NAME

ibv_get_cq_event, ibv_ack_cq_events – get and acknowledge completion queue (CQ) events

SYNOPSIS

#include <infiniband/verbs.h>

int ibv_get_cq_event(struct ibv_comp_channel **channel*,
        struct ibv_cq ***cq*, **void** ***cq_context*);

void ibv_ack_cq_events(struct ibv_cq **cq*, **unsigned int** *nevents*);

DESCRIPTION

ibv_get_cq_event() waits for the next completion event in the completion event channel *channel*. Fills the arguments *cq* with the CQ that got the event and *cq_context* with the CQ's context.

ibv_ack_cq_events() acknowledges *nevents* events on the CQ *cq*.

RETURN VALUE

ibv_get_cq_event() returns 0 on success, and −1 on error.

ibv_ack_cq_events() returns no value.

NOTES

All completion events that **ibv_get_cq_event()** returns must be acknowledged using **ibv_ack_cq_events()**. To avoid races, destroying a CQ will wait for all completion events to be acknowledged; this guarantees a one-to-one correspondence between acks and successful gets.

Calling **ibv_ack_cq_events()** may be relatively expensive in the datapath, since it must take a mutex. Therefore it may be better to amortize this cost by keeping a count of the number of events needing acknowledgement and acking several completion events in one call to **ibv_ack_cq_events()**.

EXAMPLES

The following code example demonstrates one possible way to work with completion events. It performs the following steps:

Stage I: Preparation
1. Creates a CQ
2. Requests for notification upon a new (first) completion event

Stage II: Completion Handling Routine
3. Wait for the completion event and ack it
4. Request for notification upon the next completion event
5. Empty the CQ

Note that an extra event may be triggered without having a corresponding completion entry in the CQ. This occurs if a completion entry is added to the CQ between Step 4 and Step 5, and the CQ is then emptied (polled) in Step 5.

```
cq = ibv_create_cq(ctx, 1, ev_ctx, channel, 0);
if (!cq) {
      fprintf(stderr, "Failed to create CQ\n");
      return 1;
}

/* Request notification before any completion can be created */
if (ibv_req_notify_cq(cq, 0)) {
      fprintf(stderr, "Couldn't request CQ notification\n");
      return 1;
}
```

```
.
.
.
/* Wait for the completion event */
if (ibv_get_cq_event(channel, &ev_cq, &ev_ctx)) {
        fprintf(stderr, "Failed to get cq_event\n");
        return 1;
}

/* Ack the event */
ibv_ack_cq_events(ev_cq, 1);

/* Request notification upon the next completion event */
if (ibv_req_notify_cq(ev_cq, 0)) {
        fprintf(stderr, "Couldn't request CQ notification\n");
        return 1;
}
/* Empty the CQ: poll all of the completions from the CQ (if any exist) */
do {
        ne = ibv_poll_cq(cq, 1, &wc);
        if (ne < 0) {
                fprintf(stderr, "Failed to poll completions from the CQ\n");
                return 1;
        }

        /* there may be an extra event with no completion in the CQ */
        if (ne == 0)
                continue;

        if (wc.status != IBV_WC_SUCCESS) {
                fprintf(stderr, "Completion with status 0x%x was found\n", wc.status);
                return 1;
        }
} while (ne);
```

The following code example demonstrates one possible way to work with completion events in non-blocking mode. It performs the following steps:

1. Set the completion event channel to be non-blocked
2. Poll the channel until there it has a completion event
3. Get the completion event and ack it

```
/* change the blocking mode of the completion channel */
flags = fcntl(channel->fd, F_GETFL);
rc = fcntl(channel->fd, F_SETFL, flags | O_NONBLOCK);
if (rc < 0) {
            fprintf(stderr, "Failed to change file descriptor of completion event channel\n");
            return 1;
}


/*
 * poll the channel until it has an event and sleep ms_timeout
 * milliseconds between any iteration
 */
my_pollfd.fd      = channel->fd;
my_pollfd.events  = POLLIN;
my_pollfd.revents = 0;

do {
        rc = poll(&my_pollfd, 1, ms_timeout);
} while (rc == 0);
if (rc < 0) {
        fprintf(stderr, "poll failed\n");
        return 1;
}
ev_cq = cq;

/* Wait for the completion event */
if (ibv_get_cq_event(channel, &ev_cq, &ev_ctx)) {
    fprintf(stderr, "Failed to get cq_event\n");
    return 1;
}

/* Ack the event */
ibv_ack_cq_events(ev_cq, 1);
```

SEE ALSO

ibv_create_comp_channel(3), **ibv_create_cq**(3), **ibv_req_notify_cq**(3), **ibv_poll_cq**(3)

AUTHORS

Dotan Barak
<*dotanba@gmail.com*>