The Solutions to Tutorial Questions and Lab Projects of Week 3

Tutorial Questions

1. The `Election` interface provides two remote methods:

   `vote`: with two parameters through which the client supplies the name of a candidate (a string) and the 'voter's number' (an integer used to ensure each user votes once only). The voter's numbers are allocated sparsely from the range of integers to make them hard to guess.

   `result`: with two parameters through which the server supplies the client with the name of a candidate and the number of votes for that candidate.

   Which of the parameters of these two procedures are input and which are output parameters?

   Answer

   `vote`: input parameters: name of candidate, voter's number;

   `result`: output parameters: name of candidate, number of votes

2. Discuss the invocation semantics that can be achieved when the request-reply protocol is implemented over a TCP/IP connection, which guarantees that data is delivered in the order sent, without loss or duplication. Take into account all of the conditions causing a connection to be broken.

   Answer

   A process is informed that a connection is broken

   - when one of the processes exits or closes the connection.
   - when the network is congested or fails altogether

   Therefore a client process cannot distinguish between network failure and failure of the server.

   Provided that the connection continues to exist, no messages are lost, therefore, every request will receive a corresponding reply, in which case the client knows that the method was executed exactly once.

   However, if the server process crashes, the client will be informed that the connection is broken and the client will know that the method was executed either once (if the server crashed after executing it) or not at all (if the server crashed before executing it).

   But, if the network fails the client will also be informed that the connection is broken.

   This may have happened either during the transmission of the request message or during the transmission of the reply message. As before the method was executed either once or not at all.

   Therefore we have at-most-once call semantics

3. A request-reply protocol is implemented over a communication service with omission failures to provide at-least-once RMI invocation semantics. In the first case the implementor assumes an asynchronous distributed system. In the second case the implementor assumes that the maximum time for the communication and the execution of a remote method is T. In what way does the latter assumption simplify the implementation?

Answer

In the first case, the implementor assumes that if the client observes an omission failure it cannot tell whether it is due to loss of the request or reply message, to the server having crashed or having taken longer than usual. Therefore when the request is re-transmitted the client may receive late replies to the original request. The implementation must deal with this.

In the second case, an omission failure observed by the client cannot be due to the server taking too long. Therefore when the request is re-transmitted after time T, it is certain that a late reply will not come from the server. There is no need to deal with late replies.

4. A client makes remote procedure calls to a server. The client takes 5 milliseconds to compute the arguments for each request, and the server takes 10 milliseconds to process each request. The local operating system processing time for each send or receive operation is 0.5 milliseconds, and the network time to transmit each request or reply message is 3 milliseconds. Marshalling or unmarshalling takes 0.5 milliseconds per message.

   Calculate the time taken by the client to generate and return from two requests:

   (i) if it is single-threaded, and

   (ii) if it has two threads that can make requests concurrently on a single processor.

   You can ignore context-switching times. Is there a need for asynchronous RPC if client and server processes are threaded?

Answer

i) time per call = calc. args + marshal args + OS send time + message transmission + OS receive time + unmarshall args + execute server procedure + marshall results + OS send time + message transmission + OS receive time + unmarshal args

= 5 + 4 * marshal/unmarshal + 4 * OS send/receive + 2* message transmission + execute server procedure

= 5 + 4 * 0.5 + 4 * 0.5 + 2 * 3 + 10 ms

= 5 + 2 + 2 + 6 + 10

= 25 ms.

Time for two calls = 50 ms.

ii) threaded calls:

client does calc. args + marshal args + OS send time (call 1)

= 5 + 0.5 + 0.5 = 6

then calc args + marshal args + OS send time (call 2) = 6

= 12 ms then waits for reply from first call

server gets 1st call after

message transmission + OS receive time + unmarshal args

= 6 + 3 + 0.5+ 0.5 = 10 ms, takes 10 + 1 to execute, marshal,

send at 21 ms

server receives 2nd call before this, but works on it after 21 ms taking 10 + 1, sends it at 32 ms from start

client receives it 3 + 1 = 4 ms later i.e. at 36 ms (message transmission + OS receive time + unmarshal args) later

Time for 2 calls = 36 ms.

5. Briefly discuss the advantages and disadvantages of the middleware approach.

   Answer

   Middleware is a layer of service between OS and distributed applications. The advantages of middleware approach include (1) Hide complexity and heterogeneity of distributed systems. The middleware approach makes distributed programming independent from OS, network protocols and programming language. (2) Bridge gap between lower OS communications and programming abstraction; and (3) Provide standard services (such as naming services, transaction services etc). The major disadvantage is that a lot of standard OS services must be re-implemented by the middleware.

Lab Projects

Download the source code from Week 3 block of the unit Moodle site.