NAME

ibv_get_async_event, ibv_ack_async_event - get or acknowledge asynchronous events

SYNOPSIS

#include <infiniband/verbs.h>

int ibv_get_async_event(struct ibv_context **context***,**
struct ibv_async_event **event***);**

void ibv_ack_async_event(struct ibv_async_event **event***);**

DESCRIPTION

ibv_get_async_event() waits for the next async event of the RDMA device context *context* and returns it through the pointer *event*, which is an ibv_async_event struct, as defined in <infiniband/verbs.h>.

```
struct ibv_async_event {
        union {
                struct ibv_cq  *cq;          /* CQ that got the event */
                struct ibv_qp  *qp;          /* QP that got the event */
                struct ibv_srq *srq;         /* SRQ that got the event */
                int             port_num;    /* port number that got the event */
        } element;
        enum ibv_event_type     event_type;  /* type of the event */
};
```

One member of the element union will be valid, depending on the event_type member of the structure. event_type will be one of the following events:

QP events:

IBV_EVENT_QP_FATAL Error occurred on a QP and it transitioned to error state

IBV_EVENT_QP_REQ_ERR Invalid Request Local Work Queue Error

IBV_EVENT_QP_ACCESS_ERR Local access violation error

IBV_EVENT_COMM_EST Communication was established on a QP

IBV_EVENT_SQ_DRAINED Send Queue was drained of outstanding messages in progress

IBV_EVENT_PATH_MIG A connection has migrated to the alternate path

IBV_EVENT_PATH_MIG_ERR A connection failed to migrate to the alternate path

IBV_EVENT_QP_LAST_WQE_REACHED Last WQE Reached on a QP associated with an SRQ

CQ events:

IBV_EVENT_CQ_ERR CQ is in error (CQ overrun)

SRQ events:

IBV_EVENT_SRQ_ERR Error occurred on an SRQ

IBV_EVENT_SRQ_LIMIT_REACHED SRQ limit was reached

Port events:

IBV_EVENT_PORT_ACTIVE Link became active on a port

IBV_EVENT_PORT_ERR Link became unavailable on a port

IBV_EVENT_LID_CHANGE LID was changed on a port

IBV_EVENT_PKEY_CHANGE P_Key table was changed on a port

IBV_EVENT_SM_CHANGE SM was changed on a port

IBV_EVENT_CLIENT_REREGISTER SM sent a CLIENT_REREGISTER request to a port

IBV_EVENT_GID_CHANGE GID table was changed on a port

CA events:

IBV_EVENT_DEVICE_FATAL CA is in FATAL state

ibv_ack_async_event() acknowledge the async event *event*.

RETURN VALUE

ibv_get_async_event() returns 0 on success, and -1 on error.

ibv_ack_async_event() returns no value.

NOTES

All async events that **ibv_get_async_event()** returns must be acknowledged using **ibv_ack_async_event()**. To avoid races, destroying an object (CQ, SRQ or QP) will wait for all affiliated events for the object to be acknowledged; this avoids an application retrieving an affiliated event after the corresponding object has already been destroyed.

ibv_get_async_event() is a blocking function. If multiple threads call this function simultaneously, then when an async event occurs, only one thread will receive it, and it is not possible to predict which thread will receive it.

EXAMPLES

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

1. Set the async events queue work mode to be non-blocked
2. Poll the queue until it has an async event
3. Get the async event and ack it

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

/*
 * poll the queue until it has an event and sleep ms_timeout
 * milliseconds between any iteration
 */
my_pollfd.fd      = ctx->async_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;
    }

    /* Get the async event */
    if (ibv_get_async_event(ctx, &async_event)) {
        fprintf(stderr, "Failed to get async_event\n");
        return 1;
    }

    /* Ack the event */
    ibv_ack_async_event(&async_event);
```

SEE ALSO

ibv_open_device(3)

AUTHORS

Dotan Barak <dotanba@gmail.com>