OPENFABRICS ALLIANCE

Kernel OpenFabrics Interface

Server End point setup

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
  - Completed: fi_getinfo(), fi_fabric(), fi_domain()

- Server connection setup:
  - fi_passive_ep() *pep – create a passive endpoint.
  - fi_eq_open() *event queue – where connection requests appear.
  - fi_pep_bind() bind EQ to passive end-point.
  - fi_listen(pep) post a listen for the passive endpoint.
  - fi_eq_sread(eq) blocking wait for connection request(s).

fi_passive_ep()

Create a passive endpoint. Passive endpoints belong to a fabric domain and are used to listen for incoming connection requests.

```
struct fid_pep     *pep;

int  fi_passive_ep( struct fid_fabric *fabric, struct fi_info *info,
                    struct fid_pep **pep, void *context );
```

Arguments:

- *fabric – fabric instance.
- *info – application selected fi_info record.
- **pep (out) passive endpoint pointer.
- *context – domain event context.

Return:

- rc == 0 defines success.

fi_eq_open()

Create an event queue used in waiting for and accepting connection requests.

int fi_eq_open(struct fid_fabric *fabric, struct fi_eq_attr *attr,
struct fid_eq **eq, void *context)

Arguments:

*fabric – opened fabric instance

*attr – EQ attributes

**eq – output event queue ptr.

*context – application supplied value, returned in EQ events.

Return:

0 == success.

struct fi_eq_attr

```
struct fi_eq_attr {
        size_t              size;       # min EQ depth
        uint64_t            flags;      # operational flags: FI_WRITE, can write to EQ
        enum fi_wait_obj    wait_obj;       # wait object type:
                                            FI_WAIT_NONE async
                                            FI_WAIT_UNSPEC(blocking)
                                            FI_WAIT_SET, use existing wait_set.
        int                 signaling_vector;  # core where CQ interrupts are directed.
        struct fid_wait     *wait_set;      # existing wait_set pointer;
                                            # requires FI_WAIT_SET
}
```

fi_pep_bind()

Associate a passive endpoint with an event queue

int fi_pep_bind(struct fid_pep *pep, struct fid *fid, uint64_t flags)

Arguments:

*pep – existing passive endpoint

*fid – EQ->fid to bind to the passive endpoint.

flags – operational flags.

Return

0 == success

fi_eq_sread()

Blocking read of one event from event queue.

int fi_eq_sread(struct fid_eq *eq, uint32_t *event, void *buffer, size_t len,
int timeout, uint64_t flags)

Arguments:

*eq – opened Event Queue

*event – reported event ID.

*buffer – event format defined via fi_info->ep_attr.protocol

len – sizeof(eq event).

timeout – default units (milliseconds) or flags defines units.

flags – operational flags:

FI_PEEK – read without consuming the event.

FI_TIME_US – micro-seconds

FI_TIME_MS – milli-seconds

Return: number of bytes written to buffer; at most 1 event.

Infiniband Server RC example

```
typedef struct {
        struct fi_context       context;
        struct fi_info          *prov;
        struct fid_fabric       *fabric;        // set during initialization steps
        struct fid_domain       *domain;
        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;

application_context_t       ctx = { 0 };
```

Infiniband Server RC example II

```
struct fi_eq_attr        eq_attr = {10, 0, FI_WAIT_UNSPEC, 0, NULL };
uint32_t                 event_id;
struct fi_eq_cm_entry    event;     // format derived from fi_info. ep_attr->protocol =
                                    // FI_PROTO_RDMA_CM_IB_RC;

ret = fi_passive_ep(ctx.fabric, ctx.prov, &ctx.pep, 0xCafeBabe );

ret = fi_eq_open(ctx.fabric, &eq_attr, &ctx.eq, 0);

ret = fi_pep_bind( ctx.pep->fid, ctx.eq->fid, 0);

ret = fi_listen(ctx.pep)

ret = fi_eq_sread(ctx.eq, &event_id, (void*)&event, sizeof(event), 20000, FI_TIME_MS);

    // blocked waiting for a client connection.
```

Thank you.