`fly-open --exec путь_к_fly-start-panel.desktop_указанному_выше`

– включить экранную клавиатуру, для чего скопировать файл

`/usr/share/applications/fly-vkbd.desktop` в каталоги:

`/etc/xdg/autostart/` (для всех)

`~/.config/autostart/` (для конкретного пользователя)

и добавить опции `--strut-bottom --minimizetotray` в поле `Exec`:

`Exec=fly-vkbd --strut-bottom --minimizetotray`

Можно увеличить размер (долю высоты экрана занимаемую под клавиатурой) в

`/etc/xdg/rusbitech/fly-vkbd.conf` (для всех)

~/ .config/rusbitech/fly-vkbd.conf (для конкретного пользователя)

добавив в секцию [General] строку strutBottomHeight=0.4

Если fly-vkbd уже работает, то для вступления данных изменений в силу ее надо перезапустить, выполнив:

fly-open --exec путь_к_fly-vkbd.desktop_указанному_выше

– монитор батареи /usr/share/applications/qbat.desktop скопировать в

/etc/xdg/autostart (для всех)

~/ .config/autostart (для конкретного пользователя)

и запустить его сразу командой:

qbat

– микшер звука /usr/share/applications/fly-admin-mixer.desktop скопировать в

/etc/xdg/autostart (для всех)

~/ .config/autostart (для конкретного пользователя)

и, чтобы он запускался минимизированным в трей, в

/etc/xdg/rusbitech/fly-admin-mixer.conf (для всех)

~/ .config/rusbitech/fly-admin-mixer.conf (для конкретного пользователя)

добавить в секцию [General] строку StartInTray=true и запустить его командой:

fly-admin-mixer

2) для Qt4-приложений увеличить размер шрифта (с 10 до 11) и элементов интерфейса (Global Strut) для чего в файлах (если их нет - то следует создать):

/etc/xdg/Trolltech.conf (для всех)

~/ .config/Trolltech.conf (для конкретного пользователя)

в секции [qt] задать параметры:

font="Verdana, 11, -1, 5, 50, 0, 0, 0, 0, 0"

globalStrut\width=26

globalStrut\height=26

С помощью утилиты qtconfig-qt4 можно сделать тоже самое. Однако все это можно не делать, если сразу увеличить шрифт и для Qt и для GTK (но не для Libreoffice - для него должны быть специальные настройки) в файлах:

/etc/X11/Xresources/x11-common (для всех)

~/ .Xresources (для конкретного юзера)

добавив одну строку типа

Xft.dpi: 110

dpi можно выставить и истинное (например для 10 дюймового планшета с разрешением 1280x800 это где-то 145), но тогда возможно придется уменьшать размер шрифтов.

dpi в совокупности с размерами шрифтов поможет добиться наилучшего вида.

3) для отключения обрамления окон и их максимизации при старте в файлах:

`/usr/share/fly-wm/apprc` (для всех будущих новых пользователей)

`~/.fly/apprc` (для конкретного пользователя)

раскомментировать строку, в начале которой стоит - "*".

4) Показать на панели инструментов кнопку закрытия текущего активного окна, для чего в файле:

`/usr/share/fly-wm/toolbar/close.desktop` (для всех будущих новых пользователей)

`~/.fly/toolbar/close.desktop` (для конкретного пользователя)

строку `"NoDisplay="` закомментировать или установить `NoDisplay=false`

5) Произвести настройки некоторых параметров менеджера окон и рабочего стола в файлах

`/usr/share/fly-wm/theme/default.themerc` (для всех будущих новых пользователей)

`~/.fly/theme/default.themerc` (для конкретного пользователя)

`~/.fly/theme/current.themerc` (для конкретного пользователя)

- все *Font размеры установить с 10 на 11 (или см п 2)
- `TaskbarHeight=60`, `TaskbarButtonHeight=50`, `TrayIconSize=48` (или оставить 50, 45, 32 для компактности)
- `OneClickRun=true`
- `ChoiceDialogItemMargin=5`, `MenuVerticalMargin=6`
- `PagingSize=4x1` (или оставить 2x2 для компактности)
- `FixedTaskbar=true`
- `DndDragThreshold=25` `DoubleClickDistance=25` (необязательно)

Для вступления в силу этих изменений может потребоваться перезапуск менеджера окон или повторных вход в систему.

6) Блокировщик экрана можно

- отключить полностью в файлах:

`/usr/share/fly-wm/theme/default.themerc` (для всех будущих новых пользователей)

`~/.fly/theme/default.themerc` (для конкретного пользователя)

`~/.fly/theme/current.themerc` (для конкретного пользователя) установить параметр `ScreenSaverDelay=0`

и еще его необходимо настроить на случай "ручного" запуска пользователем для чего

– отредактировать в файлах:

/etc/X11/app-defaults/XLock (для всех у кого еще нет своего XLock)

~/XLock (только если файл уже есть у конкретного пользователя),

добавив 2 строки:

XLock.startCmd: fly-vkbd -geometry 1280x250+0-0 --loginhelper

XLock.underWinCmd: fly-vkbd

и включить энергосберегающий хранитель экрана:

XLock.mode: blank

7) Настроить «невидимый курсор»:

в fly-admin-theme выбрать тему курсоров xcursor-transparent

или в файлах:

/etc/X11/Xresources/x11-common (для всех)

~/.Xresources (для конкретного пользователя)

добавить строку

Xcursor.theme: xcursor-transparent

Для уже открытых окон изменения вступят в силу только при их новом запуске.

8) Выбрать планшетный режим во fly-fm при его работе

или создав/изменив файлы:

/etc/xdg/rusbitech/fly-fm.conf (для всех)

~/.config/rusbitech/fly-fm.conf (для конкретного пользователя)

с такими строками:

[regularWindow]

tabletMode=true

TabletParams\selectFilesMode=true

TabletParams\placesInsteadOfTree=true

TabletParams\fileActionsOnToolbar=true

TabletParams\hideStatusBar=true

TabletParams\viewsTabWidgetState=@ByteArray(

\x1\0\0\0\$\0\0\0\x3\0\0\0\x10\0\0\0\x30

\0\0\0\b\0\0\0\x10\0\0\0\x2\0\0\0@\0\0\0\x10

\xff\xff\xff\xff\xff\xff\xff\xff)

Уже запущенный менеджер фалов требуется перезапустить.

9) В файле /usr/share/X11/xorg.conf.d/10-evdev.conf внести следующие изменения: в каждой секции с Identifier, содержащей слова touch или tablet нужны опции:

Option "EmulateThirdButton" "true"

```
Option "EmulateThirdButtonMoveThreshold" "30"
```

Однако последний параметр задается не в координатах экрана, а в координатах устройства ввода, которые в общем случае могут не совпадать с экранными. Так, координаты сенсорного экрана у планшета «Acer W501» имеют диапазон по оси X 0-32760 (их можно увидеть в лог-файле X-сервера), а значит, даже `EmulateThirdButtonMoveThreshold=2000` не выглядит большим.

Для вступления изменений в силу требуется перезапуск X сервера и повторный вход в систему.

10) Настройка графического логина (входа в систему):

- включить автологин в `/etc/X11/fly-dm/fly-dmrc`

```
AutoLoginEnable=true
```

```
AutoLoginUser=...
```

- увеличить размер шрифта в `/etc/X11/fly-dm/fly-dmrc`, вставив строку вида

```
StdFont=Verdana,11,...
```

при этом лучше включить `AntiAliasing=true`

- то же самое можно сделать для всех шрифтов в `/usr/share/fly-dm/themes/fly/fly.xml`, Но это небезопасно т.к. может потребовать редактирования размеров элементов темы!

11) Для автоматического показа экранной клавиатуры надо в файлах (если их нет, то надо создать):

```
/etc/xdg/Trolltech.conf (для всех)
```

```
~/.config/Trolltech.conf (для конкретного пользователя)
```

в секции `[qt]` задать параметр: `softInputPanelCall=fly-vkbd`

Клавиатура будет автоматически показываться при щелчке левой кнопкой мыши в текстовых полях всех Qt4-приложений.

12) Для «ручного» поворота экрана в портретный режим и обратно надо показать на панели инструментов соответствующую кнопку. Для этого в файлах:

```
/usr/share/fly-wm/toolbar/rotate.desktop (для всех будущих новых пользователей)
~/.fly/toolbar/rotate.desktop (для конкретного пользователя)
```

строку `NoDisplay=` закомментировать или установить `NoDisplay=false`.

Данный ярлык использует скрипт `/usr/bin/fly-rotate.sh`, который не всегда может корректно определять имя сенсорного устройства. Тогда его надо вручную получить с помощью команды `xinput` и вписать в скрипт.

13) Для демонстрации (тестирования) планшетного режима на рабочем столе в каталогах

```
/usr/share/fly-wm/Desktops/Desktop1 - для всех новых пользователей, (а
```

для текущего существующего пользователя каталог определяется командой -
xdg-user-dir DESKTOP)

можно разместить ярлыки:

a) pinchzoom.desktop:

```
[Desktop Entry]
Type=Application
Name=Multitouch zoom
Name[ru]=Мультитач зум
Exec=/usr/lib/qt4/examples/touch/pinchzoom/pinchzoom
Path=/usr/lib/qt4/examples/touch/pinchzoom/
Icon=zoom-fit-best
X-FLY-IconContext=Actions
```

Прим.: пакет qt4-demos предварительно должен быть установлен!

б) scroller.desktop:

```
[Desktop Entry]
Type=Application
Name=Kinetic scrolling
Name[ru]=Кинетический скроллинг
Exec=/usr/lib/qt4/examples/scroller/wheel/wheel
Icon=application-x-executable
X-FLY-IconContext=MimeTypes
```

Прим.: пакет qtscroller предварительно должен быть установлен!

Внимание разработчикам: оба примера поставляются с исходниками.

14) «Кнопка питания» по умолчанию выключает устройство, причем ее действие определяется на уровне базовой системы, а не рабочего стола.

Чтобы назначить ей переход в спящий режим надо:

- отменить ее действие по-умолчанию, отредактировав или удалив скрипты /etc/acpi/power*,
- назначить ей переход в спящий режим либо на уровне системы в /etc/acpi/power*, либо на уровне рабочего стола, для чего в файлах: /usr/share/fly-wm/keyshortcustrc (для всех будущих новых пользователей) ~/.fly/keyshortcustrc (для конкретного пользователя)

изменить 2 строки вида:

```
None|XF86PowerDown=FLYWM_SUSPEND
```

None|XF86PowerOff=FLYWM_SUSPEND

(по-умолчанию в этих строках задано FLYWM_SHUTDOWN)

15) Настройки для приложений не адаптированных для обработки мультитач-событий (gtk-приложения, libreoffice и другие).

При переключении в планшетный режим для таких приложений стартует программа-демон `twofing` и её ярлык размещается в автостарте (аналогично 1)). Эта программа преобразует мультитач-события в события понятные неадаптированным приложениям. Например, мультитач-жест увеличение или уменьшение преобразуется в событие `Ctrl + нажатие 4-ой или 5-ой кнопки мыши (колесико)`, которое уже понимается как увеличение-уменьшение многими программами (`firefox`, `libreoffice` и т.п.). Однако, `twofing` дополнительно нагружает систему, но самое главное: не дает работать с мультитач-событиями приложениям, разработанным с прямой поддержкой мультитач. Поэтому в версии «Смоленск 1.3» `twofing` используется как временное демо-решение и требует осторожности. (снятия с выполнения при обнаружении каких-либо сбоев в работе тачскрина).

2.10. 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`.

2.11. Приложения и мандатная политика

Для разработки графических приложений с использованием библиотек графических интерфейсов Qt 4 в ОС СН реализована библиотека `libparsec-mac-qt4-1`, 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-qt4-1-dev`.

Далее приведены примеры использования классов из библиотеки `libparsec-mac-qt4-1`.

Примеры:

1. Создание диалога для установки мандатных атрибутов объектов ФС

```
#include <QApplication>
```

```

#include <QDialog>
#include <QVBoxLayout>
#include <QDialogButtonBox>

#include <parsec-qt4/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 ParsecQt4::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:
ParsecQt4::FileMacLabelWidget* m_fileMacLabelWidget;
QDialogButtonBox* m_buttonBox;
QString m_filePath;
};

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

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

    return 0;
}

```

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

```

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

```

```

#include <parsec-qt4/loginmaclabeldialog.h>

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

#include <iostream>

#include <fly/flyfileutils.h>

using namespace ParsecQt4;

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:
        /* we must support QApplication options,
           so avoid all options, even if they are invalid */
        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, "%llx ", 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::fromLocal8Bit(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;
}

//QDialog::Rejected
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 мандатной метки объекта (подробнее об этом в Руководстве [2], подраздел 4.1). Кроме того, при принятии решения могут учитываться специальные привилегии субъекта (см. [2], подраздел 4.2).

3.2. Сетевое взаимодействие

3.2.1. Передача мандатных 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 в соответствии со стандартом RFC1108 внедряются мандатные метки, соответствующие метке объекта – сетевое соединение (сокеты). При этом метка сокета наследуется от субъекта (процесса). Прием сетевых пакетов подчиняется мандатным правилам разграничения доступа. Метка сокета может иметь тип, позволяющий создавать сетевые сервисы, принимающие соединения с любыми мандатными метками.

Метки передаются в поле опций IPv4 пакета. Для передачи меток по сети используется вариант опции DoD Security Option IPv4 (см. RFC1108 — Security Options for the Internet Protocol — Параметры безопасности для IP-протокола).

В рамках стандарта RFC1108 метка снабжается классом 0хAB (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.

Поле опций IPv4 при передаче мандатных меток представлено в таблице 1.

Т а б л и ц а 1

Тип	Длина	Описание
130	Значение	DoD Basic Security: используется для передачи классификационных меток и флагов защиты авторизации

Для передачи меток по сети используется формат DoD Basic Security, представленный в таблице 2.

Таблица 2

TYPE = 130	LENGTH	CLASSIFICATION LEVEL (Unclassified)	PROTECTION AUTHORITY FLAGS
10000010	XXXXXXXX	10101011	AAAAAAA[1 0] AAAAAA0

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.

Метка в подсистеме безопасности PARSEC в ОС CH имеет следующий формат:

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

где: level — уровень (беззнаковое целое число 256 возможных значений), categories — категории (битовая маска little-endian, всего 64 категории), структура должна быть упакована (выравнивание полей отключено). В поле PROTECTION AUTHORITY FLAGS данная 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 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 на объекте доступа является синонимом нулевой мандатной метки. Таким образом, ядро ОС, в которой все объекты и субъекты доступа имеют мандатный уровень 0, функционирует аналогично стандартному ядру ОС Linux.

3.2.2. 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 мандатными метками. Основной процесс сетевого сервиса должен обладать определенными привилегиями Linux и привилегиями подсистемы безопасности PARSEC (см. [2], подраздел 4.2);
- 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, передаваемая по сети;
- 6) в дочернем процессе осуществляется установка необходимых привилегий 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 привилегий Linux и подсистемы безопасности PARSEC.

Дальнейшая обработка информации при взаимодействии с пользователем в рам-

ках 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 4.2):

- PARSEC_CAP_PRIV_SOCKET — для создания привилегированного сокета, имеющего возможность принимать соединения с любыми мандатными метками;
- 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 привилегий. 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 C.

Пример

```
#include <string.h>
#include <linux/prctl.h>
#include <sys/socket.h>
#include <netdb.h>
#include <parsec/mac.h>
#include <parsec/parsec_integration.h>
#include <parsec/parsec_mac.h>
#include <pwd.h>

int sock = 0;
```

```

int clnt_sock = 0;

const char* user;

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

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

//объявление переменной для установки привилегий PARSEC
// и ее начальная инициализация
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_SOCKET);

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

```

```

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

//установка флага наследования привилегий
if(prctl(PR_SET_KEEPCAPS,1))
{
    // обработка ошибки
}

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

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

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

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

// ожидание запроса на соединение на привилегированном сокете
if(listen(sock,5) < 0)
{
    // обработка ошибки
}

// создание соединения на привилегированном сокете

```

```

if(!(clnt_sock = accept(sock, (struct sockaddr*)&clnt_addr, &addr_len)))
{
    // обработка ошибки
}

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

// создание дочернего процесса для обработки информации в контексте пользователя
child = fork();
if(child<0)
{
    // обработка ошибки
}

if(!child)
{
    // код, выполняемый в дочернем процессе,
    // предназначенном для обработки информации в контексте пользователя

    // объявление переменной для получения с сокета соединения
    // передаваемой по сети мандатной метки клиентского процесса пользователя
    parsec_mac_label_t mac_label;

    // 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;

    // 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)
{
    // обработка ошибки
}

// 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)
{
    // обработка ошибки
}

// 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))))
{
    // обработка ошибки
}

```

```

// получение привилегий PARSEC для дочернего процесса
if(parsec_capget(0, &cur_pcaps ) != 0)
{
    // обработка ошибки
}

// проверка наличия необходимых эффективных привилегий PARSEC
// у дочернего процесса
if(!(cur_pcaps.cap_effective&PARSEC_CAP_TO_MASK(PARSEC_CAP_SETMAC)))
{
    // обработка ошибки
}

// получение с сокета соединения передаваемой по сети
// мандатной метки клиентского процесса пользователя
if(parsec_fstatmac(clnt_sock,&mac_label))
{
    // обработка ошибки
}

// установка на дочерний процесс мандатной метки,
// полученной с сокета соединения
if(parsec_setmac(0,&mac_label.mac) < 0)
{
    // обработка ошибки
}

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

// 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))
{
    // обработка ошибки
}

// переключение дочернего процесса в работу с правами пользователя,
// прошедшего идентификацию и аутентификацию и являющегося клиентом
// сетевого сервиса
if(setuid(pwd->pw_uid))
{
    // обработка ошибки
}

// сброс привилегий Linux и PARSEC дочернего процесса сетевого сервиса
if(parsec_cur_caps_set(0,0) != 0)
{
    // обработка ошибки
}

// код, выполняемый в дочернем процессе,
// для обработки и предоставления информации пользователю
// клиентского процесса
}

```

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

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

- `PARSEC_CAP_CHMAC` — предоставляет право менять мандатную метку файла;
- `PARSEC_CAP_MAC_SOCKET` — предоставляет право менять мандатную метку 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 PostgreSQL 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 (<http://www.postgresql.org/docs/8.4/static/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 4.8).

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 необходимо выставить переменную окружения `KRB5CCNAME`. Например, для скрипта на языке PHP установка значения переменной окружения будет иметь следующий вид:

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

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

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

Задача о переносе 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 (7.2). 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 [22]).

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 с ядром версии 2.6.x. Достаточно хорошая подборка статей на эту тему предоставлена Linux-порталом IBM, как, например [23] [24]. При этом надо учесть особенности как 64-разрядной процессорной архитектуры и ее поддержки ядром Linux 2.6 [5], 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 состав ОС СН помимо новой прикладной графической библиотеки Qt 4 входит и библиотека предыдущего поколения Qt 3 (версия 3.3.8). Более ранние версии библиотек Qt (т. е. 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 пере-

работка этих модулей на базе нового интерфейса (достаточно полное его описание можно найти в руководстве [5]). Советы по переносу программ управления устройствами с платформы Windows в Linux можно найти в статье [27].

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 «программ-оберток» с помощью макросов С-препроцессора, которые будут отображать старый программный интерфейс СЗИ на новый.

7.2. Настройка среды исполнения 32-разрядных приложений

Процессоры с архитектурой x86-64 (AMD64) способны работать как в 64-разрядном, так и 32-разрядном режимах.

Если рассматривать небольшие 32-разрядные Linux-приложения для x86, которые могут быть статически слинкованы со всеми необходимыми библиотеками, то они с большой вероятностью будут успешно работать в среде 64-разрядной ОС СН.

Для работы тех приложений, которые используют динамически подгружаемые библиотеки, в ОС СН необходимо настроить среду выполнения 32-разрядных программ. Для этого следует установить пакет `ia32-libs`, который содержит основные системные разделяемые 32-разрядные библиотеки. В консоли суперпользователя выполнить команду:

```
# apt-get install ia32-libs ia32-libs-gtk
```

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
```

7.3. 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. Хорошее и последовательное описание технических решений и методов, необходимых для успешного перехода пользователей на Linux-систему можно прочитать в статьях [25] [26]. Также следует ознакомиться с рекомендациями в статье [27] при переносе программ управления устройствами, если таковые имеются в системе. Особенности компиляции программ, разработанных на 32-разрядной платформе Windows, приведены в 7.4.

7.4. 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. 7.2). Чтобы компиляция в ОС СН проходила корректно и давала на выходе 32-разрядные исполняемые приложения, необходимо использовать для компилятора gcc — ключ `-m32`, для линковщика `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, тогда компиляция требует более сложной настройки. Обзор приемов такой компиляции можно найти в статье [28].

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.
(<ftp://astra-linux.ru/docs/Ruk-Devel-AstraLinuxCE.pdf>)
- [2] РУСБ.10015-01 97 01-1 «Операционная система специального назначения «Astra Linux Special Edition». Руководство по КСЗ. Часть 1»
- [3] Осаму Аоки: Справочник по Debian
(<http://qref.sourceforge.net/Debian/reference/index.ru.html>)
- [4] Руководство по Debian Policy
(<http://www.debian.org/doc/debian-policy/>)
- [5] Д.Бовет, М.Чезати: Ядро Linux (3-е издание); БХВ-Петербург, 2007.

Стандарты открытого рабочего стола Freedesktop.org

- [6] Desktop Entry Standard
(<http://standards.freedesktop.org/desktop-entry-spec>)
- [7] Icon Theme Specifications и Icon Naming Specification
(<http://standards.freedesktop.org/icon-naming-spec>
<http://standards.freedesktop.org/icon-theme-spec>)
- [8] Shared MIME-info database
(<http://standards.freedesktop.org/shared-mime-info-spec/>)
- [9] Drag-and-Drop Protocol for the X Window System
(<http://www.freedesktop.org/wiki/Specifications/XDND>)
- [10] Extended Window Manager Hints
(<http://standards.freedesktop.org/wm-spec/wm-spec-latest.html>)
- [11] XEmbed Protocol Specifications
(<http://standards.freedesktop.org/xembed-spec/xembed-spec-latest.html>)
- [12] System Tray Protocol Specifications
(<http://standards.freedesktop.org/systemtray-spec/systemtray-spec-latest.html>)
- [13] Desktop Menu Specifications
(<http://standards.freedesktop.org/menu-spec/menu-spec-latest.html>)

- [14] Sound Theme and Naming Specifications
(<http://www.freedesktop.org/wiki/Specifications/sound-theme-spec>)
- [15] Desktop Application Autostart Specification
(<http://standards.freedesktop.org/autostart-spec/autostart-spec-latest.html>)

Книги по графической библиотеке Qt

- [16] Жасмин Бланшет, Марк Саммерфилд: QT4.Программирование GUI на C++. Изд. 2-е. Официальное руководство от Trolltech; КУДИЦ-Пресс, 2008.
- [17] Макс Шлее: Qt4. Профессиональное программирование на C++. Наиболее полное руководство; БХВ-Петербург, 2006.
- [18] Юрий Земсков: Qt 4 на примерах, БХВ-Петербург, 2008.
- [19] Daniel Molkentin: The Book of Qt4: The Art of Building Qt Applications; No Starch Press San Francisco, 2007.
- [20] Johan Thelin: Foundations of Qt Development; APress, 2007.
- [21] Alan Ezust and Paul Ezust: An introduction to Design Patterns in C++ with Qt4; Prentice Hall, 2006.

Проблемы переноса программ на платформу Astra Linux Special Edition

- [22] Анализ уязвимостей при переходе на 64-х разрядные системы
(<http://www.viva64.com/ru/a/0046/#ID0ER3AE>)
- [23] Общие вопросы переноса приложений между UNIX-платформами
(<http://www.ibm.com/developerworks/ru/library/multiunix-i/>)
- [24] Приемы переноса Linux-приложений на 64-х разрядные системы
(<http://www.ibm.com/developerworks/ru/library/l-port64/index.html>)
- [25] Руководство по миграции на Linux. Часть II
(http://www.ibm.com/developerworks/ru/library/linux_migr/part2.html)
- [26] Руководство по миграции на Linux. Часть III
(http://www.ibm.com/developerworks/ru/library/linux_migr/part3.html)
- [27] Перенос приложений управления устройствами с Windows на Linux
(<http://www.ibm.com/developerworks/ru/library/l-devctrl-migration/index.html>)
- [28] Перенос программ из Windows в UNIX
(<http://www.ibm.com/developerworks/ru/library/au-porting2/index.html>)