Krzysztof Opasiak
k.opasiak@samsung.com
Samsung R&D Institute Poland

Make your own USB device and driver with ease!

Workshop 1 – Find a suitable device

Login: lab_host01

Password: lab_host01

Task:

Modify **find_device.c** to iterate over all available USB devices, find suitable one, open it and call ***print_configuration_string()*** on given device to show string associated with first configuration.

Definition of suitable device:

idVendor == DESIRED_VID

idProduct == DESIRED_PID

manufacturer string == DESIRED_MANUFACTURER

Useful functions and data structures:

```
/* libusb error signalization convention */
ret = libusb_function()
if (ret < 0)
        printf("libusb_function() failed");
```

```
struct libusb_device_descriptor {
        /* (…) */
        uint16_t idVendor;
        uint16_t idProduct;
        /* (…) */
        uint8_t  iManufacturer;
        /* (…) */
};
int libusb_get_device_descriptor(libusb_device *dev,
                                 struct libusb_device_descriptor *desc)

int libusb_open(libusb_device *dev, libusb_device_handle **dev_handle)

int libusb_get_string_descriptor_ascii(libusb_device_handle *dev_handle,
                                       uint8_t desc_index, unsigned char *data, int length)

int strcmp(const char *s1, const char *s2)

void libusb_close(libusb_device_handle *dev_handle)

ssize_t libusb_get_device_list(libusb_context *ctx, libusb_device ***list)
```

Workshop 2 – Synchronous libusb API

Login: lab_host02

Password: lab_host02

Task:

Implement missing parts of **host_schat.c** according to simple chat protocol which goes as follow:

Single chat message transfer consist of two USB Bulk transfers:

1) length – two bytes which contains unsigned integer in little endian byte order. This value is the length of whole chat message (strlen(content) + 1 + 2 /* '\0' + sizeof(length) */)
2) content – null terminated string with single line of input

Implementation should be done using synchronous libusb API.

Useful functions and data structures:

/* libusb error signalization convention */

ret = libusb_function()

if (ret < 0)

printf("libusb_function() ***failed****");*

/

** Structure for chat message.*

** 1st USB transfer –* ***data*** *=&****m.length****,* ***length****=2*

** 2nd USB transfer –* ***data*** *=* ***m.line_buf****,* ***length****=(libusb_le16_to_cpu(****m.length****) – 2)*

** For simplicity let's assume that MAX_LINE_LENGTH is always enough*

**/*

struct message *{*

uint16_t ***length****;*

char ***line_buf****[MAX_LINE_LENGTH];*

} __attribute((packed)) ***m****;*

int ***libusb_claim_interface(****libusb_device_handle* ****dev****, int* ***interface_number)***

int ***libusb_bulk_transfer(****struct libusb_device_handle* ****dev_handle****,*
unsigned char ***endpoint****,*
unsigned char ****data****,*
int ***length****,*
int ****transferred****,*
unsigned int ***timeout)***

/ Always success */*
uint16_t ***libusb_le16_to_cpu****(uint16_t* ***x)***

Workshop 3 – FunctionFS + libaio

Login: lab_device03

Password: lab_device03

Task:

Implement missing parts of **device_achat.c** according to simple chat protocol which goes as follow:

Single chat message transfer consist of two USB Bulk transfers:

3) length – two bytes which contains unsigned integer in little endian byte order. This value is the length of whole chat message (strlen(content) + 1 + 2 /* '\0' + sizeof(length) */)
4) content – null terminated string with single line of input

Implementation should be done using FunctionFS and libaio.

Then switch to another terminal, log in using **lab_host03** pass: **lab_host03** and use **find_device, host_schat** to check your implementation.

Do you see some issues of our host side"driver"?.

Useful functions and data structures:

```
/*
* Structure for chat message.
* 1st USB request – buf =&m.length, length=2
* 2nd USB request – buf = m.line_buf, length=(libusb_le16_to_cpu(m.length) – 2)
* For simplicity let's assume that MAX_LINE_LENGTH is always enough
*/
struct message {
        uint16_t length;
        char line_buf[MAX_LINE_LENGTH];
} __attribute((packed));

typedef void (*ffs_complete_t)(struct ffs_request *)
```

/* Our simple wrapper for **struct iocb** */

struct ffs_request *{*

struct iocb ***iocb****;*

unsigned char ****buf****;*

ssize_t ***length****;*

void ****context****;*

int ***status****;*

int ***actual****;*

ffs_complete_t ***complete****;*

};

/* Our simple structure for single chat transfer */

struct transfer *{*

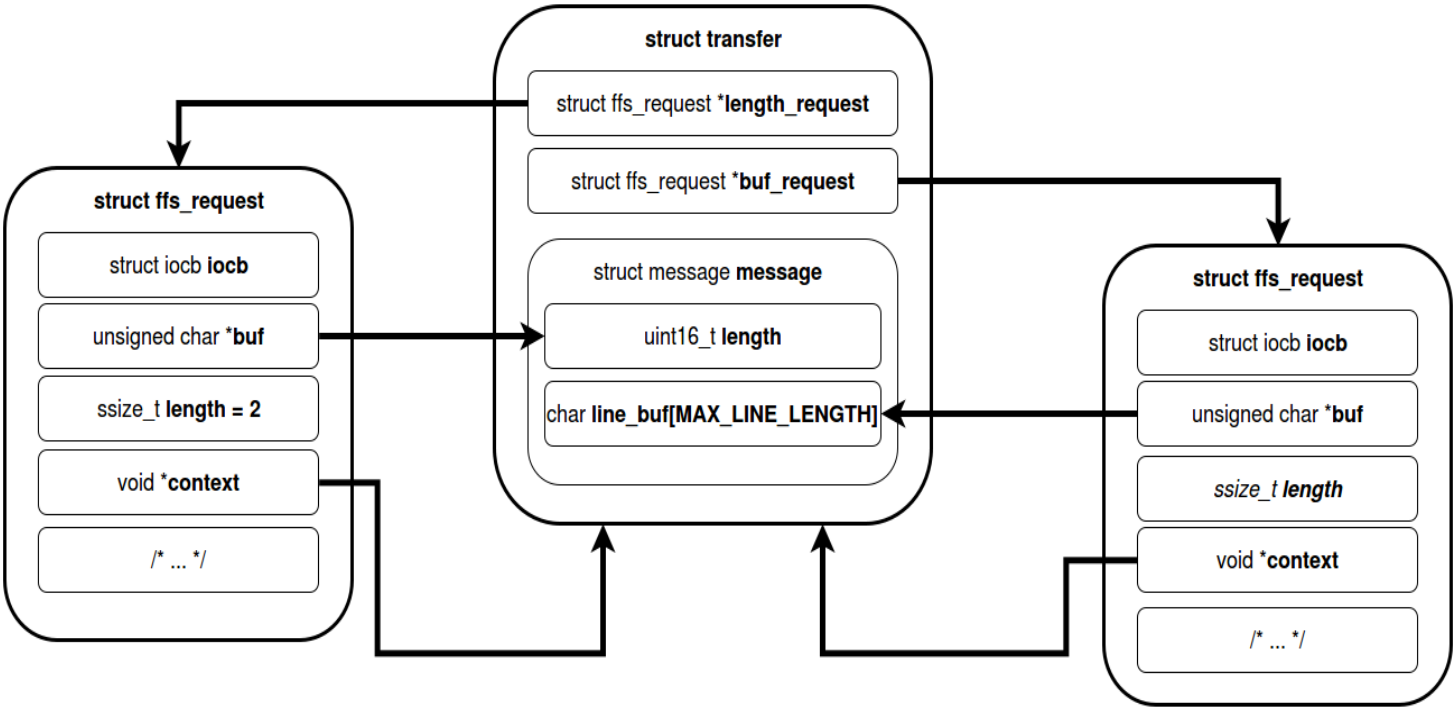
struct ffs_request ****length_request****;*

struct ffs_request ****buf_request****;*

struct message ***message****;*

/* (…) */

};



USB_DIR_IN

USB_DIR_OUT

void ***io_prep_pwrite****(struct iocb* ****iocb****, int* ***fd****, void* ****buf****, size_t* ***count****, long long* ***offset)***

void ***io_prep_pread****(struct iocb* ****iocb****, int* ***fd****, void* ****buf****, size_t* ***count****, long long* ***offset)***

ssize_t **read**(int fd, void *buf, size_t count)

ssize_t **write**(int fd, const void *buf, size_t count)

int **submit_ffs_request**(io_context_t ***ctx**, struct ffs_request ***req**)

void **fill_ffs_request**(struct ffs_request ***req**, int **dir**, int **ep_fd**, int **event_fd**,
unsigned char ***buf**,
int **length**,
ffs_complete_t **complete**,
void ***context**)

int **recv_message**(struct transfer ***in_transfer**)

int **FD_ISSET**(int **fd**, fd_set ***set**)

int **handle_events**(io_context_t ***ctx**, int **event_fd**)

BONUS WORKSHOP

Workshop 4 – Asynchronous libusb API

Login: lab_host04

Password: lab_host04

Task:

Implement missing parts of **host_achat.c** according to simple chat protocol which goes as follow:

Single message transfer consist of two USB transfers:

5) length – two bytes which contains unsigned, 2 bytes integer in little endian byte order. This byte indicates the length of whole message (strlen(content) + 1 /* '\0' / + 2 /* sizeof(length) */)
6) content – null terminated string with single line of input

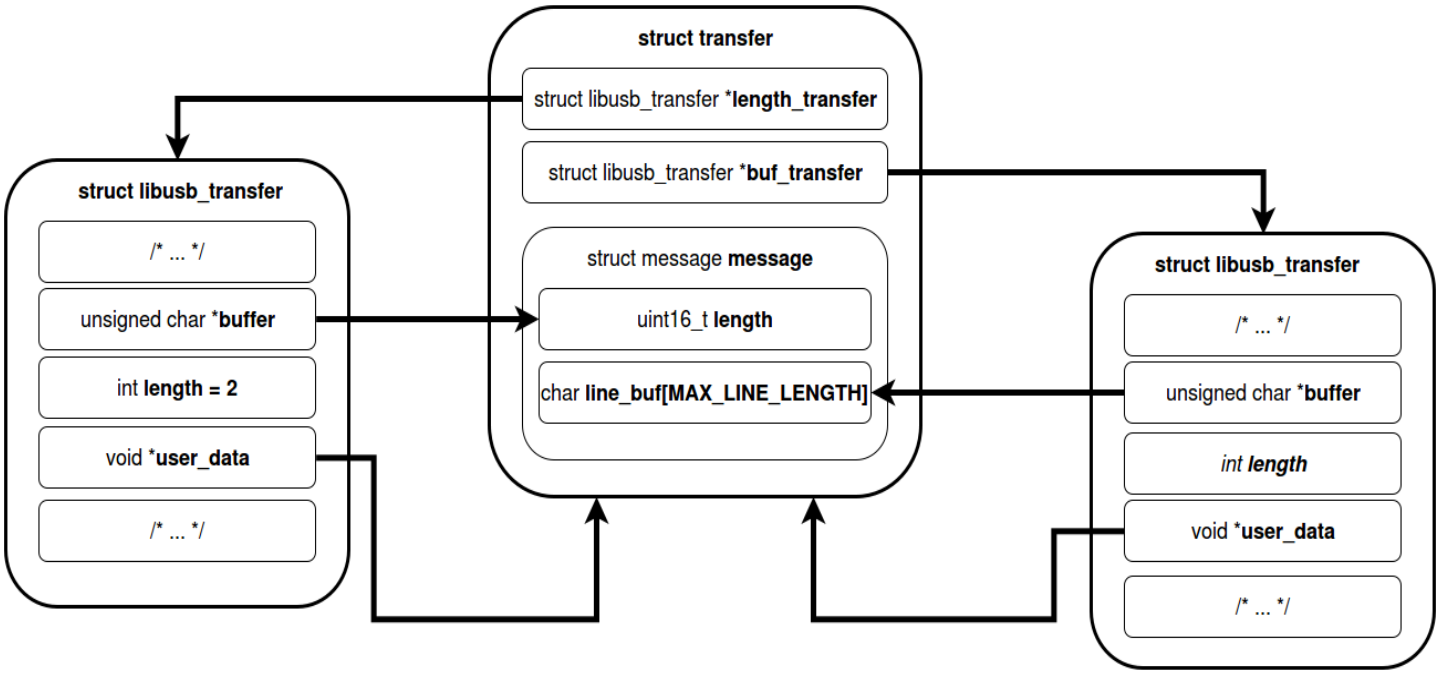
Implementation should be done using asynchronous libusb API.

Useful functions and data structures:

```
/*
* Structure for chat message.
* 1st USB request – buf =&m.length, length=2
* 2nd USB request – buf = m.line_buf, length=(libusb_le16_to_cpu(m.length) – 2)
* For simplicity let's assume that MAX_LINE_LENGTH is always enough
*/
struct message {
        uint16_t length;
        char line_buf[MAX_LINE_LENGTH];
} __attribute((packed));
```

```
struct libusb_transfer {
        libusb_device_handle *dev_handle;
        unsigned char endpoint;
        unsigned int timeout;
        enum libusb_transfer_status status;
        int length;
        int actual_length;
        libusb_transfer_cb_fn callback;
        void *user_data;
        unsigned char *buffer;
       /* (…) */
};
```

```
/* Our simple structure for single chat transfer */
struct transfer {
         struct libusb_transfer *length_transfer;
         struct libusb_transfer *buf_transfer;
         struct message message;
          /* (…) */
};
```



libusb_submit_transfer*(struct libusb_transfer* ****transfer****)*

libusb_fill_bulk_transfer*(struct libusb_transfer* ****transfer****,*
libusb_device_handle ****dev_handle****,*
unsigned char ***endpoint****,*
unsigned char ****buffer****,*
int ***length****,*
libusb_transfer_cb_fn ***callback****,*
void ****user_data****,*
unsigned int ***timeout****)*

libusb_handle_events_timeout_completed(libusb_context ***ctx**, struct timeval ***tv**,
int ***completed**)