Milestone 1

Design

Our kvs relies on a database (register) of size 1025 * 6 where key n version k is stored in the n * 6 + k key of the register. The latest version is kept track in another db of size 1025. We have three handlers: get, put and range. Select is converted to range in the ingress. Each time through the ingress we create a new returnval and put the returnval in there before recirculating if necessary.

Implementation

The request is defined below and contains all the info needed to support each of get, put, range and select. For range, we split it up into many ranges in the send.py:

```
class Request(Packet):
    fields_desc = [
    BitField("requestType", 0, 16),
    BitField("key", 0, 32),
    BitField("endKey", 0, 32),
    BitField("val", 0, 32),
    BitField("op", 0, 32),
    BitField("version", 0, 32),
    BitField("recordType", 0, 16),
    ByteField("is_first", 0)
    ]
```

We use returnval to return values where is_first is used to keep track of if the returnval is the last one:

```
class ReturnVal(Packet):
    fields_desc = [
    BitField("val", 0, 32),
    ByteField("is_first", 0)
    ]
```

We use these two for storage as described in Design:

```
register<bit<32>>(1025 * 6) db;
register<bit<32>>(1025) latest_version;
```

They are invoked and updated in the get, put and range handlers:

```
action get_handler() {
        bit<32> val;
        db.read(val, (bit<32>) ((hdr.request.key * 6) + hdr.request.version));

        hdr.returnval[0].val = val;
    }

    action put_handler() {
        bit<32> vers;
        bit<32> newVers;

        latest_version.read(vers, (bit<32>) hdr.request.key);

        db.write((bit<32>) ((hdr.request.key * 6) + vers), hdr.request.val);

        newVers = vers + 1;
        latest_version.write((bit<32>) hdr.request.key, newVers);
    }

    action range_handler() {
        bit<32> val;
        db.read(val, (bit<32>) ((hdr.request.key * 6) + hdr.request.version));

        hdr.returnval[0].val = val;

        hdr.request.key = hdr.request.key + 1;
    }
```

The get handler reads the relevant index and attaches it onto the 0th returnval.
The put handler reads the latest version, adds 1 to it and then puts the value into the db.
The range handler acts similarly to the get handler except it updates the header request start key.

To handle range and select, we push the header as below:

```
hdr.returnval.push_front(1);
hdr.returnval[0].setValid();
```

There is also some additional processing to update the range and select in special cases in hw2.p4, e.g. if range key and endKey are equal then we change it to a get request. We change the select to a range query to avoid using any if statements in the handlers.

Testing

We tried a variety of tests with different orderings to ensure functionality worked as expected. See below:

[("PUT", 0, 10), ("PUT", 0, 12), ("PUT", 2, 5),
("GET", 0, 0), ("GET", 0, 1), ("GET", 0, 2), ("GET", 2, 0),
("RANGE", 0, 0, 0), ("RANGE", 0, 0, 1), ("RANGE", 0, 2, 0),
("SELECT", 5, "<", 0), ("SELECT", 0, "<=", 0), ("SELECT", 0, "==", 1)]
Which returns 10 for the first get, 12 for the second get, 0 for the third get, 5 for the fourth get.
10 for the first range, 12 for the second range, 10, 0, 12 for the third range
10, 0, 5, 0, 0 for the first select, 10 for the second select, 12 for the third select.

[("PUT", 0, 10), ("GET", 0, 0), ("PUT", 0, 11), ("GET", 0, 0)]
Returns 10, 10

[("PUT", 0, 10), ("GET", 0, 1), ("PUT", 0, 11), ("GET", 0, 1)]
Returns 0, 11

[("PUT", 1, 10), ("GET", 1, 1), ("RANGE", 1, 10, 1) ,("PUT", 1, 11), ("GET", 1, 1), ("RANGE", 1, 10, 1)]
Returns 0, [0,0,0,0,0,0,0,0,0,0], 11, [11,0,0,0,0,0,0,0,0,0]

[("PUT", 100, 10), ("PUT", 101, 10), ("PUT", 104, 10), ("RANGE", 100, 105, 0)]
Returns [10, 10, 0, 0, 10, 0]

[("PUT", 1015, 10), ("PUT", 1017, 11), ("PUT", 1020, 13), ("RANGE", 1016, 1021, 0),
("SELECT", 1017, ">", 0), ("SELECT", 1017, ">=", 0)]
Returns [0,11,0,0,13,0], [0,0,13,0,0,0,0], [11,0,0,13,0,0,0,0]

We also tested other test cases not listed here to stress the load. Remarkably very large selects and ranges can be often handled without breaking them up.