ЗАКРЫТОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО

«Аладдин Р.Д.»

УТВЕРЖДЕН

USB-НОСИТЕЛЬ JACARTA SF/ГОСТ

СПЕЦИАЛИЗИРОВАННОЕ СРЕДСТВО ДЛЯ БЕЗОПАСНОГО ХРАНЕНИЯ И ПЕРЕНОСА ИНФОРМАЦИИ

СПЕЦИАЛИЗИРОВАННЫЙ СЪЁМНЫЙ МАШИННЫЙ НОСИТЕЛЬ ИНФОРМАЦИИ

ВСТРОЕННОЕ ПРОГРАММНОЕ СРЕДСТВО JACARTA OS

Описание программы

Листов 36

Подпись и дата

Инв. № дубл.

Взам. инв. №

Подпись и дата

Инв. № подл.

Аннотация

В данном документе приведено описание логической структуры, функций составных частей и связей между ними встроенного программного средства JaCarta OS.

Содержание

1 Общие положения

1.1 Обозначение и наименование

Полное наименование: «Встроенное программное средство JaCarta OS».

Краткие наименования: программное средство JaCarta OS, программа JaCarta OS; JaCarta OS.

1.2 Программное обеспечение, необходимое для функционирования программы

Для функционирования JaCarta OS необходимо:

- программное обеспечения микроконтроллера смарт-карты, входящее в состав средства криптографической защиты информации (СКЗИ) «Криптотокен 2 ЭП»;
- программное обеспечение «USB-носитель JaCarta SF/ГОСТ. Комплект программных средств» (далее — ПО терминала), функционирующее на терминальном оборудовании и осуществляющее обращение к JaCarta OS в соответствии со стандартом ISO/IEC 7816-4:2005 [1].

1.3 Языки программирования, на которых написана программа

Программа написана на языке Си.

2 Функциональное назначение

Программа JaCarta OS предназначена для:

- обеспечения взаимодействия программ, выполняющихся на терминальном оборудовании (рабочей станции или персональном компьютере), к которому подключается изделие JaCarta SF/ГОСТ;
- управления жизненным циклом карты памяти в составе изделия JaCarta SF/ГОСТ;
- управления доступом к разделам карты памяти в составе изделия JaCarta SF/ГОСТ.

3 Описание логической структуры

3.1 Алгоритм программы

Алгоритм программы JaCarta OS включает в себя два шага:

- запуск программы;
- рабочий цикл программы.

3.1.1 Запуск программы JaCarta OS

При подаче электропитания на ведущий микроконтроллер выполняется начальная загрузка программы JaCarta OS. Запускается загрузчик процесса ядра (startup_cortex.S). В частности, производится:

- обнуление и инициализация векторов прерывания;
- очистка оперативной памяти ведущего микроконтроллера;
- запуск ядра программы JaCarta OS.

Ядро JaCarta OS выполняет следующие действия:

- выполняет свою инициализацию;
- запускает процесс App (специальный процесс JaCarta OS, см. подраздел 3.3.1).

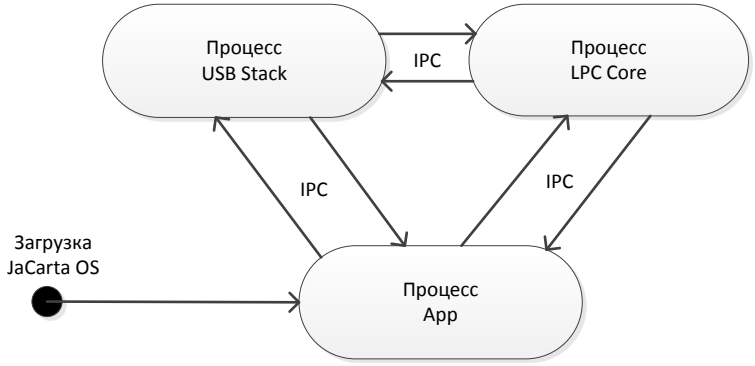
Процесс App запускает:

- процесс уровня драйверов JaCarta OS (LPC Core);
- процесс уровня аппаратной абстракции для поддержки USB-контроллера (USB Device);
- остальные процессы уровня аппаратной абстракции для поддержки периферийных устройств.

3.1.2 Рабочий цикл программы JaCarta OS

Рабочий цикл JaCarta OS состоит из последовательности переключений между рабочими циклами процессов (см. подраздел 3.2.1), выполняющих функции на различных уровнях программы. Передача управления между процессами осуществляется посредством межпроцессных сообщений (IPC — inter-process communication), см. подраздел 3.2.4.

Переключение между процессами может инициироваться самими процессами, в соответствии с их внутренней логикой, или происходить в результате обработки внешних по отношению к JaCarta OS событиями (например, прерываниями от контроллеров периферийных устройств/шин передачи данных).



LPC Core – процесс уровня библиотеки драйверов

USB Stack – процесс уровня аппаратной абстракции

App – специальный процесс JaCarta OS

Рисунок 1 — Диаграмма состояний рабочего цикла JaCarta OS (обобщённая схема)

3.1.3 Обработка событий

Обработка событий в ядре программы JaCarta OS влечёт за собой последовательность переключений между процессами разных уровней (драйверов, аппаратной абстракции и прикладного). Ниже приведены примеры последовательностей передачи управления между процессами в результате обработки типовых прерываний.

Одним из базовых событий JaCarta OS является обработка USB-пакета, поступающего от терминала (рисунок 2). Пакет данных, полученный от USB-контроллера по принципу конвейера проходит последовательную обработку в драйвере USB-контроллера (процесс LPC Core), на уровне аппаратной абстракции (процесс USB Stack), в специальном процессе JaCarta OS (процесс APP) и передаётся на уровень аппаратной абстракции смарт-карты (подпрограмма Smartcard в рамках процесса App) с последующим возвратом обработанных данных.

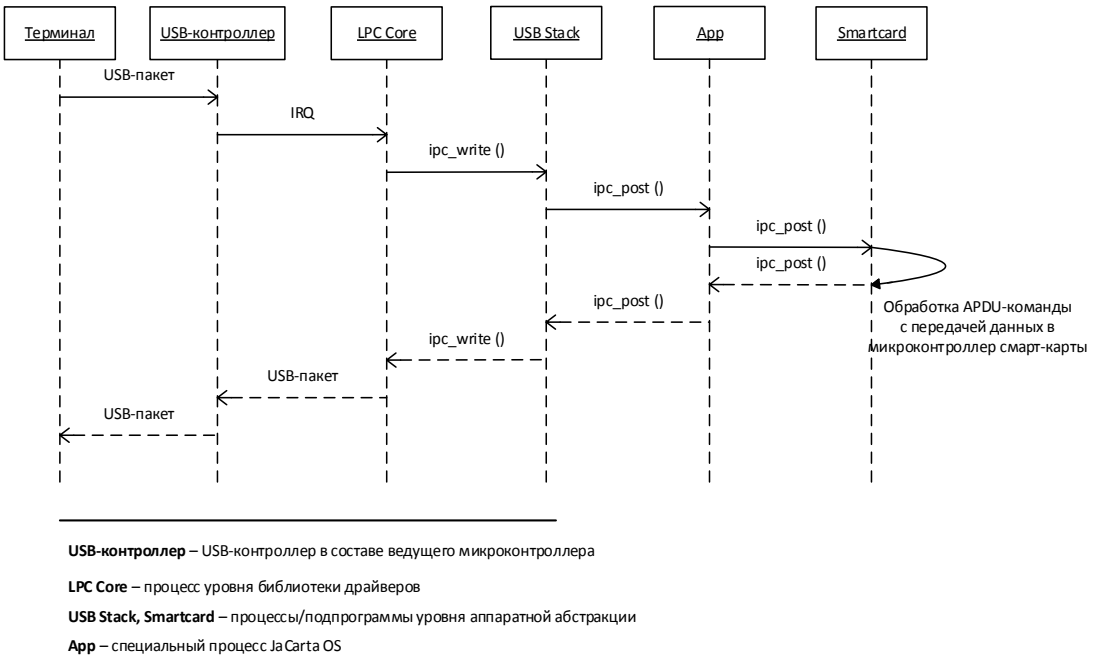


Рисунок 2 — Пример обработки команды, требующей обращения к микроконтроллеру смарт-карты

В случае если APDU-команда обращена к апплетам, функционирующим в ведущем микроконтроллере обработка данных, поступающих по USB, полностью реализуется в специальном процессе JaCarta OS (процесс APP, см. рисунок 3).

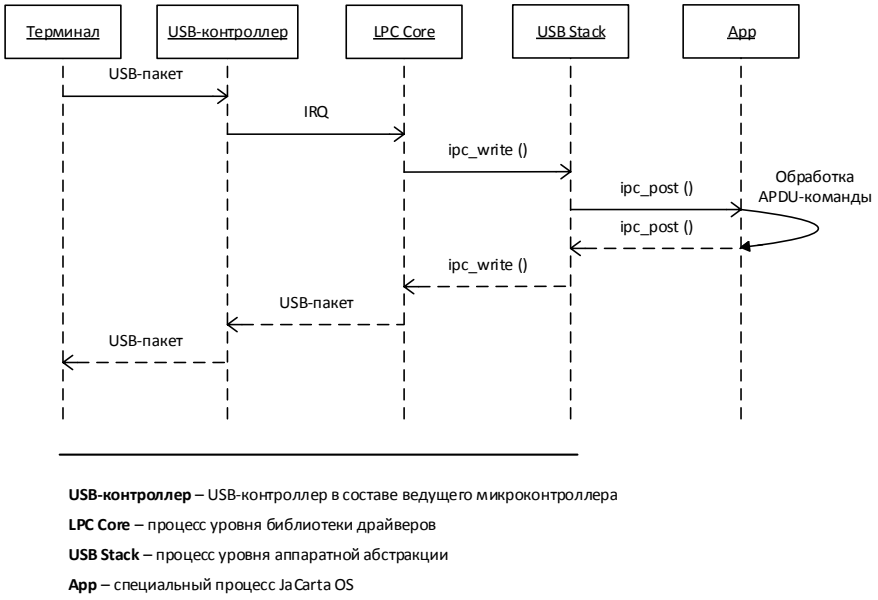


Рисунок 3 — Пример обработки команды в рамках ведущего микроконтроллера

Примером обработки внутреннего события является управление светодиодным индикатором (рисунок 4). По установленному с помощью системного таймера событию выполняется прерывание с последовательной обработкой на уровне драйверов (процесс LPC Core) и специальном процессе JaCarta OS (App), где принимается решение о задействовании инди-

катора.

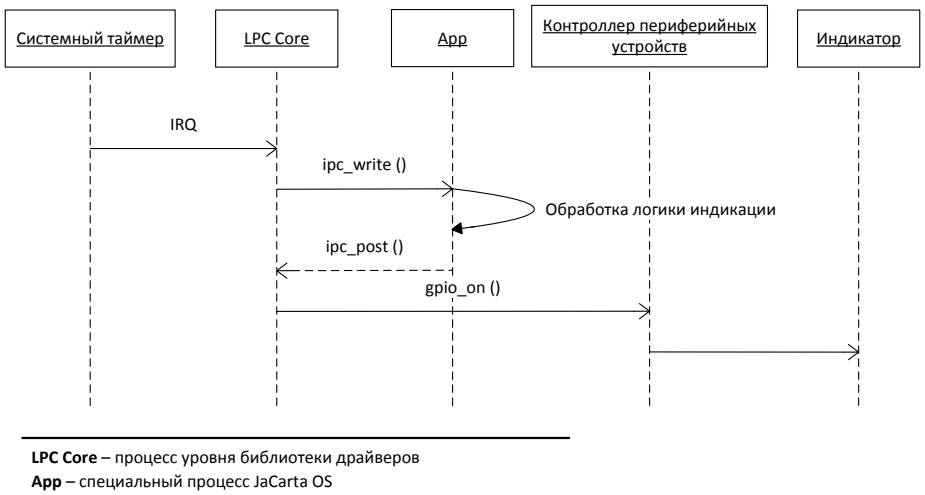


Рисунок 4 — Пример последовательности управления светодиодным индикатором

Для иллюстрации переключения между процессами при взаимодействии с микроконтроллером смарт-карты приведена диаграмма последовательности обработки прерывания I^2C (рисунок 5).

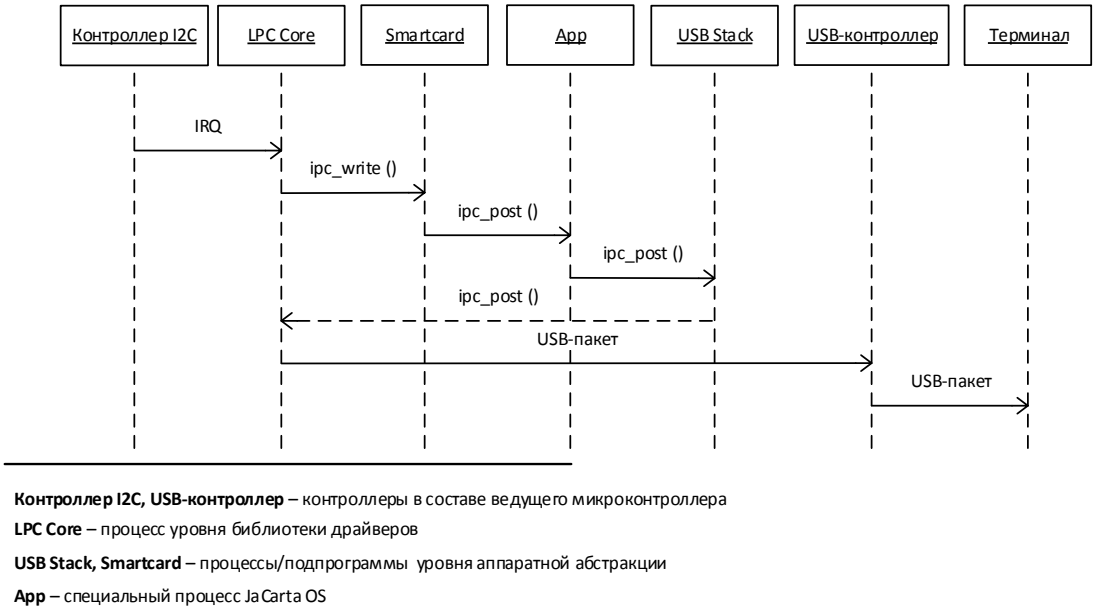


Рисунок 5 — Пример последовательности обработки прерывания от контроллера I^2C (обработка ответа от микроконтроллера смарт-карты)

3.1.4 Алгоритм апплета специального процесса JaCarta OS

При передаче управления одному из апплетов специального процесса JaCarta OS (см. подраздел 3.3.1) выполняется функция, реализующая алгоритм синтаксического анализа APDU-команды формата, соответствующего спецификации ISO/IEC 7816-4 [1]. В ходе вы-

полнения данного алгоритма производится:

- проверка принадлежности класса (поле CLA) команды множеству допустимых классов для данного апплета;
- проверка корректности значения поля INS, в контексте специального процесса JaCarta OS означающего вид команд данного апплета;
- проверка корректности значения поля P1, в контексте специального процесса JaCarta OS идентифицирующего данную команду.

После выполнения указанных проверок идентифицированная команда выполняется с использованием параметров, передаваемых в полях P2 и Data (при наличии последнего), результаты выполнения команды (поле DATA и коды состояния SW) возвращаются в виде выходных параметров функции.

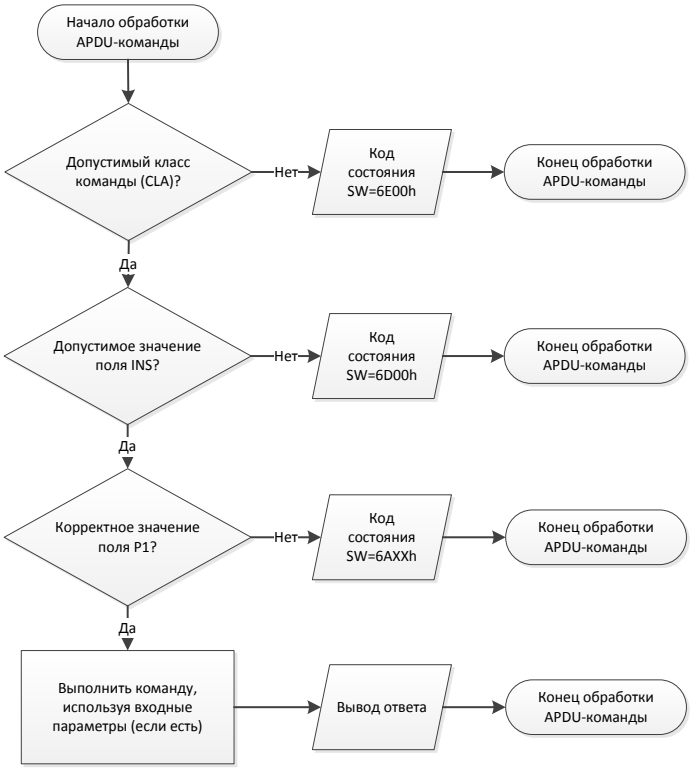


Рисунок 6 — Алгоритм обработки APDU-команды апплетом в рамках специального процесса JaCarta OS

3.2 Используемые методы

3.2.1 Системные вызовы

Системный вызов — это обращение процесса к ядру JaCarta OS.

Функция системного вызова имеет следующий синтаксис:

```
void svc_call(unsigned int num, unsigned int param1, unsigned int param2, unsigned int param3)
```

Параметры:

- num — номер системного вызова (список системных вызовов, как и сама функция находится в файле userspace/svc.h);
- param1, param2, param3 — параметры системного вызова. Специфичны для каждого из вызовов.

Во время системного вызова:

- значения параметров функции системного вызова (num, param1, param2 и param3) сохраняются на стеке;
- генерируется исключение SVCall;
- управление передаётся обработчику исключения (функции SVC_Handler);
- значения параметров системного вызова восстанавливаются из стека;
- вызывается функция супервизора с параметрами, восстановленными на предыдущем шаге.

Функция-обработчик исключения SVCall (SVC_Handler) имеет следующий синтаксис:

```
  .thumb_func
SVC_Handler:
    mrs   r0, psp
    ldmia r0, {r0-r3}
    bl    svc
```

Параметры:

- r0 – r3 — регистры общего назначения;
- psp — указатель стека процесса;
- svc — адрес вызываемой функции супервизора, получаемый на этапе компиляции JaCarta OS.

Функция супервизора имеет следующий синтаксис:

```
void svc(unsigned int num, unsigned int param1, unsigned int param2, unsigned int param3)
```

Параметры:

- num — номер системного вызова;
- param1, param2, param3 — параметры системного вызова. Специфичны для каждого из вызовов.
- Функция супервизора принимает на вход значение параметра num и в зависимости от его значения реализует функциональную логику ядра JaCarta OS, используя значения параметров param1, param2 и param3.
- Вызвать функцию супервизора напрямую из процесса нельзя, для этого необходимо выполнить системный вызов.

3.2.2 Регистрация глобальных объектов

Объект — это компонент JaCarta OS, имеющий идентификатор.

Объект процесса — это объект JaCarta OS, идентификатор которого находится в памяти принадлежащей процессу.

Для работы с объектом необходимо знать его идентификатор.

Идентификатор по-английски — handle. В JaCarta OS применяются следующие идентификаторы:

- Системные идентификаторы — указатели на системные объекты, доступные лишь для ядра.
- Пользовательские идентификаторы — идентификаторы объектов, специфичные для конкретных процессов или представляющие собой аппаратную абстракцию.

Значение идентификатора объекта становится известным только после того, как объект будет создан в процессе работы JaCarta OS.

Зарезервированы следующие идентификаторы:

- INVALID_HANDLE — неверный идентификатор;
- ANY_HANDLE — любой идентификатор. Используется как маска;
- KERNEL_HANDLE — идентификатор ядра.

Эти идентификаторы используются как именованные константы в языке Си и определены в файле userspace\types.h.

Сделать идентификатор объекта доступным всем процессам для чтения можно, зарегистрировав объект как глобальный.

Глобальные объекты — это объекты, зарегистрированные в ядре в списке глобальных объектов. При регистрации в указанный список помещаются индекс регистрируемого объекта и его идентификатор.

Индекс объекта — это заранее определённое (при написании JaCarta OS), соответствующее объекту число.

В дальнейшем узнать идентификатор глобального объекта можно, обратившись к ядру и указав индекс объекта, который всегда известен заранее.

Функция регистрации глобального объекта имеет следующий синтаксис:

```
void object_set(int idx, HANDLE object)
```

Параметры:

- idx — индекс объекта;
- object — значение любого (системного или пользовательского) идентификатора.

Здесь HANDLE — это целочисленный тип (unsigned int), определённый для наглядности как специальный тип в файле types.h.

После того, как идентификатор зарегистрирован, только владелец (процесс зарегистрировавший идентификатор) может его изменить.

Функция регистрации собственного процесса как глобального объекта имеет следую-

щий синтаксис:

```
void object_set_self(int idx)
```

Параметр idx — индекс объекта.

Эта функция помещает идентификатор и индекс процесса, из которого (в контексте которого) выполняется, в список глобальных объектов.

Функция получения идентификатора глобального объекта имеет следующий синтаксис:

```
HANDLE object_get (int idx)
```

Параметр idx — индекс объекта.

3.2.3 Процессы

Процесс — это сущность, используемая для описания функционирования программы JaCarta OS и предоставляющая заданные функциональные возможности на уровнях драйверов, аппаратной абстракции и прикладных программ.

Процесс реализуется с помощью бесконечного цикла, в рамках которого в соответствии с прикладной логикой осуществляется передача управления ядру JaCarta OS или другому процессу посредством межпроцессных сообщений (см. раздел 3.2.4).

3.2.3.1 Создание процесса

Функция создания процесса имеет следующий синтаксис:

__STATIC INLINE HANDLE process_create(const REX *rex)

Возвращает идентификатор процесса или значение INVALID_HANDLE в случае ошибки создания процесса.

Параметры:

- rex — указатель на структуру с параметрами процесса следующего вида:

```
typedef struct {
    const char* name;
    unsigned int size;
    unsigned int priority;
    unsigned int flags;
    unsigned int ipc_size;
    void (*fn) (void);
}REX;
```

где:

- name — указатель на строку с именем процесса;
- size — размер оперативной памяти процесса;
- priority — базовый приоритет процесса (чем меньше значение этого параметра,

тем выше приоритет процесса);

- flags — битовая структура параметров процесса (приведена в таблице 1):
- ipc_size — размер очереди сообщений;
- fn — точка входа процесса.

Таблица 1 — Битовая структура параметров процесса

Бит	b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Название	- (не используются)							REX_FLAG_PERSISTENT_NAME	- (не используются)																								PROCESS_FLAGS_ACTIVE
Допустимые значения битов	0	0	0	0	0	0	0	0;1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0;1

Роли битов:

- b_{31}—b_{25} зарезервированы;
- b_{24} — флаг хранения имени процесса в постоянной памяти, бит, определяющий наличие имени процесса в постоянной памяти (1 — имя присутствует в постоянной памяти; 0 — имя отсутствует в постоянной памяти);
- b_{23}—b_1 зарезервированы;
- b_0 — флаг активного процесса, бит, определяющий необходимость запуска процесса после создания (1 — запустить процесс после создания; 0 — не запускать процесс после создания).

3.2.3.2 Жизненный цикл процесса

Жизненный цикл процесса состоит из следующих этапов:

- инициализация процесса;
- бесконечный цикл (рабочий цикл).

Процесс не имеет выхода из бесконечного цикла. Он может быть только завершён извне.

Пример тела процесса:

```
void process1()
{
```

```
    IPC ipc;
    MY_STRUCT struct;
    process_init(&struct);
    bool need_post;
    for(;;)
    {
        error(ERROR_OK);
        ipc_read_ms(&ipc, 0, ANY_HANDLE);
        need_post = false;
        switch (ipc.cmd)
        {
              case CMD1:
                    need_post = process_cmd1(&struct, ipc.param1);
                    break;
              case CMD2:
                    need_post = process_cmd2(&struct, ipc.param1);
                    break;
              default:
                    break;
        }
        if (need_post)
         ipc_post_or_error(&ipc);
    }
}
```

3.2.3.3 Завершение процесса

Функция завершения любого процесса (кроме текущего) имеет следующий синтаксис:

```
void process_destroy(HANDLE process)
```

Параметр process — идентификатор процесса.

Функция завершения текущего процесса имеет следующий синтаксис:

```
void process_exit();
```

Эта функция завершает процесс в контексте которого выполняется.

3.2.3.4 Остальные функции управления процессом

Функции получения идентификатора текущего процесса имеют следующий синтаксис:

```
HANDLE process_get_current();
```

Функция получения флагов процесса имеет следующий синтаксис:

```
unsigned int process_get_flags(HANDLE process);
```

Параметр process — идентификатор процесса.

Функция установки флагов процесса имеет следующий синтаксис:

```
void process_set_flags(HANDLE process, unsigned int flags);
```

Параметры:

- process — идентификатор процесса;
- flags — флаги для установки.

Текущая версия JaCarta OS позволяет установить только флаг PROCESS_FLAGS_ACTIVE, что равносильно вызову функций process_freeze/process_unfreeze.

Функция возобновления исполнения процесса имеет следующий синтаксис:

```
void process_unfreeze(HANDLE process);
```

Параметр process — идентификатор процесса.

Функция остановки исполнения процесса имеет следующий синтаксис:

```
void process_freeze(HANDLE process);
```

Параметр process — идентификатор процесса.

Функция получения приоритета процесса имеет следующий синтаксис:

```
unsigned int process_get_priority(HANDLE process);
```

Параметр process — идентификатор процесса.

Функция получения приоритета текущего процесса имеет следующий синтаксис:

```
unsigned int process_get_current_priority();
```

Функция установки приоритета процесса имеет следующий синтаксис:

```
void process_set_priority(HANDLE process, unsigned int priority);
```

Параметры:

- process — идентификатор процесса;
- priority — значение приоритета.

Функция установки приоритета текущего процесса имеет следующий синтаксис:

void process_set_current_priority`(unsigned int priority);`

Параметр priority — значение приоритета.

Функция приостановки текущего процесса до тайм-аута имеет следующий синтаксис:

void sleep(SYSTIME* time);

При этом срабатывает исключение супервизора и происходит переключение на следующий процесс или останавливается ядро.

Параметр time — указатель на структуру SYSTIME.

Для хранения информации таймеров используется тип SYSTIME, который представляет собой следующую структуру:

```
typedef struct _SYSTIME{
    unsigned int sec;
    unsigned int usec;
} SYSTIME;
```

где:

- sec — количество секунд;
- usec — количество микросекунд.

3.2.3.5 Профилирование процессов

Профилирование процессов — деятельность по сбору информации об использовании памяти выделенной процессам, количестве объектов внутри процессов и времени их работы.

Для использования профилирования процессов предварительно должна быть включена опция #define KERNEL_PROFILING {1} в настройках ядра.

Функция тестирования переключения процессов имеет следующий синтаксис:

```
void process_switch_test();
```

Функция может использоваться для оценки производительности ядра.

Функция вывода отладочной информации о процессах и используемой памяти имеет следующий синтаксис:

```
void process_info();
```

3.2.4 Межпроцессные сообщения

Межпроцессные сообщения (IPC-сообщения) — это механизм JaCarta OS, позволяющий отправлять сообщение от одного процесса другому с одновременной передачей ему управления. Взаимодействие осуществляется при помощи вызова функции супервизора, то есть происходит обращение к ядру. При обработке межпроцессного сообщения ядром всегда используется указатель на структуру IPC, содержащийся в этом межпроцессном сообщении.

В JaCarta OS возможны IPC-сообщения двух типов:

- команда — сообщение без запроса результата выполнения, посылается в асинхронном режиме (процесс-отправитель продолжает свою работу и не ожидает ответа);
- запрос — сообщение с запросом результата выполнения, посылается в синхронном режиме (процесс-отправитель приостанавливает свою работу и ждёт ответ).

Функция отправки IPC-сообщения без ожидания ответа (асинхронный режим) имеет следующий синтаксис:

void ipc_post(IPC* ipc);

Функция помещает IPC-сообщение в очередь IPC-сообщений процесса-приёмника, не должна использоваться из контекста прерывания.

Параметр ipc — указатель на структуру IPC.

Структура IPC имеет следующий синтаксис:

```
typedef struct {
    HANDLE process;
    unsigned int cmd;
    unsigned int param1;
    unsigned int param2;
    unsigned int param3;
} IPC;
```

где:

- process — идентификатор процесса, которому направляется сообщение, либо от которого сообщение пришло. Отправить сообщение можно только другому процессу, прийти сообщение может от другого процесса или ядра, в последнем случае значение этого параметра будет KERNEL_HANDLE;
- cmd — код IPC-сообщения. Ядром программы JaCarta OS зарезервированы следующие сообщения:
  - IPC_PING — проверка доступности процесса. Все процессы обязаны отвечать на это сообщение таким же IPC.
  - IPC_STREAM_WRITE — сообщение о записи каким-либо процессом данных в поток;
  - IPC_TIMEOUT — тайм-аут программного таймера.
  - Значения кодов пользовательских сообщений должны быть не меньше константы IPC_USER;
- param1, param2, param3 — параметры, специфичные для кода запроса. В общем случае рекомендуется придерживаться следующего правила: param1 — идентификатор объекта, param2 — объект данных, param3 — размер или код ошибки.

Тип IPC-сообщения (с запросом результата выполнения или без запроса результата выполнения) задаётся при формировании кода cmd установкой флага HAL_REQ_FLAG.

В заголовочном файле userspace/ipc.h определены следующие макросы, для упрощения формирования и разбора кода IPC-сообщений:

- HAL_CMD(group, item) — макрос формирует код IPC-сообщения, без запроса результата выполнения, где: group — группа HAL сообщения, item — тип IPC-сообщения.

- HAL_REQ(group, item) — аналогичен предыдущему макросу, но формирует код IPC-сообщения, с запросом результата выполнения.
- HAL_GROUP(cmd) — макрос извлекает HAL группу из cmd IPC-сообщения.
- HAL_ITEM(cmd) — макрос извлекает IPC группу из cmd IPC-сообщения.

Данная функция аналогична предыдущей (void ipc_post(IPC* ipc)), но на вход принимает параметры IPC-сообщения и заполняет структуру IPC самостоятельно, имеет следующий синтаксис:

```
void ipc_post_inline(HANDLE process, unsigned int cmd, unsigned int param1, unsigned int param2, unsigned int param3);
```

Параметры:

- process — идентификатор процесса;
- cmd — код IPC-сообщения;
- param1, param2, param3 — параметры, специфичные для кода запроса.

Функция аналогичня ipc_post, но заменяющая значение param3 в структуре сообщения кодом последней ошибки имеет следующий синтаксис:

```
void ipc_post_or_error(IPC* ipc)
```

Параметр ipc — указатель на структуру IPC.

Функция ожидания сообщения имеет следующий синтаксис:

```
void ipc_read(IPC* ipc);
```

Параметр ipc — указатель на структуру IPC.

Функция реализует ожидание сообщения, переданного процессом с произвольным идентификатором.

Функция получения запроса сообщения от процесса с заданным идентификатором имеет следующий синтаксис:

```
bool ipc_read_ms(IPC* ipc, unsigned int ms, HANDLE wait_process);
```

Функция возвращает true, если после выхода из функции у процесса не установлено кода ошибки. Время ожидания может быть равно нулю, в таком случае процесс будет находиться в режиме ожидания до первого сообщения, соответствующего запросу.

Параметры:

- ms — время ожидания сообщения в миллисекундах;
- wait_process — идентификатор процесса, от которого ожидается сообщение. Для чтения сообщения от процесса с произвольным идентификатором необходимо использовать значение ANY_HANDLE.

Функция отправки сообщения другому процессу с ожиданием ответа (синхронный режим) имеет следующий синтаксис:

```
void call(IPC* ipc);
```

Параметр ipc — указатель на структуру IPC;

Функция совмещает в себе функции svc_call и ipc_peek. Сообщение ожидается только от того процесса, которому было отправлено. Функцию нельзя использовать, если были выполнены синхронные запросы ipc_call к процессу при условии, что к этому же процессу уже существует асинхронный запрос. IPC-сообщение обрабатывается мгновенно, без помещения сообщения в очередь процесса-приёмника.

Функция, представляющая собой модифицированную версию функции call с явным заданием параметров сообщения IPC, имеет следующий синтаксис:

```
void ack(HANDLE process, unsigned int cmd, unsigned int param1, unsigned int param2, unsigned int param3);
```

Параметры:

- process — идентификатор процесса;
- cmd — код IPC-сообщения;
- param1, param2, param3 — параметры, специфичные для кода запроса.

Функция, представляющая собой версию функции call с явным заданием параметров IPC и возвратом идентификатора от процесса, к которому осуществлялся запрос, имеет следующий синтаксис:

```
unsigned int get(HANDLE process, unsigned int cmd, unsigned int param1, unsigned int param2, unsigned int param3);
```

Параметры:

- process — идентификатор процесса;
- cmd — код IPC-сообщения;
- param1, param2, param3 — параметры, специфичные для кода запроса.

Идентификатор передаётся в param2. В случае ошибки возвращается INVALID_HANDLE и устанавливается код последней ошибки контексте текущего процесса.

3.3 Структура программы с описанием функций составных частей и связи между ними

JaCarta OS включает в себя следующие компоненты (рисунок 7):

- ядро;
- библиотеку драйверов LPC Lib;
- библиотеку уровня аппаратной абстракции, включающую в себя:
  - библиотеку управления USB-интерфейсом;
  - библиотеку управления микроконтроллером смарт-карты;
  - библиотеку управления периферийными устройствами;
- специальный процесс JaCarta OS.

Ядро и библиотеки драйверов и уровня аппаратной абстракции составляют *базовую часть JaCarta OS*. Специальный процесс JaCarta OS (App) при своей работе взаимодействует с базовой частью JaCarta OS, а именно с библиотекой уровня аппаратной абстракции.

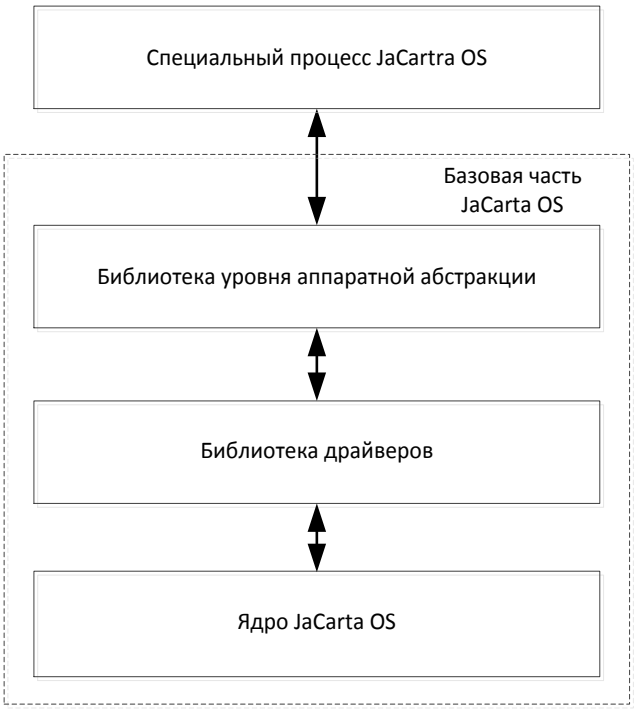


Рисунок 7 — Структура программы JaCarta OS

Классификация исходных кодов базовой части JaCarta OS в соответствии с описанной структурой приведена в приложении на с. 77.

3.3.1 Специальный процесс JaCarta OS

Специальный процесс содержит логику реализации функционального назначения JaCarta OS (см. раздел 2).

В состав специального процесса JaCarta OS (рисунок 8) входят следующие модули:

- фильтр APDU-команд (Firewall);
- диспетчер APDU-команд;
- служебный апплет;
- виртуальный апплет;
- авторизационный апплет;
- блок служебных функций специального процесса.

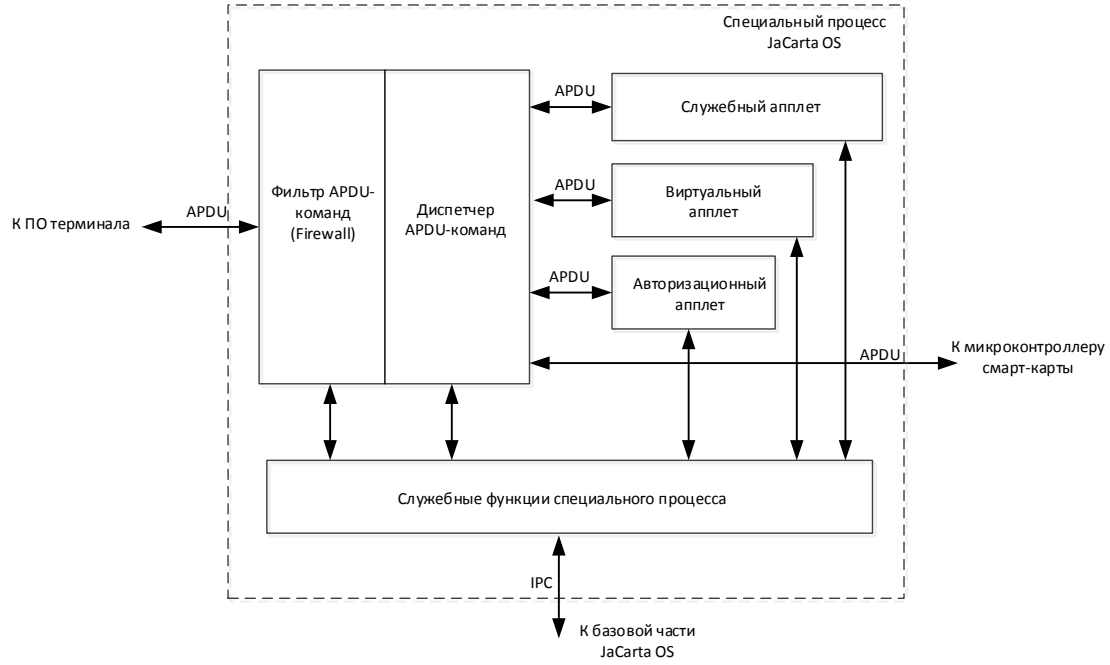


Рисунок 8 — Функциональная структура специального процесса JaCarta OS

3.3.2 Фильтр APDU-команд

Фильтр APDU-команд (Firewall) выполняет блокировку APDU-команд, не отвечающих критериям, которые задаются специальными масками (см. фильтрацию APDU-команд [2]). В случае если полученная команда не отвечает критериям маски, фильтр возвращает ошибку с кодом состояния (SW) 6785h.

3.3.3 Диспетчер APDU-команд

Диспетчер APDU-команд осуществляет маршрутизацию APDU-команд к апплету, функционирующему в рамках специального процесса JaCarta, либо в составе СКЗИ «Криптотокен 2 ЭП».

Маршрутизация APDU-команд осуществляется по принципу триггера с множеством устойчивых состояний (рисунок 9). Событием переключения триггера является получение команды выбора апплета.

После получения команды выбора апплета, все APDU-команды направляются данному апплету до очередного получения команды выбора апплета. В случае если ни один апплет не выбран, команды направляются микроконтроллеру смарт-карты.

Признак выбора апплета в случаях служебного, виртуального и авторизационного апплетов, хранится в самом апплете.

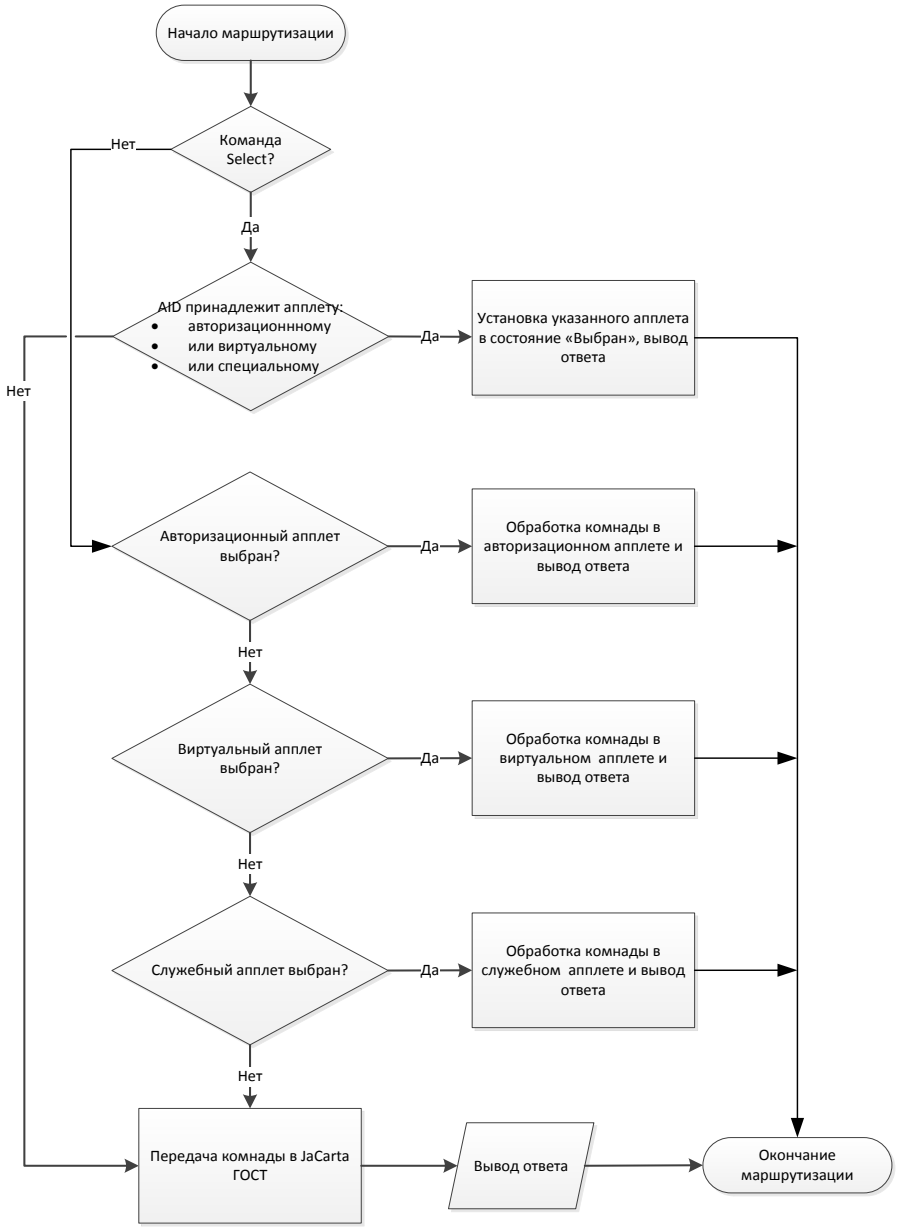


Рисунок 9 — Блок-схема маршрутизации APDU-команд

В случае передачи команды Select в JaCarta ГОСТ проверяется факт положительного завершения команды. Если команда завершилась успешно (SW 9000h), то флаг выбранного апплета сбрасывается.

3.3.4 Служебный апплет

Служебный апплет выполняет следующие функции:

- запись блока конфигурационных данных об ЭН в процессе его производства;
- получение конфигурационных данных об ЭН в процессе его эксплуатации;
- получение дополнительной служебной и эксплуатационной информации об ЭН:
  - номера версии апплета;
  - размера блока конфигурационных данных;

  - объёма свободной памяти ЭСППЗУ на микроконтроллере смарт-карты ЭН;
- получения значений эксплуатационных параметров изделия JaCarta SF/ГОСТ:
  - номера этапа жизненного цикла карты памяти;
  - счётчика подключений изделия JaCarta SF/ГОСТ к порту USB;
  - счётчика прерванных APDU-команд;
  - счётчика невыполненных APDU-команд «Отключить скрытые разделы»;
  - счётчика общего времени работы изделия JaCarta SF/ГОСТ;
  - версии JaCarta OS;
  - кода последней ошибки, сохранённого в оперативной памяти изделия JaCarta SF/ГОСТ;
- получение информации об ЭСППЗУ ведущего микроконтроллера, в частности значений:
  - полного размера доступного ЭСППЗУ;
  - текущего смещения используемой на данный момент ячейки памяти относительно начала ЭСППЗУ;
  - размера ячейки хранения служебных счётчиков [2];
  - счётчика фактов записи в текущую ячейку хранения служебных счётчиков;
  - счётчика подключений изделия JaCarta SF/ГОСТ к порту USB;
  - счётчика прерванных APDU-команд;
  - счётчика невыполненных APDU-команд «Отключить скрытые разделы» из состояния подключённых скрытых разделов;
  - счётчика общего времени работы изделия JaCarta SF/ГОСТ (с начала эксплуатации) в минутах;
- получение комплекса данных для аудита изделия JaCarta SF/ГОСТ.

Приведённые выше функции реализуются в процессе обработки набора APDU-команд служебного апплета [2]. Типовой алгоритм обработки APDU-команды приведён в разделе 3.1.4.

APDU-команда подаётся из терминала на вход апплета в соответствии с описанием из подраздела «Обработка событий» (см. диаграмму последовательностей на рисунок 3).

Входные и выходные данные служебного апплета приведены соответственно в разделах 6.1 и 6.2.

3.3.5 Виртуальный апплет

Виртуальный апплет выполняет следующие функции:

- получение серийного номера ведущего микроконтроллера;

- получение данных о виртуальном апплете, в частности:
  - версии виртуального апплета;
  - признак возможности инвалидации [3] контрольной суммы векторов прерываний (для последующего вызова загрузчика NXP USB Bootloader);
  - длины ключа доступа к загрузчику NXP USB Bootloader;
  - контрольной суммы параметров виртуального апплета;
- получение сведений о конфигурации изделия JaCarta SF/ГОСТ, в частности:
  - кода последней ошибки, сохранённого в оперативной памяти;
  - серийного номера смарт-карты;
  - счётчика включений изделия JaCarta SF/ГОСТ;
  - версии JaCarta OS;
  - поля «Служебная информация производства» [3] (mdata);
- проверка ключа доступа к NXP USB Bootloader [3], с целью авторизации на администрирование USB-носителя;
- смена ключа доступа к NXP USB Bootloader;
- включение режима прошивки средствами NXP USB Bootloader;
- запрет режима прошивки средствами NXP USB Bootloader;
- получение параметров загрузчика JC SF GOST Booter;
- управление прошивкой средствами загрузчика JC SF GOST Booter.

Приведённые выше функции реализуются в процессе обработки набора APDU-команд виртуального апплета [3]. Типовой алгоритм обработки APDU-команды приведён в разделе «Алгоритм апплета специального процесса JaCarta OS».

APDU-команда подаётся из ПО терминала на вход апплета в соответствии с описанием из подраздела 3.1.3 (см. диаграмму последовательностей на рисунок 3).

Входные и выходные данные виртуального апплета приведены соответственно в разделах 6.1 и 6.2.

3.3.6 Авторизационный апплет

Авторизационный апплет содержит основную логику поддержки карты памяти (управления её жизненными циклами [4]), обеспечения аутентификации и авторизации пользователя для работы с картой памяти, а также защищённую запись и чтение информации с карты памяти изделия JaCarta SF/ГОСТ. В частности, авторизационный апплет выполняет следующие функции:

- разметку и переразметку карты памяти на разделы (открытые и скрытые разделы RW и CD-ROM [4]);
- подключение открытых разделов на постоянной основе для записи и чтения из них

(раздел RW) или только чтения (раздел CD-ROM);

- установку размеров ISO-образов для их последующей записи на карту памяти;
- получение блока служебной информации [4] о карте памяти;
- удаление с карты памяти всех разделов и обнуление содержимого всех секторов;
- предоставление программного интерфейса для реализации протоколов аутентификации и авторизации пользователя, подключения скрытых разделов / автономного подключения скрытых разделов на карте памяти, частности:
  - получение данных о карте памяти для вычисления ключей авторизации и специального преобразования скрытых разделов;
  - запись мастер-ключа (массива ключей) авторизации / запись ключа (массива ключей) авторизации и передача данных для вычисления контрольной суммы ключа специального преобразования скрытых разделов;
  - активация мастер-ключа / ключа авторизации;
  - генерация параметров инициализации для ЭН пользователя;
  - подготовка ЭН пользователя к автономному подключению скрытых разделов;
  - подключение скрытых разделов RW и CD-ROM;
  - выработка служебной последовательности для подключения скрытых разделов на ЭН пользователя;
  - подключение скрытых разделов RW и CD-ROM на ЭН пользователя в автономном режиме;
  - инициация протокола подключения скрытых разделов RW и CD-ROM на ЭН пользователя;
  - отключение скрытых разделов RW и CD-ROM.

Приведённые выше функции реализуются в процессе обработки набора APDU-команд авторизационного апплета [4]. Типовой алгоритм обработки APDU-команды приведён в разделе 3.1.4.

APDU-команда подаётся из ПО терминала на вход апплета в соответствии с описанием из подраздела 3.1.3 (см. диаграмму последовательностей на рисунок 3).

Входные и выходные данные апплета приведены соответственно в разделах 6.1 и 6.2.

3.3.7 Блок служебных функций специального процесса

Служебные функции специального процесса включают в себя:

- функции работы с микроконтроллером смарт-карты;
- функции прикладного уровня работы со смарт-картой;

- функции прикладного уровня поддержки интерфейса USB CCID [5];
- функции прикладного уровня поддержки интерфейса USB [6] — [7];
- функции поддержки работы с CD-ROM-разделами карты памяти;
- функции контроля целостности ПО JaCarta OS;
- функции вычисления контрольных сумм CRC32;
- функции прикладного уровня для специального преобразования скрытых разделов карты памяти;
- функции управления ячейкой хранения служебных счётчиков [2] в ЭСППЗУ;
- функции хранения массива ключей авторизации в ЭСППЗУ;
- функции прикладного уровня для работы с ЭСППЗУ;
- функции эмуляции временных дисков на период записи ISO-образов в разделы CD-ROM карты памяти;
- функции работы с файловой системой;
- функции прикладного уровня поддержки светодиодного индикатора;
- функции прикладного уровня для обращения к скрытым разделам карты памяти;
- функции специального преобразования скрытых разделов карты памяти [8];
- функции вычисления контрольной суммы с использованием ключа специальных преобразований [8];
- функции вычисления контрольной суммы [8].

Вызов служебных функций осуществляется непосредственно из функциональных модулей (апплетов, диспетчера APDU-команд) специального процесса.

3.4 Связи программы с другими программами

В процессе функционирования ПО JaCarta OS взаимодействует со следующими программами:

- ПО терминала;
- СКЗИ «Криптотокен 2 ЭП»;
- ПО JaCarta OS, функционирующее в составе другого экземпляра изделия JaCarta SF/ГОСТ.

3.4.1 Связи программы с ПО терминала

Взаимодействие программы с ПО терминала осуществляется в соответствии со спецификациями USB [6] — [7], CCID [5] и ISO/IEC 7816-4 [1] в рамках реализации функционального назначения программы (см. раздел 2).

3.4.2 Связи программы с ПО микроконтроллера смарт-карты

Взаимодействие программы с ПО микроконтроллера смарт-карты осуществляется в рамках реализации протокола подключения скрытых разделов (см. «Ключевая схема управления доступом к скрытым разделам флеш-памяти» [9]).

3.4.3 Связи программы с ПО JaCarta OS другого экземпляра изделия JaCarta SF/ГОСТ

Взаимодействие программы с ПО JaCarta OS другого экземпляра изделия JaCarta SF/ГОСТ осуществляется посредством ПО терминала при инициализации электронного носителя пользователя и подключении скрытых разделов (см. «Ключевая схема управления доступом к скрытым разделам флеш-памяти» [9]).

4 Используемые технические средства

Программа JaCarta OS поставляется в составе USB-носителя JaCarta SF/ГОСТ и функционирует на микроконтроллерах серии NXP LPC18x. Помимо указанного аппаратного обеспечения для корректного функционирования программы требуется наличие:

- микроконтроллера смарт-карты с установленными и настроенными на нём компонентами СКЗИ «Криптотокен 2 ЭП»;
- карты microSD (*карты памяти*).

5 Вызов и загрузка

Загрузка программы JaCarta OS выполняется автоматически при подаче электропитания на микроконтроллеры в составе изделия JaCarta SF/ГОСТ.

6 Входные и выходные данные

Параметры входных и выходных данных программы JaCarta OS при взаимодействии с микроконтроллером смарт-карты определяется спецификациями I2C [10] и ISO/IEC 7816-4 [1].

Параметры входных и выходных данных JaCarta OS при взаимодействии с ПО терминала, определяются спецификациями USB [6]—[7], CCID [5] и ISO/IEC 7816-4 [1].

6.1 Входные данные

Входными данными JaCarta OS являются APDU-команды, поступающие от ПО терминала.

Формат APDU-команд, адресованных служебному апплету, приведён в описании APDU [2] данного апплета.

Формат APDU-команд, адресованных виртуальному апплету, приведён в описании APDU [3] данного апплета.

Формат APDU-команд, адресованных авторизационному апплету, приведён в описании APDU [4] данного апплета.

Формат APDU-команд, которые адресованы СКЗИ «Криптотокен 2 ЭП», приведён в описаниях APDU [11] и [12] данного СКЗИ.

6.2 Выходные данные

Выходными данными JaCarta OS является информация, формируемая:

- служебными функциями специального процесса (см. коды ошибок JaCarta OS [3]);
- служебным апплетом (см. описание APDU-ответов [2] данного апплета);
- виртуальным апплетом (см. описание APDU-ответов [3] данного апплета);
- авторизационным апплетом (см. описание APDU-ответов [4] данного апплета),

а также APDU-команды, передаваемые диспетчером средству криптографической защиты «Криптотокен 2 ЭП».

Перечень принятых сокращений

APDU	–	Application protocol data unit, блок данных прикладного протокола
CCID	–	Circuit(s) Cards Interface Device, устройство сопряжения смарт-карт
GPIO	–	General Purpose Input/Output, интерфейс ввода-вывода общего назначения
I2C	–	Inter-Integrated Circuit, интерфейс I2C
IO	–	Input/output, ввод-вывод
IPC	–	Inter-process communication, межпроцессная коммуникация
NVIC	–	Nested vectored interrupt controller, контроллер вложенных векторных прерываний
REX	–	Realtime Exokernel, экзоядро реального времени
SVC	–	Supervisor call, механизм запроса к супервизору из прикладного процесса
UART	–	Universal asynchronous receiver-transmitter, универсальный асинхронный приёмопередатчик
ВМК	–	Ведущий микроконтроллер
ПО	–	Программное обеспечение
СКЗИ	–	Средство криптографической защиты информации
ЭН	–	Электронный носитель

Перечень принятых терминов

ПО терминала	–	программное обеспечение «USB-носитель JaCarta SF/ГОСТ. Комплект программных средств», функционирующее на терминальном оборудовании и осуществляющее обращение к JaCarta OS в соответствии со стандартом ISO/IEC 7816-4:2005 [13]
Изделие JaCarta SF/ГОСТ	–	«USB-носитель JaCarta SF/ГОСТ. Специализированный съёмный машинный носитель информации»
Межпроцессное сообщение	–	механизм JaCarta OS, позволяющий отправлять сообщение от одного процесса другому с одновременной передачей ему управления
Системный вызов	–	обращение процесса к ядру JaCarta OS
Процесс	–	сущность, используемая для описания функционирования программы JaCarta OS и предоставляющая заданные функциональные возможности на уровнях драйверов, аппаратной абстракции и прикладных программ
Объект	–	компонент JaCarta OS, имеющий идентификатор
Объект процесса	–	объект JaCarta OS, идентификатор которого находится в памяти принадлежащей процессу
Системный идентификатор	–	указатель на системные объекты, доступные лишь для ядра
Пользовательский идентификатор	–	идентификатор объекта, специфичный для конкретного процесса или представляющий собой аппаратную абстракцию
Глобальный объект	–	объект, зарегистрированный в ядре в списке глобальных объектов
Индекс объекта	–	заранее определённое (при написании JaCarta OS), соответствующее объекту число

Приложение А

(рекомендуемое)

Классификация исходных текстов базовой части JaCarta OS

В настоящем приложении приведены основные файлы базовой части JaCarta OS, с отнесением их к разным уровням архитектуры данной программы.

Полный список файлов ПО JaCarta OS с описанием их назначения приведён в тексте программы [8].

Ядро JaCarta OS

Ядро ядра (Kernel)

- startup_cortex.S - файл инициализации (первоначальная загрузка) ведущего микроконтроллера (ВМК);
- kcortexm.c/h — обработка ошибок ядра ВМК;
- dbg.c/h — отладка ядра JaCarta OS;
- kernel.c/h — ядро JaCarta OS;
- kio.c/h — библиотека ввода/вывода JaCarta OS;
- kipc.c/h — библиотека IPC;
- kirq.c/h — библиотека IRQ;
- kobject.c/h — библиотека для работы с объектами;
- kprocess.c/h — библиотека для работы с процессами;
- kstream.c/h — библиотека для работы с потоками;
- ksystime.c/h — библиотека системного время JaCarta OS.

Библиотеки ядра

- lib_lib.c/h — файл содержащий настройки библиотек для работы в JaCarta OS;
- lib_std.c/h — работа с памятью;
- lib_stdio.c/h — работа с вводом выводом;
- lib_systime.c/h — работы с системным временем;
- lpc_lib_gpio.c — работа JaCarta OS с портами GPIO;
- pool.c/h — работа памяти;
- printf.c/h — реализация printf.

Библиотеки уровня драйверов

- lpc_core_private.h — описание структуры ядра процесса работы с драйверами пе-

риферии ВМК;

- lpc_core.c/h — логика процесса работы с драйверами периферии ВМК;
- lpc_eep.c/h — библиотека работы с EEPROM;
- lpc_gpio.c/h — библиотека работы с периферийными устройствами;
- lpc_i2c.c/h — библиотека работы с шиной I^2C;
- lpc_power.c/h — библиотека работы с ядром ВМК (установка рабочей частоты, питания и пр.);
- lpc_timer.c/h — библиотека работы с аппаратными таймерами;
- lpc_uart.c/h — библиотека работы с UART;
- lpc_usb.c/h — библиотека работы с USB;
- lpc11uxx_bits.h — описание регистров.

Библиотеки уровня аппаратной абстракции

- ccidd.c/h — CCID уровень;
- usbd.c/h — USB стек;
- leds.c/h, userspace/gpio.h — библиотеки поддержки пассивных периферийных устройств;
- sys.h; stdio.c/h; stdlib.c/h — служебные библиотеки.

Список литературы

1 International Standard ISO/IEC 7816-4. Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange [Текст]. — Second edition. — 2005-01-15. — ISO/IEC, 2005. — 90 с.

2 Служебный апплет. Описание APDU [Текст]

3 Виртуальный апплет. Описание APDU [Текст]

4 USB-носитель JaCarta SF/ГОСТ. JaCarta OS. Авторизационный апплет. Описание APDU [Текст] / ЗАО «Аладдин Р.Д.»

5 Universal Serial Bus. Device Class: Smart Card. CCID. Specification for Integrated Circuit(s) Cards Interface Devices. [Текст] — Revision 1.1 — April 22rd, 2005 — 123 с.

6 Universal Serial Bus. Mass Storage Class. Specification Overview. [Текст] — Revision 1.4 — February 19, 2010 — с.14

7 Universal Serial Bus. Mass Storage Class. Bulk-Only Transport. [Текст] — Revision 1.0 — September 31, 1999 — с.22

8 USB-носитель JaCarta SF/ГОСТ. Комплект программных средств. Программное средство JaCarta OS. Текст программы [Текст]

9 JaCarta SF/ГОСТ. Ключевая схема управления доступом к скрытым разделам флеш-памяти. [Текст] / ЗАО «Аладдин Р.Д.»

10 UM10204. I2C-bus specification and user manual. [Текст] — Rev. 6. — 4 April 2014 — NXP Semiconductors, 2014 — 65 с.

11 Средство криптографической защиты информации «Криптотокен 2 ЭП» в составе изделия JaCarta ГОСТ. Описание APDU для прикладных программ [Текст]

12 Средство криптографической защиты информации «Криптотокен 2 ЭП» в составе изделия JaCarta ГОСТ. Описание APDU для администрирования [Текст]

13 International Standard ISO/IEC 7816-4. Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange [Текст]. — Second edition. — 2005-01-15. — ISO/IEC, 2005. — 90 с.