# DPDK IoKit SCSI Target Brief

*By March Luo <unix.march@gmail.com, march@huawei.com>*

Version 1.2

**Keywords**: dpdk, user space, real time, SCSI target, storage

## Motivation & Background

SCSI target stack is crucial to develop SCSI interface based hardware, however established SCSI target stack is mostly under kernel-space or tightly couples with iSCSI under user-space.

In addition, in order to meet  with enterprise IO scenario, herein the trait is low latency and will meet with heavy IO pressure, thus, regularly these software prefers to be developed under kernel space rather than adopting POSIX interface in user-space.

Nevertheless, software coupled with kernel-space is nightmare in terms of either development progress or stability. Therefore, developers are already considering the feasibility to move to user-space, in which through OS-bypass technology to overcome and eliminate the leverage of slowness of operation system. instead of regular POSIX based programming model.

Fortunately, there are established facilities ready for us to help for building them, the open-source DPDK by Intel, hugepages mechanism by Linux since 2.6.xx and possibility of user mode hardware driver on Linux.

Our project tends to use Intel's DPDK as the fundamental low level component, avoiding reinvention of chart wheels.

Low level drivers based on DPDK are originally developed for network packet forwarding purpose, thus the APIs and model are less considered for IO purpose, the dpdk-iokit will design a suit of IO-purpose APIs based on DPDK.

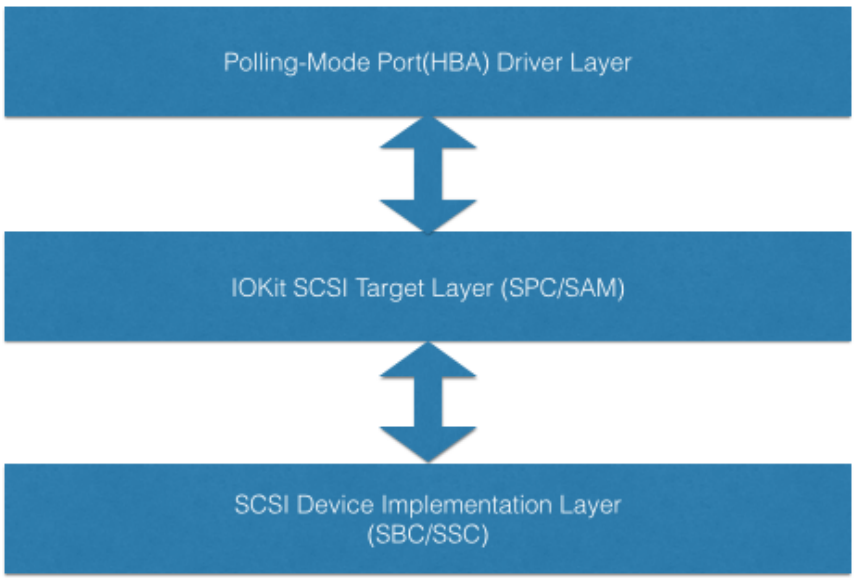
Furthermore, dpdk-iokit will extend the original interrupt mechanism, the original mechanism is through POSIX ***select()*** function to transfer interrupt to user space, because the mechanism is merely designed for management purpose. By patching the Linux kernel, the interrupt address in user space which is registered into kernel is able to be invoked through setting EIP register aka. preemption of original execution address of user space process. In the laboratory the preemption based interrupt pass costs 200-300 cycles.

The project dpdk-iokit is designed for delivering a complete IO stack based on Intel open source DPDK (dpdk.org), relating complete SCSI stack, work queue implementation which is used for IO schedule, and an extension of interrupt mechanism.

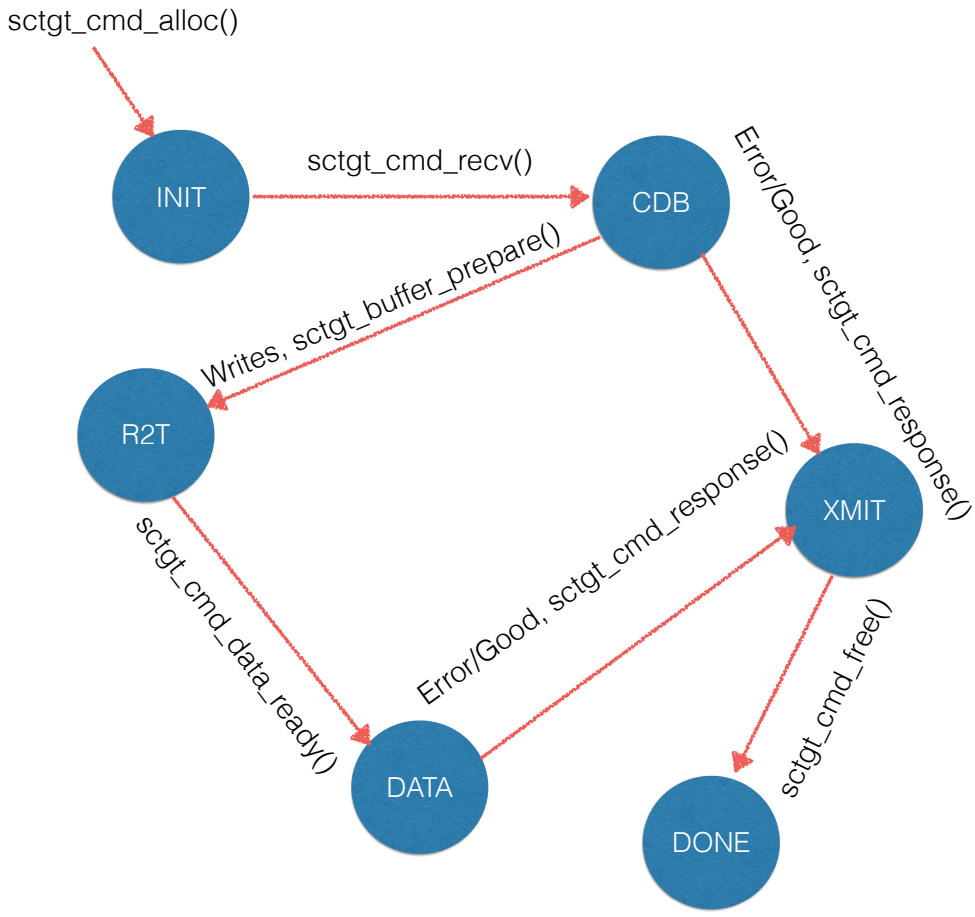
# Architecture of SCSI Stack

As the standard papers described in the SCSI architecture model there are three layers: the physical link layer which conveys packed packet on wires; the logical architecture layer, which is responsible for management of topology, name space and task; and the specified device protocol implementation.

Thus give the consideration above the DPDK SCSI stack consists of three layers: the HBA (host bus adapter) port PMD (polling mode driver) layer, which is responsible for packing and unpacking SCSI packets on wires; the middle logical layer, which implements SPC and SAM, for instance responsible for name space management, aka the LUN space management; and device protocol implementation layer which implements corresponding device protocols, such as SCSI disk and SCSI tape.



# SCSI Command State Changes



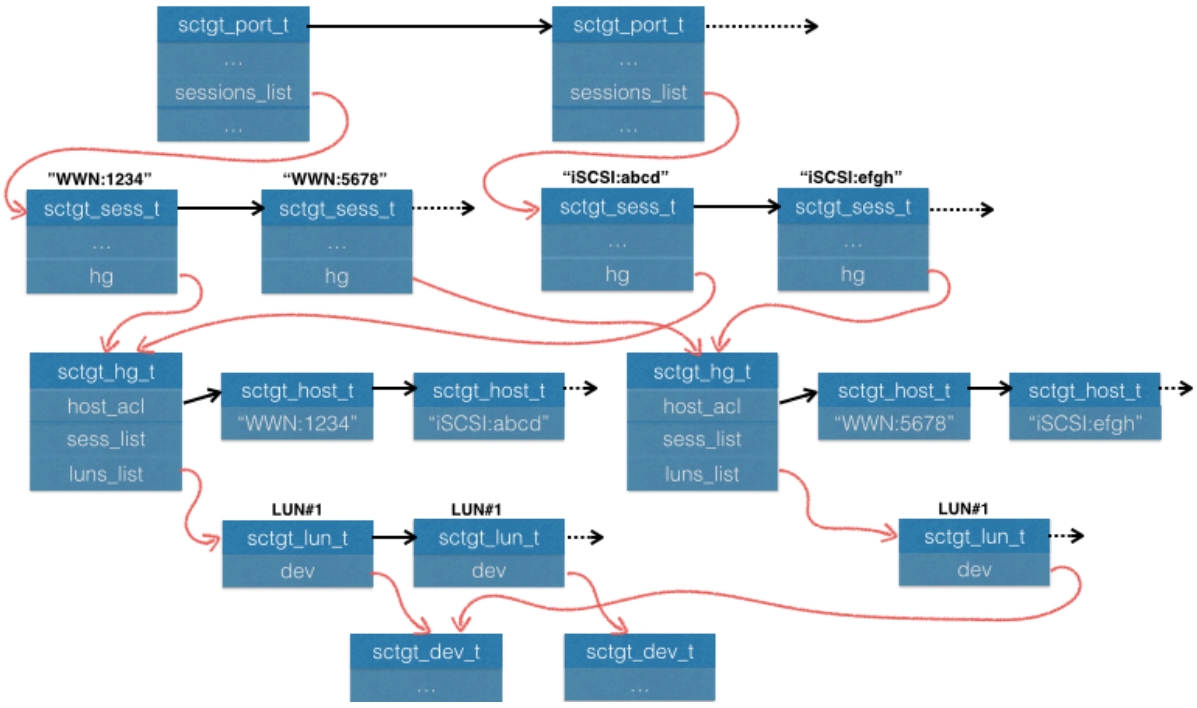

| | | |
|---|---|---|
| SCTGT_SCPH_INIT | - | HBA driver calls ***sctgt_cmd_alloc()*** to allocate a new command data structure and the initial phase is set to SCTGT_SCPH_INIT |
| SCTGT_SCPH_CDB | - | HBA driver calls ***sctgt_cmd_recv()*** to receive SCSI header information, once such information is verified the phase is set to SCTGT_SCPH_CDB |
| SCTGT_SCPH_R2T | - | If the received SCSI command is write command, the SCSI stack will call ***sctgt_buffer_prepare()*** for preparing buffer, and then set the phase to SCTGT_SCPH_R2T, marking the SCSI target is ready for receiving data from initiator |
| SCTGT_SCPH_DATA | - | Once initiator transferred data to target, the HBA driver will call ***sctgt_cmd_data_ready()*** for marking data ready for backing device layer. |
| SCTGT_SCPH_XMIT | - | The ***sctgt_cmd_response()*** will be called if the target will response command to initiator, and the command will be marked as SCTGT_SCPH_XMIT |
| SCTGT_SCPH_DONE | - | When ***sctgt_cmd_free()*** is called, the command will be marked as SCTGT_SCPH_DONE |

# Core Data Structures

DPDK SCSI stack is designed for supporting dynamic front-end port registering and back-end protocol endpoint, besides these, as well as the stack will support multiple LUN name space rather than single name space.

The following picture revealed out the relationship among data structures. Inside the picture: the ***sctgt_port_t*** is designed for representing the HBA port entity, such HBA port is dynamically registered into the SCSI stack through ***sctgt_port_register(*)***; ***sctgt_sess_t*** is used for recording the SAM I_T nexus (the connection context between SCSI initiator and the SCSI target), the SCSI cmds occurred on the connection will be associated with such session; ***sctgt_hg_t*** is used for maintaining the LUN namespace inside a host group, difference initiator (SCSI host) can only see the LUN namespace which such initiator is added through ACL control command by users; ***sctgt_host_t*** is only used for ACL purpose to record an ACL entry, in which the credence in ACL entry is in the form of string, vary from the physical protocol, for example the FCP card will use peer WWN, the iSCSI protocol will use InitiatorName; ***sctgt_lun_t*** is the LUN intermediate data structure inside the host group for representing the SCSI LUN, and each LUN will eventually attach to a specific backing device; ***sctgt_dev_t*** is the entity which is registered by backing device layer as the SCSI protocol endpoint.

DPDK SCSI stack won't supply the specific SCSI disk SBC protocol implementation, but DPDK SCSI stack has a simple virtual disk implementation for evaluation purpose.



**HBA Driver Data Structure *sctgt_port_t***

```
struct sctgt_port {
    sctgt_pt_t          ptype;
    unsigned int        poll;
    char                name[SCTGT_NAME_LEN];
    unsigned int        sname_len;
    unsigned int        max_cmds;
    unsigned int        cmd_ext_sz;
    unsigned long       dma_mask;
    unsigned int        max_sgl;
    unsigned int        cpu_id;
    unsigned int        mcp;

    void          (poll_cmd)(sctgt_port_t *port);
    void          (poll_data_ready)(sctgt_port_t *port);
    void          (poll_status_completion)(sctgt_port_t *port);
    void          (*xmit_response)(sctgt_cmd_t *cmd);
    void          (*xfer_ready)(sctgt_cmd_t *cmd);
    void          (*release_session)(sctgt_session_t *sess);
    void          (*task_mgmt_done)(sctgt_mgmt_t *mgmt, int result);

    int           (*control)(sctgt_port_t *port, sctgt_ptctl_t cmd);
    void          (*release_cmd)(sctgt_cmd_t *cmd);
    void          (*cmd_ext_constructor)(sctgt_cmd_t *cmd);
    void          (*cmd_ext_destructor)(sctgt_cmd_t *cmd);

    list_entry_t            list[SCTGT_MAX_CPUS];
    unsigned int            port_id;
    struct rte_mempool      *cmds_pool;

    unsigned int            next_cmd_seqno;
    list_entry_t            sessions_list[SCTGT_MAX_CPUS];
    unsigned long           sessions_count;
    unsigned long           next_session_id;
    iokit_hash_t            sessions_hash[SCTGT_MAX_CPUS];

    unsigned long           refcnt;
    unsigned long           cmds_outstanding;
    unsigned long           mgmt_outstanding;

    void                    *cookie;

};
```

```
  ptype  -     HBA offers and indicate the physical port type, the
               options are:
               SCTGT_PT_FCP          - Regular fibre channel
               SCTGT_PT_SOFT_ISCSI   - Software iSCSI
               SCTGT_OFE_ISCSI       - Offload Engine iSCSI
               SCTGT_LOOP            - Virtual HBA loops to local OS
```

poll - HBA offers, 0 is to indicate the driver is not on polling mode, otherwise will indicate the driver is on polling mode and the polling function set on port will be invoked.

name[] - HBA offers and should include the hardware identifier information.

sname_len - HBA offers, indicates the string length of session name.

max_cmds - HBA offers, indicates the maximum SCSI command queue length the hardware could support.

cmd_ext_sz - HBA offers, indicates the private storage area size per sctgt_cmd_t provided by SCSI stack.

dma_mask - HBA offers, indicates the DMA address boundary since not all address will work with real hardware

max_sgl - HBA offers, stands for the ability that the hardware support SGL (scatter-gather-list), if the hardware does not support SGL, the value should be 1 to indicate the hardware will use flat memory.

mcp - HBA offers, value 1 will indicate the hardware will support multiple polling queue, otherwise the polling will always happen on specified processor by member ->cpu_id.

cpu_id - HBA offers, combines with the member ->mcp will indicate the polling processor, but if the ->mcp is 1, the ->cpu_id will be ignored.

poll_cmd() - HBA offers, it is the polling function is responsible for collecting incoming SCSI CDB.

poll_data_ready() - HBA offers, it is the polling function is responsible for collecting the write DMA (I->T) completion signals.

poll_status_completion()() - HBA offers, it is the polling function is responsible for collecting the SCSI phase completion signals.

xmit_response() - HBA offers, it is the function for transmitting completed SCSI command back to initiator.

xfer_ready() - HBA offers, is is the function for notifying to the initiator that the target is ready to receive data.

release_session() - HBA offers, when a session is not needed such session could be destroyed asynchronously, such function will be called before the target stack is going to release the allocated memory.

task_mgmt_done() - HBA offers, when SCSI task management command is done the target stack will call such function for acking the result back to initiator.

control() - HBA offers, it is used for control purpose, such enable/disable

release_cmd() - HBA offers, when a SCSI command is for example aborted there will be no normal way to release the SCSI command, instead the

target stack will call this function to actively release the command.

cmd_ext_constructor() - HBA offers, the SCSI command data structure sctgt_cmd_t will provide private storage area for low level hardware, such function will be called to initialize the private area.

cmd_ext_destructor() - HBA offers, before the SCSI stack is going to free the memory for sctgt_cmd_t, the stack will call it for finishing the private area.

list[] - Internally use, the sctgt_port_t will be linked on multiple list per processor.

port_id - Internally use

cmds_pool - Internally use, provide the SCSI command allocator pool.

next_cmd_seqno - Internally use, atomic number, as the hallmark of SCSI command.

sessions_list[] - Internally use, all sessions will be linked on the corresponding list by processor ID.

sessions_count - Internally use, atomic number

next_session_id - Internally use, atomic number

sessions_hash[] - Internally use, per processor hash table used for acceleration of session lookup operation.

refcnt - Internally use, atomic number

cmds_outstanding - Internally use, atomic number

mgmt_outstanding - Internally use, atomic number

cookie - Internally use, for anchoring the private data by HBA driver.

## I_T Connection Data Structure *sctgt_session_t*

```
struct sctgt_session {
    list_entry_t            plist;
    list_entry_t            hlist;
    list_entry_t            rlist;
    unsigned int            cpu_id;
    unsigned long           sid;
    char                    sname[MAX_SNAME_LEN];
    unsigned long           refcnt;
    sctgt_hg_t              *hg;
    sctgt_port_t            *port;
    unsigned long           cmds_outstanding;
    unsigned long           mgmt_outstanding;
    void                (*remove_done)(sctgt_session_t *session);
    void                *cookie;
};
```

| | | |
|---|---|---|
| plist | - | Internally use, as the list anchor on port->sessions_list[cpu_id] |
| hlist | - | Internally use, as the list anchor on hg->sessions_list[cpu_id] |
| rlist | - | Internally use, as the list anchor on asynchronous deletion list |
| cpu_id | - | Internally use, record the receiving processor ID |
| sid | - | Internally use, the session ID. |
| sname | - | Internally use, string, the session name |
| refcnt | - | Internally use, atomic number |
| hg | - | Internally use, point to the host group that current session belongs to |
| port | - | Internally use, point back to the HBA driver data structure |
| cmds_outstanding | - | Internally use, atomic number |
| mgmt_outstanding | - | Internally use, atomic number |
| remove_done() | - | Internally use, store the callback pointer which will be called when the session is removed asynchronously. |
| cookie | - | Internally use, store the private data which will be used by HBA driver possibly. |

## Host Group Data Structure *sctgt_hg_t*

```
struct sctgt_hg {
    list_entry_t      list;
    unsigned int      hg_id;
    char              name[SCTGT_MAX_NAME_LEN];

    list_entry_t      hosts_acl;
    unsigned int      hosts_count;
    unsigned int      next_host_id;

    sctgt_lun_t       **luns;
    unsigned int      luns_count;


    unsigned char     *report_luns_buf;
    unsigned int      report_luns_buf_len;

    list_entry_t      sessions_list[SCTGT_MAX_CPUS];
    unsigned long     sessions_count;

    unsigned long     cmds_outstanding;
    unsigned long     mgmt_outstanding;

    unsigned long     refcnt;
};
```

```
list   -    Internally use, list anchor on host group global list
hg_id  -    Internally use
name[] -    Internally use, string
hosts_acl   -    Internally use, the ACL list, the node on the ACL
                 is sctgt_host_t
hosts_count      -    Internally use, amount of ACL entries.
next_host_id     -    Internally use, atomic number
luns        -    Internally use, will point to LUNs.
luns_count  -    Internally use

report_luns_buf   -    Internally use, will store the constructed
                       command block for namespace discovery request
                       REPORT_LUNS.
report_luns_buf_len    -    Internally use, indicate the REPORT_LUNS
                            buffer length.
sessions_list[]   -    Internally use, store the attached sessions
                       on current host group indexed by processor
                       ID.
sessions_count    -    Internally use, atomic number
cmds_outstanding  -    Internally use
mgmt_outstanding  -    Internally use
refcnt            -    Internally use
```

**Host Group ACL Entry Data Structure *sctgt_host_t***

```
struct sctgt_host {
    list_entry_t      list;
    int               id;
    char              name[MAX_SNAME_LEN];
    sctgt_hg_t        *hg;
};
```

| | | |
|---|---|---|
| list | - | Internally use, the list anchor on ACL list of host group |
| id | - | Internally use |
| name | - | Internally use, string, store the name will be matched with the session name |
| hg | - | Internally use, point back to host group. |

**LUN Data Structure *sctgt_lun_t***

```
struct sctgt_lun {
    list_entry_t      dlist;
    lun_t             lun_id;
    sctgt_hg_t        *hg;
    sctgt_dev_t       *dev;
    unsigned int      ua;

    unsigned int      _3rdpty;
    unsigned int      _3rdpty_id;

    unsigned int      excl_sid;
    unsigned char     excl_sname[MAX_SNAME_LEN];
    unsigned int      excl_sname_len;

    unsigned long     refcnt;
    unsigned long     cmds_outstanding;
};
```

| | | |
|---|---|---|
| dlist | - | Internally use, list anchor on attached backing device |
| lun_id | - | Internally use |
| hg | - | Internally use, point to host group it belongs to |
| dev | - | Internally use, point to backing device object |
| ua | - | Internally use, indicate wether there is an unit attention message |
| _3rdpty | - | Internally use |
| _3rdpty_id | - | Internally use, store third party locking information according to SPC |
| excl_sid | - | Internally use |
| excl_sname | - | Internally use, store the exclusive lock owner session name according to SPC. |
| excl_sname_len | - | Internally use, indicate the session name length |
| refcnt | - | Internally use |
| cmds_outstanding | - | Internally use |

**Backing Device Data Structure *sctgt_dev_t***

```
struct sctgt_dev {
    char              name[SCTGT_NAME_LEN];
    unsigned int      type;

    void   (*parser)(sctgt_dev_t *dev, sctgt_cmd_t *cmd);

    int    (*prepare_buffer)(sctgt_dev_t *dev, sctgt_cmd_t *cmd);
    void   (*release_buffer)(sctgt_dev_t *dev, sctgt_cmd_t *cmd)

    list_entry_t      list;
    list_entry_t      luns_list;
    unsigned int      luns_count;

    unsigned int      dev_id;

    unsigned long     refcnt;
    unsigned long     cmds_outstanding;


    void              *cookie;
} ;
```

| | | |
|---|---|---|
| name | - | Device driver offers, an unique name is required |
| type | - | Device driver offers, indicates the device type according to SAM, for instance SCTGT_DT_DISK |
| parser() | - | Device driver offers, the device specified SCSI command will be implemented in this function. |
| prepare_buffer() | - | Device driver offers, during writes such method will be called by SCSI stack to prepare buffer for receiving data from initiator, and during reads the backing device driver will call ***sctgt_buffer_prepare()*** by itself. |
| release_buffer() | - | Device driver offers, before the SCSI command is really released the method will be called. |

| | | |
|---|---|---|
| list | - | Internally use, list anchor on global device list |
| luns_list | - | Internally use, all LUNs attached to current device will be linked |
| luns_count | - | Internally use, atomic number |
| dev_id | - | Internally use |
| refcnt | - | Internally use |
| cmds_outstanding | - | Internally use |
| cookie | - | Internally use, device driver will store its private data on this member. |

**SCSI Command Data Structure *sctgt_cmd_t***

```
struct sctgt_cmd {
    list_entry_t      list;
    sctgt_port_t      *port;
    unsigned long     seqno;
    sctgt_scph_t      phase;
    sctgt_session_t   *session;
    lun_t             lun_id;
    sctgt_lun_t       *lun;
    lba_t             lba;
    unsigned int      len_expected;
    sctgt_xdir_t      xdir;
    unsigned char     cdb[MAX_CDB_SIZE];
    unsigned int      cdb_len;
    unsigned long     tag;
    sctgt_qt_t        qt;
    uint32_t          nblocks;
    unsigned int      scsi_status;
    unsigned char     sense[SENSE_SIZE];
    unsigned int      sense_len;

    unsigned int      sgl;
    void              *buffer;

    void              *ext;
};
```

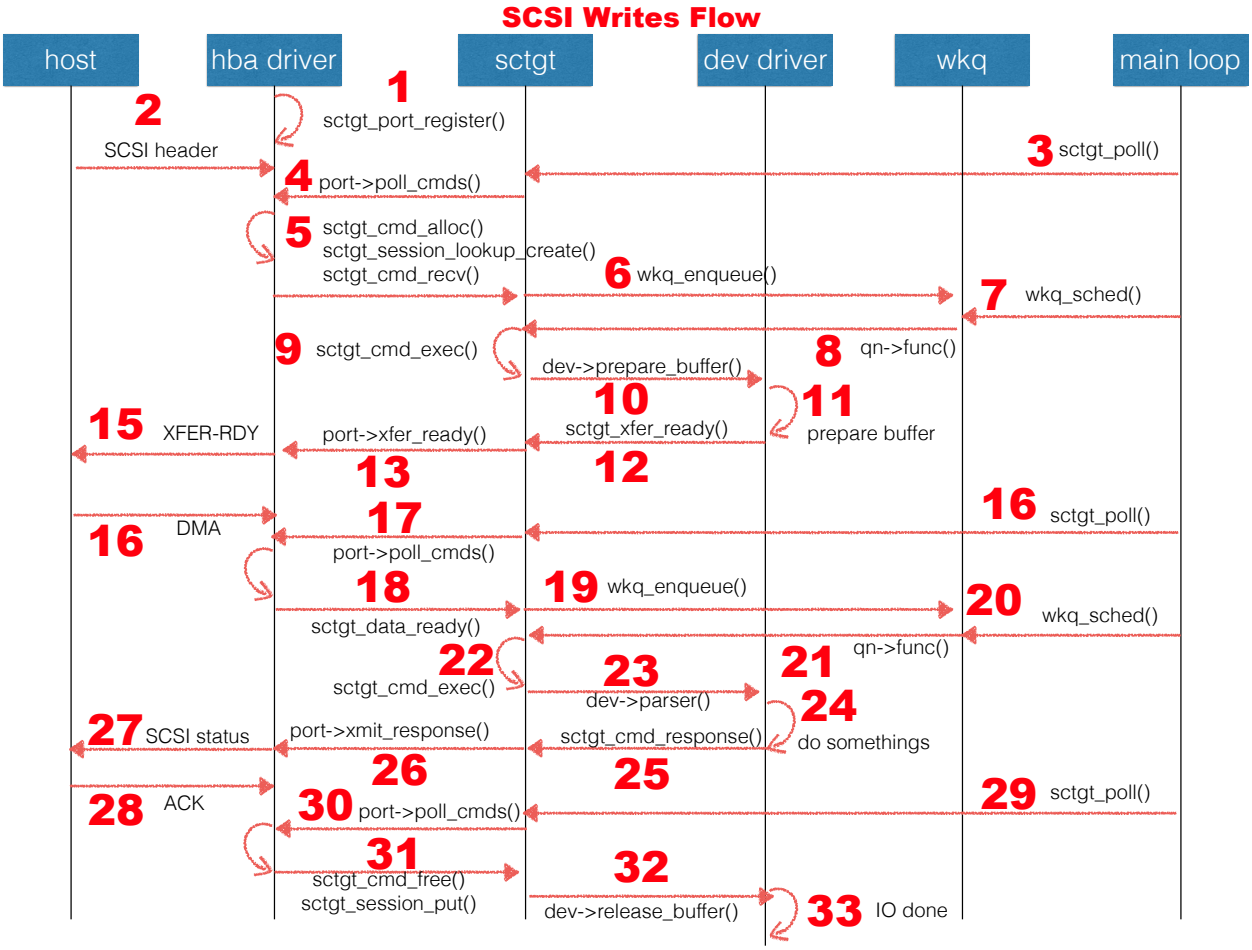
| | | |
|---|---|---|
| list | - | Internally use, SCSI stack will use the entry to link on list |
| port | - | Internally use, point to the current hardware |
| seqno | - | Internally use |
| phase | - | Internally use, indicate the SCSI phase |

**SCSI Task Management Data Structure *sctgt_mgmt_t***

```
struct sctgt_mgmt {
    list_entry_t            list;
    unsigned long           seqno;
    sctgt_session_t         *session;
    sctgt_tmc_t             tmc;
    unsigned long           tag;
    lun_t                   lun_id;
    sctgt_lun_t             *lun;
    void                    *cookie;
};
```

| | | |
|---|---|---|
| list | - | Internally use |
| seqno | - | Internally use |
| session | - | Internally use, point to the corresponding session |
| tmc | - | Internally use, task management code |
| tag | - | Internally use, command tag |
| lun_id | - | Internally use |
| lun | - | Internally use |
| cookie | - | Internally use, point to low lower HBA driver private data |

# Data Flow and Function Calling



# Sample HBA Driver

```
static sctgt_port_t isp_port_template = {
    .ptype                  = SCTGT_PT_FCP,
    .name                   = "Qlogic ISP xxxx Tgt",
    .sname_len              = SCTGT_FCP_SNAME_LEN,
    .max_cmds               = ISP_MAX_CMDS,
    .cmd_ext_sz             = sizeof(isp_cmd_t),
    .dma_mask               = ~0UL,
    .max_sgl                = ISP_MAX_SGL,
    .mcp                    = 0,
    .cpu_id                 = 0,
    .poll_cmd               = isp_poll_cmd,
    .poll_data_ready        = isp_poll_data_ready,
    .poll_status_completion = isp_poll_status_completion
    .xmit_response          = isp_xmit_response,
    .xfer_ready             = isp_xfer_ready,
    .release_session        = isp_release_session,
    .task_mgmt_done         = isp_task_mgmt_done,
    .control                = isp_control,
    .release_cmd            = isp_release_cmd,
    .cmd_ext_constructor    = isp_cmd_ext_constructor,
    .cmd_ext_destructor     = isp_cmd_ext_destructor,
}
```

```
int isp_probe(...)
{
    isp_port_t *ip;

    ...

    ip = alloc();

    ...

    /*Register as a PCI low-level PCI device*/
    ret = ret_intr_callback_register(...);
    ret = rte_eal_pci_register(...);

    ...

    /*Register as a SCSI target port driver*/
    memcpy(     &ip->port_driver,
                &isp_port_template,
                sizeof(ip->port_driver));

    snprintf(   ip->port_driver.name,
                sizeof(ip->port_driver.name),
                "Qlogic ISP xxxx  Tgt #%d",
                next_inst_id);

    sctgt_port_set_cookie(&ip->port_driver, ip);
    ...
    ret = sctgt_port_register(&ip->port_driver);
    ...

    return ret;
}
```

```
void
isp_poll_cmd(void)
{
    sctgt_cmd_t *cmd;
    sctgt_session_t *sess;

    cmd = sctgt_cmd_alloc(&port->port_driver);
    sess = sctgt_session_lookup_create(...);
    sctgt_cmd_recv(cmd,...);

}

void
isp_poll_data_ready(void)
{
    isp_hw_get_msg(msg, MSG_R2T);
    cmd = lookup from msg->tag;
    sctgt_cmd_data_ready(cmd);
}

void
isp_poll_status_completion(void)
{
    isp_hw_get_msg(msg, MSG_COMP);
    cmd = lookup from msg->tag;
    sctgt_session_put(cmd->session);
    sctgt_cmd_free(cmd);
}


void
isp_xmit_response(...)
{
    isp_hw_put_msg(msg, cmd, MSG_XMIT);
}


void
isp_xfer_ready(...)
{
    isp_hw_put_msg(msg, cmd->tag, MSG_XFER);
}
```

```
void
isp_release_session(...)
{
    prv = sctgt_session_get_private(session);

    ...

}

void
isp_task_mgmt_done(mgmt, ret)
{

    isp_hw_put_msg(msg, mgmt, MSG_TASK_MGMT);
    ...
}

void
isp_release_cmd()
{
        ...
    sctgt_cmd_free(cmd);
}


int isp_cmd_ext_constructor(cmd)
{
    priv = sctgt_cmd_get_cookie(cmd);

    return isp_init_priv(priv);
}

void
isp_cmd_ext_destructor(cmd)
{
    priv = sctgt_cmd_get_cookie(cmd);

    isp_fini_priv(priv);
}
```

# SCSI Block Device Interface

```
struct sdi_disk {

    char                name[SCTGT_NAME_LEN];
    char                vendor_id[SDI_VENDOR_LEN + 1];
    char                model_id[SDI_MODEL_LEN + 1];
    char                revision_id[SDI_REVISION_LEN + 1];
    char                sn[SDI_P80_SN_LEN + 1];

    uint64_t            sectors;

    uint32_t            block_size;
    unsigned int        write_cache;
    unsigned int        read_only;
    unsigned int        support_queue;
    sdi_ops_t           *ops;


    unsigned int        id;

    unsigned long       refcnt;

    sctgt_dev_t         *dev;
    void                *cookie;
};
```

| | | |
|---|---|---|
| name | - | Driver offers, disk name |
| vendor_id | - | Driver offers, vendor identifier in SCSI INQUIRY |
| model_id | - | Driver offers, production identifier in SCSI INQUIRY |
| revision_id | - | Driver offers, hardware revision identifier in SCSI INQUIRY |
| sn | - | Driver offers, serial number by SCSI INQUIRY page 0x80 |
| sectors | - | Driver offers, sectors |
| block_size | - | Driver offers, block size of sector |
| write_cache | - | Driver offers, 1 stands for write back, 0 stands for write through |
| read_only | - | Driver offers, 1 stands for read only, 0 stands for regular read-write |
| support_queue | - | Driver offers, support queue |
| ops | - | Driver offers, operation function set defined by ***sdi_ops_t*** |
| id | - | Internally uses |
| refcnt | - | Internally uses |
| dev | - | Internally uses, points to device object of ***sctgt*** |
| cookie | - | Internally uses, saves driver private data |

```
struct sdi_ops {
    int    (*write_begin)(sdi_disk_t *disk, sctgt_cmd_t *cmd);
    void   (*write_end)(sdi_disk_t *disk, sctgt_cmd_t *cmd);
    void   (*write_async)(sdi_disk_t *disk, sctgt_cmd_t *cmd);
    void   (*read_async)(sdi_disk_t *disk, sctgt_cmd_t *cmd);
};
```

write_begin() - ***sdi*** will call it for preparing buffer before writes
write_end() - once ***write_async()*** completed, ***sdi*** will call it for ending buffer
write_async() - ***sdi*** wil pass write request to the function after ***write_begin()*** called
read_async() - ***sdi*** wil pass read request to the function

```
int sdi_disk_register(sdi_disk_t *disk);

void sdi_disk_unregister(sdi_disk_t *disk);
```

sdi_disk_register() - register a SCSI disk
sdi_disk_unregister() - unregister a SCSI disk

```
int sdi_cmd_complete(sctgt_cmd_t *cmd, int error);
```

sdi_cmd_complete - complete a SCSI command

# Management Guide

1. Create an ACL host group (namespace)
   ***sctgt_hg --add --hg db***

2. Add ACL entries
   2.1. Add a FCP host with its name "***11:22:33:44:55:66***"
      ***sctgt_hg --add --host "11:22:33:44:55:66"***
   2.2. Add a local loop virtual host for virtual HBA#0
      ***sctgt_hg --add --host "loop0"***
   2.3. Add an iSCSI host with its ***InitiatorName***
      ***sctgt_hg --add --host "iqn.2009-05.com.test:test.1"***

3. Create a ramdisk for example
   ***ramdisk_adm --add --name "rdisk1" --sectors 128MB***

4. Attach ramdisk to the LUN of defined host group for example
   ***sctgt_lun --attach --hg db --lun 0 --dev "rdisk1"***

5. Launch ramdisk program for example
   ***ramdisk -c <cpu-mask> --pci-list <pci1,pci2,...>***