EXS – EXtended Sockets

Robert D. Russell <rdr@iol.unh.edu>
Patrick MacArthur <pmacarthy@iol.unh.edu>

InterOperability Laboratory
University of New Hampshire
121 Technology Drive, Suite 2
Durham, New Hampshire 03824-4716, USA
Extended Sockets API (ES-API)

- Published by The Open Group in 2005
  - opengroup.org/bookstore/catalog/c050.htm

- Defines 2 major new extensions to “normal” sockets
  - memory registration for zero-copy I/O
  - event queues for asynchronous I/O

- Designed to give programmer access to RDMA
EXS Goals

- Expose RDMA features to programmer
  - do not totally hide RDMA from programmer
  - provide a more convenient interface than verbs

- Extend well-known sockets API
  - “normal” sockets are inadequate for direct RDMA use
  - add a few new functions and data types
  - repurpose many existing functions and data types

- Target audience
  - new applications intended for RDMA
  - porting existing applications requires source code changes
UNH-EXS

- Based on Open Group's ES-API
  - with additional extensions in order to run entirely in user space (because ES-API is expected to be integrated into existing kernel sockets)
- Runs on InfiniBand, iWARP, and RoCE
- Provides both `SOCK_SEQPACKET` and `SOCK_STREAM` connections using RC only
- Library designed for use by user threads in Linux
- Implemented entirely with user-space OFS verbs
- Requires no change to OFS or Linux
## UNH-EXS stack

<table>
<thead>
<tr>
<th>user space</th>
<th>Application Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNH-EXS Library</td>
<td></td>
</tr>
<tr>
<td>OFS Verbs Library</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel space</th>
<th>OFS Kernel Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfiniBand Driver</td>
<td>RoCE Driver</td>
</tr>
<tr>
<td>RoCE NIC</td>
<td>iWARP RNIC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CA</th>
<th>InfiniBand HCA</th>
<th>RoCE NIC</th>
<th>iWARP RNIC</th>
</tr>
</thead>
</table>

| wire | InfiniBand Fabric | 10Gig Ethernet |
EXS event queues

- Extensions to deal with asynchronous events
- New “event queue” and “event” data structures
  - exs_qhandle_t
  - exs_event_t
- New queue manipulation functions
  - exs_qcreate() - creates new event queue
  - exs_qdelete() - deletes existing event queue
  - exs_qdequeue() - removes events from existing event queue
  - exs_qmodify() - modifies existing event queue
  - exs_qstatus() - returns event queue attributes
EXS event queue usage

- `send()`, `recv()`, `accept()`, `connect()`, `close()` have extended versions: `exs_send()`, `exs_recv()`, etc.
  - all these extended operations just start an action
  - control returns immediately to user
  - operation proceeds in parallel to user code

- Extended operations have extra parameters, 1st is
  - `exs_qhandle_t` parameter required to designate event queue

- When I/O operation completes, EXS library adds
  - `exs_event_t` containing status to designated event queue
EXS memory registration

- Extensions to deal with registered memory

- New “memory region” data structure
  - `exs_mhandle_t`

- Two new registration functions
  - `exs_mregister()` - creates new `exs_mhandle_t` by registering user-defined virtual memory
  - `exs_mderegister()` - destroys existing `exs_mhandle_t` by unregistering its memory region
EXS memory region usage

- New `exs_send()` and `exs_recv()` functions designate “memory region” with additional parameter
  - `exs_mhandle_t` result of previous registration

- Normal address and length parameters must refer to memory entirely within designated “memory region”
Parameters to `exs_send()`

- **Four “normal socket” parameters**
  - `fd` – socket descriptor
  - `address` – of data to be sent
  - `length` – number of data bytes to send
  - `flags`

- **Three new “extension” parameters**
  - `event_queue` – for posting completion event
  - `request_id` – user-defined transaction id
  - `memory_region` – must cover all data bytes
Parameters to `exs_recv()`

- Four “normal socket” parameters
  - `fd` – socket descriptor
  - `address` – of where to put received data
  - `length` – maximum number of data bytes to receive
  - `flags`

- Three new “extension” parameters
  - `event_queue` – for posting completion event
  - `request_id` – user-defined transaction id
  - `memory_region` – must cover all data bytes
How EXS maps transfers onto verbs

- `exs_recv()` issues RDMA SEND to “advertise” its “metadata” to other side
  - address – where to put data
  - length – maximum number of bytes of data to receive
  - remote “key” from the `memory_region`

- `exs_send()` matches its “metadata” with advertised “metadata” and issues RDMA WRITE_WITH_IMM to transfer data

- on both sides, EXS library gets completion status and enqueues it in `event_queue` along with user-defined `request_id`
Typical EXS Data Transfer

- user App
- EXS library
- OFS
- CA
- wire
- CA
- OFS
- EXS library
- user App

- exs_recv
- exs_send
- exs_qdequeue
- exs_qdequeue

parallel activity

RDMA Send metadata

RDMA Write-with-Imm data
Other UNH-EXS functions

- `exs_accept()` – ES-API standard
- `exs_bind()` – UNH extension
- `exs_close()` – UNH extension
- `exs_connect()` – ES-API standard
- `exs_fcntl()` – UNH extension
- `exs_init()` – ES-API standard
- `exs_listen()` – UNH extension
- `exs_socket()` – UNH extension
Tuning UNH-EXS with exs_fcntl()

- Modeled on “normal UNIX” fcntl()
- Allows user to:
  - set maximum “small packet” size
  - set maximum “inline data” size
  - set completion thread’s CPU affinity
  - turn on “busy-polling” for completions
  - set receive buffer size (for SOCK_STREAMs only)
  - turn off use of receive buffer (for SOCK_STREAMs only)
  - set maximum “advertisement” credits
Obtaining UNH-EXS

- Complete source code tar file
  - [www.iol.unh.edu/services/research/unh-exs](http://www.iol.unh.edu/services/research/unh-exs)
  - includes README giving installation instructions
  - includes overview document for programmers

- User overview documentation (how to use it)
  - [www.iol.unh.edu/services/reseach/unh-exs/exs-overview.pdf](http://www.iol.unh.edu/services/reseach/unh-exs/exs-overview.pdf)
  - describes each EXS function in detail
  - has examples of converting existing sockets code to EXS
Relationships between EXS and “normal” socket and UNIX functions

- EXS memory regions, event queues, and fds can NOT be inherited by a child process

- UNH-EXS fds cannot be used with “normal” socket or UNIX I/O functions, such as:
  - `read()`, `write()`, `poll()`, `select()`, `fcntl()`, `fstat()`, etc.

- UNH-EXS is thread safe, but not thread cancellation safe
EXS blast throughput over FDR
EXS blast CPU usage over FDR
EXS throughput performance

- The bigger the message, the smaller the CPU usage (for fixed number of outstanding messages)
- The more simultaneously outstanding messages, the higher the throughput (for fixed message size)
- Reasonable “sweet spot”: 512 Kibibytes, 4 messages
  - throughput: 45.6 Gigabytes/second
  - CPU usage: 14.0% user, 9.4% kernel, 23.4% total
- Ideal “sweet spot”: 2 Mibibytes, 4 messages
  - throughput: 47.9 Gigabytes/second
  - CPU usage: 4.2% user, 2.3% kernel, 6.5% total
EXS ping-pong round-trip time over FDR
EXS ping-pong CPU usage over FDR

![Graph showing CPU utilization over message size with and without CPU pinning.](image)
EXS ping-pong performance

- Small messages very sensitive to 2 factors:
  - “busy-polling” vs “wait-for-notify” for completions
  - “pinning” threads to CPUs or not
    - two threads to pin: completion thread, mainline thread
    - together, “busy-polling” and “pinning” reduce RTT by 1/3, from 30 microseconds to 10 microseconds
      - one-way time reduced from 15 to 5 microseconds
- “busy-polling” is expensive in CPU usage
  - total for 2 threads increases from about 60% to 200%
- “wait-for-notify” not cheap due to kernel involvement
Acknowledgments

- This material is based upon work supported by the National Science Foundation under award number OCI-1127228.

- Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
QUESTIONS?
THANK YOU!