Технологии QNX и КПДА в России

Санкт-Петербург, 30 октября 2018

«Подключение специализированных устройств ввода в среде QNX и ЗОСРВ Нейтрино»

Александр Покид, СВД Встраиваемые Системы

СВД Встраиваемые Системы

SWD Software

Подсистема ввода/вывода

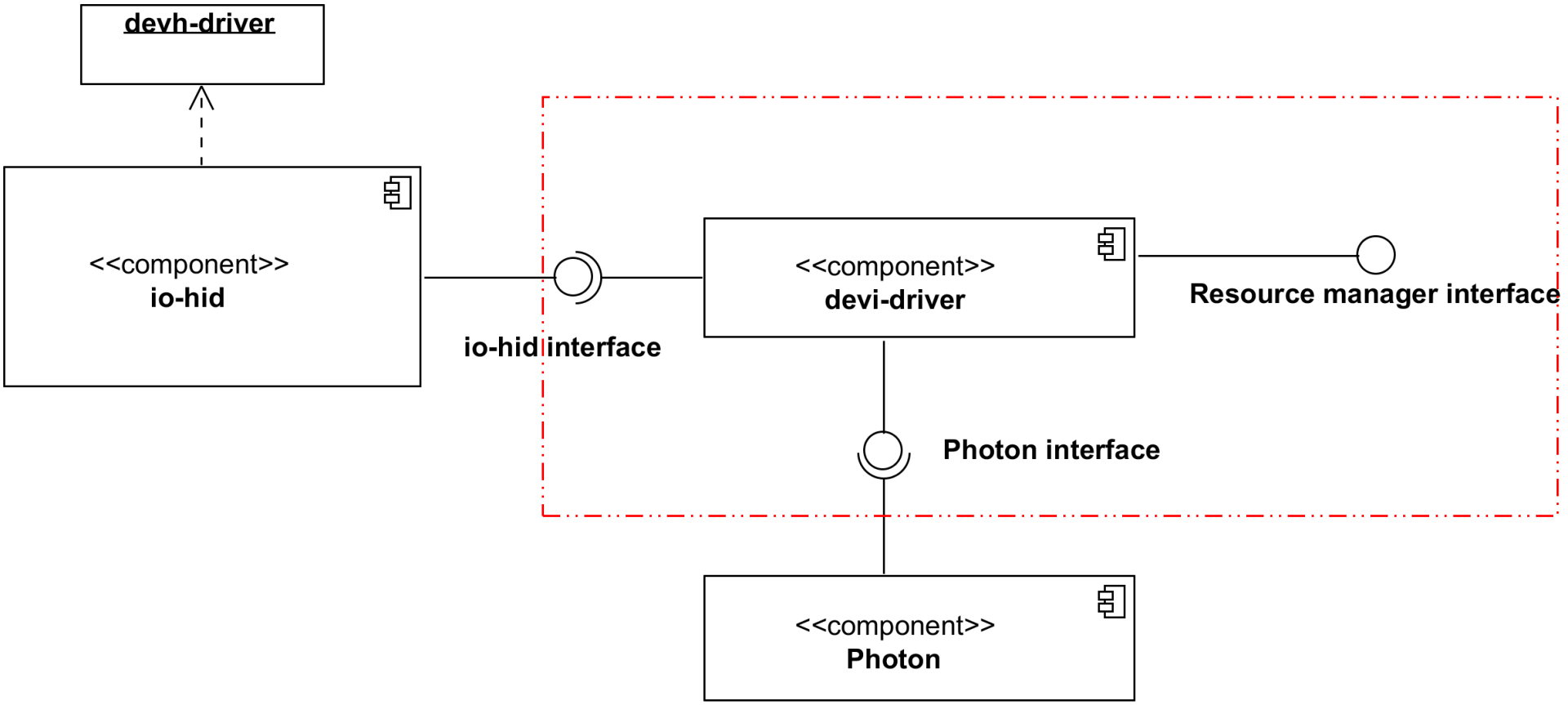
Менеджеры ввода обеспечивают поддержку следующих классов устройств:

- Клавиатуры;
- Устройства абсолютного позиционирования;
- Устройства относительного позиционирования.

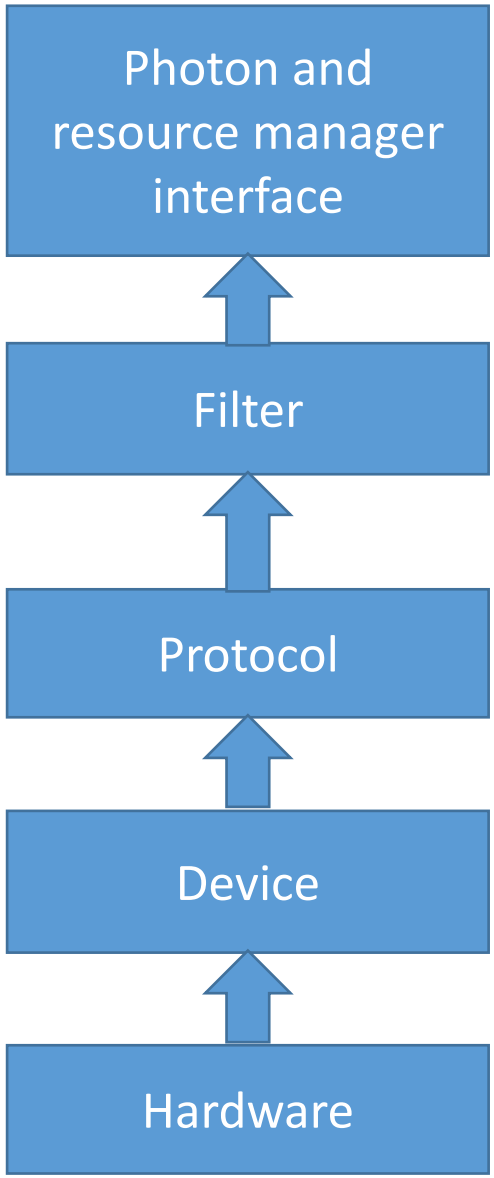
Содержание

- Общая схема работы подсистемы ввода;
- Драйверы семейства devi-*:
  - Принципы работы;
  - Сборка и запуск;
  - Общие алгоритмы инициализации и ввода данных;
  - Доступный API;
- Драйвера семейства devh-*:
  - Общий принцип работы;
  - Сборка и запуск;
  - Общие алгоритмы инициализации и ввода данных;
  - Основные используемые структуры данных.

Общая структура подсистемы ввода



Event bus



События:

- Относительные;
- Абсолютные;
- Клавиатуры.

СВД Встраиваемые Системы

SWD Software

Запуск драйвера

```
devi-<имя_драйвера> <опции_драйвера>
<протокол> <опции протокола>
[<устройство> <опции_устройства>]
[<фильтр> <опции_фильтра>]
```

Пример:

```
# devi-hirun ps2 kbd -2 &
```

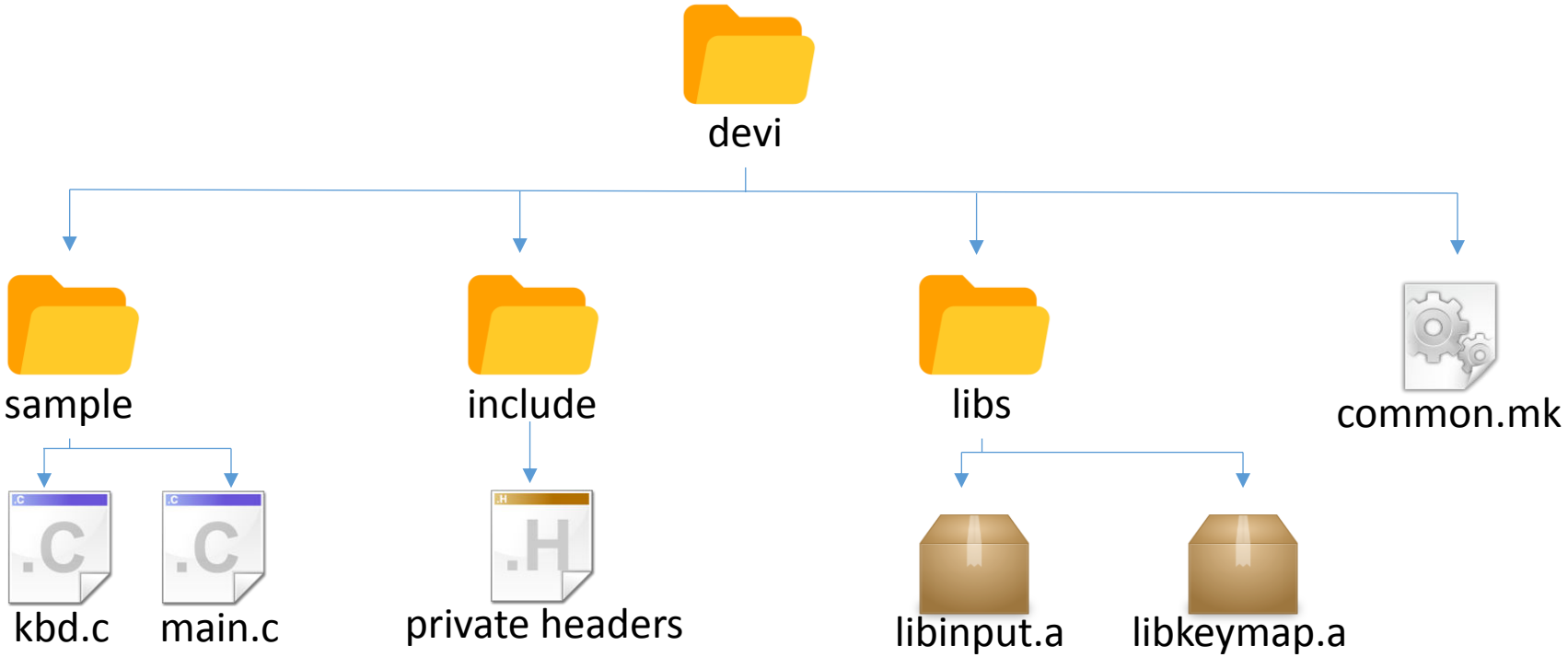
Интерфейсы и взаимодействие

Реализуемые интерфейсы:

1. Photon;
2. Менеджер ресурсов.

```
#include <sys/dcmd_input.h>
...
int main()
{
  struct _keyboard_ctrl kbctrl;
  /* Some initialization */
  ...
  int fd = open("/dev/keyboard0", O_RDWR);
  if (devctl(fd, _KEYBOARDGETCTRL, &kbctrl, sizeof(kbctrl),
NULL) != EOK) {
            /* Error handling*/
      }
  /* Keyboard control information now in structure kbctrl */
}
```

Дерево сборки



Дескриптор модуля

```
struct _input_module {
    input_module_t *up;     /* Up module in the bus line */
    input_module_t *down;   /* Down module in the bus line */
    struct Line* line;      /* Bus line */
    int flags;              /* flags */
    int type;               /* type of module */
    char name[12];          /* Module name */
    char date[12];          /* Date of compilation */
    const char* args;       /* Module arguments */
    void* data;             /* Private module data */
    /* Callbacks */
    ...
}
```

Дескриптор модуля (продолжение)

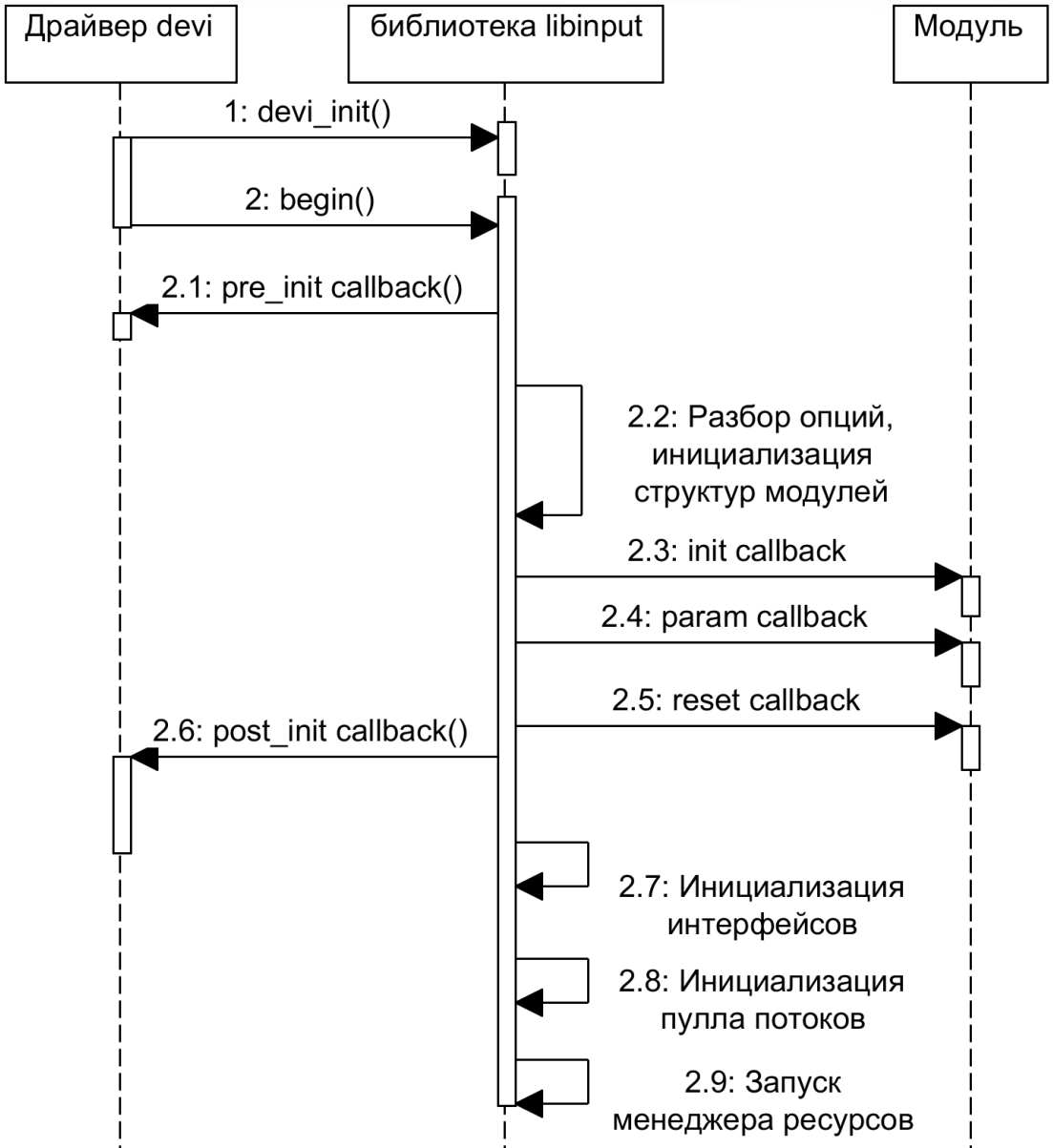
```
struct _input_module {
/* Module parameters */
...
int (*init)     (input_module_t* module);
int (*reset)    (input_module_t* module);
int (*input)    (input_module_t* module, int num, void* data);
int (*output)   (input_module_t*, void* data, int num);
int (*pulse)    (message_context_t* ctx, int code,
                 unsigned flags, void* data);
int (*parm)     (input_module_t* module, int code, char* optarg);
int (*devctrl)  (input_module_t* module, int event, void* data);
int (*shutdown)(input_module_t* module, int delay);
 }
```

Инициализация драйвера

- Заполнить структуру дескриптора модуля;
- Заполнить структуру _common_callbacks;
- Выполнить инициализацию библиотеки libdevinput, вызвав функцию: `devi_init(_common_callbacks*)`
- Выполнить всю необходимую инициализацию прочих библиотек и ресурсов;
- Передать управление библиотеке libdevinput, вызвав функцию `begin(int argc, char** argv).`

```
struct _common_callbacks {
    int nCallbacks;
    int (*pre_init)();
    int (*post_init)();
    int (*pre_shutdown)();
    int (*post_shutdown)();
}
```

Диаграмма последовательности инициализации



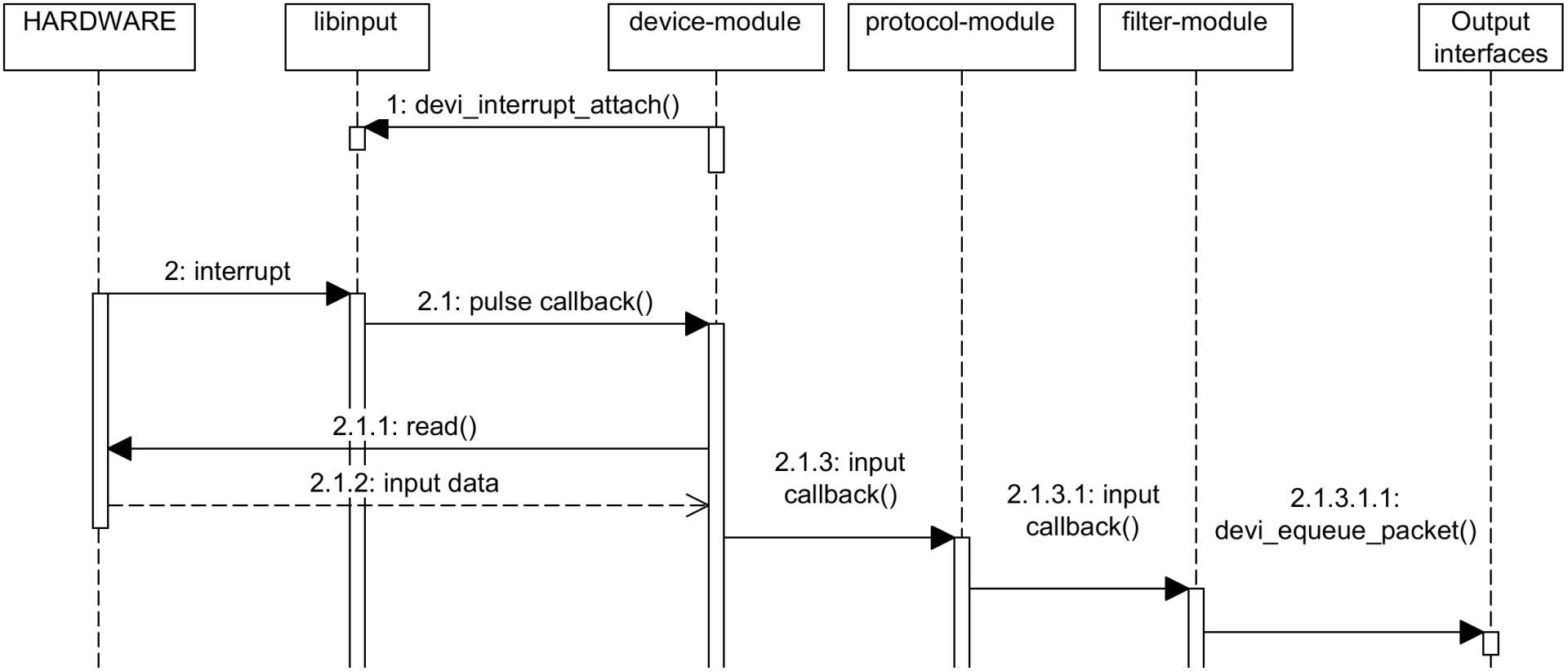
Обработка событий (pulse)

- Вызывается в ответ на возникновение события;
- Обычно реализуется в модулях устройств;
- Выполняет сбор данных из устройства;
- Возможные способы регистрации обработчика:
  - devi_register_interrupt();
  - devi_register_pulse();
  - devi_register_timer();
- Для передачи данных далее по Busline используется функция `module->up->input()`.

Ввод данных (input)

- Вызывается в моменты, когда модуль более низкого уровня готов к передаче данных;
- Модуль устройства:
  - Выполнить сбор данных;
  - Упаковать их в согласованный с модулем протокола формат;
  - Передать их на уровень модуля-протокола, используя функцию `module->up->input()`.
- Модуль протокола:
  - Преобразовать данные в стандартную структуру пакета, соответствующего обслуживаемому устройству (packet_kbd, packet_abs, packet_rel);
  - Передать их на уровень модуля-протокола, используя функцию `module->up->input()`.
- Модуль фильтра:
  - Выполнить необходимую коррекцию принятых данных;
  - Передать данные на реализуемые интерфейсы, используя функцию `devi_enqueue_packet()`.

Диаграмма ввода данных



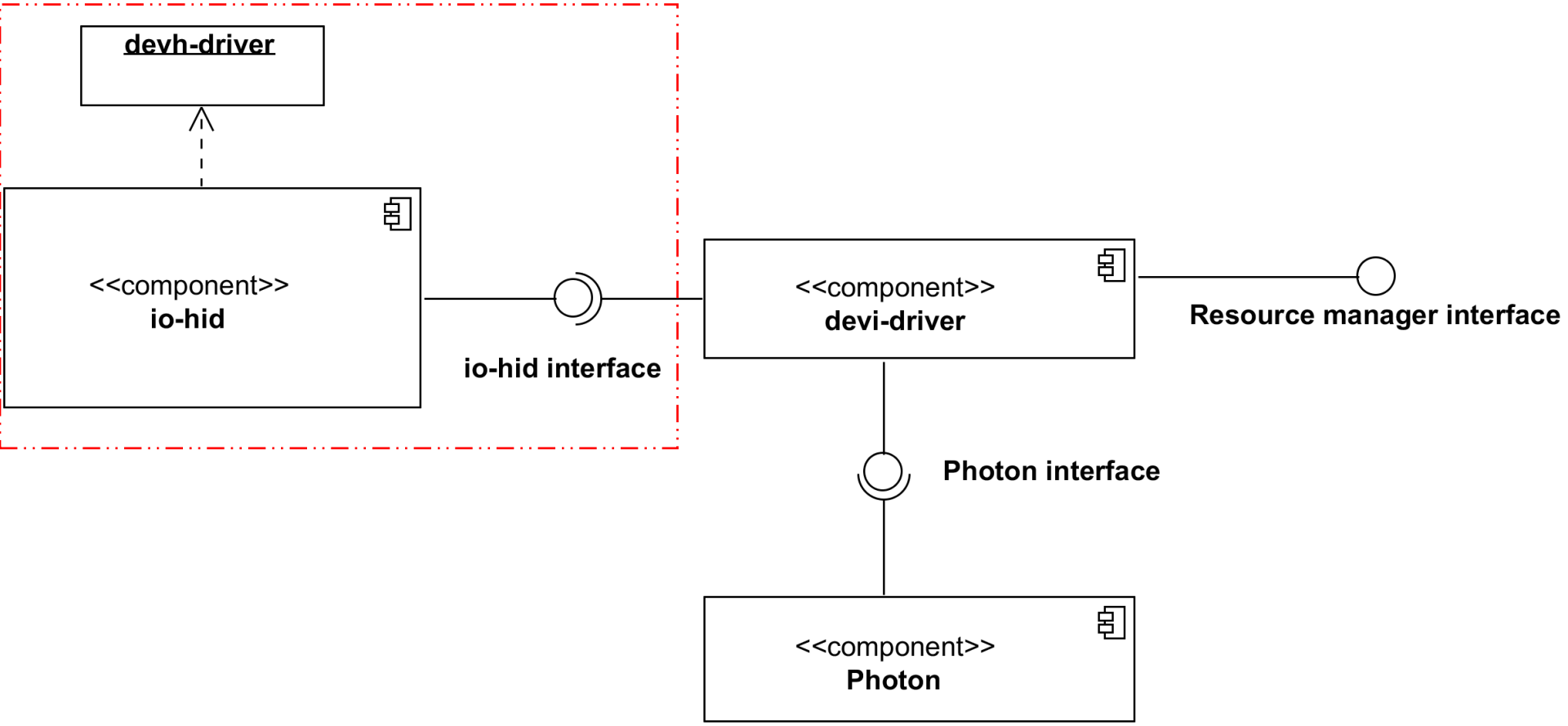
Управление устройством (devctl)

- Реализует управление модулем и реакцию на внешние devctl-команды;
- Собственные devctl-команды можно зарегистрировать, используя константы из заголовочного файла **include/const.h**;
- Модуль устройства:
  - Если тип события известен – выполнить обработку;
  - Если тип события не известен – вернуть значение -1 и установить переменную `errno` в значение `EINVAL`;
- Модуль протокола:
  - Если тип события известен – выполнить обработку;
  - Если тип события не известен и имеется связанный модуль нижнего уровня – передать событие далее по Busline;

Доступный API библиотеки libinput

- Работа с циклическими буферами (функции семейства buff_*);
- Системные функции:
  - clk_get();
  - devi_request_iorange();
- Передача данных на/от реализуемых интерфейсов:
  - devi_enqueue_packet();
- Регистрация обработчиков событий (devi_register_*);
- Доступ к данным менеджера ресурсов:
  - devi_get_coid();
  - devi_get_dispatch_handle()/devi_set_dispatch_handle();
- Работа с менеджером io-hid/usb:
  - devi_hid_init();
  - devi_hid_register_client()/devi_hid_unregister_client();
  - devi_hid_server_connect()/devi_hid_server_disconnect();
  - devi_scan_to_usb()/devi_usb_to_scan().

Общая структура подсистемы ввода



Запуск драйвера

Вместе с менеджером io-hid:

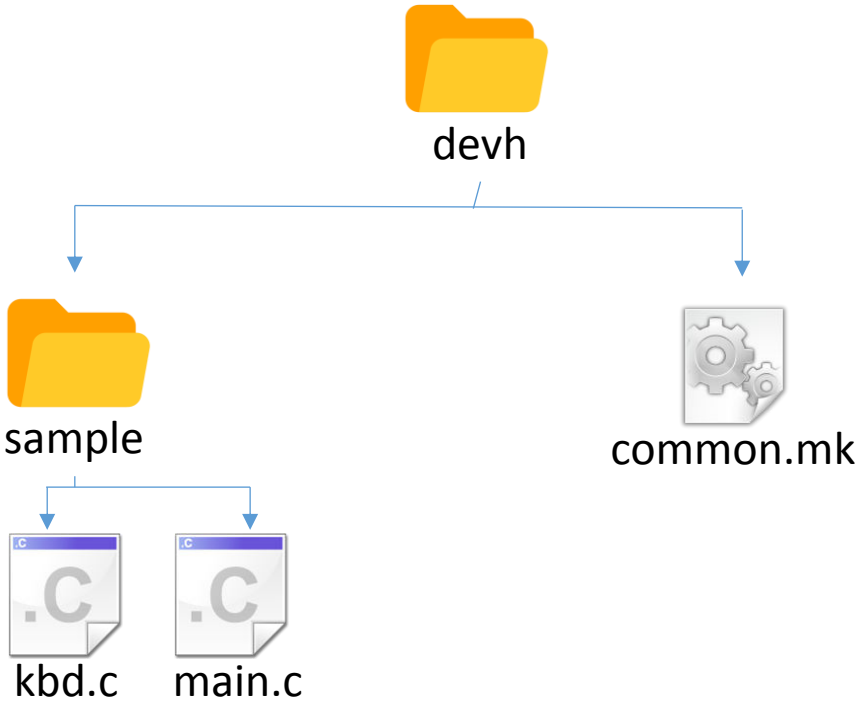
```
io-hid –d <имя_драйвера>|<путь к библиотеке>
[параметры]
```

С помощью утилиты mount:

```
mount –T io-hid <путь к библиотеке>
[параметры]
```

```
umount /dev/io-hid/<имя библиотеки>
```

Дерево сборки



Инициализация

Дескриптор драйвера:

```
struct io_hid_dll_entry {
    char* name;
    int nfuncs;
    int (*init) (void* dll_hdl, dispatch_t *dpp, io_hid_self_t *ioh,
char* options);
    int (*shutdown)(void* dll_hdl)
}
```

Дескриптор io-hid:

```
struct io_hid_self_t {
    _Uint32t nfuncs;
    int (*reg) (void* dll_hdl, io_hid_registrant_t *registrant,
                int *reg_hdlp);
    int (*dereg)(void* dll_hdl);
    int (*get_buffer)(int reg_hdlp, void** buffer);
    int (*send_report)(int registrant_hdl, void* data, _Uint32t dlen);
}
```

Алгоритм работы

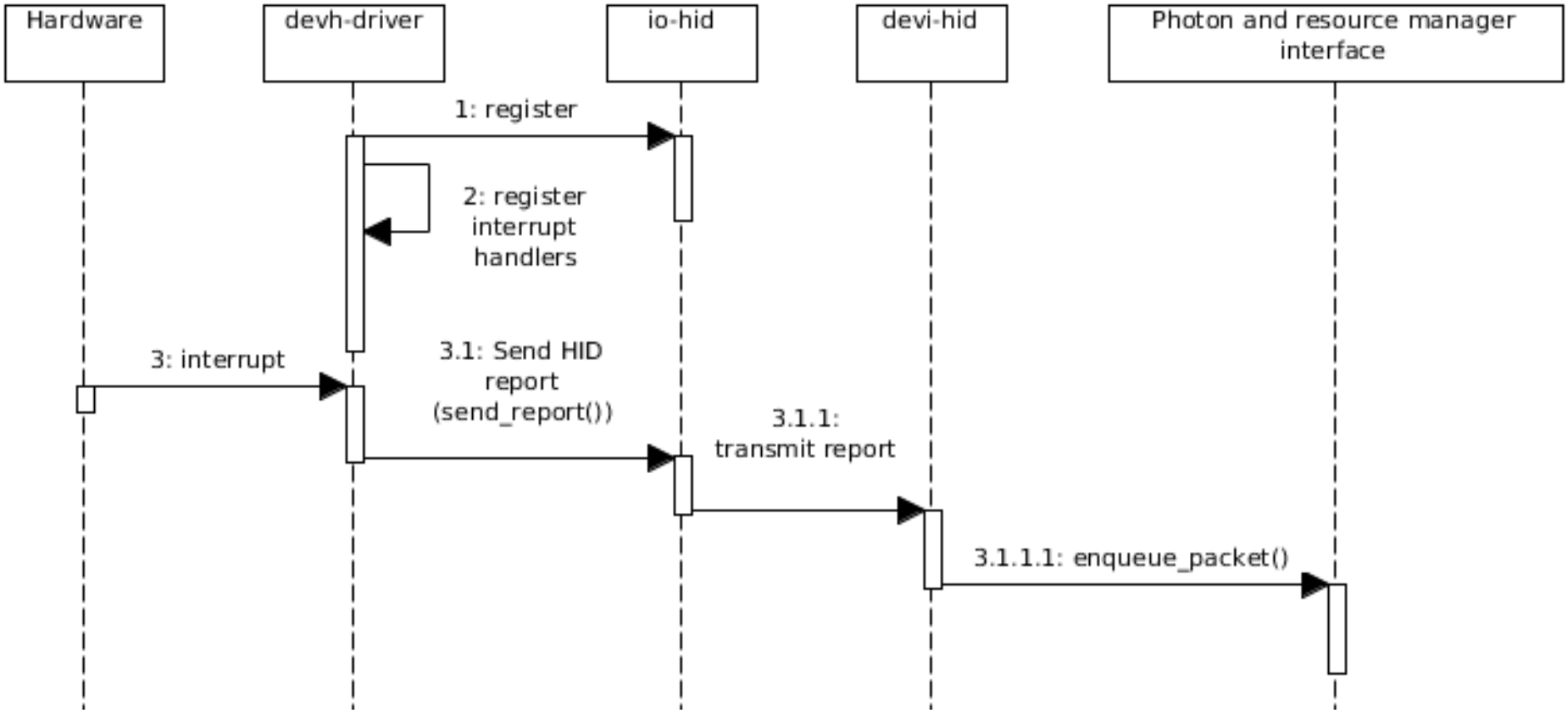
При инициализации:

1. Заполнить дескриптор устройства (`io_hid_registrant`);
2. Установить обработчики прерываний и инициализировать ресурсы драйвера;
3. Зарегистрировать драйвер, вызвав функцию `reg()` менеджера io-hid (содержится в структуре `io_hid_self_t`).

При получении данных от устройства:

1. Заполнить HID-report;
2. Передать HID-report менеджеру io-hid, вызвав функцию `send_report()` менеджера io-hid.

Временная диаграмма работы драйвера



Дескриптор HID устройства

```
struct io_hid_registrant_t {
    _Uint32t flags;
    hidd_device_ident_t *device_ident;      /* vid,pid */
    void *desc;                            /* binary descriptor */
    _Uint16t dlen;                /* binary descriptor length */
    _Uint8t reserved[2];
    void *user_hdl;
    io_hid_registrant_funcs_t *funcs;     /* callbacks */
    _Uint8t reserved2[4];
}
```

Дескриптор функций драйвера

Дескриптор функций драйвера (структура `io_hid_registrant_funcs`):

- Подключение/отключение клиента (`client_attach/client_detach`);
- Управление памятью (`rbuffer_alloc/rbuffer_free`);
- Чтение/Запись HID пакетов (`report_read/report_write`);
- Управление частотой обмена (`get_idle/set_idle`);
- Управление режимом работы (`get_protocol/set_protocol`);
- Получение дополнительных данных от устройства (`string`, `indexed_string`);
- Перезагрузка устройства (`reset`).

Спасибо за внимание

СВД Встраиваемые Системы

Санкт-Петербург, ул. Варшавская, д. 11
e-mail: training@kpda.ru | http://kpda.ru/

SWD Software