NAME

ibv_alloc_dm, ibv_free_dm, ibv_memcpy_to/from_dm − allocate or free a device memory buffer (DMs) and perform memory copy to or from it

SYNOPSIS

#include <infiniband/verbs.h>

struct ibv_dm *ibv_alloc_dm(struct ibv_context **context*, **struct ibv_alloc_dm_attr** **attr*);

int ibv_free_dm(struct ibv_dm **dm*);

DESCRIPTION

ibv_alloc_dm() allocates a device memory buffer for the RDMA device context *context*. The argument *attr* is a pointer to an ibv_alloc_dm_attr struct, as defined in <infiniband/verbs.h>.

ibv_free_dm() free the device memory buffer *dm*.

```
struct ibv_alloc_dm_attr {
        size_t length;          /* Length of desired device memory buffer */
        uint32_t log_align_req; /* Log base 2 of address alignment requirement */
        uint32_t comp_mask;     /* Compatibility mask that defines which of the following variable
};
```

Address alignment may be required in cases where RDMA atomic operations will be performed using the device memory

In such cases, the user may specify the device memory start address alignment using the log_align_req parameter

in the allocation attributes struct.

Accessing an allocated device memory

In order to perform a write/read memory access to an allocated device memory, a user could use the ibv_memcpy_to_dm and ibv_memcpy_from_dm calls respectively.

int ibv_memcpy_to_dm(struct ibv_dm **dm*, **uint64_t** *dm_offset*, **void** **host_addr*, **size_t** *length*);

int ibv_memcpy_from_dm(void **host_addr*, **struct ibv_dm** **dm*, **uint64_t** *dm_offset*, **size_t** *length*);

dm_offest
is the byte offset from the beginning of the allocated device memory buffer to access.

host_addr
is the host memory buffer address to access.

length
is the copy length in bytes.

Device memory registration

User may register the allocated device memory as a memory region and use the lkey/rkey inside sge when posting receive or sending work request. This type of MR is defined as zero based and therefore any reference to it (specifically in sge) is done with a byte offset from the beginning of the region.

This type of registration is done using ibv_reg_dm_mr.

int ibv_reg_dm_mr(struct ibv_pd **pd*, **struct ibv_dm** **dm*, **uint64_t** *dm_offset*, **size_t** *length*, **uint32_t** *access*);

pd
the associated pd for this registration.

dm
the associated dm for this registartion.

dm_offest
is the byte offset from the beginning of the allocated device memory buffer to register.

length
the memory length to register.

access
mr access flags (Use enum ibv_access_flags). For this type of registration, user must set the IBV_ACCESS_ZERO_BASE flag.

RETURN VALUE

ibv_alloc_dm() returns a pointer to an ibv_dm struct or NULL if the request fails.

ibv_free_dm() returns 0 on success, or the value of errno on failure (which indicates the failure reason).

ibv_reg_dm_mr() returns a pointer to an ibv_mr struct on success or NULL if request fails.

ibv_memcpy_to_dm()/ibv_memcpy_from_dm() returns 0 on success or the failure reason value on failure.

NOTES

ibv_alloc_dm() may fail if device has no more free device memory left, where the maximum amount of allocated memory is provided by the *max_dm_size* attribute in *ibv_device_attr_ex* struct. **ibv_free_dm()** may fail if any other resources (such as an MR) is still associated with the DM being freed.

SEE ALSO

ibv_query_device_ex(3),

AUTHORS

Ariel Levkovich <lariel@mellanox.com>