Exercise #3: Low-Latency Protocols

General

The goal of this exercise is to compare the Rendezvous protocol to a trivial "eager" case.
Please refer to Lecture #2 for additional information on Eager vs. Rendezvous.

Part 1 – Eager Client-Server (30%)

Write a C/C++ single-threaded server-client application with Verbs API: The client connects to the server (use the same code, with IBV_WR_SEND) and sends two kinds of requests (for part 1: key + value < 4KB):

1. GET request: Given a (string) key – retrieve the (string) value from the server (default is "").
2. SET request: Given a key and a value – store it on the server (possibly overwriting the old value). The server can store the key-value mapping in a simple array (or any other database), to simplify implementation.

The client should implement the following API and test both kv_set() and kv_get() throughput:

```
int kv_open(char *servername, void **kv_handle); /*Connect to server*/
int kv_set(void *kv_handle, const char *key, const char *value);
int kv_get(void *kv_handle, const char *key, char **value);
void kv_release(char *value);/* Called after get() on value pointer */
int kv_close(void *kv_handle); /* Destroys the QP */
```

*Note that `kv_handle` is a struct you define, create and use with every API call except for `kv_release`.

Part 2 – RDMA & Rendezvous protocol (30%)

Extend the application to also support the use of RDMA (reads or writes). When a user calls set/get with value >= 4KB (you can assume the key size < 4KB), the client will send the server a "control message", and the server will act according to its content. For messages <4KB – use part 1. You should also pay attention to the following:

1. The goal of Rendezvous is to achieve "zero-copy" – send large buffers from their location. You shouldn't call memcpy() on more than 4KB.
2. Send as few control messages as possible.
3. Actions like memory allocation (malloc) and registration (ibv_reg_mr) are expensive. You must do them in parallel to communication whenever possible. For example, if you can send a message and allocate memory – send the message first (it may be impossible if you need to send the pointer to that memory). Please note the application is single-threaded.

"Control message" may be interpreted in different ways – the interpretation is deliberately open and is part of this exercise.

Part 3 – Run single server with multiple clients (40%)

Run a single KV server to serve multiple (at least 2) clients at the same time.

1. Run 1 server, 2 clients or more clients, each client should use its own input file (see Client1.txt, Client2.txt)
2. Client should be able to have multiple outstanding SET() requests.
3. Servers should handle both clients in parallel (more than one operations at a time)

As in previous exercises, the same executable should be used for both client and server, depending on the command-line.