

libcxlmi a CXL Management Interace library

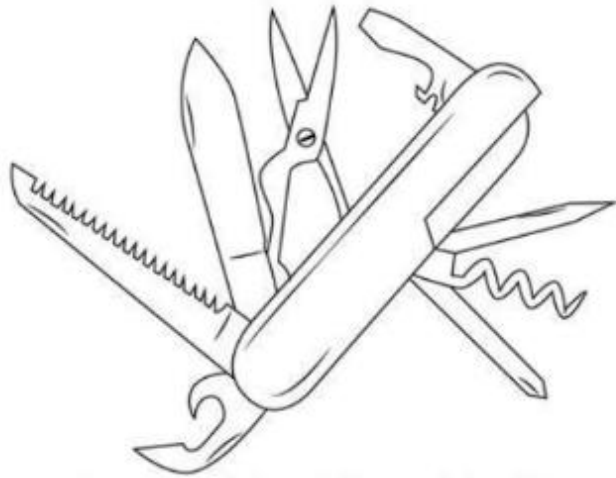
Linux Plumbers Conference

Davidlohr Bueso. Vienna, Austria. September 2024.

<https://github.com/computexpresslink/libcxlmi>

Influenced by fmaps-tests + libnvme

CXL 3.1 Sections 9.19 (*Manageability Model for CXL Devices*) &
9.20 (*Component Command Interface*)



- Is **not** intended to compete with libcxl from ndctl.
 - Supports OoB (MCTP) and Linux ioctl (mbox) based CCIs.
- Users: BMC, Fabric managers, Firmware, etc.
 - Type3 SLD, Type3 MLD (FM owned) or a CXL Switch.
 - Does not support G-FAM devices.
- You get what you ask for. Users must provide the correct command to the correct CXL component.
- Concurrency is left to users. Libraries should not hold locks.
- Endianness-aware.
- Ideal scenario: provide a **standard and open** building block for FMs, openBMC, etc. State of the art is very fragmented and obscure.

Example: Simple MCTP component discovery/DBUS

```
// link with -lcxlmI
#include <libcxlmi.h>

int main(int argc, char **argv)
{
    int rc = EXIT_FAILURE, num_ep;
    struct cxlmi_ctx *ctx;
    struct cxlmi_endpoint *ep, *tmp;

    ctx = cxlmi_new_ctx(stdout, DEFAULT_LOGLEVEL);
    if (!ctx) {
        fprintf(stderr, "cannot create new context object\n");
        return rc;
    }

    num_ep = cxlmi_scan_mctp(ctx);
    if (num_ep < 0) {
        fprintf(stderr, "dbus scan error\n");
        goto free_ctx;
    } else if (num_ep == 0)
        printf("no endpoints found\n");
    else
        printf("found %d endpoint(s)\n", num_ep);

    cxlmi_for_each_endpoint_safe(ctx, ep, tmp) {
        rc = show_device_info(ep);
        cxlmi_close(ep);
    }

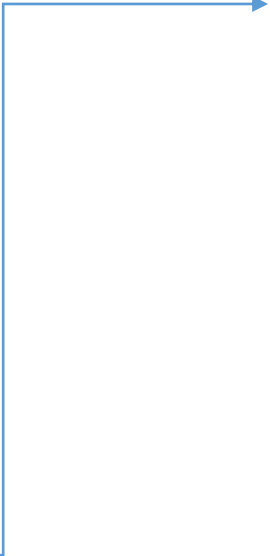
free_ctx:
    cxlmi_free_ctx(ctx);
    return rc;
}

int show_device_info(struct cxlmi_endpoint *ep)
{
    int rc;
    struct cxlmi_cmd_identify id;

    rc = cxlmi_cmd_identify(ep, NULL, &id);
    if (rc)
        return rc;

    printf("serial number: 0x%lx\n", (uint64_t)id.serial_num);
    printf("Vendor ID:%04x Device ID:%04x\n", id.vendor_id, id.device_id);

    return 0;
}
```



- Setup the path for CCI commands to be sent.
 - Individual MCTP nid:eid endpoint. (*cxlmi_open_mctp()*)
 - Enumerate and open all MCTP endpoints (DBus). (*cxlmi_scan_mctp()*)
 - Individual, Linux-specific sysfs device endpoint. (*cxlmi_open()*)
- By default, it will also probe the endpoint to get the CXL component this belongs to: either a CXL Switch or a Type3 device.
- While the library context can track different representations of CCIs for the same underlying CXL component, duplicates of each type is forbidden.
 - This matches the component requirement of 1:1 MCTP and a primary Mailbox.

API: Sending CCI Commands (1/5)

- Best express: what command to send to what *endpoint*, and how (direct or tunneled)
- Same name for both the functions to send CXL commands and the respective payload data structure(s):

`cxlmi_cmd_[memdev|fmapi_]<cmd_name>`

```
int cxlmi_cmd_memdev_sanitize(struct cxlmi_endpoint *ep, struct cxlmi_tunnel_info *ti);
```

```
int cxlmi_cmd_fmapi_get_qos_status(struct cxlmi_endpoint *ep,  
    struct cxlmi_tunnel_info *ti,  
    struct cxlmi_cmd_fmapi_get_qos_status *ret);
```

```
struct cxlmi_cmd_fmapi_get_qos_status {  
    uint8_t backpressure_avg_percentage;  
};
```

```
int cxlmi_cmd_transfer_fw(struct cxlmi_endpoint *ep,  
    struct cxlmi_tunnel_info *ti,  
    struct cxlmi_cmd_transfer_fw *in);
```

```
struct cxlmi_cmd_transfer_fw {  
    uint8_t action;  
    uint8_t slot;  
    ...  
    uint8_t data[];  
};
```

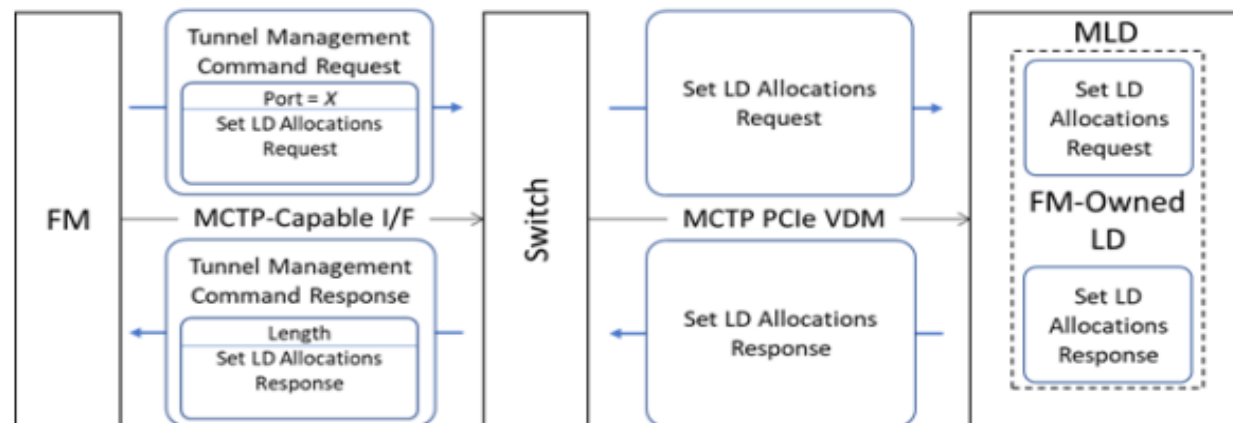
```
int cxlmi_cmd_get_supported_logs_sublist(struct cxlmi_endpoint *ep,  
    struct cxlmi_tunnel_info *ti,  
    struct cxlmi_cmd_get_supported_logs_sublist_req *in  
    struct cxlmi_cmd_get_supported_logs_sublist_rsp *ret);
```

API: Sending CCI Commands (2/5)

- Tunneling Commands to an MLD through a CXL Switch

```
struct cxlmi_cmd_fmapi_set_ld_allocations_req *alloc_req;
struct cxlmi_cmd_fmapi_set_ld_allocations_rsp *alloc_rsp;
struct cxlmi_tunnel_info ti = {
    .level = 1,
    .port = 3, /* cmd sent to a Switch */
};

/* ... prepare payload buffers */
rc = cxlmi_cmd_fmapi_set_ld_allocations(ep, &ti, alloc_req, alloc_rsp);
if (rc) {
    /* handle error */
}
```

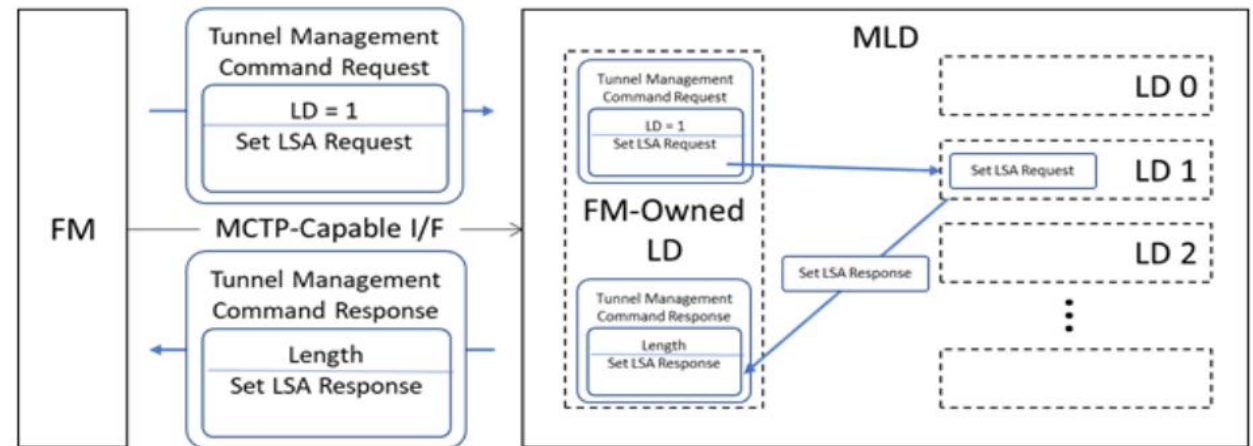


API: Sending CCI Commands (3/5)

- Tunneling Commands to an LD in an MLD

```
struct cxlmi_cmd_memdev_set_lsa *lsa = arm_lsa(offset, data);  
struct cxlmi_tunnel_info ti = {  
    .level = 1,  
    .ld = 1, /* cmd sent to an MLD */  
};
```

```
rc = cxlmi_cmd_memdev_set_lsa(ep, &ti, lsa);  
if (rc) {  
    /* handle error */  
}
```

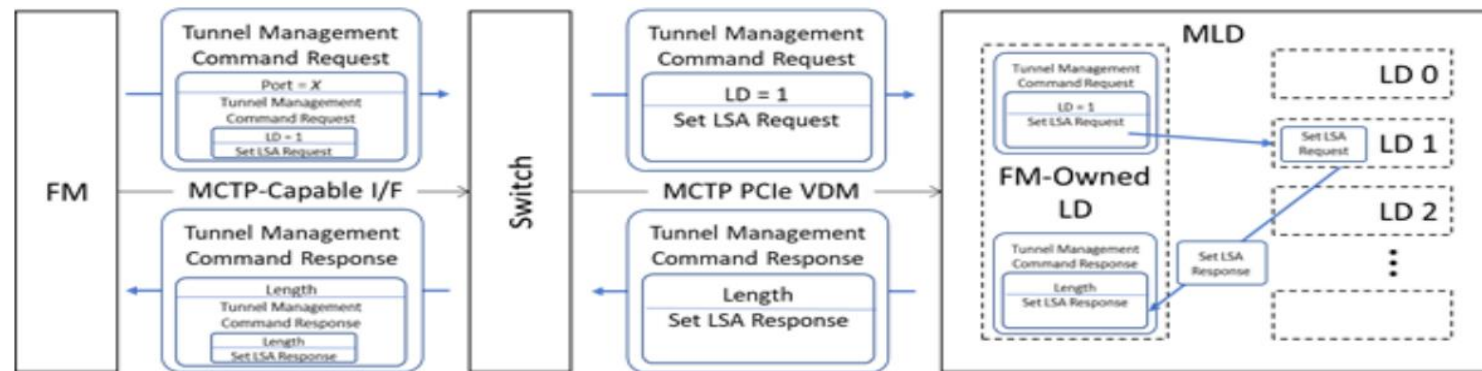


API: Sending CCI Commands (4/5)

- Tunneling Commands to an LD in an MLD through a CXL Switch

```
struct cxlmi_cmd_memdev_set_lsa *lsa = arm_lsa(offset, data);  
struct cxlmi_tunnel_info ti = {  
    .level = 2,  
    .port = 3, /* outer tunnel */  
    .ld = 1, /* inner tunnel */  
};
```

```
rc = cxlmi_cmd_memdev_set_lsa(ep, &ti, lsa);  
if (rc) {  
    /* handle error */  
}
```



- Return values are either:

-1: internal error (library level), with `errno` set,

or,

the respective CXL-specific return code (`cxlmi_cmd_retcode` enum).

```
err = cxlmi_cmd_activate_fw(ep, NULL, &fw); // cmd is known to be bg-capable
if (err && err != CXLMI_RET_BACKGROUND) {
    if (err > CXLMI_RET_SUCCESS) // > 0
        fprintf(stderr, "could not activate fw: %s\n", cxlmi_cmd_retcode_tostr(err));
    return err;
}
```

- Is this of value to the general community?
 - There has been no release, hence nothing is set in stone.
- Opens, TODO:
 - Testing, testing, testing.
 - qemu has been very valuable.
 - MCTP-capable i2c controller (*aspeed-i2c*).
 - Kernel i2c transport driver in v5.18.
 - Tunneling commands to the LD-Pool CCI in an MHD.
 - cxl.io (mbox CCI)
 - Finish adding the full CXL 3.1 command set.
 - Many commands are there, but still many missing (fairly trivial to add).

Thank you.

THE NEXT CREATION STARTS HERE

Placing **memory** at the forefront of future innovation and creative IT life



- Management Component Transport Protocol (MCTP). Server-oriented components.
 - Introduced in Linux v5.16 (Jeremy Kerr).
 - Common hardware transports:
 - i2c/SMBus (kernel device-tree), PCIe Vendor Defined Messages (VDM) and serial (ttySn).
 - Standards are produced by DMTF, ie:
 - [MCTP over i2c/SMBus \(DSP0237\)](#).
 - [CXL Fabric Manager API over MCTP Binding Specification \(DSP0324\)](#).
- MCTP stack management software:
 - [CodeConstruct/mctp: MCTP userspace tools \(github.com\)](#)
- Standard socket API.
 - *struct sockaddr_mctp*
 - Extend 255 endpoint support by adding the network-id for address space.
 - socket will only receive messages sent on that network (unless MCTP_NET_ANY)
 - Networks need to be on physically separate busses.
 - Kernel handles tags automatically (requests using MCTP_TAG_OWNER).