Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spring Term 2014

ADVANCED COMPUTER NETWORKS
Project P1: Introduction to RDMA Programming

Assigned on: **17 April 2014**
Due by: **8 May 2014, 23:59**

1 Introduction

The goal of this project is to give an introduction to RDMA programming [2, 3] and related software [5]. This is a group project with the maximum group size of 2. This exercise assumes basic knowledge of Linux and C. Start with downloading the project handout tarball we provide from the website and extract it into a folder of your choice.

2 Setting up the RDMA Framework for Development

In this section we provide the detailed instructions to install and configure the software/packages needed to solve this exercise. For this programming assignment, you will need a 64-bit Linux system (Ubuntu, Debian) with a kernel that is version 2.6.36.2 or newer. One option can be installing an image inside VirtualBox (http://www.virtualbox.org/). We have tested our solution using various configurations: a 64-bit system running Ubuntu (11.04 - the Natty Narwhal) and 64-bit Ubuntu image running inside VirtualBox. Kernel versions varied between 2.6.36 and 2.6.39.

For the given VM image, you should just build and install SoftiWARP. Remember to load the relevant kernel modules after a fresh reboot.

2.1 Installation of RDMA Related Packages

Here we describe the steps required to get the RDMA framework ready for development. In order to see all RDMA related packages type:

```
apt-cache search rdma
```

Among these packages the necessary ones to be installed for this assignment are the following:

```
ibverbs-utils, libibverbs-dev, libibverbs1, librdmacm-dev, librdmacm1,
rdmacm-utils
```

In addition, make sure that you need the commonly used Linux tools such as:

```
libtool, autoconf, automake, linux-tool-common, linux-headers
```

You can install these packages using `sudo apt-get install` command.

2.2 Building the SoftiWARP Kernel Driver and User Library

The SoftiWARP RDMA device comes with a kernel module and a user space library. Find the softiwarp.tar file inside the assignment folder and extract it to a folder of your own choice. Build the SoftiWARP kernel driver as follows:

Go into the `kernel/softiwarp` that exists under the directory you extracted the softiwarp code. Build it using:

```
atr@localhost$ make clean
atr@localhost$ make
```

Then go to the `userlib/libsiw-0.1` directory inside your extracted softiwarp directory and build the SoftiWARP user library as follows:

```
atr@localhost$ ./configure --prefix=/home/acn/local
atr@localhost$ make clean
atr@localhost$ make install
```

If you are doing a custom installation then make sure to use right directory with the `--prefix` parameter. The prefix is where the libsiw will be installed locally for the user.

Then, set the LD_LIBRARY_PATH with the command:

```
atr@localhost$ export LD_LIBRARY_PATH=/home/acn/local/lib
```

You have to make sure every terminal window you open to run has to have this variable set. Eventually you might want to put this in your bash profile (.bashrc).

Make sure that you have the `siw.driver` file placed correctly in your system's `/etc/libibverbs.d` directory. If you do not find this file in that directory, you need to run the following commands:

```
sudo mkdir /etc/libibverbs.d
sudo cp /home/acn/local/etc/libibverbs.d/siw.driver /etc/libibverbs.d/
```

2.3 Installation of the udev Rules File for Ubuntu Systems

On Ubuntu systems, you will need to install a new udev rules file in order to be able to use the packages and execute the application properly.

- Find the `90-ib.rules` file inside the assignment handout folder.
- Copy this file to /etc/udev/rules.d directory in your system using the following command: `sudo cp 90-ib.rules /etc/udev/rules.d/`
- Reboot your system in order for this modification to take effect (or restart the udev manager).

2.4 Loading Kernel Modules

We use Open Fabric Enterprise Distribution (OFED) RDMA framework [1]. The OFED RDMA stack is part of the Linux kernel distribution and contains kernel modules. SoftiWARP kernel modules built in the last step is a device specific module, which upon loading is linked to the rest of the OFED module. To load all required modules into the system, we have provided a script called `addmodules.sh` in your assignment folder.

- Find `addmodules.sh` in your assignment folder and make it executable (if it is not already) using the `chmod +x` command.
- Open this file and edit the path of the siw.ko module to point to your own siw.ko (see Section 2.2) file inside `softiwarp/kernel/softiwarp`.
- Execute the script using the command `sudo ./addmodules.sh`. If you experience any permission problems with the script, try browsing into the `softiwarp/kernel/softiwarp` directory, and install the module directly using `sudo insmod siw.ko`

At this point you should be able to see the siw module loaded in your system. You can verify this by doing `lsmod | grep "siw"`.

2.5 Verify

To verify if the system is ready for the RDMA development you should be able to see a few RDMA capable devices using command `ibv_devices`. A sample output is shown below:

```
atr@localhost:~$ ibv_devices
    device                 node GUID
    ------              ----------------
    siw_wlan0           6c881470928c0000
    siw_eth0            3c970e986e2f0000
    siw_lo              7369775f6c6f0000
atr@localhost:~$
```

2.6 In Case of Problems

In general, if you are having trouble until this point, make sure that you have the following things done right:

- Installation of all the necessary RDMA packages.
- `LD_LIBRARY_PATH` is set correctly. Verify using `echo $LD_LIBRARY_PATH`
- `siw` related modules are loaded. Verify using `lsmod | grep "siw"`
- `strace` command can also be useful for debugging. Do `man strace` to see how to use this command.

If nothing else works, email Patrick or Animesh (atrivedi@student.ethz.ch) about the problem.

3 The Sample RDMA Client Server Application

In this section we describe the main task that you need to do in the context of this assignment. We start with giving a brief overview about the code that has been given out. The code implements a simple RDMA server client program. The code is split into:

- `rdma_common.[ch]` : contains some common RDMA work routines that come in handy while coding such as memory allocation and registration etc.
- `rdma_server.c` : contains RDMA server side logic.
- `rdma_client.c` : contains RDMA client side logic.

For a better understanding read the code, which is well commented.

The goal of the client is to copy a string buffer (give by the user using `-s` parameter) from a source buffer (identified by `src`) to a destination buffer (identified by `dst`) using a remote server buffer as a temporary buffer. The required logical steps can be summarized as below:

For the RDMA server, refer all symbols and functions in `rdma_server.c`

a) Start an RDMA server and wait until a client connects, see `start_rdma_server()`.

b) Setup and prepare client specific resources for a new incoming connection, see `setup_client_resources()`.

c) Accept the client connection, see `accept_client_connection()`.

d) Send server's buffer RDMA metadata (address, length, STAG) to the client, see `send_server_metadata_to_client()` (you will implement).

e) Wait for disconnect from the client, then clean up and shutdown, see `disconnect_and_cleanup`.

For the RDMA client, refer all symbols and functions in `rdma_client.c`

a) Setup and prepare connection resources for a new client connection, see `client_prepare_connection()`.

b) Prepare communication buffers where the client will receive the metadata from the RDMA server, see `client_pre_post_recv_buffer()`. This function shows how to prepare a receive work request.

c) Connect to the server, see `client_connect_to_server()`.

d) Send client side RDMA metadata (address, length, STAG) to the server and wait until the client has received metadata for the remote buffer from the server, see `client_send_metadata_to_server()`. This function shows how to prepare a send work request. RDMA read and write requests are prepared in a similar manner.

e) Write a local `src` buffer to the remote server buffer, whose credentials client received in the last step, using RDMA write. Then read the remote server buffer into another local `dst` buffer using RDMA read. See `client_remote_memory_ops()` (you will implement).

f) Verify that `src` and `dst` buffers contain the same content, see `check_src_dst()`.

g) Disconnect and cleanup the client connection, see `client_disconnect_and_clean()`.

See also figure 1 for the control flow.

3.1 Your Task

The provided code sets up RDMA connection related resources for the server and the client. The client successfully connects to the server and waits to receive RDMA metadata from the server. You task is to complete following functions in the code:

- `send_server_metadata_to_client()` in `rdma_server.c`
- `client_remote_memory_ops()` in `rdma_client.c`

3.2 Debugging

For debugging there is a compile time macro `debug` in `rdma_common.h`. Enable it by defining `ACN_RDMA_DEBUG`. For rdma related calls read the man page or refer to [4].

To see what is happening at the device level, you can compile `siw` module with `DPRINT_MASK` defined to `DBG_ALL` in `siw_debug.h` file. The output of the device can be seen with `dmesg` command while running the experiment. These messages can show you common mistakes such as base and bound violations, permission errors, invalid stags etc. for RDMA operations. Also, you can also enable selective debugging by choosing an appropriate `DPRINT_MASK`. Read the comments in the file. You need to unload (`sudo rmmod siw`) and reload (`sudo insmod siw.ko`) the `siw` module everytime you compile it for changes to take effect.

3.3 Sample Correct Output

Once you have correctly implemented the missing parts of the code, you should verify that the `src` and `dst` buffers contain the same content.

Server

```
atr@localhost$ ./rdma_server
Server is listening successfully at: 0.0.0.0 , port: 20886
A new connection is accepted from 127.0.0.1
Client side buffer information is received...
---------------------------------------------------------
buffer attr, addr: 0xa58010 , len: 36 , stag : 0xf6e826
---------------------------------------------------------
The client has requested buffer length of : 36 bytes
A disconnect event is received from the client...
Server shut-down is complete
```

Client

```
atr@localhost$ ./rdma_client -s "This is a test string for project P1"
Passed string is : This is a test string for project P1 , with count 36
Trying to connect to server at : 127.0.0.1 port: 20886
The client is connected successfully
---------------------------------------------------------
buffer attr, addr: 0x673f70 , len: 36 , stag : 0x47d0e0
---------------------------------------------------------
...
SUCCESS, source and destination buffers match
Client resource clean up is complete
```

4 Hand-In Instructions

Please commit the solution to your SVN repo under the folder of project1 by May 8th.

4.1 Demo

At the next assignment session (May 8th) you are suppose to show a working demo of the code. When demonstrating your solution you should be able to show a working solution and explain what have you written, and how it works.

References

[1] OFED for Linux. `https://www.openfabrics.org/index.php/resources/ofed-for-linux-ofed-for-windows/linux-sources.html`.

[2] RDMA Consortium. `http://www.rdmaconsortium.org/`.

[3] Work on RDMA host software at IBM Research Zurich. `http://www.zurich.ibm.com/sys/rdma/`.

[4] Detailed blog about RDMA. `http://www.rdmamojo.com/`.

[5] SoftiWARP: Software iWARP kernel driver and user library for Linux. `http://gitorious.org/softiwarp`.

Appendix

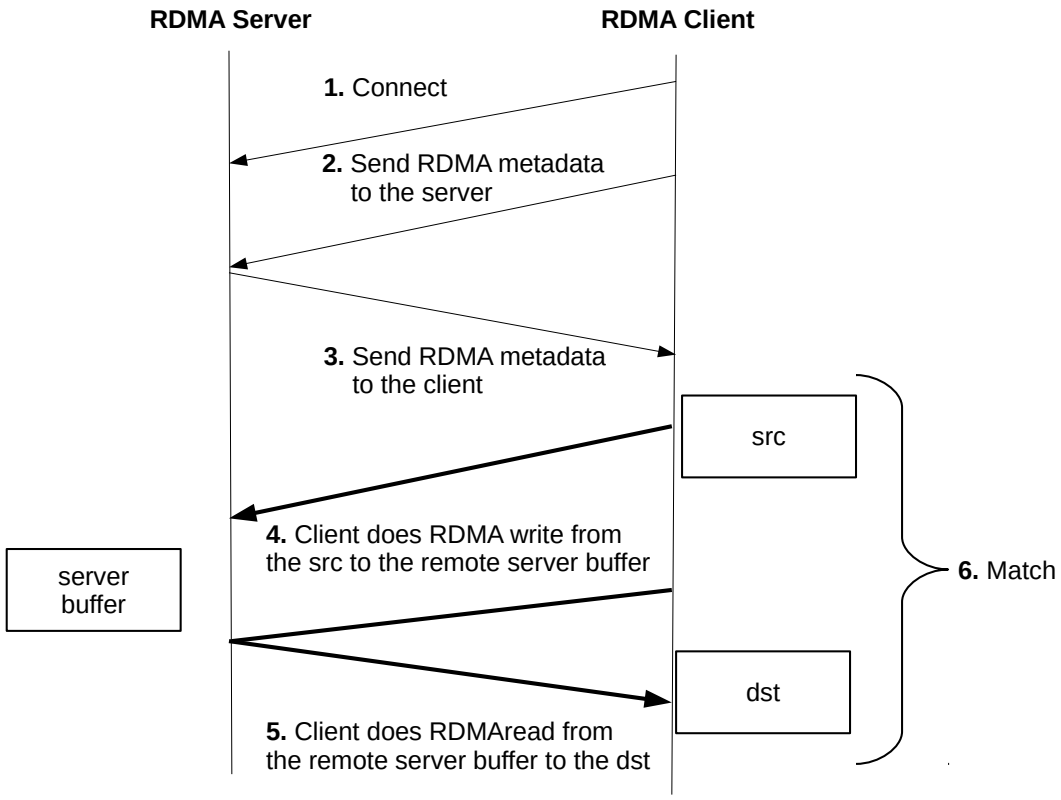

Figure 1: The interaction between server and client.