Written examination in Distributed Systems

July 08, 2022

Last name: _____

First name: _____

Student number: _____

Signature: _____

Mit dem Bearbeiten dieser schriftlichen Prüfung (Klausur) bestätigen Sie das Sie diese alleine bearbeiten und das sie sich gesund und prüfungsfähig fühlen. Mit dem Erhalt der Aufgabenstellung gilt die Klausur als angetreten und wird bewertet.

By attending this written exam, you confirm that you are working on it alone and feel healthy and capable to participate. Once you have received the examination paper, you are considered to have participated in the exam, and it will be graded.

- Use the provided sheets. Do *not* use own paper.
- You are allowed to use a *self prepared*, *double sided DIN-A4 sheet* in the exam.
- You are allowed to use a non-programmable calculator.
- Do *not* use a red pen.
- Please note that the pages are printed on *both sides*.
- Time limit: *90 minutes*
- Turn off your mobile phones!

*Result:

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	10	10	6	13	12	11	7	7	9	4	89
Score:											

1.0: 89-84.5, **1.3**: 84.5-80, **1.7**: 80-75.5, **2.0**: 75.5-71, **2.3**: 71-67,
2.7: 67-62.5, **3.0**: 62.5-58, **3.3**: 58-53.5, **3.7**: 53.5-49, **4.7**: 49-44.5, **5.0**: <44.5

Question 1

Points: ..... (max. 10 points)

Decide whether the following statements are correct or wrong and explain shortly why.

(a) When asymmetrical addressing is used only the receiver (server) names the other side. ☐ True ☑ **Wrong**

> **Solution:** Only the sender (client) names the address of the server.

(b) Synchronous communication can be implemented with less memory resources than asynchronous communication. ☑ **True** ☐ Wrong

> **Solution:** No buffer is required.

(c) A binder in an RPC system can be used to register an interface. ☑ **True** ☐ Wrong

> **Solution:** The server exports its interface to the binder.

(d) UNIX user IDs are organized in flat name spaces. ☑ **True** ☐ Wrong

> **Solution:** UNIX UIDs do not establish a hierarchy.

(e) A name service can be used to lookup files in a file system, a directory service is used to lookup directories. ☐ True ☑ **Wrong**

> **Solution:** A name service provides name resolution.

(f) When data is often modified more replications are preferable to increase the performance of the write operations. ☐ True ☑ **Wrong**

> **Solution:** Modifications can only happen on the primary and are negative for the performance with many replicas.

(g) An NTP server with a higher stratum level is more accurate. ☐ True ✓ **Wrong**

Solution: A lower stratum level is closer to a reference time source.

(h) Cristian's algorithm can be used to synchronize vector clocks. ☐ True ✓ **Wrong**

Solution: It is used to synchronize computer clocks to a reference time source.

(i) `PUT` and `POST` are idempotent operations. ☐ True ✓ **Wrong**

Solution: Only PUT is idempotent.

(j) When using the automounter in NFS typically no action is performed at boot time. ✓ **True** ☐ Wrong

Solution: With automounter the actual mounting happens only when the directory is access for the first time.

Question 2

Points: ..... (max. 10 points)

Question	Answer
What needs to be implemented in order to use a *at-most-once* semantics in an RPC system?	**Solution:** Duplicate detection on the server.
What is the *orphan problem* in an RPC system?	**Solution:** A client process dies after invoking a RPC.
How is *location transparency* guaranteed in NFS?	**Solution:** Not at all — only by convention.
What is the validity period of the key used for symmetric encryption in TLS?	**Solution:** It is only valid for one session.
Which communication pattern allows for transparent sending of a message to multiple receivers?	**Solution:** Publish/Subscribe
Which property has to be given for a hash function in order to be used as a cryptographic hash function?	**Solution:** One-way function
Which relationship must be given for a pair of *lamport timestamps* if two events are concurrent?	**Solution:** The causal (in)dependence cannot be determined using lamport clocks.
Why does the application developer need to know the error semantics of a given RPC system?	**Solution:** In order to potentially deal with the corresponding consequences of an error in the application.
What does a GPS based clock provide?	**Solution:** An approximation of the real physical time.
What is meant as *strict consistency* for a (distributed) file system?	**Solution:** All modifications are immediately visible for all participants.

Question 3

Points: .....(max. 6 points)

Consider the following network protocols

Mark in the following table which of the given network protocols work connection-oriented or connection-less.

Abb.	**Protocol**	**conn.-oriented**	**conn.-less**
ICMP	Internet Control Message Protocol		✓
IP	Internet Protocol		✓
SMTP	Simple Mail Transfer Protocol	✓	
TCP	Transmission Control Protocol	✓	
HTTP	HyperText Transfer Protocol	✓	
UDP	User Datagram Protocol		✓
SNMP	Simple Network Management Protocol		✓
FTP	File Transfer Protocol	✓	
TFTP	Trivial File Transfer Protocol		✓
NTP	Network Time Protocol		✓
TELNET	Remote Terminal Protocol	✓	
SSH	Secure Shell	✓	

Question 4

Points: ..... (max. 13)

The following stub code sends a string of variable length over a previously opened socket.

```
#include <stdint.h>
#include <string.h>
#include <unistd.h>

int write_string(int sock, char *string)
{
  int len = strlen(string);
  int16_t length_hdr = len;
  uint8_t *message = malloc(len + sizeof(length_hdr));
  memcpy(&message[0], &length_hdr, sizeof(length_hdr));
  memcpy(&message[sizeof(length_hdr)], string, len);
  if(write(sock, message, len  + sizeof(length_hdr)) < 0)
  {
    return -1;
  }
  free(message);
  return 0;
}
```

(a) What is the maximum length of a string to be sent in this function? Why? (1)

> **Solution:** Since the variable `length_hdr` has 16 bit and is signed the maximum value is $2^{15} - 1 = 32767$. This is the maximum string length.

(b) What happens if this limit is exceeded? (1)

- ☐ Program abort due to dereferencing an invalid pointer
- ☐ No abort, but only a part of the string gets transmitted
- √ **Undefined (cannot be answered without knowledge of the receive function)**

(c) Would it be better if the variables `len` and `length_hdr` in the code above would **not** be declared as *signed*[1]? (2)

- ☐ **If no**: Why not? Which problems would result in this case?

- √ **If yes: What are the consequences for the answers to the questions a) and b)?**

  Maximum string length in that case: $2^{16} = 65536$ **bytes**

- ☐ Program abort due to dereferencing an invalid pointer
- √ **No abort, but only a part of the string gets transmitted**
- ☐ Undefined (cannot be answered without knowledge of the receive function)

[1] i. e., as `unsigned int` resp. `uint16_t`

(d) Put the content of the message which has been sent over the socket into the form below. Assume that the function gets called as shown below on an *x86* architecture, i. e., a little endian system: (2)

```
send_string(sock, "Hello\n");
```

Note: One box represents one byte (8 bit). Put either a double digit hexadecimal number or an ASCII character in each box.

0x06	0x00	'H'	'e'	'l'	'l'	'o'	'\n'				

(e) Outline an appropriate receive function below. On success the function shall return a pointer to the null-terminated received string. The required memory for the string should be allocated with `malloc()`. In the error case the function shall return `NULL` and not allocate any memory. (For the reception of a message the well-known system call `read(int sock, uint8_t *buffer, size_t length)` shall be used.) (3)

```
#include .... (like above)

char *read_string(int sock)
{

  int16_t length_hdr;

  char *retval;
```

Solution:

```
  if(read(sock, &length_hdr, sizeof(length_hdr)) < 0)
  {
    return NULL;
  }
  if(length_hdr < 0)
  {
    return NULL;
  }
  if((retval = malloc(length_hdr + 1) == NULL)
  {
    return NULL;
  }
  if(read(sock, retval, length_hdr) < 0)
  {
    free(retval);
    return NULL;
  }
  retval[length_hdr] = '\0'
```

```
  return retval;
}
```

(f) As long as all participating nodes use the same architecture (here: x86) everything works as expected. As soon as one of the nodes (either sender or receiver) uses a SPARC or PowerPC processor (big endian architectures!), errors occur. Why? Which behavior would you expect for the transmitted string in task d) on the receiver side? Would the receiver crash? (2)

Solution: The length of the string is transmitted incorrectly because of the differing data representation. In the example the receiver expects a length of $0x0600 = 1536$ instead of $0x0006 = 6$. The receiver would interpret any further bytes received afterwards as continuation of the string. The receiver would probably not crash but block until 1536 bytes in total have been received.

(g) How would you solve this problem? Complete the code below by using the one or multiple of the following functions: `htons()`, `htonl()`, `ntohs()`, `ntohl()`. Describe your changes in one or two sentences. (2)
Description:

Solution: The length must be converted from host byte order into network byte order. Since it is a 16 bit (short) value, the corresponding function is `htons()`.

```
#include .... (like above)

int send_string(int sock, char *string)
{
  int len = strlen(string);

  int16_t length_hdr = len;

  uint8_t *message = malloc(len + sizeof(length_hdr));
```

Solution:

```
 length_hdr = htons(len);
```

```
  memcpy(&message[0], &length_hdr, sizeof(length_hdr));

  memcpy(&message[sizeof(length_hdr)], string, len);

  if(write(sock, message, len  + sizeof(length_hdr)) < 0)
  {

    return -1;
  }

  free(message);

  return 0;
}
```

Question 5

Points: ..... (max. 12)

Consider the following two C code snippets of function pairs for **connection-less** message exchange with **direct addressing**:

SYNOPSIS A:

```
#include <messageinterface.h>

/**
 * Send message pointed to by
 * <msg> to <peer>
 */
void SEND_A(
      node_t peer,
      const message_t *msg);

/**
 * Receive message from <peer>,
 * store it to buffer pointed to
 * by <msg>
 */
void RECEIVE_A(
      node_t peer,
      message_t *msg);
```

SYNOPSIS B:

```
#include <messageinterface.h>

/**
 * Send message pointed to by
 * <msg> to <peer>
 */
void SEND_B(
      node_t peer,
      const message_t *msg);

/**
 * Receive message from anyone.
 * Store it to buffer pointed to
 * by <msg>, store originator
 * of message to <peer>
 */
void RECEIVE_B(
      node_t *peer,
      message_t *msg);
```

(Assume that a header file `messageinterface.h` defines the type `node_t` for addressing a peer and `message_t` for the structure of message.)

(a) Are we dealing with implicitly typed, explicitly typed, or untyped messages? (Don't forget to explain why!) (2)

> **Solution:** These are implicitly typed messages: The structure of the data is specified and implicitly expected from all participants. No explicit type information is transmitted.

(b) Why is the pointer to the message (`msg`) for the `SEND` functions marked as `const` but not for the `RECEIVE` functions? (1)

> **Solution:** The content of the message buffer is only read for sending. For the reception it is written.

(c) Which of the given code snippets is suited for the use in a server and which is suited for a client? (2)

> **Solution:** Version „A" is suited for a client, „B" for a server: a server is typically used by multiple clients and hence cannot know its communication partner when calling `RECEIVE()` but will only get to know its address afterwards.

(d) The following outlines the main application of a simple RPC server. Put the calls to the selected funtion pairs at the correct spots and complete the server application in the process. (4)

```
#include <messageinterface.h>

static int fun;

void main()
{
      node_t client_id;
      message_t message_buffer;
      int result;
      int server_running = 1;
      unsigned int arg1, arg2, arg3;
```

```
    while ( server_running )
    {
```

Solution:

```
        RECEIVE_B(&client_id, &message_buffer); /* 2P */
```

```
        fun = server_unmarshal(&message_buffer, &arg1, &arg2, &arg3);


        result = call_server_fun(arg1, arg2, arg3);


        server_marshal(&message_buffer, result);
```

Solution:

```
        SEND_B(client_id, &message_buffer); /* 2P */
```

```
    }


}

/**
 * Helper function for invoking server services.
 * For simplicity, all services have identical API.
 */
int call_server_fun(int arg1, int arg2, int arg3)
{
    int result;
    switch(fun)
    {
      case 0:
        result = fun0(arg1, arg2, arg3);
        break
      case 1:
        result = fun1(arg1, arg2, arg3);
        break
      case 2:
        result = fun2(arg1, arg2, arg3);
        break
      .......
      case X:
        result = funX(arg1, arg2, arg3);
        break
    }
    return result;
}
```

(e) Assume that the communication channel has a capacity of N messages.[2] How many requests by clients can the server handle in parallel? (Don't forget to explain why!) (1)

Solution: Exactly one. The server is not multithreaded.

(f) Could the number of client requests which can be handled concurrently increased by running the `while()` in parallel multiple times – for instance, by using threads. Why or why not? *Note:* The function `call_server_fun()` as specified above remains unchanged. (2)

Solution: No, the function is obviously not reentrant. Already because of the use of the static variable `fun`.

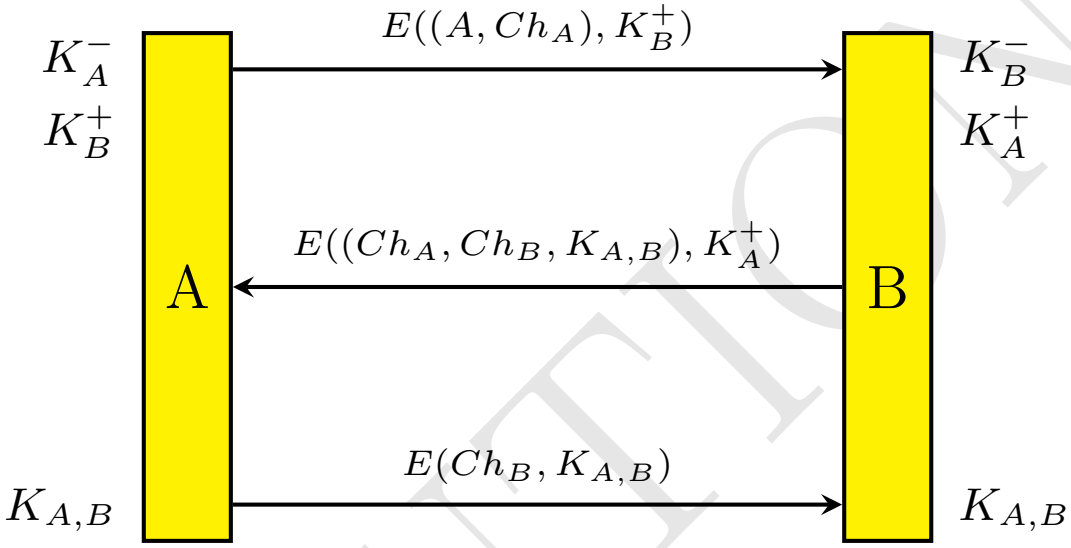
[2] I.e., `SEND()` can be called up to N times before the function blocks ($N \in \mathbb{N}, N > 0$).

Question 6

Points: ..... (max. 11)

Describe the principle of a **challenge-response protocol** for **authentication** using **public** keys: the nodes A and B have the following objects and methods available:

- Private key of A: K_A^-
- Public key of A: K_A^+
- Cipher: $E()$
- Random number generator to create challenges: $R()$
  Can be used to generate random numbers, which can be used as challenges, e. .g., $Ch_A = R()$. Each call to $R()$ provides a new, non-predictable number.



The communication is initiated at node A by sending a communication request „A“ to B (see above).

(a) Is a symmetric or asymmetric encryption used for authentication? Why? (1)

Solution: Asymmetric. A and B each owns a pair of keys.

(b) Complete the diagram above with the required messages. (2)

(c) After the depicted communication A and B can communicate in an encrypted form. Which key is used for this and who has generated it? (2)

Solution: The symmetric session key $K_{A,B}$ is created by B.

(d) Do A and B have to exchange information (e. g., keys) before the depicted communication? Why or why not? (1)

Solution: No. They require only the corresponding public key of the other participant.

(e) Can an attacker that overhears the first message repeat this message? Which benefit would they have? (2)

Solution: Yes, but without the private key of A it cannot decipher the response of B correctly and hence, cannot read the generated session key. I. e., it does not gain anything by doing so.

(f) What must be assured in order to consider the method secure? (2)

Solution: The association of a public key to the real person must be ensured (trustworthy PKI). Appropriate symmetric and asymmetric encryption methods with sufficiently long keys have to be selected.

(g) Which protection goals are accomplished with the challenge-response protocol? (1)

- √ **Confidentiality**
- √ **Privacy**
- √ **Integrity**
- √ **Authenticity**
- √ **Accountability**
- □ Availability

Question 7

Points: ..... (max. 7)

(a) What is the advantage of resolving names via broadcast? (1)

Solution: No previous knowledge/well-known address is required.

(b) To resolve IPv4 addresses the *Address Resolution Protocol (ARP)* uses a broadcast procedure. Why is this protocol not well-suited to resolve domain names in the Internet? (1)

Solution: Broadcast methods do not scale and are not suited beyond subnet boundaries.

(c) The URIs *urn:ietf:rfc:2141* and *https://tools.ietf.org/html/rfc2141* both points to the same document. Which URI scheme is it in each case and how do these schemes differ? (2)

Solution: The first is a Uniform Resource Name (URN), the second is a Uniform Resource Locator (URL). A URN identifies a resource in persistent and location-independent manner, a URL specifies location and access method.

(d) Are email addresses unique? (1)

Solution: Yes.

(e) How is the namespace for email addresses structured? (1)

Solution: In a hierarchical namespace.

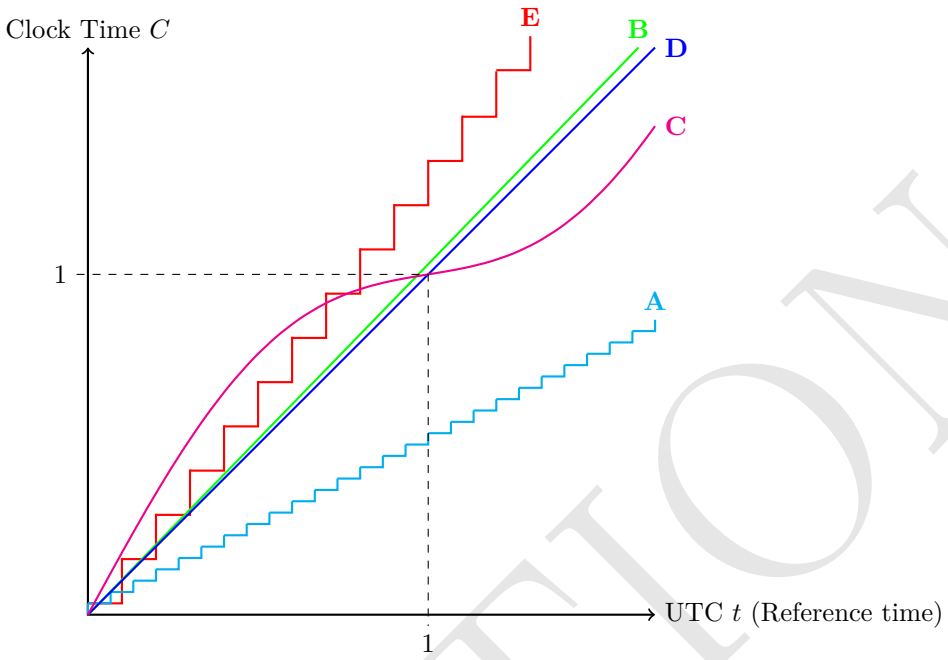
(f) The web server of the university is reachable via the IPv4 address `192.109.234.218`. Why does the server deliver a different web page if you enter `https://192.109.234.218` in the address bar of your browser compared to when entering `https://www.frankfurt-university.de/`? (1)

Solution: The server may deliver multiple web pages. Only by specifying the host name the request unambiguously specifies which web page shall be delivered.

Question 8

Points: .....(max. 7 points)

The following diagram shows the progress of time measured with the different clocks A, B, C, D, and E compared to the reference time.



Characterize the properties of these clocks wrt

- Accuracy
- Stability, and
- Resolution.

- Clock „A“:

  Solution: Inaccurate (too slow), good stability, low resolution

- Clock „B“:

  Solution: Rather good accuracy (slightly too fast), good stability, high resolution

- Clock „C“:

  Solution: Medium accuracy, but low stability, high resolution

- Clock „D“:

  Solution: ideal clock

- Clock „E“:

  Solution: Inaccurate (too fast), good stability, low resolution

Question 9

Points: ..... (max. 9)

(a) Name at least four problems which needs to be solved by a distributed file system: (4)

Solution:

- Visible overall structure
- Naming
- Programming interface
- Mapping to physical memory
- Integrity
- Security and access control
- Concurrency control
- Access model
- Location
- Replication
- Availability

(b) In a distributed system the Java runtime environment (JRE) shall be provided via a distributed file system. Which consistency semantics would you recommend? (1)

Solution: Read-Only files.

(c) For which cases would you recommend a stateful server in a distributed file system? (2)

Solution: If performance is important, the load on the network shall be reduced, or file locks shall be provided.

(d) In order to increase scalability, a company would like to launch a storage network. For which use cases would you recommend a *Storage Area Network (SAN)* and when you rather recommend a *Network Attached Storage (NAS)* system? Explain your decision. (2)

Solution: SAN systems are well suited for applications working on block storage (e.g., database systems), NAS systems for applications which are based on file access. Both systems can be combined.

Question 10

Points: ..... (max. 4)

Given a UNIX system with a clock which has a positive drift with respect to a reference time. The clock runs stable with a high resolution.

(a) The clock is used by multiple processes in order to assign timestamps to events. Does the clock fulfills the clock condition? (1)

> **Solution:** Clock condition: $a \rightarrow b \Rightarrow C(a) < C(b)$
> Here $C(e)$ refers to the (logical) time at which the event e happens.
> The clock fulfills the condition since the generated time stamps are steady and monotonously increasing.

(b) Periodically an accurate timestamp from a reference time source is obtained. Which possibilities exist to manipulate the clock using the provided timestamp of the reference time source in order to get closer to the reference time? (2)

> **Solution:** The function `settimeofday` can be used to compensate the offset. The function `adjtime` can be used to compensate the drift.

(c) Which of the possibilities from task b would effect the clock condition? (1)

> **Solution:** Using `settimeofday` may hurt the clock condition since the clock would be reset (run backwards).
> `adjtime` can be used without affecting the clock condition.