DTU Compute
Technical University of Denmark
Building 324
DK–2800 Lyngby
Denmark

HHL 06–11–2024

02158 CONCURRENT PROGRAMMING FALL 2024

Examination Requirements (Syllabus)

The written examination for 02158 Concurrent Programming takes place on **Monday, December 9, 2024** . The duration of the exam is **4 hours**. Only *non-electronic* aids are allowed (i.e. books and notes, but **no computers**).

The exam paper will be in **English only**. Solutions may be written in either English or Danish. Hand-in will be **on paper**.

Texts

1. Hans Henrik Løvengreen: *Basic Concurrency Theory*, Version 1.2
   Chapters 1–5 **except** sections 4.4, 4.5.1-4.5.4, 5.4.2.
2. Gregory R. Andrews: *Foundations of Multithreaded, Parallel, and Distributed Programming*
   Chapters 1–5, 7–8 with *emphasis* on sections 1.1–1.3, 1.9, 2.1–2.5, 2.8, 3.1–3.3.2, 3.4–3.4.2, 3.6, 4.1-4.5, 5.1–5.2, 7.1–7.4, 7.6–7.6.3, 8.1–8.4.

   Furthermore Section 9.1 and Part3 Intro. (pp. 527–532).
3. A. Silberschatz, P.B. Galvin and G. Gagne: *Deadlocks* (Chapter 7 from "Operating Systems Concepts (8th ed.)").
4. Hans Henrik Løvengreen: Concurrent Programming Practice *Processes, threads and tasks*, Version 2.0 (preliminary v.3). Sections 1–2. 6 pages.

[All intervals stated above are inclusive.]

Furthermore, all exercises, programming labs and mandatory assignments offered during the course are assumed known. Also remember to consult the *errata* list.

Notes

1. In *Basic Concurrency Theory*, you should pay attention to:

   Chapter 1: You must know the definitions of Petri Nets and the firing rules. You should be able to model synchronization aspects of concurrent systems with Petri Nets.

   Chapter 2: You should understand transition systems as a basic process model and interleaving as a way of modelling concurrency. You must know transition diagrams (aka program graphs) as a syntactic means of expressing processes.

   Chapter 3: You must be able to state safety properties formally as invariants and to give inductive, systematic proofs for these. In arguments, you may use the formal notation for pre/post values introduced in section 3.5.2. The semaphore invariant must be known.

Chapter 4: You must be able to express basic liveness properties (e.g. fairness) as formulas of temporal logic. You will not be asked about proof lattices.

Chapter 5: The notion of a monitor invariant must be known and you must be able to express basic safety properties as well as absence of unnecessary waiting. Also, you should be able to give systematic operational proofs of monitor invariants.

2. In Andrews' textbook, you should be familiar with the following concepts and principles:

   Chapter 1: Andrews' programming notation.

   Chapter 2: Atomic actions and the interleaving model. Synchronization. Safety and liveness properties.

   Chapter 3: Properties of critical regions. Busy-wait synchronization techniques.

   Chapter 4: Semaphore definition. Techniques: Mutual exclusion. Counting.

   Chapter 5: The monitor concept. Signal-and-continue vs. signal-and-wait semantics. Signalling techniques. Spurious wakeups. You must be able to construct efficient monitor solutions to given synchronization problems. *Unless otherwise stated, you should for condition queues assume FIFO ordering, signal-and-continue semantics, and that spurious wakeups do not occur.*

   Chapter 7: Asynchronous communication. Servers based on asynchronous communication. Synchronous communication in the form of CSP. Use of selection constructs (**if**/**do**).

   Chapter 8: Andrews' module concept. Operation implementation using RPC. Operation implementation using rendezvous with server process (**in**). Combinations of RPC and rendezvous. (The asynchronous **send** form is not emphasized.)

   Section 9.1 + Part 3 Intro: The notion of *speedup* must be known. You are expected to have a a basic understanding of task-based parallel processing using a bag-of-task with a number of worker threads (thread pool), cf. Mandatory Assignment 1.

3. Regarding resource control, you should know the three principal ways of treating deadlock (prevention, avoidance, detection/break), resource allocation graphs and the principles of determining whether a situation is safe/in deadlock. You should be able to apply the principle of the Banker's Algorithm.

4. You should be able to demonstrate a basic understanding of how a computation problem may be divided into tasks and how tasks may be executed on a thread pool.

Hans Henrik Løvengreen