



COURSE DESCRIPTION CARD - SYLLABUS

Course name

Safety-Oriented Programming [S2Inf1-SRC>MBP]

Course

Field of study

Computing

Year/Semester

1/1

Area of study (specialization)

Distributed and cloud systems

Profile of study

general academic

Level of study

second-cycle

Course offered in

Polish

Form of study

full-time

Requirements

compulsory

Number of hours

Lecture

30

Laboratory classes

30

Other

0

Tutorials

0

Projects/seminars

0

Number of credit points

5,00

Coordinators

dr hab. inż. Paweł Wojciechowski prof. PP
pawel.t.wojciechowski@put.poznan.pl

Lecturers

dr hab. inż. Paweł Wojciechowski prof. PP
pawel.t.wojciechowski@put.poznan.pl

Prerequisites

The students starting this course should have basic knowledge in the field of concurrent and distributed systems and knowledge of at least one modern programming language. They should be able to solve basic problems of synchronization among concurrent threads or processes. They also should be able to obtain information from English-language sources. They should also understand the need to expand their competences and be ready to cooperate as part of a team. In addition, in terms of social competences, the students must present attitudes such as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, and respect for other people.

Course objective

The goal of the lectures is to discuss some programming methods that allow you to write concurrent programs that are correct and efficient. Firstly, we will see how to use monitors correctly and efficiently. Next, we will discuss the alternative synchronization method, i.e. transaction memory. Secondly, we will present selected automatic verification methods. Thirdly, we will discuss some issues regarding the correctness of concurrent programs aimed for modern computer architectures. Finally, we will show example methods and languages that make it easy writing of parallel and distributed programs. During lab classes, students carry out exercises and programming projects. Other goals: - developing students' ability to solve problems of concurrent programming and the ability to use advanced synchronization mechanisms to solve synchronization problems, - shaping teamwork skills in students during the implementation of the exercises and projects during laboratory classes.

Course-related learning outcomes

Knowledge:

has a structured and theoretically underpinned general knowledge of secure programming languages and paradigms (k2st_w2)
has advanced detailed knowledge of selected issues in computer science, such as modern methods, languages and tools for concurrent and distributed programming (k2st_w3)
has knowledge of development trends and the most significant new developments in computer science and other selected related scientific disciplines in the field of secure programming languages and paradigms (k2st_w4)
has advanced and detailed knowledge of the processes occurring in the life cycle of software information systems (k2st_w5)
has advanced knowledge of methods, techniques and tools used in solving complex engineering tasks and conducting research work in the area of computer science that concerns concurrent programming (k2st_w6)

Skills:

is able to acquire information from literature, databases and other sources (in polish and english), integrate them, interpret and critically evaluate, draw conclusions and formulate and fully justify opinions (k2st_u1)
is able to use analytical, simulation and experimental methods to formulate and solve engineering tasks and simple research problems (k2st_u4)
is able - when formulating and solving engineering tasks - to integrate knowledge from different areas of computer science (and, if necessary, also knowledge from other scientific disciplines) and apply a system approach, taking into account also non-technical aspects (k2st_u5)
is able to assess the usefulness and possibility of using new achievements (methods and tools) and new it products (k2st_u6)
is able to critically analyze existing technical solutions and propose their improvements (enhancements) (k2st_u8)
is able to assess the usefulness of methods and tools for solving an engineering task involving the construction or evaluation of an it system or its components, including recognizing the limitations of these methods and tools (k2st_u9)
is able - using, among others, conceptually new methods - to solve complex information technology tasks, including atypical tasks and tasks with a research component (k2st_u10)

Social competences:

understands that in computer science, knowledge and skills become obsolete very quickly (k2st_k1),
understands the importance of using the latest knowledge of computer science in solving research and practical problems (k2st_k2)

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Learning effects are verified as follows:

Forming assessment:

a) in the scope of lectures -- based on the answer to questions about the material discussed in previous lectures,

b) in the field of laboratories -- based on the assessment of the current progress of the implementation of tasks and programming projects.

Summary rating:

a) In the scope of lectures, verifying the assumed education effects is carried out by a written test. To be included in sufficient grade, it is necessary to obtain above half the possible points.

b) In the field of laboratories, verifying the assumed education effects is carried out by programming projects and a written test that can be combined with a written test at the lecture.

The final assessment may be influenced by activity during classes, e.g. the combination of additional aspects of the issue, and improving didactic materials.

Programme content

Writing concurrent programs for contemporary computer architectures, which do not guarantee sequential consistency, is usually significantly more difficult than writing analogous sequential programs. In addition, testing such programs is difficult, and detecting and removing all programming errors in the code is not always possible. The purpose of the lectures is to discuss some programming methods that allow you to write concurrent programs that are correct and efficient.

Course topics

The core subject curriculum is a subset of the following topics, some of which are covered in more than one class:

Lectures

1. Classical issues of concurrent programming in the context of modern hardware architectures, in particular low-level constructs of atomicity (mutual exclusion), conditional synchronization, and barriers.
2. Concurrent programming and synchronization on the example of monitors in C # / Java:
 - basic monitor operations, correct access to shared data, invariants, correct design patterns, incorrect practices (e.g. double-check locking),
 - deadlocks, starvation, efficiency problems with lock conflicts and priority inversion, advanced synchronization problems and optimization (e.g. avoiding spurious wake-ups and spurious lock conflicts).
3. Detection of the race condition during program execution on the example of the Eraser tool: lockset algorithm, optimization (initialization of variables, read-only shared data, read-write locks), correctness and completeness of the algorithm.
4. Detection of the high-level data race condition by static analysis: the algorithm and its correctness and completeness (false positives - unnecessary warnings, false negatives - unnoticed errors).
5. Detection of programming errors (divide by zero, race condition, deadlock) based on model checking, on the example of Java Pathfinder tool.
6. Conditional critical regions (CCR) and transactional memory: low-level operations on transactional memory, implementation of CCR using these operations, stack structure, data structures (ownership records and transaction descriptors), algorithm of atomic writing to transactional memory.
7. Correctness of concurrent access to shared objects: sequential and concurrent histories, linearizability property, concurrent FIFO queue and register, linearizability properties (locality, blocking vs. non-blocking).
8. Memory model on the example of Java language: memory model as a specification of correct semantics of concurrent programs and legal implementations of compilers and virtual machines, weakness vs. strength of the memory model, contemporary limitations of the classical model of sequential consistency, global analysis and code optimization, happens-before memory model and its weakness, memory model including circular causality, examples of controversial program code transformations.
9. Parallel programming on the example of the Cilk language: programming constructs, the acyclic directed graph (DAG) as a model of multithreaded computing, performance measures on a multi-core processor, greedy scheduling and upper limits for computation time.
10. Distributed programming with guarantees of eventual consistency on the example of Conflict-Free Replicated Types (CRDTs) and Cloud Types.
11. Distributed programming in the message-passing model on the example of the Erlang language (actors model, fault tolerance) and/or the Nomadic Pict language (process mobility, verification of the correctness of communication during compilation by static types).

Laboratory classes

In laboratory classes, students will learn about various concurrent programming constructs, with particular emphasis on safety in programming for modern computer architectures that do not guarantee sequential consistency. Example algorithms implementing synchronization mechanisms and concurrent, non-blocking data structures will also be discussed. This is complementary material to lectures.

If time permits, OCaml will also be discussed -- a standard functional programming language with roots in the ML language from the 70s. Ocaml has become an inspiration for many other contemporary programming languages, e.g. Rust. By discussing the language constructions, emphasis will be placed on expression and programming safety. OCaml is an elegant combination of the best features of programming languages, i.e. it supports modules and interfaces, first-order functions, abstract types, type variables, pattern matching, immutability, static type checking and automatic memory management. All these features together naturally support safe programming.

Teaching methods

- a) lectures - multimedia presentation illustrated with examples on the board, demonstration on the computer, discussion moderated by the lecturer.
- b) Laboratory exercises - multimedia presentation, discussion of examples on a board or on a computer, practical exercises at the computer consisting in performing tasks by students, team work, implementing programming projects by students, discussion moderated by the lecturer.

Bibliography

Example scientific articles (all are available by the Main Library of the Poznan University of Technology and / or are made available to students by the teacher):

1. An Introduction to Programming with C# Threads. Andrew D. Birrell
2. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson
3. High-level Data Races. Cyrille Artho, Klaus Havelund, Armin Biere
4. Language Support for Lightweight Transactions. Tim Harris, Keir Fraser
5. Linearizability: a correctness condition for concurrent objects. Maurice P. Herlihy, Jeannette M. Wing
6. The Java Memory Model. Jeremy Manson, William Pugh, Sarita V. Adve
7. A Minicourse on Multithreaded Programming. Charles E. Leiserson, Harald Prokop
8. MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean, Sanjay Ghemawat
9. Erlang - A survey of the language and its industrial applications. Joe Armstrong
10. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages. Paweł T. Wojciechowski
11. Sinfonia: A New Paradigm for Building Scalable Distributed Systems. Marcos K. Aguilera, Arif Merchant, Mehul Sha
12. The Part-Time Parliament. Leslie Lamport
13. Cloud Types for Eventual Consistency. Sebastian Burckhardt¹, Manuel Fahndrich, Daan Leijen, and Benjamin P. Wood
14. Conflict-free Replicated Data Types. Nuno Preguica, Carlos Baquero, Marc Shapiro
15. Developing Applications with OCaml. Emmanuel Chailoux, Pascal Manoury and Bruno Pagano

Breakdown of average student's workload

	Hours	ECTS
Total workload	125	5,00
Classes requiring direct contact with the teacher	60	2,50
Student's own work (literature studies, preparation for laboratory classes/tutorials, preparation for tests/exam, project preparation)	65	2,50