Written examination in Distributed Systems

July 15, 2024

Last name: ______________________________

First name: ______________________________

Student number: ______________________________

Signature: ______________________________

Written examination in Distributed Systems

July 15, 2024

Please write only your student number — but **not your name** — on this or any of the following sheets. By omitting your name a pseudonymized correction of your exam can be achieved. The first page with your name will be removed before correction and consequently the corrector cannot be biased when correcting your exam. By putting your student number on all pages you make sure that even in the case the stapling gets lost each page can be attributed to your exam.

Student number: ______________________________

Result:

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	10	10	13	5	12	11	8	9	8	4	90
Score:											

1.0: 90-85.5, **1.3**: 85-81, **1.7**: 80.5-76.5, **2.0**: 76-72, **2.3**: 71.5-67.5,
2.7: 67-63, **3.0**: 62.5-58.5, **3.3**: 58-54, **3.7**: 53.5-49.5, **4.7**: 49-45, **5.0**: <45

Question 1

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

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

(a) When a component in a distributed system can be moved without changing the user interface it is called *location transparent*. ☐ True ☐ Wrong

(b) TCP is a good choice for the transport layer protocol if latency is most important. ☐ True ☐ Wrong

(c) Sockets can be used in almost any programming language on most operating systems. ☐ True ☐ Wrong

(d) Shared memory can be used for IPC in a distributed system. ☐ True ☐ Wrong

(e) RPC can be used to achieve access transparency. ☐ True ☐ Wrong

(f) LDAP is a popular example for a directory service. ☐ True ☐ Wrong

(g) An ISBN is a pure name. ☐ True ☐ Wrong

(h) GPS receivers or atomic clocks can be used as reference time sources. ☐ True ☐ Wrong

(i) Lamport or vector clocks can be used to solve the causality problem. ☐ True ☐ Wrong

(j) Interactions in a REST service require sufficient memory on the server to store the state. ☐ True ☐ Wrong

(k) Distributed file systems require always a dedicated file server. ☐ True ☐ Wrong

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?	
What is the *orphan problem* in an RPC system?	
How is *location transparency* guaranteed in NFS?	
What is the validity period of the key used for symmetric encryption in TLS?	
Which communication pattern allows for transparent sending of a message to multiple receivers?	
Can an HTTP method be safe but not idempotent?	
Which relationship must be given for a pair of *Lamport timestamps* if two events are concurrent?	
Why does the application developer need to know the error semantics of a given RPC system?	
What does a GPS based clock provide?	
What is meant as *strict consistency* for a (distributed) file system?	

Question 3

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)

(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: ______________ 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.

(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;














  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)

(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:

```
#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));



  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 4

Points: .... (max. 5 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		
FTP	File Transfer Protocol		
NTP	Network Time Protocol		
TELNET	Remote Terminal Protocol		
SSH	Secure Shell		

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:	**SYNOPSIS B:**

```
#include <messageinterface.h>

/**
 * Send message pointed to by
 * <msg> to <peer>
 */
void SEND_A(
      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_A(
      node_t *peer,
      message_t *msg);
```
	```
	#include <messageinterface.h>

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

	/**
	 * Receive message from <peer>,
	 * store it to buffer pointed to
	 * by <msg>
	 */
	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)

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

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

(d) The following outlines the main application of a simple RPC server. Put the calls to the selected function 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 )
     {


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



          result = call_server_fun(arg1, arg2, arg3);



          server_marshal(&message_buffer, result);


     }


}
```

```
/**
 * 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)

(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)

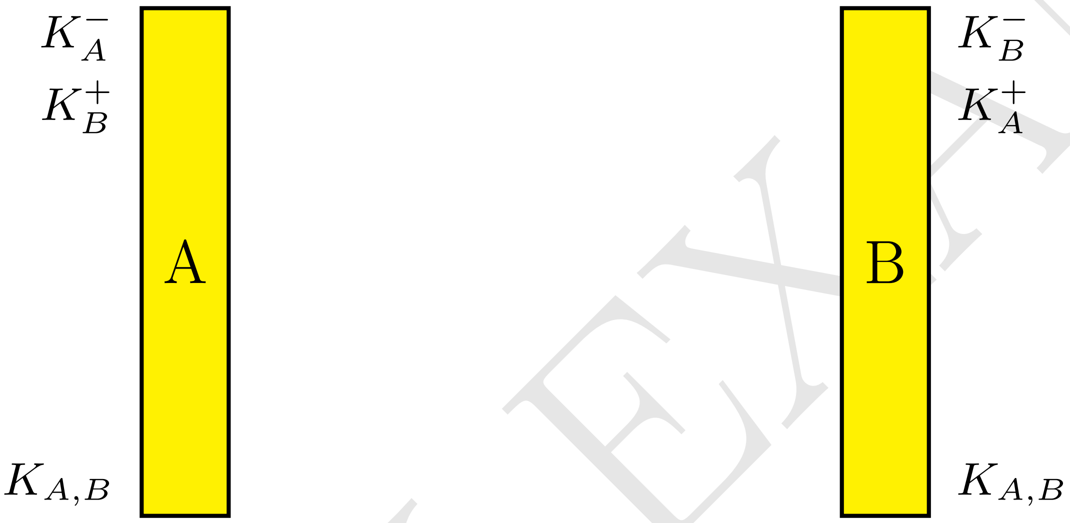
[2]I.e., `SEND()` can be called up to N times before the function blocks ($N \in \mathbb{N}, N > 0$).

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)

(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)

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

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

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

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

- ☐ Confidentiality
- ☐ Privacy
- ☐ Integrity
- ☐ Authenticity
- ☐ Accountability
- ☐ Availability

Question 7 **Points:** ....(max. 8)

(a) Describe how to peers can communicate with each other without knowing each others addresses using the MQTT protocol. (2)

(b) Describe how a MQTT client can select the required *error semantics* for the application. (1)

(c) Inspect the following code and describe what it does. What prerequisite needs to be fulfilled? Check if the logic of the contains makes sense. (3)

```
const char *topic = "roomman/room/+/reserved";
MQTTClient_message pubmsg = MQTTClient_message_initializer;
pubmsg.payload = "true";
pubmsg.payloadlen = strlen(msg) + 1;
MQTTClient_publishMessage(client, topic, &pubmsg, &token);
MQTTClient_waitForCompletion(client, token, ROOMMAN_MQTT_TIMEOUT);
```

The function `MQTTClient_publishMessage()` can be used to publish the message included in `pubmsg`. The parameter `client` points to MQTT client object. We assume that it has been properly initialized before. The parameter `topic` is used to specify the name of the topic. The parameter `token` is an output parameter and irrelevant for this task.

(d) The paho MQTT library provides an API to **publish** messages asynchronously or synchronously. For subscribing it only provides an asynchronous API. Explain the reasoning for this design decision. (2)

Question 8 **Points:** ....(max. 9)

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

(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)

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

(d) Assume that the directory `/home` is exported by an *NFS* and an *AFS* server. Explain how this directory may become accessible on a corresponding client for both systems. (2)

Question 9

Points: ....(max. 8)

Consider the following interface definition for a SunRPC based application:

```
struct roomman_create_room_arg {
        char building_name[32];
        char room_name[32];
        uint16_t     capacity;
};

struct roomman_lookup_arg {
        char building_name[32];
        char room_name[32];
};

struct roomman_update_capacity_arg {
        int32_t rid;
        uint16_t capacity;
};

program ROOMMAN_CREATE_PROG {
        version ROOMMAN_CREATE_VERS {
                int CREATE_ROOM(roomman_create_room_arg) = 1;
        } = 1;
} = 0x47110001;

program ROOMMAN_LOOKUP_PROG {
        version ROOMMAN_LOOKUP_VERS {
                int LOOKUP(roomman_lookup_arg ) = 1;
        } = 1;
} = 0x47110002;

program ROOMMAN_DELETE_PROG {
        version ROOMMAN_DELETE_VERS {
                int DELETE(int ) = 5;
        } = 1;
} = 0x47110003;

program ROOMMAN_UPDATE_PROG {
        version ROOMMAN_UPDATE_VERS {
                int UPDATE(roomman_update_capacity_arg ) = 6;
        } = 1;
} = 0x47110004;
```

(a) Describe the necessary steps to create client application based on this interface description. (2)

(b) Which service needs to be running on the host which runs the RPC server? (1)

(c) Describe the function of numbers `0x47110001`, `0x47110002` .... (1)

(d) How many services does the above interface description provide? Explain whether you would have designed the interface description similarly or what you would change. (2)

(e) Instead of `char building_name[32]` the parameter could be specified as `string building_name<32>`. Explain the difference and name an advantage for one of the solutions. (2)

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)

(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)

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