527 — Computer Networks and Distributed Systems — Tutorial 2: RPC Implementation

E.C. Lupu

Note that the solution notes below only briefly list (some of) the key points that should be included in an answer. They are by no means complete. In an exam, you are expected to spell out the solution more fully and include a detailed explanation of your reasoning.

1 RPC Implementation

A client process accessing a remote server using Remote Procedure Call (RPC) has the following structure:

```
import IfType.h    // interface type specification
process client1 {
    IfType sref;
    ....
    sref  = lookup("sname", IfType, "nameserver")
                       //bind interface reference sref
    ....
    sref.op1(p1, p2, ....pn, result) // RPC on  op1 of server "sname"
    ....
}
```

Outline a simple optimised RPC communication protocol, assuming the IMPORT has already been successfully accomplished. The protocol must recover from communication errors and detect duplicate messages but assume input and output parameters fit into a single message. Your solution must indicate the actions performed at both client and server with respect to dealing with sequence numbers, how communication errors are recovered and what state information is maintained.

Solution Notes:

Client

Block after sending request so only one message transaction at a time. We can use a counter to generate transaction numbers, and hold requests until acknowledged. Send message with new transaction ID and start communications timeout.

If there is no reply when timeout expires, retransmit message. Repeat timeout and retransmission until the retry limit is exceeded then signal transaction failure to user and cancel timeout.

When $reply$ or ack received cancel timeout.

If the transaction ID in the reply is not equal to current transaction ID, it must be an old or duplicate reply so discard and send ack.

If $reply$ is valid send ack and pass $reply$ to user which is now unblocked.

Server

Saves active sender's source address, transaction IDs, and last transmitted reply in a table.

```
new request received:
  check table for source address and transaction ID.
      if (transactionID not in table} { // it is a new request
          pass the request to application process;
          enter transaction ID (& source) in table
```

```
      } else { // it is a duplicate request
         if (reply saved) { send reply }
        else { // the operation has not yet returned
         ack and discard request }
      }

application process returns:
       Get transaction ID from table;
       insert in message; send to source,
       save reply, start timeout.

       If ack received with correct transaction ID
       or new request received with  transaction ID > than one stored in table
       then   {cancel timeout:  discard reply but save transaction ID.

Timeout expires:
       retransmit reply.
       repeat  timeout  and retransmits until retry limit  exceeded;
       then discard reply.
```