SCST user space device handler interface description

Vladislav Bolkhovitin

Version 3.1.0

Contents

1 Introduction

SCST user space device handler module scst_user is a device handler for SCST, which provides a way to implement in the user space complete, full feature virtual SCSI devices in the SCST environment.

This document assumes that the reader is familiar with the SCST architecture and the states through which SCSI commands go during processing in SCST. Module scst_user basically only provides hooks to them. Their description could be found on the SCST web page on http://scst.sf.net.

2 User space API

Module scst_user provides /dev/scst_user character device with the following system calls available:

- **open()** - allows to open the device and get a file handle, which will be used in all subsequent actions until close() is called
- **close()** - closes file handle returned by open()
- **poll()** - allows to wait until some pending command from SCST to process is available.
- **ioctl()** - main call, which allows commands interchange with the SCST core.

Device /dev/scst_user could be opened in blocking or non-blocking mode using O_NONBLOCK flag. In the blocking mode ioctl() SCST_USER_REPLY_GET_CMD function blocks until there is a new subcommand to process. In the non-blocking mode if there are no pending subcommands SCST_USER_REPLY_GET_CMD function returns immediately with EAGAIN error code, and the user space device handler can use poll() call to get notification about new subcommands arrival. The blocking mode is the default.

The module scst_user API is defined in scst_user.h file.

3 IOCTL() functions

There are following IOCTL functions available. All of them has one argument. They all, except SCST_USER_REGISTER_DEVICE return 0 for success or -1 in case of error, and errno is set appropriately.

3.1 SCST_USER_REGISTER_DEVICE

SCST_USER_REGISTER_DEVICE registers new virtual user space device. The argument is:

```
struct scst_user_dev_desc
{
        aligned_u64 version_str;
        aligned_u64 license_str;
        uint8_t type;
        uint8_t sgv_shared;
        uint8_t sgv_disable_clustered_pool;
        int32_t sgv_single_alloc_pages;
        int32_t sgv_purge_interval;
        uint8_t has_own_order_mgmt;
        struct scst_user_opt opt;
        uint32_t block_size;
        char name[SCST_MAX_NAME];
        char sgv_name[SCST_MAX_NAME];
},
```

where:

- **version_str** - protocol version, shall be DEV_USER_VERSION.
- **license_str** - license of this module, for instance, "GPL", "GPL v2", or "Proprietary". This field serves the same purpose as macroses EXPORT_SYMBOL/EXPORT_SYMBOL_GPL of the Linux kernel. You can find more info about it, if you ask at license@scst-tgt.com e-mail address.
- **type** - SCSI type of the device.
- **sgv_shared** - true, if the SGV cache for this device should be shared with other devices. False, if the SGV cache should be dedicated.
- **sgv_disable_clustered_pool** - disable usage of clustered pool for this device. Normally, 2 independent SGV pools created and used for each device - normal and clustered. Clustered pool creates and contains SG vectors, in which coalesced paged merged (clustered) in single SG entries. This is good for performance. But not all target drivers can use such SG vectors, plus in some cases it is more convenient to have a single memory pool. So, this option provides such possibility.
- **sgv_single_alloc_pages** - if 0, then the SGV cache for this device will work in the set of power 2 size buffers mode. If >0, then the SGV cache will work in the fixed size buffers mode. In this case it sets the size of each buffer in pages. See the SGV cache documentation (http://scst.sourceforge.net/sgv_cache.txt) for more details.
- **sgv_purge_interval** - sets the SGV cache purging interval. I.e. an SG buffer will be freed if it's unused for time t purge_interval <= t < 2*purge_interval. If purge_interval is 0, then the default interval will be used (60 seconds). If purge_interval <0, then the automatic purging will be disabled. Shrinking by the system's demand will also be disabled.
- **has_own_order_mgmt** - set it in non-zero, if device implements own ORDERED commands management, i.e. guarantees commands execution order requirements, specified by SAM.
- **opt** - device options, see SCST_USER_SET_OPTIONS/SCST_USER_GET_OPTIONS below
- **block_size** - block size, shall be divisible by 512 for block devices
- **name** - name of the device
- **sgv_name** - name of SGV cache for this device

SCST_USER_REGISTER_DEVICE returns registered device's handler or -1 in case of error, and errno is set appropriately.

In order to unregister the device, either call SCST_USER_UNREGISTER_DEVICE function, or close on its file descriptor.

3.2 SCST_USER_UNREGISTER_DEVICE

SCST_USER_UNREGISTER_DEVICE is obsolete and should not be used. Just close the device's fd instead.

3.3 SCST_USER_SET_OPTIONS/SCST_USER_GET_OPTIONS

SCST_USER_SET_OPTIONS/SCST_USER_GET_OPTIONS allows to set or get correspondingly various options that control various aspects of SCSI commands processing.

The argument is:

```
struct scst_user_opt
{
	uint8_t parse_type;
	uint8_t on_free_cmd_type;
	uint8_t memory_reuse_type;
	uint8_t partial_transfers_type;
	uint32_t partial_len;

	uint8_t tst;
	uint8_t queue_alg;
	uint8_t tas;
	uint8_t swp;
	uint8_t d_sense;

	uint8_t has_own_order_mgmt;
},
```

where:

- **parse_type** - defines how the user space handler wants to process PARSE subcommand. Possible values are:
  - **SCST_USER_PARSE_STANDARD** - tells SCST use standard internal parser for this SCSI device type.
  - **SCST_USER_PARSE_CALL** - tells SCST generate SCST_USER_PARSE for all SCSI commands
  - **SCST_USER_PARSE_EXCEPTION** - tells SCST generate SCST_USER_PARSE for unknown SCSI commands or SCSI commands that produce errors in the standard parser.

- **on_free_cmd_type** - defines how the user space handler wants to process ON_FREE_CMD subcommand. Possible values are:
  - **SCST_USER_ON_FREE_CMD_CALL** - tells SCST generate SCST_USER_ON_FREE_CMD for all SCSI commands
  - **SCST_USER_ON_FREE_CMD_IGNORE** - tells SCST do nothing on this event.
- **memory_reuse_type** - defines how memory allocated by the user space handler for a SCSI commands data buffers is then reused by the SCST core as data buffer for subsequent commands. Possible values are:
  - **SCST_USER_MEM_NO_REUSE** - no memory reuse is possible, for each commands the user space handler will each time allocate a dedicated data buffer
  - **SCST_USER_MEM_REUSE_READ** - memory reuse by only READ-type commands (i.e. which involve data transfer from target to initiator) is allowed. For all WRITE-type commands (i.e. which involves data transfer from initiator to target) the user space handler will each time allocate a dedicated data buffer
  - **SCST_USER_MEM_REUSE_WRITE** - memory reuse by only WRITE-type commands is allowed. For all READ-type commands the user space handler will each time allocate a dedicated data buffer
  - **SCST_USER_MEM_REUSE_ALL** - unlimited memory reuse is possible.
- **partial_transfers_type** - defines if the user space handler supports partial data transfers, when a SCSI command, which required big data transfer, is broken on several subcommands with smaller data transfers. This allows to improve performance by simultaneous data transfers from/to remote initiator and to/from the underlying storage device as well as lower allocation memory requirements for each (sub)command. All subcommands will have the same unique value in "parent_cmd_h" field and SCST_USER_SUBCOMMAND flag in "partial" field of struct scst_user_scsi_cmd_exec. The final subcommand will also have in that field SCST_USER_SUBCOMMAND_FINAL flag. All the subcommands will have the original unmodified CDB. Possible values are:
  - **SCST_USER_PARTIAL_TRANSFERS_NOT_SUPPORTED** - the partial data transfers are not supported
  - **SCST_USER_PARTIAL_TRANSFERS_SUPPORTED_ORDERED** - the partial data transfers are supported, but all the subcommands must come in order of data chunks. Could be used, e.g., for tape devices.
  - **SCST_USER_PARTIAL_TRANSFERS_SUPPORTED** - the partial data transfers are supported without limitations.
- **tst**, **queue_alg**, **tas**, **swp**, **d_sense** - set values for TST, QUEUE ALGORITHM MODIFIER, TAS, SWP and D_SENSE fields from control mode page correspondingly, see SPC.
- **has_own_order_mgmt** - true, if the user space handler has full commands execution order management, i.e. guarantees commands execution order as required by SAM. False otherwise.

Flags *parse_type* and *on_free_cmd_type* are designed to improve performance by eliminating context switches to the user space handler, when processing of the corresponding events isn't needed.

Flag *memory_reuse_type* is designed to improve performance by eliminating memory allocation, preparation and then freeing each time for each commands, if the same memory will be allocated again and again. See SCST_USER_ALLOC_MEM description below for more info.

SCST_USER_SET_OPTIONS should not be used from the same and the only thread, which also handles incoming commands, otherwise there could be a "deadlock", when SCST_USER_SET_OPTIONS waits for active commands finish, but nobody handles them. This "deadlock" will be resolved only when initiator, which sent those commands, aborts them after timeout.

IMPORTANT! It is duty of the caller to serialize SCST_USER_SET_OPTIONS invocations. The kernel code does not have any locking around modifying above properties.

3.4 SCST_USER_REPLY_AND_GET_CMD

SCST_USER_REPLY_AND_GET_CMD allows at one call reply on the current subcommand from SCST and get the next one. If 0 is returned by ioctl(), SCST_USER_REPLY_AND_GET_CMD returns a SCST subcommand in the argument, which is defined as the following:

```
struct scst_user_get_cmd
{
        uint32_t cmd_h;
        uint32_t subcode;
        union {
                uint64_t preply;
                struct scst_user_sess sess;
                struct scst_user_scsi_cmd_parse parse_cmd;
                struct scst_user_scsi_cmd_alloc_mem alloc_cmd;
                struct scst_user_scsi_cmd_exec exec_cmd;
                struct scst_user_scsi_on_free_cmd on_free_cmd;
                struct scst_user_on_cached_mem_free on_cached_mem_free;
                struct scst_user_tm tm_cmd;
        };
},
```

where:

- **cmd_h** - command handle used to identify the command in the reply.
- **subcode** - subcommand code, see 4.1 below
- **preply** - pointer to the reply data or, if 0, there is no reply. See SCST_USER_REPLY_CMD for description of struct scst_user_reply_cmd fields

Other union members contain command's specific payload.

For all received subcommands the user space device handler shall call SCST_USER_REPLY_AND_GET_CMD or SCST_USER_REPLY_CMD function to tell SCST that the subcommand's processing is finished, although some subcommands don't return a value.

You can see description of possible subcommands in section 4 (subcommands).

3.5 SCST_USER_REPLY_AND_GET_MULTI

SCST_USER_REPLY_AND_GET_MULTI allows at one call reply on the multiple subcommands from SCST and get the multiple next subcommands.

Its argument is defined as:

```
struct scst_user_get_multi {
        aligned_u64 preplies;
        int16_t replies_cnt;
        int16_t replies_done;
        int16_t cmds_cnt;
        struct scst_user_get_cmd cmds[0];
},
```

where:

- **preplies** - pointer to array of replies with size *replies_cnt*. See SCST_USER_REPLY_CMD for description of struct scst_user_reply_cmd fields
- **replies_cnt** - number of entries in *preplies* array. If 0, there are no replies
- **replies_done** - returns how many replies were processed by SCST. If there are unprocessed replies, the user space device handler must retry the unprocessed replies.
- **cmds_cnt** - on entry: number of available entries in *cmds* array; on exit - number of valid subcommands in *cmds* array
- **cmds** - returned array of subcommands

Returns 0 on success or -1 in case of error, and errno is set appropriately.

3.6 SCST_USER_REPLY_CMD

SCST_USER_REPLY_CMD IOCTL function allows the user space handler to return the result of a command's execution. Its argument is defined as:

```
struct scst_user_reply_cmd
{
        uint32_t cmd_h;
        uint32_t subcode;
        union {
                int32_t result;
                struct scst_user_scsi_cmd_reply_parse parse_reply;
                struct scst_user_scsi_cmd_reply_alloc_mem alloc_reply;
                struct scst_user_scsi_cmd_reply_exec exec_reply;
        };
},
```

where:

- **cmd_h** - command handle used to identify the command in the reply.
- **subcode** - subcommand code, see 4.1

Union contains the subcommand's specific payloads:

result - subcommand's result code

OR

```
struct scst_user_scsi_cmd_reply_parse
{
        uint8_t queue_type;
        uint8_t data_direction;
        uint16_t cdb_len;
        aligned_i64 lba;
        uint32_t op_flags;
        aligned_i64 data_len;
        int32_t bufflen;
        int32_t out_bufflen;
},
```

where:

- **queue_type** - SCSI task attribute (queue type). NOTE! In current implementation setting changing this field from the provided value affects commands execution only when then produced by iSCSI-SCST target. With all other target drivers, this field is ignored. This is because for them commands queueing is set before parse() called.
- **data_direction** - command's data flow direction, one of SCST_DATA_* constants
- **cdb_len** - length of CDB
- **lba** - LBA of the command, if any
- **op_flags** - commands flags, one or more scst_cdb_flags bits, see above. At least SCST_INFO_VALID must be set for correct processing. SCST_IMPLICIT_HQ not implemented (yet) for single stage init target drivers (all, except iSCSI), because custom parse can reorder commands due to multithreaded processing.
- **data_len** - command's data length. Could be different from bufflen for commands like VERIFY, which transfer different amount of data, than process, or even none of them
- **bufflen** - command's buffer length
- **out_bufflen** - command's out buffer length (for bidirectional commands)

OR

```
struct scst_user_scsi_cmd_reply_alloc_mem
{
        uint64_t pbuf;
},
```

where:

- **pbuf** - pointer to command's data buffer

OR

```
struct scst_user_scsi_cmd_reply_exec
{
        int32_t resp_data_len;
        uint64_t pbuf;

        uint8_t reply_type;

        uint8_t status;
        uint8_t sense_len;
        aligned_u64 psense_buffer;
},
```

where:

- **resp_data_len** - length of the response data
- **pbuf** - pointer to command's data buffer. Used only when in the original SCST_USER_EXEC subcommand pbuf field is 0
- **reply_type** - could be one of the following constants:
  - **SCST_EXEC_REPLY_BACKGROUND** - tells SCST send to the remote initiator GOOD status, but the command not yet completed by the user space handler, it is being executed in the background. When it completed, the user space handler must call SCST_USER_REPLY_CMD again with reply_type SCST_EXEC_REPLY_COMPLETED. This mode can be used only for WRITEs and with no mem reuse for them. Also in this mode SCST_USER_ON_FREE_CMD_IGNORE supposed to be used. SCST_USER_ON_FREE_CMD_IGNORE should be OK, because in this mode the user space handler knows when this memory can be reused (SCST_EXEC_REPLY_COMPLETED time). In case if the user space handler finishes before SCST sent reply to the initiator, SCST can still have reference to that memory, but it is harmless, because SCST is not going to touch this memory anyhow in this processing path, because WRITEs are not sending data, so this memory can be safely reused for other needs.
  - **SCST_EXEC_REPLY_COMPLETED** - the user space handler completed the command
- **status** - SAM status of the commands execution
- **sense_len** - length of sense data in psense_buffer, if any
- **psense_buffer** - pointed to sense buffer

3.7 SCST_USER_FLUSH_CACHE

SCST_USER_FLUSH_CACHE - flushes SGV cache for the corresponding virtual user space device and queues for all cached memory buffers corresponding SCST_USER_ON_CACHED_MEM_FREE subcommands.

During execution of SCST_USER_FLUSH_CACHE at least one another thread must process all coming subcommands, otherwise after timeout it will fail with EBUSY error.

SCST_USER_FLUSH_CACHE doesn't have any parameters.

SCST_USER_FLUSH_CACHE returns 0 on success or -1 in case of error, and errno is set appropriately.

3.8 SCST_USER_DEVICE_CAPACITY_CHANGED

SCST_USER_DEVICE_CAPACITY_CHANGED - queues CAPACITY DATA HAS CHANGED Unit Attention or corresponding Asynchronous Event to the corresponding virtual device. It will notify remote initiators, connected to the device, and allow them to automatically refresh new device size. You should use SCST_USER_DEVICE_CAPACITY_CHANGED after resize of the device.

SCST_USER_DEVICE_CAPACITY_CHANGED doesn't have any parameters.

SCST_USER_DEVICE_CAPACITY_CHANGED returns 0 on success or -1 in case of error, and errno is set appropriately.

3.9 SCST_USER_GET_EXTENDED_CDB

SCST_USER_GET_EXTENDED_CDB - requests extended CDB, if CDB size is more than SCST_MAX_CDB_SIZE bytes. In this case SCST_USER_GET_EXTENDED_CDB returns additional CDB data beyond SCST_MAX_CDB_SIZE bytes.

SCST_USER_GET_EXTENDED_CDB has the following arguments:

```
struct scst_user_get_ext_cdb {
        uint32_t cmd_h;
        aligned_u64 ext_cdb_buffer;
},
```

where:

- **cmd_h** - command handle used to identify the command in the reply.
- **ext_cdb_buffer** - pointer to buffer, where extended CDB will be copied.

SCST_USER_GET_EXTENDED_CDB returns 0 on success or -1 in case of error, and errno is set appropriately.

3.10 SCST_USER_PREALLOC_BUFFER

SCST_USER_PREALLOC_BUFFER - asks to preallocate a buffer.

It has the following arguments:

```
union scst_user_prealloc_buffer {
        struct scst_user_prealloc_buffer_in in;
        struct scst_user_prealloc_buffer_out out;
},
```

where:

- **in** - provides data about buffer to preallocate
- **out** - returns information about preallocated buffer

Structure *scst_user_prealloc_buffer_in* has the following definition:

```
struct scst_user_prealloc_buffer_in {
        aligned_u64 pbuf;
        uint32_t bufflen;
        uint8_t for_clust_pool;
},
```

where:

- **pbuf** - pointer to the buffer
- **bufflen** - size of the buffer
- **for_clust_pool** - if 1, then the buffer will be preallocated in the clustered pool. If 0, then the buffer will be preallocated in the normal pool.

Structure *scst_user_prealloc_buffer_out* has the following definition:

```
struct scst_user_prealloc_buffer_out {
        uint32_t cmd_h;
}
```

where:

- **cmd_h** - handle used to identify the buffer in SCST_USER_ON_CACHED_MEM_FREE subcommand.

SCST_USER_PREALLOC_BUFFER returns 0 on success or -1 in case of error, and errno is set appropriately.

4 SCST_USER subcommands

4.1 SCST_USER_ATTACH_SESS

SCST_USER_ATTACH_SESS notifies the user space handler that a new initiator's session is about to be attached to the device. Payload contains struct scst_user_sess, which is defined as the following:

```
struct scst_user_sess
{
        uint64_t sess_h;
        uint64_t lun;
        uint16_t threads_num;
        uint8_t rd_only;
        uint16_t scsi_transport_version;
        uint16_t phys_transport_version;
        char initiator_name[SCST_MAX_NAME];
        char target_name[SCST_MAX_NAME];
},
```

where:

- **sess_h** - session's handle, may not be 0
- **lun** - assigned LUN for this device in this session
- **threads_num** - specifies amount of additional threads, requested by the corresponding target driver
- **rd_only** - if true, this device is read only in this session
- **scsi_transport_version** - version descriptor value for SCSI transport of this session
- **phys_transport_version** - version descriptor value for physical transport of this session
- **initiator_name** - name of the remote initiator, which initiated this session
- **target_name** - name of the target, to which this session belongs

When SCST_USER_ATTACH_SESS is returned, it is guaranteed that there are no other commands are being executed or pending.

After SCST_USER_ATTACH_SESS function completed, the user space device handler shall reply using "result" field of the corresponding reply command.

4.2 SCST_USER_DETACH_SESS

SCST_USER_DETACH_SESS notifies the user space handler that the corresponding initiator is about to be detached from the particular device. Payload contains struct scst_user_sess, where only handle field is valid.

When SCST_USER_DETACH_SESS is returned, it is guaranteed that there are no other commands are being executed or pending.

This command doesn't reply any return value, although SCST_USER_REPLY_AND_GET_CMD or SCST_USER_REPLY_CMD function must be called.

4.3 SCST_USER_PARSE

SCST_USER_PARSE returns SCSI command on PARSE state of the SCST processing. The PARSE state is intended to check validity of the command, determine data transfer type and the necessary data buffer size. This subcommand is returned only if SCST_USER_SET_OPTIONS parse_type isn't set to SCST_USER_PARSE_STANDARD. In this case the standard SCST internal parser for this SCSI device type will do all the job.

Payload contains struct scst_user_scsi_cmd_parse, which is defined as the following:

```
struct scst_user_scsi_cmd_parse
{
        uint64_t sess_h;

        uint8_t cdb[SCST_MAX_CDB_SIZE];
        uint16_t cdb_len;

        aligned_i64 lba;

        uint32_t timeout;
        int32_t bufflen;
        aligned_i64 data_len;
        int32_t out_bufflen;

        uint32_t op_flags;

        uint8_t queue_type;
        uint8_t data_direction;

        uint8_t expected_values_set;
        uint8_t expected_data_direction;
        int32_t expected_transfer_len;
        int32_t expected_out_transfer_len;

        uint32_t sn;
},
```

where:

- **sess_h** - corresponding session handler
- **cdb** - SCSI CDB
- **cdb_len** - SCSI CDB length
- **lba** - LBA of the command, if any

- **timeout** - CDB execution timeout
- **bufflen** - command's buffer length
- **data_len** - command's data length. Could be different from bufflen for commands like VERIFY, which transfer different amount of data, than process, or even none of them
- **out_bufflen** - for bidirectional commands command's OUT, i.e. from initiator to target, buffer length
- **op_flags** - CDB flags, one or more scst_cdb_flags bits, see below.
- **queue_type** - SCSI task attribute (queue type)
- **data_direction** - command's data flow direction, one of SCST_DATA_* constants
- **expected_values_set** - true if expected_data_direction and expected_transfer_len contain valid values
- **expected_data_direction** - remote initiator supplied command's data flow direction
- **expected_transfer_len** - remote initiator supplied transfer length
- **expected_out_transfer_len** - remote initiator supplied out, i.e. from initiator to target, transfer length for bidirectional commands.
- **sn** - command's SN, which might be used for task management

Bits of scst_cdb_flags can be:

- **SCST_TRANSFER_LEN_TYPE_FIXED** - this command uses fixed blocks addressing
- **SCST_SMALL_TIMEOUT** - this command needs small timeout
- **SCST_LONG_TIMEOUT** - this command needs a long timeout
- **SCST_UNKNOWN_LENGTH** - data buffer length for this command is unknown
- **SCST_INFO_VALID** - bits of op_flags are valid
- **SCST_IMPLICIT_HQ** - this command is an implicit HEAD OF QUEUE command
- **SCST_SKIP_UA** - Unit Attentions shouldn't be delivered for this command
- **SCST_WRITE_MEDIUM** - this command writes data on the medium, so should be forbidden for read-only devices
- **SCST_LOCAL_CMD** - this command can be processed by SCST core.

In the PARSE state of SCSI commands processing the user space device handler shall check and provide SCST values for command data buffer length, data flow direction and timeout, which it shall reply using the corresponding reply command.

In case of any error the error reporting should be deferred until SCST_USER_EXEC subcommand, where the appropriate SAM status and sense shall be set.

4.4 SCST_USER_ALLOC_MEM

SCST_USER_ALLOC_MEM returns SCSI command on memory allocation state of the SCST processing. On this state the user space device handler shall allocate the command's data buffer with bufflen length and then return it to SCST using the corresponding reply command. Then SCST internally will convert it in SG vector in order to use it itself and by target drivers.

If the memory reuse type is disabled (i.e. set to SCST_USER_MEM_NO_REUSE) there are no special requirements for buffer memory or its alignment, it could be just what malloc() returned. If the memory reuse type is enabled, the buffer shall be page size aligned, for example using memalign() function.

Payload contains struct scst_user_scsi_cmd_alloc_mem, which is defined as the following:

```
struct scst_user_scsi_cmd_alloc_mem
{
        uint64_t sess_h;

        uint8_t cdb[SCST_MAX_CDB_SIZE];
        uint16_t cdb_len;

        int32_t alloc_len;

        uint8_t queue_type;
        uint8_t data_direction;

        uint32_t sn;
},
```

where:

- **sess_h** - corresponding session handler
- **cdb** - SCSI CDB
- **cdb_len** - SCSI CDB length
- **alloc_len** - command's buffer length
- **queue_type** - SCSI task attribute (queue type)
- **data_direction** - command's data flow direction, one of SCST_DATA_* constants
- **sn** - command's SN, which might be used for task management

Memory allocation, preparation and freeing are ones of the most complicated and expensive operations during SCSI commands processing. Module scst_user provides a way to almost completely eliminate those operations by reusing once allocated memory for subsequent SCSI commands. It is controlled by *memory_reuse_type* option, which could be set by SCST_USER_SET_OPTIONS function. If any type memory reusage is enabled, then SCST will use its internal SGV cache in order to cache allocated and fully built SG vectors for subsequent commands of this type, so for them SCST_USER_ALLOC_MEM subfunction will not be called and in SCST_USER_EXEC pbuf pointer will point to that reused buffer.

SGV cache is a backend cache made on top of Linux kernel kmem cache. It caches unused SG vectors for future allocations to improve performance. Then, after some time of inactivity or when the system is under memory pressure, the cache entries will be freed and the user space handler will be notified using SCST_USER_ON_CACHED_MEM_FREE. See the SGV cache documentation <http://scst.sourceforge.net/sgv_cache.txt> for more details.

Since the SGV cache caches SG vectors, which can be bigger, than actual data sizes of SCSI commands, alloc_len field could also be bigger, than actually required by the SCSI command.

The memory reuse could be used in both SCSI tagged and untagged queuing environments. In the SCSI tagged queuing environment the SGV cache will take care that several commands don't use the same buffer simultaneously by asking the user space handler to allocate a new data buffer, when all cached ones are busy.

Some important notes:

1. If the user space handler needs to call fork(), it must call madvise() with MADV_DONTFORK flag for all allocated data buffers, otherwise parent or child process could loose the connection with them, which could lead to data corruption. See <http://lwn.net/Articles/171941/> for details.
2. The interface assumes that all allocated memory by the user space handler is DMA'able by the target hardware. This is almost always true for most modern systems, except if the target hardware isn't capable of using 64-bit address space and the system has >4GB of memory or the memory addresses are in address space, which is unavailable with 32-bit addresses.

In case of any error the error reporting should be deferred until SCST_USER_EXEC subcommand, where the appropriate SAM status and sense should be set.

4.5 SCST_USER_EXEC

SCST_USER_EXEC returns SCSI command on execution state of the SCST processing. The user space handler should execute the SCSI command and reply using the corresponding reply command.

In some cases for performance reasons for READ-type SCSI commands SCST_USER_ALLOC_MEM subcommand isn't returned before SCST_USER_EXEC. Thus, if pbuf pointer is 0 and the SCSI command needs data transfer, the user space handler should be prepared to allocate the data buffer with size alloc_len, which could be bigger (due to the SGV cache), than actually required by the SCSI command. But field bufflen will contain the correct value. All the memory reusage rules, described for SCST_USER_ALLOC_MEM, apply to SCST_USER_EXEC as well.

Payload contains struct scst_user_scsi_cmd_exec, which is defined as the following:

```
struct scst_user_scsi_cmd_exec
{
        uint64_t sess_h;

        uint8_t cdb[SCST_MAX_CDB_SIZE];
        uint16_t cdb_len;

        aligned_i64 lba;
        aligned_i64 data_len;
        int32_t bufflen;
        int32_t alloc_len;
        uint64_t pbuf;
        uint8_t queue_type;
        uint8_t data_direction;
        uint8_t partial;
        uint32_t timeout;

        aligned_u64 p_out_buf;
        int32_t out_bufflen;

        uint32_t sn;

        uint32_t parent_cmd_h;
        int32_t parent_cmd_data_len;
        uint32_t partial_offset;
},
```

where:

- **sess_h** - corresponding session handler
- **cdb** - SCSI CDB
- **cdb_len** - SCSI CDB length
- **lba** - LBA of the command, if any
- **data_len** - command's data length. Could be different from bufflen for commands like VERIFY, which transfer different amount of data, than process, or even none of them
- **bufflen** - command's buffer length
- **alloc_len** - command's buffer length, which should be allocated, if pbuf is 0 and the command requires data transfer
- **pbuf** - pointer to command's data buffer or 0 for SCSI commands without data transfer.
- **queue_type** - SCSI task attribute (queue type)
- **data_direction** - command's data flow direction, one of SCST_DATA_* constants
- **partial** - specifies, if the command is a partial subcommand, could have the following OR'ed flags:
  - **SCST_USER_SUBCOMMAND** - set if the command is a partial subcommand
  - **SCST_USER_SUBCOMMAND_FINAL** - set if the subcommand is a final one
- **timeout** - CDB execution timeout
- **p_out_buf** - for bidirectional commands pointer on command's OUT, i.e. from initiator to target, data buffer or 0 for SCSI commands without data transfer

- **out_bufflen** - for bidirectional commands command's OUT, i.e. from initiator to target, buffer length
- **sn** - command's SN, which might be used for task management
- **parent_cmd_h** - has the same unique value for all partial data transfers subcommands of one original (parent) command
- **parent_cmd_data_len** - for partial data transfers subcommand has the size of the overall data transfer of the original (parent) command
- **partial_offset** - has offset of the subcommand in the original (parent) command

It is guaranteed that only commands of the same queue_type per session can be returned simultaneously.

In case of any error it should be reported via appropriate SAM status and sense. If it happens for a subcommand of a partial data transfers command, all other subcommands of this command, which already passed the the user space handler or will be passed in the future, will be aborted by scst_user, the user space handler should ignore them.

4.6 SCST_USER_ON_FREE_CMD

SCST_USER_ON_FREE_CMD returns SCSI command when the command is about to be freed. At this stage, the user space device handler could do any necessary cleanups, for instance, free allocated for data buffer memory.

NOTE! If the memory reusage is enabled, then the data buffer must not be freed, it will be reused by subsequent SCSI commands. The buffer must be freed only on SCST_USER_ON_CACHED_MEM_FREE event.

Payload contains struct scst_user_scsi_on_free_cmd, which is defined as the following:

```
struct scst_user_scsi_on_free_cmd
{
        uint64_t pbuf;
        int32_t resp_data_len;
        uint8_t buffer_cached;
        uint8_t aborted;
        uint8_t status;
        uint8_t delivery_status;
},
```

where:

- **pbuf** - pointer to command's data buffer or 0 for SCSI commands without data transfer.
- **resp_data_len** - length of the response data
- **buffer_cached** - true, if memory reusage is enabled for this command
- **aborted** - true, if command was aborted
- **status** - SAM status of the commands execution

- **delivery_status** - status of cmd's status/data delivery to remote initiator. Can be:
  - **SCST_CMD_DELIVERY_SUCCESS** - delivery succeeded
  - **SCST_CMD_DELIVERY_FAILED** - delivery failed

The user space handler should reply using the corresponding reply command. No error code is needed.

4.7 SCST_USER_ON_CACHED_MEM_FREE

SCST_USER_ON_CACHED_MEM_FREE subcommand is returned, when SGV cache decided that this buffer isn't needed anymore. This happens after some time of inactivity or when the system is under memory pressure.

Payload contains struct scst_user_on_cached_mem_free, which is defined as the following:

```
struct scst_user_scsi_cmd_alloc_mem
{
        uint64_t pbuf;
},
```

where:

- **pbuf** - pointer to buffer, which should be freed

4.8 SCST_USER_TASK_MGMT_RECEIVED

SCST_USER_TASK_MGMT_RECEIVED subcommand notifies that a task management function has been received. Payload contains struct scst_user_tm, which is defined as the following:

```
struct scst_user_tm
{
        uint64_t sess_h;
        uint32_t fn;
        uint32_t cmd_h_to_abort;
        uint32_t cmd_sn;
        uint8_t cmd_sn_set;
},
```

where:

- **sess_h** - corresponding session handler
- **fn** - task management function, see below
- **cmd_h_to_abort** - handle of command to abort. Valid only if fn is SCST_ABORT_TASK
- **cmd_sn** - if cmd_sn_set is set, contains maximum commands SN, which this task management function affects. See iSCSI RFC 3720 10.5.1 for more details.

- **cmd_sn_set** - specifies if cmd_sn is valid

On this notification dev handler should do the best to ensure that all aborted by this TM command SCSI commands complete ASAP.

Possible values of *fn* field:

- **SCST_ABORT_TASK** - cmd_h_to_abort shall be aborted
- **SCST_ABORT_TASK_SET** - task set on the device shall be aborted
- **SCST_CLEAR_ACA** - ACA status shall be cleared
- **SCST_CLEAR_TASK_SET** - task set on the device shall be cleared
- **SCST_LUN_RESET**, **SCST_TARGET_RESET** - reset of the device shall be done
- **SCST_NEXUS_LOSS_SESS** - notifies about nexus loss event for the session
- **SCST_ABORT_ALL_TASKS_SESS** - all tasks in the session shall be aborted
- **SCST_NEXUS_LOSS** - notifies about global nexus loss event
- **SCST_ABORT_ALL_TASKS** - all tasks shall be aborted

The "result" field of the corresponding reply command is ignored for this subcommand.

4.9 SCST_USER_TASK_MGMT_DONE

SCST_USER_TASK_MGMT_DONE subcommand notifies that all aborted by task management function commands have finished, so the dev handler can perform actual actions required by this TM command. For instance, reset all MODE PAGES variables to default values.

Payload contains struct scst_user_tm, which was defined above.

After the TM function is completed, the device handler shall reply using "result" field of the corresponding reply command.

Possible return values are:

- **SCST_MGMT_STATUS_SUCCESS** - success
- **SCST_MGMT_STATUS_TASK_NOT_EXIST** - task does not exist
- **SCST_MGMT_STATUS_LUN_NOT_EXIST** - LUN does not exist
- **SCST_MGMT_STATUS_FN_NOT_SUPPORTED** - task management function not supported
- **SCST_MGMT_STATUS_REJECTED** - task management function was rejected
- **SCST_MGMT_STATUS_FAILED** - task management function failed

5 Commands processing flow example.

As the example consider a simple synchronous VTL, which serves one virtual SCSI tape device and can process only one command at time from any initiator.

- At the beginning the VTL opens using **open()** call /dev/scst_user/ in the default blocking mode.
- Then it using **SCST_USER_REGISTER_DEVICE** ioctl() function registers the tape device. Since only one command at time is supported, the allocated command's data memory could be reused for both READ-type (i.e. which involve data transfer from target to initiator) and WRITE-type (i.e. which involve data transfer from initiator to target) commands. So the device is configured with **parse_type** *SCST_USER_PARSE_STANDARD*, **on_free_cmd_type** *SCST_USER_ON_FREE_CMD_IGNORE*, **memory_reuse_type** *SCST_USER_MEM_REUSE_ALL* and **partial_transfers_type** *SCST_USER_PARTIAL_TRANSFERS_NOT_SUPPORTED*.
- Then it prepares struct **scst_user_get_cmd** with reply set to *0*, calls **SCST_USER_REPLY_AND_GET_CMD** ioctl() and waits until some initiator connects to its tape device. On that event the VTL receives **SCST_USER_ATTACH_SESS** subcommand. Since the VTL doesn't use any initiator specific data, it can do nothing on that subcommand, so it prepares scst_user_reply_cmd structure, where:
  - **cmd_h** set to returned by SCST_USER_REPLY_AND_GET_CMD ioctl() cmd_h
  - **subcode** set to *SCST_USER_ATTACH_SESS*
  - **result** set to *0*
- Then it prepares **struct scst_user_get_cmd** with reply set to the prepared scst_user_reply_cmd structure, calls **SCST_USER_REPLY_AND_GET_CMD** ioctl() and waits for some SCSI command arrives from the initiator.
- If the received SCSI command is READ-type one, SCST does the necessary preparations, then the VTL receives **SCST_USER_EXEC** subcommand, where **bufflen** and **data_len** fields set correctly, but memory for buffer isn't allocated, so **pbuf** field is *0*. The VTL then allocates the data buffer with size **alloc_len**, e.g. using malloc(). Then the VTL reads the data from disk in it, e.g. using O_DIRECT read() function, then prepares scst_user_reply_cmd structure, where:
  - **cmd_h** set to returned by SCST_USER_REPLY_AND_GET_CMD ioctl() cmd_h
  - **subcode** set to *SCST_USER_EXEC*
  - **exec_reply.resp_data_len** set to length of the read data
  - **exec_reply.pbuf** set to the data buffer, where the data were read
  - **exec_reply.reply_type** set to *SCST_EXEC_REPLY_COMPLETED*
  - **exec_reply.status** set to the SAM defined status of the operation
  - **exec_reply.sense_len** set and **exec_reply.psense_buffer** filled with sense data, if necessary
- Then it prepares **struct scst_user_get_cmd** with reply set to the prepared scst_user_reply_cmd structure, calls **SCST_USER_REPLY_AND_GET_CMD** ioctl() and waits for the next SCSI command arrives from the initiator.

- That's all for this SCSI command. For the next command the used data buffer will be reused.

For WRITE-type SCSI commands the processing is the same, but **SCST_USER_ALLOC_MEM** will be returned before **SCST_USER_EXEC**, since the data transfer from the initiator precedes the commands execution.

In case, if the first command requires 4K data buffer, but the second one - 1M, for it the VTL also will be asked to allocate the buffer. Then, if no more 4K commands come for some time, for it **SCST_USER_ON_CACHED_MEM_FREE** subcommand will be returned to the VTL in order to ask it to free that buffer.