STUDY MODULE DESCRIPTION FORM

Name of the module/subject	Code
Safe Programming Methods	**1010512311010514020**

Field of study	Profile of study (general academic, practical)	Year /Semester
Computing	**general academic**	**1 / 1**
Elective path/specialty: **Distributed Systems**	Subject offered in: **Polish**	Course (compulsory, elective): **obligatory**
Cycle of study: **Second-cycle studies**	Form of study (full-time,part-time): **full-time**	

No. of hours	No. of credits
Lecture: **30** Classes: **-** Laboratory: **30** Project/seminars: **-**	**5**

Status of the course in the study program (Basic, major, other)	(university-wide, from another field)
major	**from field**

Education areas and fields of science and art	ECTS distribution (number and %)

Responsible for subject / lecturer:

dr hab. inż. Paweł Wojciechowski, prof. nadzw.
email: Pawel.T.Wojciechowski@put.poznan.pl
tel. 61 665 3021
Wydział Informatyki
ul. Piotrowo 3, 60-965 Poznań

Prerequisites in terms of knowledge, skills and social competencies:

1	**Knowledge**	Students taking this course should have basic knowledge in the area of concurrent and distributed systems, and be familiar with at least one modern programming language.
2	**Skills**	The students should be able to solve classical synchronisation problems of concurrent threads (or processes) and the ability to obtain knowledge from English language sources.
3	**Social competencies**	The students should also understand the need to expand her/his knowledge and be ready to work in a team. Moreover, the student should respect common social behaviours and must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, respect for other people.

Assumptions and objectives of the course:

1. To provide students with basic knowledge in the field of modern methods, languages ??and tools of safe programming, that is, those that guarantee programming free of a given class of programming errors,

2. Discussing examples of functional programming methods on the example of OCaml functional language, exemplary methods and tools for safe programming of concurrent and distributed systems, with particular emphasis on transaction memory and functional programming techniques, mechanisms (or algorithms) used in exemplary tools, basic properties of correctness of concurrent programs, which are applicable in the context of the discussed methods and tools for safe programming of concurrent systems,

3. Developing students' reasoning skills on the correctness of concurrent programs using both traditional methods and the latest methods (and using examples of correct and incorrect programs),

4. Teaching students the skills of teamwork through the seminar nature of some activities, with emphasis on discussion and joint development of proposals, as well as through implementation of programming projects.

Study outcomes and reference to the educational results for a field of study

Knowledge:

1. has a structured and theoretically founded general knowledge in the field of languages and paradigms of safe programming - [K2st_W2]

2. has advanced detailed knowledge of selected IT problems, such as modern methods, languages and tools for concurrent and distributed programming - [K2st_W3]

3. has knowledge about development trends and the most important new achievements in computer science and other, selected, related scientific disciplines in the field of languages and paradigms of safe programming - [K2st_W4]

4. has advanced and detailed knowledge of the processes taking place in the life cycle of software information systems - [K2st_W5]

5. knows advanced methods, techniques and tools used to solve complex engineering tasks and conduct research in the field of computer science, which is related to concurrent programming - [K2st_W6]

Skills:

1. can obtain information from literature, databases and other sources (in Polish and English), integrate them, make their interpretation and critical evaluation, draw conclusions and formulate and fully justify opinions - [K2st_U1]

2. can use analytical, simulation and experimental methods to formulate and solve engineering problems and simple research problems - [K2st_U4]

3. can - when formulating and solving engineering tasks - integrate knowledge from various areas of computer science (and if necessary also knowledge from other scientific disciplines) and apply a systemic approach, also taking into account non-technical aspects - [K2st_U5]

4. can assess the usefulness and the possibility of using new achievements (methods and tools) and new IT products - [K2st_U6]

5. can make a critical analysis of existing technical solutions and propose their improvements (improvements) - [K2st_U8]

6. can assess the usefulness of methods and tools for solving an engineering task, consisting in the construction or evaluation of an IT system or its components, including the limitations of these methods and tools - [K2st_U9]

7. can - using m.in. conceptually new methods - solve complex IT tasks, including atypical tasks and tasks containing a research component - [K2st_U10]

Social competencies:

1. understands that in informatics knowledge and skills quickly become obsolete, - [K2st_K1]

2. understands the importance of using the latest knowledge in the field of computer science in solving research and practical problems - [K2st_K2]

Assessment methods of study outcomes

The learning outcomes presented above are verified in the following way:

Forming rating:

a) lectures:

- based on answers to questions about the material discussed in previous lectures,

b) laboratories:

- on the basis of an assessment of the current progress of task implementation.

Summary rating:

a) in case of lectures, verification of the assumed learning outcomes is carried out by:

- assessment of knowledge and skills demonstrated on the problematic written exam. The exam consists in a written answer to 3 questions, selected from a list of several dozen questions, which is made available to students in advance. One can get 9 points for answering all the questions correctly. To qualify for a satisfactory grade, one must get min. 4 points.

b) in case of laboratories, verification of the assumed learning outcomes is carried out by:

- assessment of skills related to the implementation of the programming project; the assessment also covers the ability to work in a team because the projects are usually carried out by two students,

- assessment and defense of the presentation by the student on the basis of scientific articles indicated by the teacher or preparation of laboratory exercises in which a group of students takes part. In both cases the discussion is moderated by the teacher. In the case of presentation, the assessment consists of clarity of explaining the motivation for a given solution, using appropriate examples, and the ability to work in a team (in the case of a two-person presentation). In the case of laboratory exercises, the assessment consists of clarity of explaining the problem and its solution as well as the ability to work with a group of students implementing the exercises.

The assessment may be increased for distinguishing activities during classes, and especially for:

- discuss additional aspects of the issue,

- effectiveness of using the acquired knowledge while solving a given problem,

- remarks related to the improvement of didactic materials.

Course description

The basic program of the subject includes the following topics:

1. Concurrent programming and synchronization on the example of monitors in C # / Java: basic monitor operations, correct access to shared data, invariants, correct design patterns, wrong practices (eg double-check locking),

2. Concurrent programming and synchronization on the example of monitors in C # / Java: deadlock, starvation, problems with efficiency in lock conflicts and reversing priorities, advanced synchronization problems and optimizations (eg avoiding spurious wake-ups and spurious lock conflicts),

3. Functional languages: verification by strong typing, type polymorphism, function assembly, partial execution (currying), functional objects (closures), references and safe memory management, anonymous functions [lab],

4. Functional languages: examples of typed data structures, aggregation structures (eg map-reduce or mapping with

reduction, filtering, and folding), pattern matching, correct recursion (tail recursion) [lab],

5. Correctness of concurrent access to shared objects: sequential and concurrent histories, linearizability, examples: FIFO queue and registers, formalisation and linearity properties (eg locality, blocking vs. non-blocking),

6. Dynamic error detection in concurrent programming on the example of Eraser: the happens-before relationship and its limitations, the lockset algorithm for detection of race conditions, algorithm optimization including variable initialization, read-only shared data and read-write locks,

7. High-level data race condition: definition, algorithm of detection, correctness and completeness of the algorithm (false positives - unnecessary warnings, false negatives - unnoticed errors),

8. Transaction memory: conditional critical regions (CCR) and other language structures, low-level transactional transaction operations, CCR implementation using these operations, stack structure, ownership records and transaction descriptors, atomic write algorithm for transactional memory (on examples),

9. Correctness of program transactional memory: classical properties and their limitations in the context of transactional memory: linearizability, serializability, global atomicity (indivisibility) and recoverability, transactional memory model, opacity as safety property of transactional memory,

10. Limitations of transaction memory and transactions: problems when converting locks into optimistic transactions, strong vs. weak atomicity and correctness of programs, problem with native methods (irreversible operations), problem with sequential composition of transactions. Comparison of the theoretical speed of execution of atomic transactions and critical sections,

11. Memory model on the example of Java language: memory model as a specification of the 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 sequential coherence model, global analysis and code optimization, memory model of happen-before and its weakness, memory model including circular causality, model formalization, examples of controversial program code transformations,

12. Safe parallel programming on the Cilk example (with support for C / C ++): Cilk programming abstractions, multithreaded computational model based on an acyclic directed graph (DAG), multicore processor performance measure, greedy ordering and upper limits for computation time,

13. Secure batch computations scattered on a large scale on the example of Google: the technique of distributed (parallel) mapping and reduction (map-reduce), basic programming constructions, system architecture, fault tolerance, transparency,

14. Safe distributed programming in the message-passing model: model of distributed actors (Erlang) or objects (NPict), messaging operations embedded in the programming language, verification of the correctness of network communication by types, process mobility (NPict), [lab]

15. Minitranslation - an alternative to message passing approach to the construction of distributed systems on the example of Sinfonia: semantics and examples of minitranskacji, caching and cohesion, failure tolerance, system architecture, minutes of approving minitransactions.

Every year, the above list is extended with additional topics in the field of the latest knowledge. Issues marked by [lab] are implemented as part of laboratory exercises.

Teaching methods:

1. lecture: multimedia presentation, presentation illustrated with examples given on the board, demonstration on the computer,
2. laboratory exercises: practical exercises, discussion, team work, case studies, demonstration on a computer.

Basic bibliography:

1. Sample scientific articles (all of them are available at the Main Library of Poznan University of Technology and / or are made available to students by the teacher): Developing Applications with OCaml, Emmanuel Chailloux, Pascal Manoury and Bruno Pagano, O'Reilly France, English translation
2. An Introduction to Programming with C# Threads. Andrew D. Birrell
3. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. Leslie Lamport
4. Linearizability: a correctness condition for concurrent objects. Maurice P. Herlihy, Jeannette M. Wing
5. Language Support for Lightweight Transactions.Tim Harris, Keir Fraser
6. On the Correctness of Transactional Memory Rachid Guerraoui, Michał Kapałka
7. General and Efficient Locking without Blocking. Yannis Smaragdakis, Anthony Kay, Reimer Behrends, Michal Young
8. Subtleties of Transactional Memory Atomicity Semantics. Colin Blundell, E Christopher Lewis, Milo M. K. Martin
9. Deconstructing Transactional Semantics: The Subtleties of Atomicity. Colin Blundell, E Christopher Lewis, Milo M. K. Martin
10. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson
11. High-level Data Races. Cyrille Artho, Klaus Havelund, Armin Biere
12. A Minicourse on Multithreaded Programming. Charles E. Leiserson, Harald Prokop
13. Erlang - A survey of the language and its industrial applications. Joe Armstrong
14. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages. Paweł T. Wojciechowski
15. The Java Memory Model. Jeremy Manson, William Pugh, Sarita V. Adve
16. Sinfonia: A New Paradigm for Building Scalable Distributed Systems. Marcos K. Aguilera, Arif Merchant, Mehul Sha
17. Typed First-class Communication Channels and Mobility for Concurrent Scripting Languages, Paweł T. Wojciechowski

Additional bibliography:

1. Threading in C#. Joseph Albahari
2. Exceptions and side-effects in atomic blocks. Tim Harris
3. Process structuring, synchronization, and recovery using atomic actions. David Lomet
4. Transactions are Back-but How Different They Are? Relating STM and Database Consistency Conditions Hagit Attiya, Sandeep Hans
5. Pathological Interaction of Locks with Transactional Memory. Haris Volos, Neelam Goyal, Michael M. Swift
6. Ad Hoc Synchronization Considered Harmful Weiwei Xiong, Soyeon Park, Jiaqi Zhang, Yuanyuan Zhou, Zhiqiang Ma
7. Mnesia A Distributed Robust DBMS for Telecommunications Applications. Hakan Mattsson, Hans Nilsson, Claes Wikstrom
8. Threads Cannot be Implemented as a Library. Hans-J. Boehm
9. The Java Memory Model is Fatally Flawed. William Pugh
10. A Classification of Concurrency Failures in Java Components. Brad Long, Paul Strooper
11. Google's MapReduce Programming Model -- Revisited. Ralf Lammel
12. Atomizer: A Dynamic Atomicity Checker For Multithreaded Programs. Cormac Flanagan, Stephen N. Freund
13. Concurrent Haskell. Simon Peyton Jones, Andrew Gordon, Sigbjorn Finne
14. Developing a High-Performance Web Server in Concurrent Haskell. Simon Marlow
15. Beautiful concurrency. Simon Peyton Jones
16. Transactions with Isolation and Cooperation. Yannis Smaragdakis Anthony Kay Reimer Behrends Michal Young
17. Ownership Types for Safe Programming: Preventing data races and deadlocks. Chandrasekhar Boyapati, Robert Lee, Martin Rinard
18. Types for Atomicity: Static Checking and Inference for Java. Cormac Flanagan, Stephen N. Freund, Marina Lifshin, Shaz Qadeer
19. Semantics of Transactional Memory and Automatic Mutual Exclusion. Martin Abadi, Andrew Birrell,

Result of average student's workload

Activity	Time (working hours)
1. participation in laboratory classes: 15 x 2 hours	30
2. preparation of presentations or laboratory exercises	10
3. writing the program and its verification as part of the programming project (time outside laboratory classes)	23
4. participation in consultations related to the implementation of the education process (some of the consultations can be carried out electronically)	2
5. participation in lectures	30
6. familiarization with the indicated literature / didactic materials (10 pages of scientific text = 2 hours), 100 pages	10
7. preparation for the exam and presence on the exam: 18 hours + 2 hours	20

Student's workload		
Source of workload	**hours**	**ECTS**
Total workload	125	5
Contact hours	64	3
Practical activities	53	2