ELEC 477 Lab 1

Lucas Srigley (20289448)

Jamie Bell (20259237)

Nolan Steed (20295240)

February 3, 2026

Design Document

This lab implements a basic RPC by using a user datagram protocol (UDP). All programs run on the same machine and communicate over the loopback interface (127.0.0.1) using assigned UDP ports.

The design includes a Reverse Polish Notation calculator, which provides a 4-element operand stack that supports the following:

- push(value): inserts a value at the top of the stack, shifting stack values down and discarding the bottom value.
- pop(): removes the top value and shifts the stack upward.
- read(): returns the value at the top of the stack.
- swap(): swaps the top two elements of the stack.
- Arithmetic operations (+ - * /): remove the top two elements and apply the operation using the second stack value as the first operand, finally pushing the result.

The server owns the calculator state, which executes operations on behalf of the client. Communication between the two is done using UDP sockets and serialized messages with Google Protocol Buffers. Using a oneof keyword, all the procedures are handled with one protobuf message, which allows a single port to handle the operations.

Each RPC message begins with a header that contains:

- a magic number that identifies valid protocol messages
- a version number that allows future protocol changes
- a message identifier that identifies each request

A stub file is implemented in *rpn_client.cpp* that provides a local interface to the calculator. Each RPC call follows the following order in the file:

1. Increment the message identifier.
2. Construct a protobuf message with the header and the appropriate request payload.
3. Serialize the message and send it to the server using *sendto.*
4. Wait for a response using *recvfrom.*
5. Validate the magic number, version, and message identifier.
6. Return the result to the caller.

For the magic number, we selected a fixed 32-bit constant with version 1 to validate quickly.

The *rpn_server.cpp* creates the udp socket and binds it to the localhost port, which then enters a while loop that waits for incoming messages. The server follows the same algorithm that is outlined in the “Socket Programming” section of the lab handout. We used a receive timeout to avoid indefinite blocking if UDP packets are lost. On timeout, the stub returns a failed status rather than retransmitting.

Testing Document

As the assignment states, there will be no test results in this report. Instead, the console output from each test can be found in the individual text files that is included in our submission. All test outputs are in the folder test_outputs/. Each file has pass/fail checks printed from our client test programs showing that the expected behaviour has been occurred.

Test 1 - Push / Read / Pop

Output file: test_outputs/test1_output.txt

This test verifies the RPC correctness for stack operations listed below:

- Push(value): pushes a single value onto the stack and confirms that the RPC call succeeded.
- Read(): reads the top of the stack and confirms it matches the value that was pushed.
- Pop(): pops the stack and confirms the RPC call succeeds.
- Read() after pop: confirms that popping shifts the stack so the top value changes to the next element that is underneath.

In the output file, look for the clear “pass” messages for push and pop, and a confirmation that the read-back value is correct before and after the pop.

Test 2 - Independent Servers

Output file: test_outputs/test2_output.txt

This test verifies that calculator state is owned by each server and is isolated when multiple servers run on different ports:

- Two server instances are used (on two different ports – tested using 3600 and 3601).
- A separate client interacts with each server and pushes a different value to each one.
- Each client then reads back from its server and confirms the values aren’t mixed.
- A final independence check confirms each server still returns its own top of stack value.

In the output file, look for a section that shows operations on server 1 and server 2 (different ports), followed by a pass/ fail showing the servers are independent.

Test 3 - All Operations

Output file: test_outputs/test3_output.txt

This test validates the full RPC procedure set and the 4 element stack semantics:

- Arithmetic operations: addition, subtraction, multiplication, division. Each operation is tested by pushing two operands performing the operation and confirming the computed result matches the expectation.
- Swap(): Pushes two values, prints the top value before swap, performs swap, then prints the new top value after swap too confirm the top two elements were exchanged.
- 4 element push shifting and replication behavior on pop (“lift”): Pushes 4 values to fill the stack and confirm the top is the most recently pushed value. Pops repeatedly and reads the top after each pop to confirm the stack shifts upward correctly. Continues popping to show replication behavior.

In the output file, look for the “pass/fail” for each arithmetic operation and then a clear “before swap / after swap” section for swap, and a sequence of “Top after pop N” checks that ends with a final message confirming the stack replication (“lift”) behavior was seen.