lingi2255 2018 Software engineering project

6 credits	30.0 h + 30.0 h	Q1

Teacher(s)	Mens Kim ;
Language :	English
Place of the course	Louvain-la-Neuve
Main themes	• Shared memory objects and synchronization • Performance and Scalability • Hardware support for synchronization • Consistency and progress • Obstruction-free, wait-free and lock-free shared data structures • Searching, sorting and counting with multiple cores • Non-Uniform Memory Access (NUMA) and impact on performance • Optimistic execution and transactional memory
Aims	Given the learning outcomes of the "Master in Computer Science and Engineering" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • INFO2.1-4 • INFO4.1-4 • INFO5.1-3 • INFO6.2-4 Given the learning outcomes of the "Master [120] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • SINF1.M3 • SINF2.1-4 • SINF4.1-4 • SINF5.1-3 • SINF6.2-4 Given the learning outcomes of the "Master [60] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • 1SINF1.M3 • 1SINF2.1-4 • 1SINF3.1-4 • 1SINF4.1-3 1 • 1SINF5.2-4 At the outcome of this course, the students will have acquired the necessary competences to build a large-scale software system under semi-professional working conditions. More specifically, students having completing this course with success will be able to: • Describe the differences among several major process models (e.g., waterfall, iterative, and agile); • Differentiate among the phases of software development (specification, architecture, design, implementation, validation, documentation); • Complete, in a rigorous and systematic way, the artefacts produced in these different software life cycle phases; • Apply a software development methodology currently practiced in industry; • Work efficiently in a team to develop a medium-to large-scale software system; • Manage the coordination and communication between the different team members; • Interact with a client to identify his requirements, to clarify imprecise specifications, and to take into account requested modifications throughout the development process; • Describe the functional requirements of a software system using, for example, use cases or users stories;' • Estimate the time and resources needed to complete such a software development project, plan the tasks to be executed and the deliverables to be produced, and respect this planning; • Use some project management tool to assign and follow the planned software development tasks; • Put in practice different methods and techniques to assure the quality of the produced software; • Understand the problems inherent to the development of large software systems having different stakeholders and that consist of multiple components. - - - -

	The contribution of this Teaching Unit to the development and command of the skills and learning outcomes of the programme(s) can be accessed at the end of this sheet, in the section entitled “Programmes/courses offering this Teaching Unit”.
Evaluation methods	• Projets (40% de la note finale) • Examen (60% de la note finale)
Teaching methods	• Courses and exercises • Practical sessions, rated • Final exam
Content	For a long time, general-purpose CPUs focused on supporting efficiently a single thread of execution. Improvements in chip manufacturing allowed packing more transistors on the same surface of a silicon wafer, and run resulting CPUs at higher frequencies. Single-threaded applications would simply run faster with every new processor generation. This era is now over. The industry hit several limitations known as the power wall, the memory wall and the ILP wall. No longer able to vertically scale-up CPUs supporting a single thread of execution, chip manufacturers have started packing together multiple, simpler units of execution, or cores. Exploiting the power of multiple cores requires exploiting parallelism in applications using multiple threads. Writing concurrent code requires identifying and managing concurrency, and introducing the necessary synchronization for correctness. Writing scalable and performant concurrent code requires understanding this tradeoff between synchronization and parallelism and mastering efficient implementations of shared data structures and algorithms for concurrent execution. Finally, multicore CPUs employ a complex memory layout, and the assumption of uniform memory access times is no longer valid. Understanding the impact of non-uniform memory accesses (NUMA) is therefore important to write efficient code for multicore CPUs. This course will provide students with the necessary tools and knowledge to write efficient and scalable code for modern multicore CPUs. It will detail the mechanisms available for synchronization, starting from the implementation of language constructs such as locks, monitors or condition variables, to the direct uses of CPU-provided synchronization primitives (e.g. compare-and-swap) to build efficient and scalable data structures. It will emphasize the performance aspects of multicore programming: the impact of synchronization primitives, the impact of non-uniform memory access, and the impact of multiple-level memory hierarchies. It will finally offer an opening to the future of multicore programming with an introduction to transactional memory and to the support for speculative execution in modern CPUs (e.g. Intel Haswell), and discuss the execution model for concurrent code running on GPUs.
Inline resources	*http://moodleucl.uclouvain.be/course/view.php?id=7599*
Bibliography	**Obligatoire** • Slides fournis en classe et en ligne (Moodle) • Exercices sur Moodle • Questionnaire d'auto-évaluation hebdomadaire sur Moodle **Optionnel** • The Art of Multiprocessor Programming, Maurice Herlihy and Nir Shavit, Morgan Kaufmann. ISBN 978-0-12-370591-4. UCL library reference 10.620.426 • Documents de recherche (pour la partie NUMA)
Other infos	• Algorithms and data structures (LINFO1103 & LINFO1121) • Performance measurements (LFSAB1402) • Language constructs for concurrent programming (LINFO1131)
Faculty or entity in charge	INFO

Programmes containing this learning unit (UE)

Program title	Acronym	Credits	Prerequisite	Aims
Master [120] in Computer Science and Engineering	INFO2M	6		🔍
Master [60] in Computer Science	SINF2M1	6		🔍
Master [120] in Computer Science	SINF2M	6		🔍