Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spring Term 2015

ADVANCED COMPUTER NETWORKS
Project P1: Introduction to RDMA Programming

Assigned on: **16 April 2015**
Due by: **29 April 2015, 23:59**

1 Introduction

The goal of this project is to give an introduction to RDMA programming [3, 4] and related software [6]. 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.

```
tar -xzvf rdma_assignment_handout.tar.gz
cd  rdma_assignment_handout
tar -xzf softiwarp.tar.gz
```

You will find some reference material and helper scripts here.

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 3.13.0 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 (14.04.02 LTS Trusty Tahr) and 64-bit Ubuntu image running inside VirtualBox. We are providing a VM image where everything is already installed. You can import this VM in your virtualbox and it should give you working setup. The username for the VM is `acn` (password: `nopass`) and you should have `sudo` access with this user.

You only need to insert relevant kernel modules after a fresh reboot. You can use following script inserting proper kernel modules:

```
rdma_assignment_handout/addmodules.sh
```

2.1 Installation of RDMA Related Packages

In case you want to use some other machine for the development, then you can use following instructions as a guideline to setup the development environment. Following instructions assume Ubuntu/Debian based system. Please adapt the instructions based on your setup. You can also refer to [1] for additional information about installation, and for help in debugging installation related issues.

2.1.1 Installing kernel source packages

We will need to install kernel source as we will be needing it to compile the kernel modules needed for RDMA iWARP.

Here are the packages needed.

```
sudo apt-get install libtool autoconf automake linux-tools-common
sudo apt-get install fakeroot build-essential crash kexec-tools makedumpfile kernel-wedge
sudo apt-get build-dep linux
sudo apt-get install git-core libncurses5 libncurses5-dev libelf-dev
sudo apt-get install linux-headers-$(uname -r)
```

2.1.2 Installing RDMA dependencies

Here we describe the steps required to get the RDMA framework ready for development.

```
sudo apt-get install libibverbs1 libibcm1 libibcm-dev ibverbs-utils libibverbs-dev
sudo apt-get install libibverbs1 librdmacm-dev librdmacm1 rdmacm-utils
```

2.1.3 Installation of the udev Rules File for Ubuntu Systems

Copy following udev rules file in proper location to make sure that your system will create RDMA related devices in `/udev/infiniband/` correctly when proper kernel modules are inserted.

```
sudo cp rmda_assignment_handout/90-ib.rules /etc/udev/rules.d/
```

Typically, udev will detect the new rules and incorporate them automatically. You may have to Reboot your system in order for this modification to take effect (or restart the udev manager) if it does not work for you directly.

2.1.4 Building the SoftiWARP Kernel Driver

The SoftiWARP RDMA device comes with a kernel module and a user space library. You can build the SoftiWARP kernel driver as follows:

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

```
make clean
make
sudo make install
```

2.1.5 Building the SoftiWARP userspace libary

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

```
./autogen.sh
./autogen.sh
./configure
make clean
sudo make install
sudo ldconfig
```

In case you are using custom location for installation, then you may have to set the LD_LIBRARY_PATH environment variable accordingly:

```
export LD_LIBRARY_PATH=/your/custom/location/
sudo ldconfig
```

Also, you need to make sure that the `siw.driver` file placed correctly in your system's `/etc/libibverbs.d` directory. You can do that by creating following soft link to the default installation directory (shown bellow), or by copying the file directly in the `/etc/libibverbs.d` directory.

```
sudo ln -s /usr/local/etc/libibverbs.d /etc/libibverbs.d
```

2.2 Loading Kernel Modules

We use Open Fabric Enterprise Distribution (OFED) RDMA framework [2]. 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. This script is fairly simple and tries to make sure that all the requirements are met. Take a look into the script if you are getting any error.

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"`.

2.3 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_eth0         3c970e986e2f0000
    siw_lo           7369775f6c6f0000
atr@localhost:~$
```

2.4 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 Pravin 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 [5].

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 the deadline.

4.1 Demo

At the next assignment session 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] Detailed blog about Installation. http://www.reflectionsofthevoid.com/2011/03/how-to-install-soft-iwarp-on-ubuntu.html.

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

[3] RDMA Consortium. http://www.rdmaconsortium.org/.

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

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

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

Appendix

When the provided code is run following output is generated on the client side:

Server:

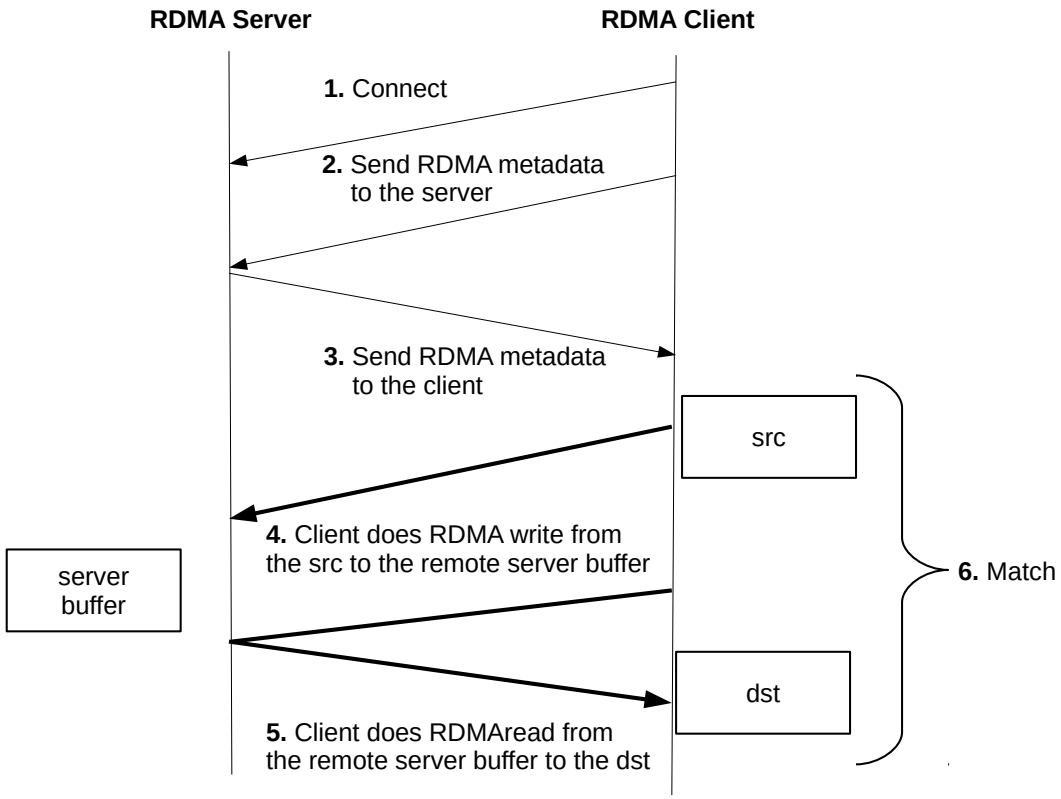

Figure 1: The interaction between server and client.

```
acn@eucan-VirtualBox:code$ ./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: 0x1ea4010 , len: 10 , stag : 0x340a26
------------------------------------------------------
The client has requested buffer length of : 10 bytes
rdma_server.c : 297 : ERROR : This function is not yet implemented
rdma_server.c : 423 : ERROR : Failed to send server metadata to the client, ret = -38
```

Client:

```
acn@eucan-VirtualBox:code$ ./rdma_client -s "helloWorld"
Passed string is : helloWorld , with count 10
Trying to connect to server at : 127.0.0.1 port: 20886
The client is connected successfully
rdma_common.c : 185 : ERROR : Work completion (WC) has error status: -5 (means: Work Request Flus
rdma_client.c : 295 : ERROR : We failed to get 2 work completions , ret = -5
rdma_client.c : 459 : ERROR : Failed to setup client connection , ret = -5
```

After an rdma_accept on the server side, the client is suppose to receive an ESTABLISHED event. But if server terminates (which happens in our case) before the client retrieves the event, the

connection is gone and the client will get -22 on the event channel. But if the server is bit slower, the client might retrieve this event before the server exits. It is a non-deterministic case as network latencies are comparable to OS latencies.

To get a better picture you can insert this code snippet in the unimplemented function on the server side. Here the server does not terminate immediately but waits to receive client side credentials but then fails to send its own.

```
struct ibv_wc wc;
int ret = -1;
ret = process_work_completion_events(io_completion_channel, &wc, 1);
if (ret != 1) {
    rdma_error("Failed to receive , ret = %d \n", ret);
    return ret;
}
printf("Client side buffer information is received...\n");
show_rdma_buffer_attr(&client_metadata_attr);
printf("The client has requested buffer length of : %u bytes \n",
        client_metadata_attr.length);

rdma_error("This function is not yet implemented \n");
```

In this case, not the client fails to retrieve a valid receive event on the work completion index 1. Work completion index 0, that belonged to the send work request, is successful.