Distributed Systems

Unassessed Coursework No. 4

Lecturer: M. Sloman

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

```
PROCESS client1

    USE iftype.h    -- interface type specification

    DECLARE  sref: iftype CLIENT

BODY

{   ....

    sref  := IMPORT ("sname", iftype, "nameserver", result)

                -- 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, detect duplicate messages but does not perform segmentation i.e. 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.

Distributed Systems

Solution Coursework No. 4

Client

Blocked after sending request so only one message transaction at a time - can use counter to generate transaction numbers, hold request until acknowledged
Send message with new transaction ID and start comms timeout.
If no reply when timeout expires retransmit message.
Repeat timeout and retransmits until retry limit exceeded then signal transaction failure to user and cancel timeout.
When reply or ack received cancel timeout .
If transaction ID in reply not equal to current transaction ID, must be old or duplicate reply so discard and send ack.
If reply 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.
If transaction ID > entry in table or no entry in table
then {new request so pass to user
and enter transaction ID (& source) in table
}
else // duplicate request
if reply saved
then send reply
else ack and discard request as receiver not yet generated reply.

User generates reply:
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.