InfiniBand Midlayer Locking

This guide is an attempt to make explicit the locking assumptions made by the InfiniBand midlayer. It describes the requirements on both low-level drivers that sit below the midlayer and upper level protocols that use the midlayer.

Sleeping and interrupt context

With the following exceptions, a low-level driver implementation of all of the methods in struct ib_device may sleep. The exceptions are any methods from the list:

- create_ah
- modify_ah
- query_ah
- destroy_ah
- post_send
- post_recv
- poll_cq
- req_notify_cq

which may not sleep and must be callable from any context.

The corresponding functions exported to upper level protocol consumers:

- rdma_create_ah
- rdma_modify_ah
- rdma_query_ah
- rdma_destroy_ah
- ib_post_send
- ib_post_recv
- ib_req_notify_cq

are therefore safe to call from any context.

In addition, the function

- ib_dispatch_event

used by low-level drivers to dispatch asynchronous events through the midlayer is also safe to call from any context.

Reentrancy

All of the methods in struct ib_device exported by a low-level driver must be fully reentrant. The low-level driver is required to perform all synchronization necessary to maintain consistency, even if multiple function calls using the same object are run simultaneously.

The IB midlayer does not perform any serialization of function calls.

Because low-level drivers are reentrant, upper level protocol consumers are not required to perform any serialization. However, some serialization may be required to get sensible results. For example, a consumer may safely call ib_poll_cq() on multiple CPUs simultaneously. However, the ordering of the work completion information between different calls of ib_poll_cq() is not defined.

Callbacks

A low-level driver must not perform a callback directly from the same callchain as an ib_device method call. For example, it is not allowed for a low-level driver to call a consumer's completion event handler directly from its post_send method. Instead, the low-level driver should defer this callback by, for example, scheduling a tasklet to perform the callback.

The low-level driver is responsible for ensuring that multiple completion event handlers for the same CQ are not called simultaneously. The driver must guarantee that only one CQ event handler for a given CQ is running at a time. In other words, the following situation is not allowed:

```
      CPU1                                    CPU2

  low-level driver ->
    consumer CQ event callback:
      /* ... */
      ib_req_notify_cq(cq, ...);
                                      low-level driver ->
```

```
        /* ... */                                  consumer CQ event callback:
                                                     /* ... */
        return from CQ event handler
```

The context in which completion event and asynchronous event callbacks run is not defined. Depending on the low-level driver, it may be process context, softirq context, or interrupt context. Upper level protocol consumers may not sleep in a callback.

Hot-plug

A low-level driver announces that a device is ready for use by consumers when it calls ib_register_device(), all initialization must be complete before this call. The device must remain usable until the driver's call to ib_unregister_device() has returned.

A low-level driver must call ib_register_device() and ib_unregister_device() from process context. It must not hold any semaphores that could cause deadlock if a consumer calls back into the driver across these calls.

An upper level protocol consumer may begin using an IB device as soon as the add method of its struct ib_client is called for that device. A consumer must finish all cleanup and free all resources relating to a device before returning from the remove method.

A consumer is permitted to sleep in its add and remove methods.