Evaluation of practical materials in distributed systems course

Olha Shepelenko
University of Tartu
olha89@ut.ee
Supervised by Artjom Lind

INTRODUCTION

Evaluation of the course is performed in order to make a try to improve the course practical materials in Distributed Systems. To make an evaluation we have to analyze materials and decide whether they are covered well enough or they have to be improved somehow. We should also explore practical materials that are proposed by other universities that have similar course structure. Therefore, in this research work I will compare topics that are covered in different universities and related practical assignments to these topics.

1 LECTURE TOPICS AT THE DIFFERENT UNIVERSITIES

Structure of the course can vary significantly depending on universities. Some topic are not covered at all (in comparison to the University of Tartu), some of them just mentioned. Each university has unique syllabus, so it is difficult to compare all course from one university to the course at UT.

Let consider lecture's structure at the University of Tartu, University of Mumbai, Johns Hopkins University and Lomonosov Moscow State University:

Lecture topics at the University of Tartu:

- Characterization of distributed systems; System models
- Networking and internetworking; Interprocessor communication
- Indirect communication
- Remote invocation; Distributed objects and components
- Peer-to-peer systems
- Web services
- Security
- Operating system support; Distributed files systems; Name services
- Designing distributed systems: Google case study; Big Data paradigm
- Coordination and agreement
- Transactions and concurrency control. Distributed transactions

Course includes 48 lecture hours, 13 practical programming exercises, covering specific topic; 3 homework. Textbook for reading is Distributed Systems: Concepts and Design by George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair.

Lecture topics at the University of Mumbai:

- Fundamentals
- Communication
- Processes
- Synchronization
- Consistency and Replication
- Distributed Technology and Frameworks
- Service Oriented Architecture

Course includes 48 lecture hours, at least 10 practical experiments, 1 mini project and assignments. Textbooks for reading are Distributed Computing by Sunita Mahajan, Seema Shah and Distributed Systems : Principles and paradigms by Andrew S. Tanenbaum & Maarten van Steen.

Lecture topics at the Johns Hopkins University

- Introduction and basic protocols.
- Synchronous Models for Consensus.
- Multicast & Group Communication Services.
- Overlay Networks.
- Asynchronous Models for Consensus.
- Distributed Transactions.
- Replication.
- Intrusion-Tolerant Replication.

- Intrusion-Tolerant Messaging.
- Large-scale Data Stores & Probabilistic Protocols.
- Communication and Knowledge.

Course includes 48 lecture hours, 2 theoretically-based homework, 3 practical programming exercises and project. Textbooks for reading is Reliable Distributed Systems: Technologies, Web Services, and Applications 2005th Edition by Kenneth P. Birman.

Lecture topics at the Lomonosov Moscow State University:

- Introduction
- Communication
- Processes
- Synchronization
- Distributed shared memory
- Distributed file systems (NFS, GFS)
- Fault tolerance
- Examples of distributed systems (Hadoop, Amazon)

Course includes 34 lecture hours, 8 homework including practical and theoretical exercises. Textbooks for reading are Distributed Systems Principles and Paradigms, Tannenbaum and Van Steen and V.A. Kryukov, V.A. Bakhtin. Distributed systems. http://sp.cs.msu.su in the "Information" section (tutorial designed by professors from MSU).

As we can see, some topics are taught in all universities and some only in one of them. It is logically to compare topics and respective practical tasks that are similar to the University of Tartu program.

2 PRACTICAL TASKS

1.1 UDP socket communication

UDP socket communication is the first topic that was given in the Distributed systems course. Students had to implement client-server communication over UDP protocol: client sends the message to the server, the latter receives this message, displays the sender address and replies the client with the timestamp. Moreover, we have to add the third component to the chat – monitors, which are displaying messages that are sent by clients to the server. Server does not show client's messages, it redirects messages to all subscribed monitors. Each monitor sends "subscribed" and "keep-alive" notification after processing the message, if monitor has not sent "keep-alive" during 60 seconds it is removed as outdated. Notice that server has to remember on which socket each monitor locates.

In the task we have to send and display short messages, we do not care about packet loss, if message is not delivered to the server, client can try again, that is why we are using UDP, we do not need to set up connection before send a message.

In my opinion, first practical materials are covered excellent. The task was explained clearly, interaction between each component of the chat was illustrated on sequence diagrams, example of the code for client-server communication was provided as well as theoretical explanation: covered OSI model, structure of UDP datagram, basic of UDP socket communication.

Searching for similar tasks suggested by other universities, I have founded that students from the University of Mumbai also deal with UDP datagrams. One of their tasks is to calculate the factorial of the entered number. Client send the number to the server using UDP socket, server calculates the factorial and sends the result back to the client. Second task is to define whether the number is even or odd.

In the materials, I have just found examples of the code. None of theoretical explanations or guidelines are given.

Course at Johns Hopkins University proposes students to design a tool that transfers files from one machine to another in reliable way in the first part of assignment they are using unreliable UDP/IP protocol. Since UDP is unreliable delivery protocol, packet loss can occur, especially when big chunks of data are transferred, so in addition they have to calculate the loss rate percentage. In a second

assignment, students should apply UDP and implement a program that multicasts packets among N machines that locate in the same network as one multicast message may receive to all machines. In the practical materials examples of the code are given, as well as detailed description of the task, also provided a list a useful functions that can be used during writing a program and listed main features of UDP.

1.2 TCP socket communication

Second task is to implement client-server communication using TCP. The idea of the task is to create dump server, we have to check if the server has enough space on the disk for storing the file or not, if yes, client sends the file and server replies with notification that file is received successfully.

Here we are using request-response model: client sets up the connection with the server, server responses him and closes the connection. Moreover, we are using fragmentation, when transmit the files. Hence, the task is not just to implement client-server communication over TCP protocol, student have to understand features of using different protocols.

As for materials, the implementation example of the client-server communication are given. To solve the task student have to refer to the sequence diagram. It seems to me very useful to use diagrams, because they help to clarify the logic of the program, which step should be done next.

In comparison to the University of Tartu, students from the University of Mumbai implement program to find the prime number. In this case client sends TCP socket containing the number, server has to define if the number is prime or not and reply with TCP socket. Second task is to implement TCP based chatting application (just sending and receiving "Hello", "My name is…" messages). As in previous UDP socket connection task, only examples of the code are provided.

At the Johns Hopkins University students have to implement a program that transfers files using TCP protocol as a second part of the first assignment. Furthermore, comparison of performance using UDP and TCP is done.

1.3 Summary of UDP/TCP assignments

To my mind, UDP/TCP communication tasks are excellently taught in the University of Tartu. Topics are covered in details, examples of the code are given. It is also mentioned that we always have to pay attention to encoding text, because we frequently transmit text data over UDP/TCP. Note that in the current task we are using request response model, however in the further tasks we also leave communication channel open (due to task nature), and we use UDP broadcast and multicast for pear discovery.

Tasks are designed in a way to provide deep understanding of protocols and make them not so trivial as in other educational institutions. For example, as was shown above, University of Mumbai proposed to calculate something without using any additional peculiarities of UDP/TCP. Notice that, Lomonosov Moscow State University has not covered this topic at all. However, Johns Hopkins University proposes complex tasks using UDP and TCP. Tasks at JHU are similar to the tasks that are given at UT. Nevertheless there are some differences in the tasks between UT and JHU, for instance, at the University of Tartu we do not implement dump server using UDP, that is why we do not face the packet loss problem, because we send small pieces of data through the network. In addition, at the Johns Hopkins University UDP multicast is utilized not only for peer discovery, as it is done at the University of Tartu, but also for transferring data through the network, that is showing one additional feature of UDP.

In the following tables below you can find what features of UDP/TCP protocols are covered at the University of Tartu, University of Mumbai and Lomonosov Moscow State University.

Table 1 – UDP protocol.

UDP		UT	UM	JHU	MSU
IP header		+	-	-	-
UDP datagram structure		+	-	+	-
UDP peculiarities:	connectionless	+	+	+	-
	unreliable	+	+	+	-
	no package ordering	+	+	+	-
fragmentation		-	-	+	-
broadcast		+	+	+	-
multicast		+	+	+	-

Table 2 – TCP protocol.

TCP		UT	UM	JHU	MSU
IP header		+	-	-	-
TCP structure		+	-	+	-
TCP peculiarities:	connection oriented	+	+	+	-
	reliable	+	+	+	-
	package ordering	+	+	+	-
fragmentation		+	-	-	-
request - response		+	-	-	-
open channel		+	-	-	-

2.4 Applying threads in network applications

At the University of Tartu threads are considered in the networking context. Course considers main concepts of threads and processes, architectures for multithreaded servers: worker pool architecture; thread-per-request, thread-per-socket, thread-per-object architecture, methods that can be applied when programming threads.

The third practical task is to implement 3-component calculator: client sends computing task to the server, server registers a client's task and redirect it to the third component – back-end that has to compute the result of the task and send it back to the server. Server does not reply to the client when the result is ready, instead the client has to control whether the server has the result or not. In our case server is multi-threaded, clients and back-ends are single threaded. We are using thread per socket model – when client/back-end connects to the server, the latter creates thread for serving this client/back-end and assign the client/back-end socket to the client/back-end thread (respectively). For managing clients that are connected to the server we are utilizing delegation pattern – server receives the task from the client and delegates it to a particular thread. Therefore, for each client's task thread is reserved that will reply directly to the client. For managing back-ends we are using producer-consumer pattern – server produces task for consumers (back-ends). Server puts the task into the queue; back-ends retrieve tasks from the queue, calculate and send results to the server.

Practical materials contain code example of 3-component chart. As in the previous tasks, sequence diagrams are given as well as detailed task description.

Distributed systems course at the University of Mumbai also includes Threads topic in the curriculum. However, they do not focus on reviewing threads in networking context. It is difficult to say how deeply this topic is covered, but for sure, they do not cover architecture for multithreaded servers, synchronization.

Students solve some practical assignments using threads. The task is to implement a two-phase commit protocol (is covered at UT also). Messages are transferred between n participants and actions are committed or aborted regarding the reply of other participants. In this case threads are used for serving participants, they also utilize thread per socket model.

The main idea of implementation protocol: coordinator sends a VOTE_REQUEST to all participants, who have to reply with COMMIT (ready to commit) or ABORT. The coordinator stores all the participants' votes. If coordinator has received COMMIT message from all participants, it sends GLOBAL_COMMIT to all, if coordinator receives at least one ABORT it sends GLOBAL_ABORT. In case when participant gets GLOBAL_COMMIT it locally commits the transaction, else it aborts the local transaction.

Practical materials contain only examples of the code.

Lomonosov Moscow State University covers Threads topic, however they does not focus their attention on threads in the networking scope. They consider distributed systems more in the scope of one machine (distributed system is deployed inside computer). As a practical task students have to implement P-operation and V-operation for counting semaphore relying on mechanism of binary semaphore.

At the University of Tartu Semaphores are not covered, so I provide a bit explanation about it. Semaphores are utilized to synchronize access to a shared resource (for both thread and process). «A semaphore appears to be a simple integer. A thread waits for permission to proceed and then signals that it has proceeded by performing a P operation on the semaphore. The semantics of the operation are such that the thread must wait until the semaphore's value is positive, then change the semaphore's value by subtracting one from it. When it is finished, the thread performs a V operation, which changes the semaphore's value by adding one to it. It is crucial that these operations take place atomically—they cannot be subdivided into pieces between which other actions on the semaphore can take place. In the P operation, the semaphore's value must be positive just before it is decremented. In both P and V operations, the arithmetic must take place without interference. If two V operations are performed simultaneously on the same semaphore, the net effect should be that the semaphore's new value is greater than it was» [9]. At the MSU they consider several types of semaphores: binary (can have only 2 values), counting (can have more than 2 values).

Practical materials contain only the task explanations, student can refer to the tutorial that is released by academic staff.

At the Johns Hopkins University threads are not covered at all.

2.5 Applying processes in network applications

The forth task at the University of Tartu is to implement multi-process networking application with shared state. Server is holding a shared counter and clients can change the value of the counter: increase and decrease it. Here we need 2 types of processes: for increasing and decreasing counter as well as the main process for finding the winner regarding termination conditions. Practical materials contain detailed description of the task and code examples. As the example of the counter that is shared between processes we were told what multiple access and synchronization are.

At the University of Mumbai students have to implement any election algorithm (Bully algorithm or ring-based election algorithm). These algorithms are designed for selecting a coordinator that is required in many algorithms used in distributed systems. Overall, all processes can play role of a coordinator. Elections, probably, are required when the system is initialized, or if the coordinator goes down. Therefore, at the UM processes are used in this context, they do not apply processes in network application. Practical materials contain examples of the code.

As I mentioned above, Lomonosov Moscow State University covers Threads topic that is closely related to Processes topic. As a practical assignment students are proposed to implement task of the readers and writers based on a mechanism of binary semaphores such that process-writer must obtain exclusive access to the database (there should not be any writers or readers). An arbitrary number of processes-readers can run simultaneously, but any reader can access only if there are no working writers.

Distributed systems course at the Johns Hopkins University do not cover Processes topic.

2.6 Summary of threads/processes task

At the University of Tartu Threads and Processes are not covered as deeply as it is done at the Lomonosov Moscow State University. At the MSU they consider how to synchronize access to shared resource, they study the synchronization of threads & processes in details. Also another type of process interaction such as message passing is covered.

However, at the University of Tartu we do not consider how to synchronize threads and processes, what is the difference between thread and processes synchronization. We do not have to implement synchronization by ourselves, we do not use low level primitives, we just use existing mechanisms for synchronization, for instance, such as threading module in Python. For every shared resource we create Lock object, when we need access the recourse we call acquire() method in order to lock it, and release() method to release the lock. Despite that at UT Threads and Processes are not covered in the low level details, I think that students are provided with all basic knowledge to be able to program threads & processes in networking scope.

As I understood from the assignments from University of Mumbai they also did not cover these topic in details comparing to Lomonosov Moscow State University.

2.7 Remote Procedure Call

The fifth task at the University of Tartu is related to Remote Procedure Call. Student have to do simple task to show their understanding of RPC, for example make such operations as subtraction, addition, matrix multiplication, concatenation of the strings. All these operations are implemented on the server side and invoked by the client, we are using existing libraries for RPC.

At the University of Mumbai RPC is covered and students have to show the implementation of RPC. One of their task is to implement calculator (+, -, *, /) and another task is to calculate square & square root as well as cube & cube root of the entered number.

In both universities the code examples are provided.

At the Lomonosov Moscow State University and Johns Hopkins University practical tasks are not given for RPC topic.

2.8 Remote Method Invocation

The sixth task is to implement the shared counter (similar to the 4th seminar task) with player instances using Python Pyro or Java RMI. In this task client reports the server with his strength, speed, name and action (increase/decrease). Server modifies the counter value according to the player's strength. Here we are using remote objects, we do not have to apply TCP sockets.

At the University of Mumbai Remote Method Invocation is covered. As a task for practical assignment students have to write a program to show the object communication using RMI, for example, implement RMI based application that shows current date and time or converts digits to words.

At the Lomonosov Moscow State University and Johns Hopkins University practical tasks are not provided for RMI topic.

2.9 Summary of RPC/RMI

RPC and RMI are covered at three universities out of four. However, practical tasks are provided only by two universities.

I have found example of implementation RPC task from the University of Mumbai. The task does not reflect the main idea of RPC. In case of RPC, on the server side new methods can be added, but client does not know about these changes and he has to have an opportunity to ask server about new available methods. However, there is no option to ask such information from the given code example. Therefore, I think this task is not useful, it does not cover the concept of RPC.

Opposed to UM, UT proposes the task that shows features of RPC. On the server side

additional methods can be implemented and client is able to request the list of methods provided by server.

At the Lomonosov Moscow State University students do not have the practical assignments on RPC topic. However, they cover RPC, specifically, the principle of the implementation of Remote Procedure Call, review synchronous and asynchronous RPC.

At both universities where practical tasks based on RMI are provided, students use existed libraries for RMI and show their understanding of RMI by implementing some tasks. However, I want to notice that task provided by UM is not well designed, because in provided code example constructor is empty:

```
publicServerDate() throws Exception
{
}
```

So, there are no unique objects, each object ServerDate() will be identical.

At the Lomonosov Moscow State University RMI topic is covered, but not so deeply, just the main concepts. Practical tasks are not provided.

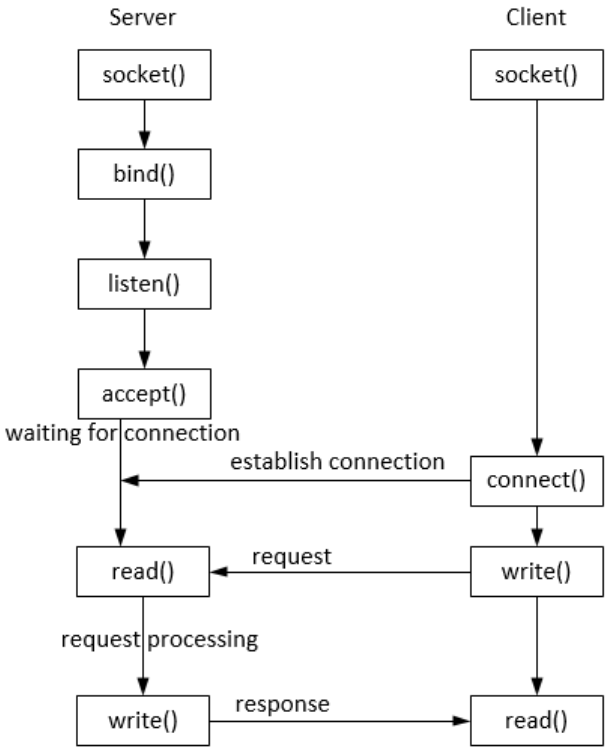
At the Johns Hopkins University RPC and RMI topic are not covered neither in theory nor in practice.

3 SUGGESTIONS FOR IMPROVEMENT THE PRACTICAL MATERIALS

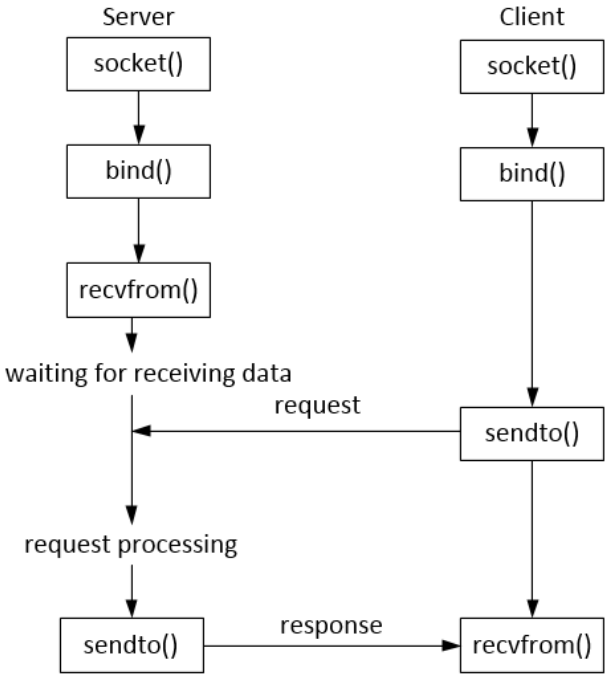
3.1 UDP & TCP

My proposition is to add to the materials the following charts (picture 1 and 2) for simplifying and easy understanding of client server interactions. Charts show, which methods we should apply when we are using UDP & TCP (given methods used in Python).

Moreover, I think it would be interesting to implement dump-server task over UDP protocol and have an opportunity to observe the main features of UDP protocol, specifically unreliable delivery and absence of packet ordering, because when you are transferring big chucks of data, some packets may be lost or come in incorrect order.



pic. 1 - Establishing TCP connection



pic. 2 - Data exchange using UDP

3.2 Threads/processes

Probably, it would be better to consider Threads and Processes more detailed because it is impossible to develop distributed applications using one thread/process, sooner or later your application will become multithreaded / multiprocessed. Therefore, synchronization is crucial in such kind of applications, we have to be able to design multithreaded / multiprocessed apps with elimination of access conflicts, when more the one thread/process is accessing the same resource.

My suggestion is to consider semaphores as a seminar topic and implement synchronization method by ourselves.

3.3 RPC/RMI

RPC and RMI tasks are well prepared at the University of Tartu. They shows the main features of RCP/RMI.
However, looking for RMI task given at the University of Mumbai we can also implement the program that asks current date and time, we should ask time not from random object, but form specific time zone object to highlight the object communication when you are applying RMI.

4 CONCLUSION

To summarize, I have analyzed practical materials from four universities and I proposed some tasks that can be performed in the future as well as some advice for improving practical materials.

REFERENCES

1. Practical materials in distributed systems course. https://courses.cs.ut.ee/2015/ds/fall/Main/Seminars
2. Distributed Systems: Concepts and Design (5th Edition) [George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair]
3. Distributed Systems: Principles and Paradigms (2nd Edition) [Andrew S.Tanenbaum, Maarten Van Steen]
4. Johns Hopkins University. Distributed Systems course. http://www.cnds.jhu.edu/courses/cs437/
5. Lomonosov Moscow State University. Distributed Systems course. http://sp.cs.msu.su/courses/os/
6. Lomonosov Moscow State University. Lecture materials. http://parallel.ru/krukov/
7. University of Mumbai. http://udcs.mu.ac.in/webpages/courses_msc.html
8. University of Mumbai. Practical materials. http://www.slideshare.net/sonalizoya/distributed-systems-40504037?from_action=save
9. Oracle documentation. Multithreaded Programming Guide https://docs.oracle.com/cd/E19683-01/806-6867/sync-27385/index.html