O2iblnd to userland

2007/07/20

1 Introduction

To run o2iblnd to userspace , we need to:

- Emcompasses all OFA verbs, uses userland-verbs for u-o2iblnd
- maintain a global MR to map all known memory (always map on demand will be too slow)
- Uses libcfs APIs to replace all kernel lock/schedule APIs
- Build system fix for userland
- Misc fix (initialize and finalize etc)

We will not discuss libcfs APIs and build system in this document, because they are simular to any other porting to userland.

2 Function specification

2.1 CM APIs

CM APIs used in o2iblnd

Kernel CM APIs	User CM APIs	Description
rdma_create_id	rdma_create_id	no callback in userspace, need to call rdma_get_cm_event to retreive communication event
rdma_destroy_id	rdma_destroy_id	
N/A	rdma_create_event_channel	create communication channel.
N/A	rdma_destroy_event_channel	destroy communication chane
N/A	rdma_get_cm_event	retreive pending communication event
N/A	rdma_ack_cm_event	acknowledgs, all events that are reported must be acknowledged
rdma_connect / rdma_disconnect	rdma_connect / rdma_disconnect	
rdma_bind_addr	rdma_bind_addr	
rdma_resolve_addr	rdma_resolve_addr	
rdma_resolve_route		
rdma_create_qp / rdma_destroy_qp	rdma_create_qp / rdma_destroy_qp	
rdma_listen / rdma_accept / rdma_reject	rdma_listen / rdma_accept / rdma_reject	

2.2 Verbs

Verbs used in o2iblnd

Kernel verbs	User verbs	Description
ib_alloc_pd	ibv_alloc_pd	different parameter type
ib_dealloc_pd	ibv_dealloc_pd	
ib_create_cq	ibv_create_cq	no callback in u-verbs, need to wait for completion event on event channel
ib_destroy_cq	ibv_destory_cq	
ib_req_notify_cq	ibv_req_notify_cq	
ib_poll_cq	ibv_poll_cq	
N/A	ibv_get_cq_event	wait for the next completion event in the completion event channel
N/A	ibv_ack_cq_event	acknowledge, all completion events which are returned by ibv_get_cq_event() must be acknowledged
N/A	ibv_create_comp_channel	Create a completion event channel
N/A	ibv_destroy_comp_channel	Destroy a completion event channel
ib_get_dma_mr	N/A	return a memory region for system memory that is usable for DMA
ib_dereg_mr	N/A	deregister a memory region and removes it from the HCA translation table
N/A	ibv_reg_mr	register memory region
N/A	ibv_dereg_mr	deregister memory region
ib_create_fmr_pool	N/A	
ib_fmr_pool_map_phys	N/A	
ib_fmr_pool_unmap	N/A	
ib_dma_map_single / ib_dma_unmap_single	N/A	
ib_dma_map_sg / ib_dma_unmap_sg	N/A	
ib_sg_dma_addrees / ib_sg_dma_len	N/A	
ib_post_send / ib_post_recv	ibv_post_send / ibv_post_recv	

2.3 Memory region

The verbs API to send and receive data does not specify memory addresses directly. Instead, a memory region is constructed and a Lkey or Rkey is used to refer to the region

2.3.1 Kernel space memory regions

kernel memory region is created by ib_get_dma_mr(), it returns a pointer to struct ib_mr which contains the 'lkey' and 'rkey' fields. The memory region represents all of physical memory so no base address or length is needed when creating it. The addresses used for the 'addr' field of struct ib_sge need to be hardware device addresses suitable for DMA access by RDMA devices. Since this mapping may be device specific, there are a set kernel verbs functions corresponding to the DMA mapping functions.

```
ib_dma_map_single();
......
ib_post_send();
......
ib_dma_unmap_single();
```

2.3.2 User space memory regions

User space memory regions are created by calling ibv_reg_mr(), It returns a pointer to a struct ibv_mr which contains the 'lkey' field and 'rkey' field, simular to kernel memory region. The lkey should be copied into the 'lkey' field of struct ibv_sge when posting buffers with ibv_post_send(), ibv_post_recv(), The address space in ibv_sge(from ibv_sge->addr to ibv_sge->addr+ibv_sge->length) should be between the address and address + length passed to ibv_reg_mr(). A memory region is destroyed by calling ibv_dereg_mr().

```
buf = malloc(size);
mr = ibv_reg_mr(pd, buf, size...);
......
ibv_post_recv();
......
ibv_dereg_mr(mr);
```

2.3.3 Growing memory region

We can't just map on demand, it will be too slow. We will need to map all known memory and extend the memory region if we're ever asked to do RDMA on memory that we've not mapped yet.

We keep a global MR object, if the new required memory region is covered by the MR object, we increase the reference count the MR object, otherwise we create a new MR object to replace the old one, the new MR object is extension of the orginal MR object. the old MR object is destroyed when reference count is zero.

2.4 CM thread

rdma_create_id() can't take callback

```
kiblnd_data.cm_chanel = rdma_create_event_channel();:
rc = rdma_create_id(kiblnd_data.cm_chanel, &id, peer, RDMA_PS_TCP);
void *o2ib_cm_thread(void *arg)
{
    ......
    while (1) {
        rdma_get_cm_event(kiblnd_data.cm_chanel, &event);
        kiblnd_cm_callback(event->id, event);
        rdma_ack_cm_event(event);
    }
}
```

2.5 CQ thread

ibv_create_cq() can't take callback, we need to create a global completion event channel and create a thread to get completion event, all CQs share the completion event channel(creating of CQ will increase reference count of comp_channel).

```
kiblnd_data.kib_cq_channel = ibv_create_comp_channel(cm_id->verbs);
cq = ibv_create_cq(cm_id->verbs, ... kiblnd_data.kib_cq_channel...);
void *o2ib_cq_thread(void *arg)
{
    ...
    while (1) {
        ibv_get_cq_event(kiblnd_data.cq_channel, &ev_cq, &ctx);
        ibv_req_notify_cq(cq, 0);
        kiblnd_cq_completion(cq, ctx);
    }
}
```

2.6 Inter-operation with kernel o2iblnd

RDMA descriptors of userspace peer and kernel space peer are different, userspace RDMA descriptors may be in fewer and large fragments and kernel RDMA descriptor have up to 256 page fragments.

kiblnd_init_rdma() generates a set of RDMA fragments that are compatible with both destination and source kib_rdma_desc_t . For example, RDMA descriptor of user peer has one 1 fragment, size is 16K, RDMA descriptor of kernel peer has 4 frament, size is 4K, then we will setup 4*4K RDMA fragments for both sides.

```
while (resid > 0) {
    ......
    wrknob = MIN(MIN(srcfrag->rf_nob, dstfrag->rf_nob), resid);
    ......
    sge->addr   = srcfrag->rf_addr;
    sge->length = wrknob;
    ......
    wrq->wr.rdma.remote_addr = dstfrag->rf_addr;
    ......
    resid -= wrknob;
    if (wrknob < srcfrag->rf_nob) {
        srcfrag->rf_nob  -= wrknob;
        srcfrag->rf_addr += wrknob;
    } else {
        srcfrag++;
        srcidx++;
    }
    if (wrknob < dstfrag->rf_nob) {
        dstfrag->rf_nob  -= wrknob;
        dstfrag->rf_addr += wrknob;
    } else {
        dstfrag++;
        dstidx++;
    }
}
```

3 Code structure

A new file will be created (o2iblnd_verbs.c or o2iblnd_lib.c), which will encompass verbs and APIs for both user space and kernel space.

3.0.1 int kiblnd_create_dev(kib_dev_t **devpp)

Create kib_dev. Allocate a communication identifier, allocate protection domains, and get mr(kernel only)

3.0.2 void kiblnd_destroy_dev(kib_dev_t *dev)

deregister mr(kernel only), release protection domain, destroy cmid

3.0.3 __u64 kiblnd_msg_map(struct ib_device *dev, void *msg, size_t size, enum dma_data_direction direction)

map msg to DMA address. Call ib_dma_map_single in kernel, call ibv_reg_mr in user space.

3.0.4 void kiblnd_msg_unmap(struct ib_device *dev, __u64 addr, size_t size, enum dma_data_direction direction)

unmap the mapping, call ib_dma_unmap_single in kernel, call ibv_unreg_mr in user space.

3.0.5 int kiblnd_create_cq(struct rdma_cm_id *cm_id, void *ctxt, int cqe, struct ib_cq **cqpp)

Create completion queue, pass in callback in kernel, pass in completion channel in user space

3.0.6 void kiblnd_destroy_cq(struct ib_cq *cq);

destroy completion queue

3.0.7 int kiblnd_rd_setup (lnet_ni_t *ni, kib_tx_t *tx, kib_rdma_desc_t *rd, unsigned int niov, struct iovec *iov, int offset, int nob)

Setup rdma descriptor, call ib_dma_map_sg() in kernel, call ibv_reg_mr() in user space (if there is fragment not covered by existed MR object, call ibv_reg_mr() to create a new MR object, which is extension of the old MR object. If all fragments are covered by exited MR object, increase reference count of MR object)

3.0.8 void kiblnd_rd_clean(lnet_ni_t *ni, kib_tx_t *tx)

Destroy rdma descriptor, call ib_dma_unmap_sg() in kernel, call ibv_dereg_mr() in user space (decrease reference count of MR object, if refcount is ZERO, call ibv_dereg_mr() to destroy mapping)

3.0.9 others

encompassing of other verbs and APIs is straightway, we will not discuss here.

4 Conclusion

To port o2iblnd to userland, we need to encompass OFA verbs, add CM thread and CQ thread, maintain global MR object, and add some code for startup/cleanup.

Estimation: (NEW: 300 LOC, Change 400 LOC, Total 700 LOC)