Kernel OpenFabrics Interface

Connection finalization

Stan Smith Intel SSG/DPD

March, 2015

Steps

- The Big Picture
  - Initialization
  - Server connection setup
  - Client connection setup
  - Connection Finalization*
  - Data transfer
  - Shutdown
- Current State
  - Server waiting for connection request via fi_eq_sread()
  - Client has sent connection request via fi_connect()
- Client connection finalization:
  - fi_eq_sread(eq) check for connection completion
- Server connection finalization:
  - Server creates a new endpoint based on connection request info.
  - Rx buffer memory registered
  - Receive posted
  - fi_accept()s connection
  - fi_eq_sread(eq) …RTU

Memory registration

Register KVA memory ‘regions’ with permissions granted for access by fabric resources. A memory region must be registered with a domain before it can be referenced locally or as the target of a remote RMA or atomic data transfer.

int fi_mr_reg(struct fid_domain *domain, const void *buf, size_t len, uint64_t access,
uint64_t offset, uint64_t requested_key, uint64_t flags, struct fid_mr **mr, void *ctx)

ARGUMENTS

domain : Resource domain

buf : Memory buffer to register with the fabric hardware

len : Length, in bytes, of memory buffer to register

access : Memory access permissions associated with registration

offset : Optional specified offset for accessing specified registered

requested_key : Optional requested remote key associated with registered buffers.

flags : Additional flags to apply to the operation.

**mr : Memory region pointer

ctx : User specified context associated with the memory region.

Connected addresses

fi_getname / fi_getpeer

The fi_getname and fi_getpeer calls may be used to retrieve the local or peer endpoint address, respectively. On input, the addrlen parameter should indicate the size of the addr buffer. If the actual address is larger than what can fit into the buffer, it will be truncated. On output, addrlen is set to the size of the buffer needed to store the address, which may be larger than the input value.

```
int fi_getname(fid_t fid, void *addr, size_t *addrlen);

int fi_getpeer(struct fid_ep *ep, void *addr, size_t *addrlen);
```

Infiniband user context example

```
typedef struct {
        struct fi_context       context;
        struct fi_info          *prov;
        struct fid_fabric       *fabric;        // set during initialization steps
        struct fid_domain       *domain;        // Active: set during initialization
        struct fid_ep           *ep;
        struct fid_pep          *pep;
        struct fid_eq           *eq;
        struct fid_cq           *send_cq;
        struct fid_cq           *recv_cq;
        struct fid_mr           *mr;
        char                    *buf;
} application_context_t;

// ctx.* is an application defined struct which contains kOFI object pointers.

application_context_t       ctx = { 0 };
```

Infiniband Server RC example

```
struct fi_info                  *prov;
uint32_t                        event_id;
#define EVENT_SIZE sizeof(struct fi_eq_cm_entry) + 128
char                            ebuf[EVENT_SIZE];
struct fi_eq_cm_entry       *event = sizeof(struct fi_eq_cm_entry) ebuf;
                                    // fi_info.ep_attr->protocol = FI_PROTO_RDMA_CM_IB_RC;

ret = fi_listen(ctx.pep);     // previously issued.
ret = fi_eq_sread(eq, &event_id, (void*)&event, sizeof(event), TIMEOUT, FI_TIME_MS);
        // error handling

// blocked waiting for a client connection request.
if (event_id != FI_CONNREQ) {
                print_err("unexpected event %d\n", event);
                rc = -FI_EOTHER;  goto errout;       // connection reject
}
prov = event.info;     // save fi_info which describes IF which the conn request appeared.
```

Infiniband Server RC example II

```
// Create a new endpoint based on connection request info; same fabric as listen.
ret = fi_domain(ctx->fabric, prov, &ctx->domain, NULL);
        // error handling – reject connection

// Create a communications End Point - set QP Work Request depth
prov->ep_attr->tx_ctx_cnt = (size_t) post_depth;
prov->ep_attr->rx_ctx_cnt = (size_t) post_depth;

/* set ScatterGather max depth */
prov->tx_attr->iov_limit = 1;
prov->rx_attr->iov_limit = 1;
prov->tx_attr->inject_size = 0; // no INLINE support

ret = fi_endpoint(ctx->domain, prov, &ctx->ep, app_context);
        // error handling – connection reject
```

Infiniband Server RC example III

```
// Create Send & Receive Completion Queues

// set event format: [context, msg(ctx, flags, len), data(ctx, flags, len, buf, data)]
cq_attr.format      = FI_CQ_FORMAT_MSG;
cq_attr.wait_obj    = FI_WAIT_NONE;
cq_attr.wait_cond   = FI_CQ_COND_NONE;
cq_attr.flags       = FI_SEND;

ret = fi_cq_open(ctx->domain, &cq_attr, &ctx->send_cq, NULL);    // Send CQ
  // error handling

cq_attr.flags       = FI_RECV;
ret = fi_cq_open(ctx->domain, &cq_attr, &ctx->recv_cq, NULL);    // Recv CQ
      // error handling

ret = fi_bind(&ctx->ep->fid, &ctx->send_cq->fid, FI_SEND);
      // error handling
```

Infiniband Server RC example IV

```
ret = fi_bind(&ctx->ep->fid, &ctx->recv_cq->fid, FI_RECV);
  // error handling

ret = fi_enable(ctx->ep);  //transition endpoint to the ‘active’ state.
  // error handling

// register receive buffer
ret = fi_mr_reg(ctx->domain, ctx->buf, ctx->buf_len, 0, 0, 0, 0, &ctx->mr, NULL);
  // error handling

// post a receive prior to accept
ret = fi_recv(ctx->ep, ctx->msg, msg_len, fi_mr_desc(ctx->mr), ctx->msg);
 // error handling
posted = TRUE;

ret = fi_accept(ctx->ep, NULL, 0);
 // error handling
```

Infiniband Server RC example V

```
// Wait for Client to complete connection.
uint32_t                        event_id;
struct fi_eq_cm_entry       *event;// fi_info.ep_attr->protocol = FI_PROTO_RDMA_CM_IB_RC;

n = fi_eq_sread(ctx->eq, &event_id, event, EVENT_SIZE, TIMEOUT, 0);
if (n != sizeof entry) {
        print_err("fi_eq_sread %d\n", (int) n);
        return (int) n;
}
if (event_id != FI_CONNECTED) {
        print_err("unexpected fi_accept() connection event %d\n", event_id);
        return -FI_EOTHER;
}
// event.fid->context == fi_ep_open(app_context); for multi-connection support
if (event->fid != &ctx->ep->fid) {       // expected fid?
        print_err("fid %p != %p\n", event->fid, &ctx->ep->fid);
        return -FI_EOTHER;
}
```

Infiniband Server RC example VI

```
/* Previously defined:
#define EVENT_SIZE sizeof(struct fi_eq_cm_entry) + 128
char                        ebuf[EVENT_SIZE];
struct fi_eq_cm_entry  *event = sizeof(struct fi_eq_cm_entry) ebuf;
 */

/* validate private data */
If ( strncmp(event->data, “Private Data”, PD_SIZE) )  {
      /* Private Data check failed – reject connection.
}

// ctx->ep is ‘now’ fully IB/RC connected.
```

Infiniband Server RC example VII

```
// Error handling: rejection, rc == error code
char reject_reason[64];
errout:
    len = snprintf(reject_reason,”Rejected(err %d) ‘%s’”,rc,fi_errstring(rc));
    fi_reject(ctx->pep, info->connreq, reject_reason, len);
    if (posted)
        fi_cancel(ctx->ep, ctx->msg);
    fi_close(ctx->scq);
    fi_close(ctx->rcq);
    fi_close(ctx->ep);
    fi_close(ctx->mr);
    fi_close(ctx->domain);
    free_info(info);
    return rc;
```

Infiniband RC Client

- Current State:
  - Server is waiting for a connection request.
  - Client has posted a connection request via fi_connect()

- Client connection finalization:
  - fi_eq_sread() check for connection completion

Infiniband Client RC example

```
struct fi_info              *info;
struct fi_eq_cm_entry       event;
size_t                      n;
struct sockaddr_in          addr = { 0 };

addr.sin_family = AF_INET;
addr.sin_port = htons(SVR_PORT);
ret = in4_pton(svr_ipaddr, strlen(svr_ipaddr),
                    (u8 *)&addr.sin_addr.s_addr, '\0', NULL);

ret = fi_connect( ctx.ep, (void*)addr, (void*)private_data, sizeof(*private_data) );
  // Error handling

 // do something useful or fall into waiting for the server to fi_accept()
n = fi_eq_sread(ctx.eq, &event_id, (void*)&event, sizeof(event), TIMEOUT, 0);
```

Infiniband Client RC example II

```
// Error handling
if (n < sizeof event) {
        struct fi_eq_err_entry eqe;
        print_err("fi_eq_sread '%s'(%d)\n", fi_strerror(n), (int) n);
        rc = fi_eq_readerr(ctx->eq, &eqe, sizeof(eqe), 0);
        if (rc)
                print_err("fi_eq_readerr() returns %d '%s'\n",rc, fi_strerror(rc));
        else {
                char buf[64];
                print_err("fi_eq_readerr() prov_err '%s'(%d)\n",
                        fi_eq_strerror(ctx->eq, eqe.prov_errno, eqe.err_data, buf, sizeof(buf)),
                        eqe.prov_errno);
                print_err("fi_eq_readerr() err '%s'(%d)\n", fi_strerror(eqe.err), eqe.err);
        }
        rc = (int) n;
        goto errout;
}
```

Infiniband Client RC example III

```
if (event_id != FI_CONNECTED) {
        print_err("unexpected event %d\n", event);
        rc = -FI_EOTHER;
        goto errout;
}

// same context specified in ep creation: fi_endpoint(…, application_context).
if (entry.fid->context != application_context) {
        print_err("entry.fid->context %lx != %lx\n",
                        (ulong)entry.fid->context, (ulong)CONTEXT);
}

printk(KERN_INFO, "*** Client Connected\n");
printk(KERN_INFO,, Client private data(len %d): '%s'\n",(n - sizeof entry), entry.data);

// ctx->ep is now fully RC connected – pass the data please…..
```

OPENFABRICS ALLIANCE

Next step
Data transfer