UNIVERSITÄT SIEGEN

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

Excercise Sheet 2

(To be processed until 31.10.)

Lecture Distributed Systems Winter Term 2024/25

Exercise 1: Communication forms in distrubited systems

There are basically two types of communication in distributed systems: reliable message streams and unreliable datagrams. What are the main characteristics of both forms of communication and what are their advantages and disadvantages? Name possible areas of application.

Exercise 2: Conversion of data formats

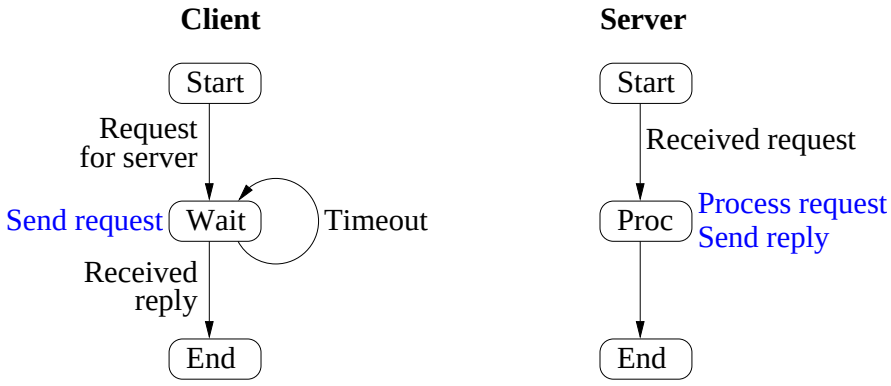
When communicating in distributed systems, it is generally necessary to convert data structures stored internally in the processes into a format suitable for transmission. For what reasons is this indispensable? Discuss necessary measures for different data types.

Exercise 3: Semantics of client/server communication (Mandatory exercise for 6 CP, submit via moodle!)

a) Client/server communication can cause problems if messages are lost during transmission between client and server. The handling of such errors can be different, resulting in different semantics for communication:

- *at least once* - the operation is executed at least once,
- *at most once* - the operation is executed at most once, and
- *exactly once* - the operation is executed exactly once.

The following figure shows state diagrams for client and server with an implementation of *at least once* semantics:



Input events (e.g. incoming messages, timeout) are shown in black, the action or output of the state machine in the respective state in blue.

Sketch analog state diagrams for implementations of the *at most once* and *exactly once* semantics using incoming messages and timeout events as above. Use sequence numbers to identify individual requests for the *exactly once* semantics.

b) For the following applications, consider whether *at least once* or *at most once* semantics is more appropriate:

- pressing an elevator button,
- translating a program,
- write/append data to a file,
- ordering a pizza,
- to get a bank account statement,
- to make an electronic transfer,
- cast one vote in an electronic voting service.

Exercise 4: Request/reply protocol

For a typical client-server protocol in request/reply style, it makes sense to use a protocol based on datagrams. Why? Design such a protocol.

Pay particular attention to all conceivable error scenarios and how they should be handled. The protocol should be as lightweight as possible. For example, the datagram layer may offer the following operations as a service:

- *send(in address, in data)* - asynchronous, unreliable transmission of data,
- *recv(out address, out data, in timeout)* - blocking receive of a packet with timeout.

The following operations are to be implemented:

- *doOperation(in address, in request, out reply)* - synchronous sending of a request to the server,
- *getRequest(out address, out request)* - blocking reception of a request (server),
- *sendReply(in address, in reply)* - sending the reply (server)

You may assume that fragmentation of the messages to be transmitted is not necessary. Furthermore, it is guaranteed that no data packets are corrupted. However, any packet loss or non-compliance with the packet sequence must be taken into account.

Exercise 5: Middleware

A reliable multicast service allows a sender to deliver reliable messages to multiple recipients. Is such a service part of a middleware layer or should it be part of an underlying layer?

Exercise 6: Transparency of RPC (Mandatory exercise for 6 CP, submit via moodle!)

If *Remote Procedure Call* (RPC) is used, it does not matter whether the called procedure (i.e. the server process) is located on the local computer node or on another node. But what happens when the procedure executes a system call? What problems could this cause, and how could they be addressed?