14th ANNUAL WORKSHOP 2018

MULTI-PROCESS SHARING OF RDMA RESOURCES

Alex Rosenbaum
Mellanox Technologies
April 2018
Why Multi-Process RDMA access?

Multi-Thread can do just as good!

REALY?

or is there anything missing?
AGENDA

- Motivation
- Different Solutions
- Examples and use cases
- Problems and limitations
MOTIVATION

- **Existing fork() based applications and frameworks**
  - Replace socket() and need to be on par
  - NGINX, Big Data, Hadoop

- **TELCO Grade Resiliency**
  - Allow design in a high availability based requirement

- **Application update without breaking connections**

- **Treat Processes as you would Thread**

- **Extended debuggability**
  - Attach to existing process and read values
ALTERNATIVE SOLUTIONS

- Shared IB Object
- fork()
- Shared Memory
SHARED IB OBJECT SOLUTION
SHARED IB OBJECT SOLUTION

- **High Level:**
  - Each process holds a reference to the same Kernel/HW object value
  - Sharing the same ib_obj through different ib_uobj and different ib_ucontext

- **Design:**
  - Open RDMA resource from user space
  - Create a Share FD or use the existing ibv_context fd
  - Associate IB object with Shared FD
  - Pass Shared FD to other process
  - Other process to open shared resource based on shared FD
  - Kernel to track resource open from all processes

- **Application has to:**
  - Pass the Shared FD between processes
  - Pass the shared objects’ handles between processes to be opened
**Example:**

- Primary processes allocate and register huge memory blocks.
- Each secondary process opens the MR’s as with Shared FD into their own PD.
- Each secondary process does RDMA operation on segments of memory which is shared and mapped once.
- Single LKey will result in higher performance.
Good for stateless objects: PD, MR, XRC
- But how do we transfer state full objects: QC, CQ, cmd_id's
FORK SOLUTION
### High Level:

- All IB/RDMA resources are created as shared fd/memory/locks.
- On fork(), all shared objects are exposed also to child.
- Kernel holds single ib_ucontext and ib_uobj instances for both processes.
- User space, parent & child, share only resources created in it’s history.
- User needs to sync processes just as it has to sync threads:
  - Pass and sync ibv/cma handles to resources each processes should handle.
  - Protect critical sections from multi-thread access, or from destroy races.
FORK() BASED SOLUTION

- Created ‘SHARED’ `ibv_context`
- All created IBV objects are allocated as shared
- Upon fork all are shared

```c
struct ibv_context *ibv_open_device_ex(
    struct ibv_device *device,
    struct ibv_context_attr *attr);

struct ibv_context_attr {    
    uint32_t flags;
};

enum ibv_context_flags {    
    IBV_CONTEXT_FLAGS_SHARED = 1 << 0
};
```
FORK() EXAMPLE

- RDMA server with fork()-ed children processes handling the traffic request/response

server_main() {
    server_create_device_shared(); /* with IBV_CONTEXT_FLAGS_SHARED */
    rdma_listen(listen_id, 0);

    while (!exit) {
        /* wait for RDMA CM new connection requests */
        rdma_get_request(listen_id, &id);

        /* create QP + CQ */
        server_create_resources(id);

        /* accept connection */
        rdma_accept(id, NULL);

        if (fork())
            continue; /* server process */
        else
            server_connection_processing(); /* child process */
    }
}
**FORK() SOLUTION LIMITATION**

- **Single Binary**
  - Upgrade/Replace of binary is impossible
  - Need to replace entire processes and release all RDMA resources

- **Adding object which is not shared to an already shared object**
  - New ‘private’ QP with shared CQ: old child will not recognize new qp_num

- **Atomicity of parent/child crash doesn’t guaranty RDMA resource usability**
SHARED MEMORY SOLUTION
**High Level Design:**
- Application to manage the shared memory
  - rdma-core allocates resource with application callback: `shared_malloc()`
- Application to allow additional processes to attach to same virtual address offset in the shared memory
- All processes modify the same shared memory DB and access the same HW mapped resource

**Allows Application logic / binary to be updated**
- Compared to `fork()` in which we have single binary
- Must keep rdma-core identical
APPLICATION SHARED MEMORY EXAMPLE

- **RDMA Server: On RDMA_CM new connection requests**
  - Create RC QP + CQ
  - Accept + Handle connection in thread

- **Launch upgrade process which attached to shared memory and takes ownership over connection and resource until ‘old’ processes can exit**
APPLICATION SHARED MEMORY SOLUTION

- Each new `ibv_context` will request application to allocate Shared Memory
- Application manages attach to shared block
- Application uses IPC to pass `ibv_obj` pointers between threads in different processes

```c
struct ibv_context *ibv_open_device_ex(
    struct ibv_device *device,
    struct ibv_context_attr *attr);

struct ibv_context_attr {
    uint32_t flags;
    struct ibv_context_memallocators *memallocators;
};

struct ibv_context_memallocators {
    void *(*alloc)(size_t size, void *priv_data);
    void (*free)(void *ptr, void *priv_data);
    void *priv_data;
};
```
APPLICATION SHARED MEMORY LIMITATION

- **Align virtual Address:**
  - Requires Disabling Address-Space Layout Randomization (ASLR) (vs fork())

- **Guarantying rdma-core binary compatibility**
  - Change in data struct will break

- **Atomicity of processes actions during process crash doesn’t guaranty RDMA resource usability**
ATOMICITY PROBLEM

- How do we protect for atomicity of multi-process failures/crashes?
  - Process crashes with new WC
  - Failure in post_send

- Can ‘other’ process recover the application state and continue managing the connection?
THANK YOU
Alex Rosenbaum
Mellanox Technologies