COURSE DESCRIPTION CARD - SYLLABUS

Course name
Distributed Computing [N1Inf1>PROZ]

Course

Field of study
Computing

Year/Semester
4/7

Area of study (specialization)
–

Profile of study
general academic

Level of study
first-cycle

Course offered in
Polish

Form of study
part-time

Requirements
compulsory

Number of hours

Lecture
16

Laboratory classes
16

Other
0

Tutorials
0

Projects/seminars
0

Number of credit points

3,00

Coordinators

dr inż. Arkadiusz Danilecki
arkadiusz.danilecki@put.poznan.pl

Lecturers

Prerequisites

Students starting this course should have a basic knowledge of algorithms and programming. Students should be able to solve basic problems in the range of design, checking the correctness and implementing algorithms in the C/C++ programming language and the ability to acquire information from the indicated sources. They should be able to understand threads and problems resulting from multithread programming, especially related to mutual exclusion, deadlock and starvation prevention. They should be able to understand the concepts of computatonal complexity (best and worst case). Students should have a necessary abilities needed to use Linux/Unix operating systems, including ability to write and compile programs. Students should also understand the necessity to broaden own competences/be ready to cooperate within the team. In addition, in the field of social competence, the student must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, respect for other people.

Course objective

1. To teach students basic knowledge about distinct characteristics of distributed systems, their fundamental differences from centralised systems, their construction, calculating the computational and communication complexity for distributed algorithms, and proving/verifying their correctness. 2. Familiarizing students with trends in distributed computing, basic issues appearing in construction of the distributed systems, problems appearing during their work and fulfillment of their typical tasks, as well as with typical solutions to the problems. 3. Enabling students to acquire necessary abilities required by distributed application development and to teach them chosen tools for such development 4. Developing the skills necessary to solving basic problems appearing in distrubuted computing domain 5. Fostering the skills necessary for team work, proper programming habits including code documentation 6. Teaching how to optimize the code by picking proper tools, algorithms and implementation methods

Course-related learning outcomes

Knowledge:
Students have expanded and deep knowledge in the domain of the distributed algorithms and their computational and communication complexity (K1st_W4)
Students know about trends in distributed computing, new methods, tools and algorithms especially as related to distributed computing (K1st_W5)
Students know basic methods, techniques and tools helping them to solve simple tasks during developoment of distributed application (K1st_W7)
Students have expanded and deep knowledge in the domain of distributed systems architecture (K1st_W4)

Skills:
Students can analyse computational and communication complexity of distributed algorithms (K1st_U8)
Students can use proper methods (analytical, simulation, experimental) for solving of specific problems from distributed computing (K1st_U4)
Students can analyse and evaluate distributed algorithms. In particular, they can participate in inspection of distributed systems and evaluate them from the point of view of extra-functional requirements. They can propose tests for functional requirements (K1st_U9)
Students can design and implement a distributed algorithm, choosing proper language for the task and using proper techniques, methods and tools (K1st_U10)
Students can design and implement distributed algorithms using chosen popular tool (K1st_U11)
Students can work in groups and are able to pick proper priorities for implementation of particular task, as demanded by them or their peers (K1st_U18)

Social competences:
Students understand that the knowledge and their skills might quickly become outdated (K1st_K1)
Students knows examples of failed or faulty distributed systemes and understands the reasons of their failures, causing significant social and financial losses (K1st_K2)

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning outcomes presented above are verified as follows:
Formative assesment:
a) For lectures:
- based on answers to questions related to subjects covered during previous lectures
b) For laboratory classes:
- based on assesment of progress of implementions of assigned tasks.
Total Assesment: Verification of assumed learning objectives is based on
- final exam, where students are given four problem questions based on issues covered during the lectures
- realisation of a group task: designing and implementing an algorithm solving chosen distributed computing problem using MPI library (in any programming language), usually based on mutual exclusion. The final grade is given by taking into account individual contribution, the quality and novelty of the solution both on theoretical (algorithm design) and practical (implementation) level. The assesment includes evaluation of the project report and final discussion where students are require to

explain their design and implementation choices.
Additional points might be given for:
- particular insights given during the classes,
- active participation in group discussions,
- covering additional aspects of discussed problems,
- giving helpful remarks related to the teaching materials,
- exceptional ability to use acquired knowledge to solve problems,
- pointing out perceptual difficulties enabling ongoing improvement of the teaching process.

Programme content

The lectures familiarize the students with examples of existing distributed systems, their most important and salient characteristics decisive for their specificity and the reasons for their development. Then, the fundamental concepts and definition of distributed computing are presented: distributed and sequential processes along with their formal models, the execution and the history of the execution, then the concepts related to the process activity, activation condition and classical request models. The formal definition of communication channels are introduced, together with other related concepts, such as communication operations or predicates describing the channel state.
Lecture continues with presentation of differences between synchronous and asynchronous communication. Different topologies of distributed computaton are presented, along with characteristics of distributed computing. Students learn about concepts such as Lamport's happened-before relation, space-time diagrams, reachable state graphs and nondeterminism inherent in distributed computation.
Students learn about issues related to the logical (virtual) time and its possible realisation with algorithms of Lamports (scalar clocks) and Mattern (vector logical clocks). The problem of correctness of distributed algorithms is discussed (progress/liveness and safety properties). Lecture then introduces the analysis of formal time and communication complexity for distributed algorithms.
The next problem discussed during the lectures is the fundamental problem of consistent global state snapshot. The basic definitions are first presented (configuration, cut, consistent cut and configuration). The possible uses for consistent snapshots are discussed, such as evaluation of global predicates. Students find out about problems involved in consistent snapshot construction in an asynchronous system. Finally, the algorithms constructing consistent snapshots of global state are presented, including Chandy-Lamport's for FIFO channels and Lai-Yang's for non-FIFO channels.
In subsequent part the reliability of distributed systems and algorithms is shortly discussed. Failure models are defined and the abstract notion of failure detector is introduced. The example implementation sof failure detectors of given characteristics are presented, together with necessary conditions for the implementation correctness. The methods providing abstract reliable communication channels based unreliable physical channels are discussed. The methods for reliable group communication are presented, using failure detectors with different properties.
Lecture also sketches the issues and problems involved with achieving consensus in a distributed system. The impossiblity of achieving consensus in fully asynchronous system in the presence of failures is presented first. Then it is shown how consensus might be achieved with certain additional assumptions (properties of available failure detectors).
The lecture contains also short introduction of the problem of termination detection in distributed systems. It introduces relevant definitions (e.g. the static and dynamic termination) and presents numerous algorithms solving the problem for the systems based on different models and with different topologies.
For most of the presented algorithms their correctness and formal complexity are analysed.
During the laboratory classes students familiarize with two environments for development of distributed applications: MPI library and, optionally, PVM. Students acquire the ability to use the proper tools and then they implement some of the algorithms presented during the lectures: they implement logical clocks, try to construct consistent snapshot of simplified distributed computation. They also create the programs for solving simple problems: breaking passwords using brute force approach and calculating π using Monte Carlo method. Finally students are assigned chosen classical problem, usually distributed mutual exclusion. They implement the assigned algorithms, individually or in pairs. They have to present the algorithm and discuss its correctness.
Each laboratory consists of problem presentation, group discussion an then implementation. The lecture is supplemented by introducing distributed mutua exclusion problem.

Course topics

The lectures sequentially cover the following topics: an introduction to distributed processing, including the motivation for creating distributed systems and examples of existing systems (e.g., Google, the Internet, grids). Subsequently, the correctness conditions for distributed algorithms, time and communication complexity, Mattern and Lamport logical clocks, consistent configuration, and consistent cut are discussed. Algorithms for constructing a consistent global state snapshot, such as Chandy-Lamport and Lai-Yang, are presented. The issue of termination detection is addressed, covering classical, static, and dynamic termination. Termination detection algorithms for the synchronous processing model, such as Dijkstra-Feijen-van Gasteren, the counter-based termination detection algorithm for the atomic model, and Misra's algorithm, are examined. Next, issues related to the presence of faults are considered: types of faults, fault modeling, and failure detectors along with their possible implementations are discussed. Finally, over three lectures, consensus problems are explored, including the Fisher-Lynch-Patterson impossibility result for solving consensus in an asynchronous model. The correctness conditions for uniform and basic consensus are defined, and the significance of consensus is explained. The Ben-Or algorithm, broadcast, and hierarchical algorithms for basic consensus are discussed, along with the hierarchical and all-acknowledgment algorithms for uniform consensus. The Raft algorithm is presented last. Proofs of correctness are provided for all algorithms. Additionally, the Ricart-Agrawala and Lamport algorithms for accessing a distributed critical section are covered.
In the laboratory sessions, the MPI environment is introduced, focusing on how to utilize simple programs (message sending and receiving, without discussing non-blocking or group communication). To familiarize students with the environment, they implement a very simple sorting algorithm, chosen to allow them to apply the functions they have learned. Next, the use of MPI with threads is demonstrated. Students can choose to implement either a simple brute-force password cracking program or a Monte-Carlo method for calculating PI. Students also implement Lamport clocks. They are given problems related to critical section access and are tasked with proposing their own algorithms (based on those discussed in the lectures). Students present their algorithms, discuss their correctness, and then implement them using MPI.

Teaching methods

1. Lecture: multimedia presentation, solving tasks, group discussion, presentation illustrated with problems sketched on a blackboard.
2. Laboratory class: task solving, practical „hands-on" exercises, group discussusion, team work, multimedia presentation.

Bibliography

Basic
1. Distributed Algorithms, N. Lynch, Morgan Kaufmann Publishers, 1996C
2. Ocena stanu globalnego w systemach rozproszonych, J. Brzeziński, Ośrodek Wydawnictw Naukowych, 2001
3. Programowanie współbieżne i rozproszone w przykładach i zadaniach, Z. Weiss, T. Gruźlewski, WNT, 1993
4. Programowanie równoległe i rozproszone, A. Karbowski (red.) E. Niewiadomska-Szynkiewicz (red.), Oficyna Wy-dawnicza Politechniki Warszawskiej, 2009
5. Introduction to Reliable and Secure Distributed Programming, C. Cachin, L. Rodrigues, R. Guerraoui, Springer-Verlag 2011

Additional
1. Distributed Algorithms and Protocols, M. Raynal, John Wiley & Sons, 1988
2. Systemy rozproszone: podstawy i projektowanie, G. Coulouris, J. Dollimore, T. Kindberg, Wydawnictwo Naukowo-Techniczne, 1998
3. Distributed Computing: Principles, Algorithms, and Systems, A. D. Kshemkalyani, M. Singhal, Cambridge University Press, 2011
4. Distributed Systems: An Algorithmic Approach, S. Ghosh, Chapman and Hall/CRC 2006
5. Podstawy programowania współbieżnego i rozproszonego, M. Ben-Ari, Wydawnictwo Naukowo Techniczne, 1990
6. Distributed computing. Fundamentals, Simulations and Advanced Topics, Attiya H., Welch J. John Wiley & Sons, 2004

Breakdown of average student's workload

	Hours	ECTS
Total workload	75	3,00
Classes requiring direct contact with the teacher	34	1,50
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	41	1,50