NAME

ibv_wr_abort, ibv_wr_complete, ibv_wr_start – Manage regions allowed to post work

ibv_wr_atomic_cmp_swp, ibv_wr_atomic_fetch_add – Post remote atomic operation work requests

ibv_wr_bind_mw, ibv_wr_local_inv – Post work requests for memory windows

ibv_wr_rdma_read, ibv_wr_rdma_write, ibv_wr_rdma_write_imm – Post RDMA work requests

ibv_wr_send, ibv_wr_send_imm, ibv_wr_send_inv – Post send work requests

ibv_wr_send_tso – Post segmentation offload work requests

ibv_wr_set_inline_data, ibv_wr_set_inline_data_list – Attach inline data to the last work request

ibv_wr_set_sge, ibv_wr_set_sge_list – Attach data to the last work request

ibv_wr_set_ud_addr – Attach UD addressing info to the last work request

ibv_wr_set_xrc_srqn – Attach an XRC SRQN to the last work request

SYNOPSIS

```
#include <infiniband/verbs.h>

void ibv_wr_abort(struct ibv_qp_ex *qp);
int ibv_wr_complete(struct ibv_qp_ex *qp);
void ibv_wr_start(struct ibv_qp_ex *qp);

void ibv_wr_atomic_cmp_swp(struct ibv_qp_ex *qp, uint32_t rkey,
                           uint64_t remote_addr, uint64_t compare,
                           uint64_t swap);
void ibv_wr_atomic_fetch_add(struct ibv_qp_ex *qp, uint32_t rkey,
                             uint64_t remote_addr, uint64_t add);

void ibv_wr_bind_mw(struct ibv_qp_ex *qp, struct ibv_mw *mw, uint32_t rkey,
                    const struct ibv_mw_bind_info *bind_info);
void ibv_wr_local_inv(struct ibv_qp_ex *qp, uint32_t invalidate_rkey);

void ibv_wr_rdma_read(struct ibv_qp_ex *qp, uint32_t rkey,
                      uint64_t remote_addr);
void ibv_wr_rdma_write(struct ibv_qp_ex *qp, uint32_t rkey,
                       uint64_t remote_addr);
void ibv_wr_rdma_write_imm(struct ibv_qp_ex *qp, uint32_t rkey,
                           uint64_t remote_addr, __be32 imm_data);

void ibv_wr_send(struct ibv_qp_ex *qp);
void ibv_wr_send_imm(struct ibv_qp_ex *qp, __be32 imm_data);
void ibv_wr_send_inv(struct ibv_qp_ex *qp, uint32_t invalidate_rkey);
void ibv_wr_send_tso(struct ibv_qp_ex *qp, void *hdr, uint16_t hdr_sz,
                     uint16_t mss);

void ibv_wr_set_inline_data(struct ibv_qp_ex *qp, void *addr, size_t length
void ibv_wr_set_inline_data_list(struct ibv_qp_ex *qp, size_t num_buf,
                                 const struct ibv_data_buf *buf_list);
void ibv_wr_set_sge(struct ibv_qp_ex *qp, uint32_t lkey, uint64_t addr,
                    uint32_t length);
void ibv_wr_set_sge_list(struct ibv_qp_ex *qp, size_t num_sge,
                         const struct ibv_sge *sg_list);

void ibv_wr_set_ud_addr(struct ibv_qp_ex *qp, struct ibv_ah *ah,
```

```
                                    uint32_t remote_qpn, uint32_t remote_qkey);
        void ibv_wr_set_xrc_srqn(struct ibv_qp_ex *qp, uint32_t remote_srqn);
```

DESCRIPTION

The verbs work request API (ibv_wr_*) allows efficient posting of work to a send queue using function calls instead of the struct based *ibv_post_send()* scheme. This approach is designed to minimize CPU branching and locking during the posting process.

This API is intended to be used to access additional functionality beyond what is provided by *ibv_post_send()*.

WRs batches of *ibv_post_send()* and this API WRs batches can interleave together just if they are not posted within the critical region of each other. (A critical region in this API formed by *ibv_wr_start()* and *ibv_wr_complete()*/*ibv_wr_abort()*)

USAGE

To use these APIs the QP must be created using ibv_create_qp_ex() which allows setting the **IBV_QP_INIT_ATTR_SEND_OPS_FLAGS** in *comp_mask*. The *send_ops_flags* should be set to the OR of the work request types that will be posted to the QP.

If the QP does not support all the requested work request types then QP creation will fail.

Posting work requests to the QP is done within the critical region formed by *ibv_wr_start()* and *ibv_wr_complete()*/*ibv_wr_abort()* (see CONCURRENCY below).

Each work request is created by calling a WR builder function (see the table column WR builder below) to start creating the work request, followed by allowed/required setter functions described below.

The WR builder and setter combination can be called multiple times to efficiently post multiple work requests within a single critical region.

Each WR builder will use the *wr_id* member of *struct ibv_qp_ex* to set the value to be returned in the completion. Some operations will also use the *wr_flags* member to influence operation (see Flags below). These values should be set before invoking the WR builder function.

For example a simple send could be formed as follows:

```
qpx->wr_id = 1;
ibv_wr_send(qpx);
ibv_wr_set_sge(qpx, lkey, &data, sizeof(data));
```

The section WORK REQUESTS describes the various WR builders and setters in details.

Posting work is completed by calling *ibv_wr_complete()* or *ibv_wr_abort()*. No work is executed to the queue until *ibv_wr_complete()* returns success. *ibv_wr_abort()* will discard all work prepared since *ibv_wr_start()*.

WORK REQUESTS

Many of the operations match the opcodes available for *ibv_post_send()*. Each operation has a WR builder function, a list of allowed setters, and a flag bit to request the operation with *send_ops_flags* in *struct ibv_qp_init_attr_ex* (see the EXAMPLE below).

Operation	WR builder	QP Type Supported	setters
ATOMIC_CMP_AND_SWP	ibv_wr_atomic_cmp_swp()	RC, XRC_SEND	DATA, QP
ATOMIC_FETCH_AND_ADD	ibv_wr_atomic_fetch_add()	RC, XRC_SEND	DATA, QP
BIND_MW	ibv_wr_bind_mw()	UC, RC, XRC_SEND	NONE
LOCAL_INV	ibv_wr_local_inv()	UC, RC, XRC_SEND	NONE
RDMA_READ	ibv_wr_rdma_read()	RC, XRC_SEND	DATA, QP
RDMA_WRITE	ibv_wr_rdma_write()	UC, RC, XRC_SEND	DATA, QP

RDMA_WRITE_WITH_IMM	ibv_wr_rdma_write_imm()	UC, RC, XRC_SEND	DATA, QP
SEND	ibv_wr_send()	UD, UC, RC, XRC_SEND, RAW_PACKET	DATA, QP
SEND_WITH_IMM	ibv_wr_send_imm()	UD, UC, RC, SRC SEND	DATA, QP
SEND_WITH_INV	ibv_wr_send_inv()	UC, RC, XRC_SEND	DATA, QP
TSO	ibv_wr_send_tso()	UD, RAW_PACKET	DATA, QP

Atomic operations

Atomic operations are only atomic so long as all writes to memory go only through the same RDMA hardware. It is not atomic with writes performed by the CPU, or by other RDMA hardware in the system.

ibv_wr_atomic_cmp_swp()

If the remote 64 bit memory location specified by *rkey* and *remote_addr* equals *compare* then set it to *swap*.

ibv_wr_atomic_fetch_add()

Add *add* to the 64 bit memory location specified *rkey* and *remote_addr*.

Memory Windows

Memory window type 2 operations (See man page for ibv_alloc_mw).

ibv_wr_bind_mw()

Bind a MW type 2 specified by **mw**, set a new **rkey** and set its properties by **bind_info**.

ibv_wr_local_inv()

Invalidate a MW type 2 which is associated with **rkey**.

RDMA

ibv_wr_rdma_read()

Read from the remote memory location specified *rkey* and *remote_addr*. The number of bytes to read, and the local location to store the data, is determined by the DATA buffers set after this call.

ibv_wr_rdma_write(), *ibv_wr_rdma_write_imm()*

Write to the remote memory location specified *rkey* and *remote_addr*. The number of bytes to read, and the local location to get the data, is determined by the DATA buffers set after this call.

The _imm version causes the remote side to get a IBV_WC_RECV_RDMA_WITH_IMM containing the 32 bits of immediate data.

Message Send

ibv_wr_send(), *ibv_wr_send_imm()*

Send a message. The number of bytes to send, and the local location to get the data, is determined by the DATA buffers set after this call.

The _imm version causes the remote side to get a IBV_WC_RECV_RDMA_WITH_IMM containing the 32 bits of immediate data.

ibv_wr_send_inv()

The data transfer is the same as for *ibv_wr_send()*, however the remote side will invalidate the MR specified by *invalidate_rkey* before delivering a completion.

ibv_wr_send_tso()

Produce multiple SEND messages using TCP Segmentation Offload. The SGE points to a TCP Stream buffer which will be segmented into MSS size SENDs. The hdr includes the entire network headers up to and including the TCP header and is prefixed before each segment.

QP Specific setters

Certain QP types require each post to be accompanied by additional setters, these setters are mandatory for any operation listing a QP setter in the above table.

UD **QPs**
ibv_wr_set_ud_addr() must be called to set the destination address of the work.

XRC_SEND **QPs**
ibv_wr_set_xrc_srqn() must be called to set the destination SRQN field.

DATA transfer setters
For work that requires to transfer data one of the following setters should be called once after the WR builder:

ibv_wr_set_sge()
Transfer data to/from a single buffer given by the lkey, addr and length. This is equivalent to *ibv_wr_set_sge_list()* with a single element.

ibv_wr_set_sge_list()
Transfer data to/from a list of buffers, logically concatenated together. Each buffer is specified by an element in an array of *struct ibv_sge*.

Inline setters will copy the send data during the setter and allows the caller to immediately re−use the buffer. This behavior is identical to the IBV_SEND_INLINE flag. Generally this copy is done in a way that optimizes SEND latency and is suitable for small messages. The provider will limit the amount of data it can support in a single operation. This limit is requested in the *max_inline_data* member of *struct ibv_qp_init_attr*. Valid only for SEND and RDMA_WRITE.

ibv_wr_set_inline_data()
Copy send data from a single buffer given by the addr and length. This is equivalent to *ibv_wr_set_inline_data_list()* with a single element.

ibv_wr_set_inline_data_list()
Copy send data from a list of buffers, logically concatenated together. Each buffer is specified by an element in an array of *struct ibv_inl_data*.

Flags
A bit mask of flags may be specified in *wr_flags* to control the behavior of the work request.

IBV_SEND_FENCE
Do not start this work request until prior work has completed.

IBV_SEND_IP_CSUM
Offload the IPv4 and TCP/UDP checksum calculation

IBV_SEND_SIGNALED
A completion will be generated in the completion queue for the operation.

IBV_SEND_SOLICTED
Set the solicted bit in the RDMA packet. This informs the other side to generate a completion event upon receiving the RDMA operation.

CONCURRENCY
The provider will provide locking to ensure that *ibv_wr_start()* and *ibv_wr_complete()/abort()* form a per−QP critical section where no other threads can enter.

If an *ibv_td* is provided during QP creation then no locking will be perfomed and it is up to the caller to ensure that only one thread can be within the critical region at a time.

RETURN VALUE
Applications should use this API in a way that does not create failures. The individual APIs do not return a failure indication to avoid branching.

If a failure is detected during operation, for instance due to an invalid argument, then *ibv_wr_complete()* will return failure and the entire posting will be aborted.

EXAMPLE
```
/* create RC QP type and specify the required send opcodes */
qp_init_attr_ex.qp_type = IBV_QPT_RC;
```

```
qp_init_attr_ex.comp_mask |= IBV_QP_INIT_ATTR_SEND_OPS_FLAGS;
qp_init_attr_ex.send_ops_flags |= IBV_QP_EX_WITH_RDMA_WRITE;
qp_init_attr_ex.send_ops_flags |= IBV_QP_EX_WITH_RDMA_WRITE_WITH_IMM;

ibv_qp *qp = ibv_create_qp_ex(ctx, qp_init_attr_ex);
ibv_qp_ex *qpx = ibv_qp_to_qp_ex(qp);

ibv_wr_start(qpx);

/* create 1st WRITE WR entry */
qpx->wr_id = my_wr_id_1;
ibv_wr_rdma_write(qpx, rkey, remote_addr_1);
ibv_wr_set_sge(qpx, lkey, local_addr_1, length_1);

/* create 2nd WRITE_WITH_IMM WR entry */
qpx->wr_id = my_wr_id_2;
qpx->send_flags = IBV_SEND_SIGNALED;
ibv_wr_rdma_write_imm(qpx, rkey, remote_addr_2, htonl(0x1234));
ibv_set_wr_sge(qpx, lkey, local_addr_2, length_2);

/* Begin processing WRs */
ret = ibv_wr_complete(qpx);
```

SEE ALSO

ibv_post_send(3), **ibv_create_qp_ex(3)**.

AUTHOR

Jason Gunthorpe <jgg@mellanox.com> Guy Levi <guyle@mellanox.com>