История изменений:

Автор	Дата	Версия	Краткое описание изменений
Андрей Потехин	15 декабря 2014	D00.01	Создание документа
Андрей Потехин	19 декабря 2014	D00.02	Добавлены новые функции интерфейса. Изменена тестовая утилита для поддержки изменённого интерфейса

1 Введение

Настоящий документ описывает архитектуру библиотеки считывателей бесконтактных смарт-карт ***Parsec PRP-08*** и ***Parsec PR-EH08***, а также процедуру установки данной библиотеки на различные версии операционной системы Linux.

2 Архитектура

Архитектура библиотеки считывателей бесконтактных смарт-карт представлена ниже (смотрите Рисунок 1). Зеленым цветом выделены стандартные компоненты операционной системы Linux и стандартные библиотеки. Голубым и желтым цветами выделены разработанные компоненты.

Считыватели бесконтактных смарт-карт ***Parsec PRP-08*** и ***Parsec PR-EH08*** подключаются к компьютеру с установленной операционной системой Linux. Подключение осуществляется с помощью интерфейса USB. Данные считыватели базируются на преобразователе интерфейсов USB / UART компании FTDI, а поэтому используют соответствующий стандартный драйвер Linux ***FTDI_SIO***.

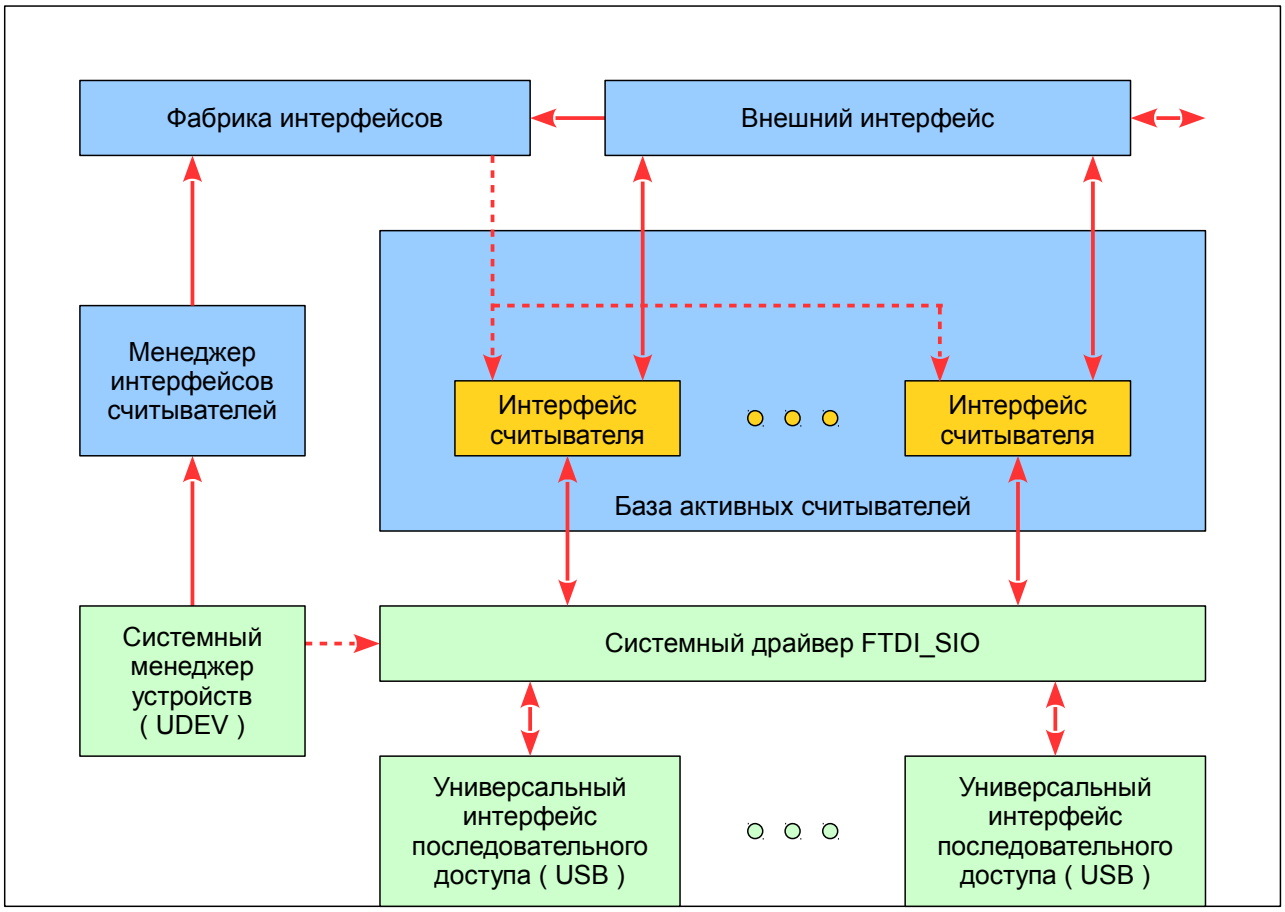

Рисунок 1: Архитектура библиотеки считывателей бесконтактных смарт-карт

Подключение устройств к системе контролируется системным менеджером устройств ***UDEV***. В момент подключения считывателя бесконтактных смарт-карт операционная система считывает специальные идентификаторы устройства, и на основе установленных правил менеджера устройств запускает драйвер ***FTDI_SIO***. Одновременно событие о подключении считывателя транслируется в менеджер интерфейсов считывателей. Менеджер интерфейсов считывателей распознаёт подключённое устройство, декодирует его тип и серийный номер, а

затем передаёт данную информацию фабрике интерфейсов. На основании этих данных фабрика интерфейсов создаёт интерфейс считывателя и помещает его в базу активных считывателей. Максимальное количество записей в базе активных считывателей задаётся константой в коде библиотеки и это значение выбирается в соответствии с предполагаемым способом использования библиотеки.

Отключение устройств от системы также контролируется системным менеджером устройств ***UDEV***. В момент отключения считывателя бесконтактных смарт-карт системный менеджер устройств транслирует событие об отключении считывателя в менеджер интерфейсов считывателей, который в свою очередь сообщает о данном событии фабрике интерфейсов. Фабрика интерфейсов уничтожает соответствующий интерфейс считывателя и удаляет его из базы активных считывателей, но только в том случае, если данный считыватель не вовлечен в транзакцию (механизм транзакций будет описан ниже). В случае транзакции интерфейс считывателя будет уничтожен сразу после её окончания.

Внешний интерфейс необходим для доступа к считывателям через активные интерфейсы. На уровне внешнего интерфейса считыватель идентифицируется по его серийному номеру. Каждое обращение к считывателю представляет собой транзакцию вида «запрос → ожидание ответа → обработка ответа». Блокировка считывателя на время транзакции осуществляется через фабрику интерфейсов, а взаимодействие с самим интерфейсом считывателя идёт напрямую. Механизм транзакций скрыт внутри библиотеки и не виден со стороны внешнего интерфейса.

Описанные блоки представлены следующими файлами:

- Внешний интерфейс:
  ReaderInterface.cpp
  reader_interface.h
- Фабрика интерфейсов и база активных считывателей:
  ReaderFabric.cpp
  ReaderFabric.hpp
- Менеджер интерфейсов считывателей:
  ReaderPlugAndPlay.cpp
  ReaderPlugAndPlay.hpp
- Интерфейсы считывателей:
  UsbSerial.cpp
  UsbSerial.hpp
  ReaderBase.cpp
  ReaderBase.hpp
  ReaderModelPrEh08.cpp
  ReaderModelPrEh08.hpp
  ReaderModelPrp08.cpp
  ReaderModelPrp08.hpp
- Модули поддержки:
  DallasCrc8.cpp
  DallasCrc8.hpp
  Logger.cpp
  Logger.hpp
  LoggerServerImpl.cpp
  LoggerServerImpl.hpp
  Thread.cpp
  Thread.hpp

3 Описание внешнего интерфейса

Внешний интерфейс описан в файле ***reader_interface.h***.

1. Установка функции логирования

```
void reader_set_log_func(
     debug_log_func_t debug_log_func_ptr );
```

Устанавливает функцию логирования для передачи служебной информации в запрашивающее приложение. В качестве параметра принимает указатель на функцию логирования. Прототип функции логирования описан в файле ***reader_interface.h***. Если логирование не требуется, то данную интерфейсную функцию можно не вызывать.

2. Установка детализации логирования

```
void reader_debug_level(
     int fabric_debug_level,
     int plug_and_play_debug_level,
     int reader_object_debug_level );
```

Задаёт детализацию логирования для различных компонентов. Каждый параметр принимает значения от 0 до 3. 0 соответствует минимальному уровню детализации, 3 — максимальному. Если логирование не требуется, то данную функцию можно не вызывать.

3. Открытие интерфейса

```
int reader_interface_init();
```

Открывает интерфейс библиотеки. Функция возвращает 0 в случае успеха, -1 в случае ошибки. Данная функция должна быть вызвана перед началом работы со считывателями.

4. Закрытие интерфейса

```
void reader_interface_term();
```

Закрывает интерфейс библиотеки. Данная функция должна обязательно быть вызвана, если в дальнейшем требуется повторное открытие интерфейса.

5. Получение списка активных считывателей

```
unsigned int reader_get_attached(
     unsigned long* serial_number_storage_ptr,
     unsigned int serial_number_storage_entries);
```

Возвращает количество активных (подключённых) считывателей. В качестве параметров принимает ссылку на массив для серийных номеров активных считывателей и его размер. Массив будет заполнен по выходу из функции. Количество записей в массиве соответствует возвращаемому значению.

6. Команда на получение информации о считывателе

```
int reader_get_info(
     unsigned long serial_number,
     Get_info_response_t* get_info_response_ptr );
```

Возвращает информацию о считывателе. В качестве параметров принимает серийный номер адресуемого считывателя и ссылку на структуру описания считывателя (описана в файле ***reader_interface.h***.). Структура описания считывателя будет заполнена по выходу из функции. Функция возвращает 0 в случае успеха, -1 в случае ошибки.

7. Команда на установку режима индикации

```
int reader_set_beep_and_blink_mode(
     unsigned long serial_number,
     unsigned char beep_on,
     unsigned char blink_on );
```

Устанавливает режим звуковой и световой индикации при поднесении карты к считывателю. В качестве параметров принимает серийный номер адресуемого считывателя и

два бинарных флага для задания режимов звуковой и световой индикации. Функция возвращает 0 в случае успеха, -1 в случае ошибки.

8. Команда на воспроизведение звукового сигнала

```
int reader_beep(
      unsigned long serial_number );
```

Воспроизводит звуковой сигнал и мигает светодиодами на считывателе. В качестве параметра принимает серийный номер адресуемого считывателя. Функция возвращает 0 в случае успеха, -1 в случае ошибки.

9. Установка опции WIEGAND

```
void reader_set_wiegand_26(
      unsigned char wiegand_26_on );
```

Устанавливает опцию WIEGAND. Данная опция является глобальной и устанавливается для всех подключенных считывателей. В качестве параметра принимает бинарный флаг.

10. Команда на чтение карты

```
int reader_read_card(
      unsigned long serial_number,
      unsigned long* card_number_ptr,
      Read_card_response_t* read_card_response_ptr );
```

Считывает код карты и / или расширенный код карты. Код карты вычисляется с учётом опции WIEGAND. В качестве параметров принимает серийный номер адресуемого считывателя, ссылку на код карты и ссылку на структуру расширенного кода карты (описана в файле ***reader_interface.h***) Код карты и структура расширенного кода карты будут заполнены по выходу из функции. Если код карты или расширенный код карты не требуется, то в качестве соответствующего параметра можно передать NULL. Функция возвращает 0 в случае успеха, -1 в случае ошибки.

4 Сборка библиотеки

Библиотека считывателей бесконтактных смарт-карт написана на языке программирования Си++, при этом внешний интерфейс поддерживает интеграцию с программами, написанными как на Си++, так и Си.

Сборка проектов обычно осуществляется с помощью утилиты ***Make*** и соответствующих файлов ***Makefile***. В данном проекте повторно используются компоненты из других проектов, а так как общая структура хранилища в данный момент не определена, то сборка осуществляется без использования утилиты ***Make***.

Для сборки проекта необходимы компилятор ***g++*** и библиотека системного менеджера устройств ***UDEV***. Также лучше установить компилятор ***gcc***, так как он может отсутствовать в Вашей системе. Установка должна осуществляться от пользователя ***root*** непосредственно или с использованием специальной утилиты ***sudo***. Установка осуществляется с помощью соответствующего пакетного менеджера выбранного дистрибутива Linux.

- OpenSUSE:

```
zypper install gcc g++ libudev-devel
```

- Ubuntu, Debian:

```
apt-get install gcc g++ libudev-dev
```

- Fedora:

```
yum install gcc g++ libudev-devel
```

Сборка динамической библиотеки считывателей бесконтактных смарт-карт осуществляется следующей командной строкой из директория с файлами проекта:

```
g++ -Wno-unused-result -O3 -fPIC -shared -o libreader.so *.cpp
```

В результате будет создана динамическая библиотека считывателей бесконтактных смарт-карт ***libreader.so***. Обратите внимание на разрядность созданной библиотеки (32 / 64 бита): без использования специального окружения разрядность библиотеки будет соответствовать разрядности операционной системы Linux, на которой производится сборка. Настоятельно рекомендуется использовать операционные системы соответствующей разрядности для сборки 32-х и 64-х битной версии библиотеки. Собранная библиотека может использоваться на различных дистрибутивах Linux до тех пор, пока базовые системные библиотеки совместимы.

5 Настройка системы для работы с библиотекой

5.1 Настройка правил UDEV

Для автоматического распознавания считывателей при их подключении и загрузки / настройки системного драйвера ***FTDI_SIO*** необходимо создать соответствующие правила системного менеджера устройств ***UDEV***. Для этого необходимо открыть следующий файл от пользователя ***root*** непосредственно или с использованием специальной утилиты ***sudo***:

```
/etc/udev/rules.d/99-parsec_reader.rules
```

и добавить в него следующие две строки:

```
ACTION=="add", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="e3b0", MODE="0666", RUN+="/sbin/modprobe ftdi_sio", RUN+="/bin/sh -c 'echo 0403 e3b0 > /sys/bus/usb-serial/drivers/ftdi_sio/new_id'"

ACTION=="add", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="e3b4", MODE="0666", RUN+="/sbin/modprobe ftdi_sio", RUN+="/bin/sh -c 'echo 0403 e3b4 > /sys/bus/usb-serial/drivers/ftdi_sio/new_id'"
```

При копировании обратите внимание на то, что указанные выше строки не должны содержать символов переноса строки посередине .

После создания указанного файла нужно либо перезапустить систему, либо выполнить следующую команду:

```
udevadm control --reload-rules
```

5.2 Сборка тестовой утилиты

Для сборки тестовой утилиты требуется компилятор ***gcc*** и библиотека системного менеджера устройств ***UDEV***. Процедура установки этих компонентов описана выше — процедуру нужно повторить только в том случае, если сборка тестовой утилиты производится на другой машине.

Для сборки тестовой утилиты нужно скопировать файлы ***reader_test.c***, ***reader_interface.h*** и ***libreader.so*** в выбранный директорий, а затем выполнить следующую командную строку в этом директории:

```
gcc -o reader_test reader_test.c -L. -lreader -lpthread -ludev
```

В результате будет создано тестовое приложение ***reader_test***.

5.3 Запуск тестовой утилиты

Для запуска тестовой утилиты необходимо скопировать библиотеку ***libreader.so*** в один из стандартных директориев Linux или прописать путь к этой библиотеке через переменную ***LD_LIBRARY_PATH***. В данном примере библиотека размещается в одном директории с тестовой утилитой ***reader_test***.

Подключите считыватель(ли) к системе. Это можно сделать и после запуска утилиты.

Для запуска тестовой утилиты необходимо перейти в указанный директорий и выполнить следующие команды:

```
export LD_LIBRARY_PATH=${PWD}
./reader_test 20 1
```

В результате тестовая утилита 20 раз запросит код поднесённой карты для каждого считывателя. Интервал между запросами задаётся вторым параметром (1 секунда). Для включения опции WIEGAND нужно добавить «1» в качестве третьего параметра при запуске:

```
export LD_LIBRARY_PATH=${PWD}
./reader_test 20 1 1
```