UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 3
Solution

Lecture Distributed Systems
Summer Term 2020

Exercise 1: RPC and unions

If the runtime system is unable to determine which value type is in the field, it cannot pack it correctly. This means that unions cannot be permitted in an RPC system, unless there is an identifying field that uniquely describes what is contained in the variant field. The identifying field must not be under the control of the user.

Exercise 2: Remote object references

Using Java we can express such an implementation e.g. by the following class:

```
public class Object_reference {
  InetAddress server_address;     // Network address of the object server
  int server_port;                // Port that identifies the server
  int object_id;                  // Identifier for this object
  URL client_code;                // (Remote) file containing the code of the
                                  // client stub
  byte[] init_data;               // Possible additional initialization
                                  // data for the stub
}
```

The object reference should contain at least the transport layer address of the server on which the object is located, including the port. We also need an object ID, because the server can contain several objects. In our implementation, we use a URL to point to a (remote) file that contains all the required client-side code. An alternative implementation would be to directly use a serialized client stub as an object reference. For example, this approach is followed in Java RMI, where proxies are passed as a reference.

Exercise 3: Exceptions

Because the exceptions are originally thrown on the server side, the server stub can only catch the exception, package it as a special error response, and return it to the client. The client stub, on the other hand, must unpack the message and throw the same exception to keep the access to the server transparent. Consequently, the exceptions must also be described in an interface definition language.

Exercise 4: Threads

a) Total time = computation of arguments + marshalling arguments + processing time in operating system send + network time for transmission + processing time in operating system receive + unmarshalling arguments + processing time in server + marshalling results + processing time in operating system send + network time for transmission + processing time in operating system receive + unmarshalling arguments

= 5 + 4 * marshalling/unmarshalling + 4* processing time in operating system + 2 * network time for transmission + processing time in server
= 5 + 4* 0.5 + 4* 0.5 + 2 * 3 + 10ms = 25ms

For two requests the client needs 2*25ms = 50ms.

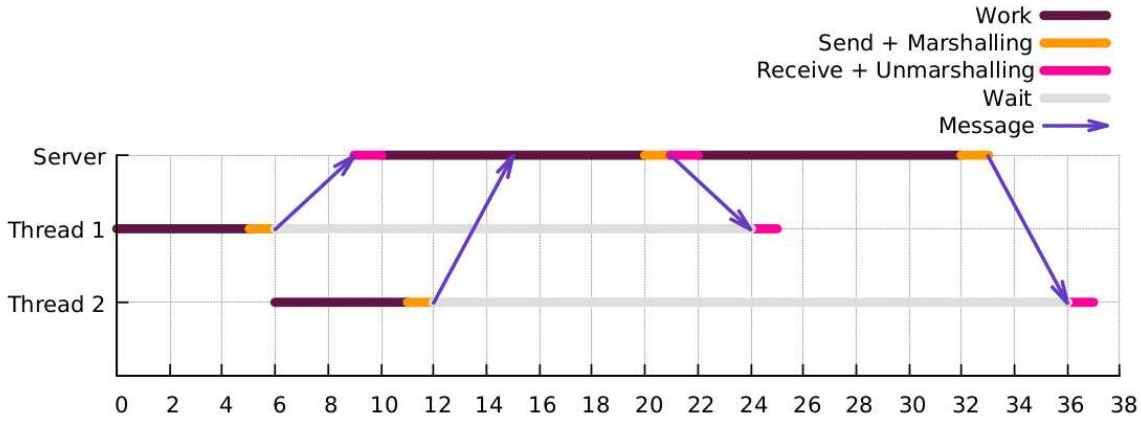
b) Client:
1st request: calculation of arguments + marshalling arguments + processing time in operating system send = 5 + 0.5 + 0.5 = 6, then
2nd request: calculation of arguments + marshalling arguments + processing time in operating system send = 5 + 0.5 + 0.5 = 6 = 12ms, then waiting for response from 1st call

Server:
The 1st call reaches the server node after 6 + 3 = 9 ms.
What follows is processing time in operating system receive + unmarshalling arguments = 0.5 + 0.5 = 1ms, so that the server process gets the request 10ms after the start.
This is followed by 10ms processing time of the server, 0.5ms each for marshalling the results and processing time in operating system send, therefore the response is sent after 21ms. The server node receives the second request in the meantime, but only processes it after 21ms, so it sends the response again 12ms later (0.5 ms each for processing time in operating system receive + unmarshalling arguments, 10ms processing time, 0.5 ms each for marshalling the results and processing time in operating system send), i.e., 33ms after the start.

Client:
The client receives the answers 3 + 1 = 4ms later.
(network time for transmission + processing time in operating system receive + unmarshal arguments)

Thus, the total time is 37ms.

In the following graphics, the execution is represented as a Gantt diagram:



Exercise 5: RMI basics and architecture

See chapter 2 and 3 of the lecture.

Exercise 6: Programming: Java RMI - Hello-World!

You will find the solution to this problem in the archive `l03eFiles.zip`[1] on the lecture's web page.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l03eFiles.zip`