Welcome to COMP2310/6310 – Concurrent and Distributed Systems

The Growing Importance of Concurrent and Distributed Systems

- computer systems are inherently concurrent
- computer-based services are inherently distributed
- advent of multicore/manycore chips, exascale computing, mobile computing, cloud computing, ubiquitous computing
  - to what extent must CDS concepts and techniques be treated explicitly in the profession?
- followup curriculum in CDS
  - COMP3300/6330 Operating Systems*
  - COMP3310/6331 Computer Networks
  - COMP3320/6464 High PerfSci. Computation
  - COMP4330/6430 Parallel Systems*
  - COMP4330/6433 Real-Time and Embedded Systems*
  - COMP4340/8320 Multicore Computing*

Course Contact

- course web site: http://cs.anu.edu.au/courses/comp2310
- course coordinator & lecturer: Peter Strazdins, CSIT N219, 6125-5140, comp2310@cs
- course tutors: Brian Lee, Beau Thompson, and Peter Strazdins (Computer Systems Group)
- forums accessible by StReAMS:
  - Announcements: postings from course staff only
  - Discussion: for your use (note the rules!)

Course Schedule

- lectures: three hours per week
- Concurrency: based on Chapters 2–8 and 10 of [Magee&Kramer ]
  - Processes and Threads, Concurrent Execution, Shared Objects & Interference, Monitors & Condition Synchronization, Deadlock, Safety and Liveness, Model-based Design, Message Passing; Architectures: Operating Systems (+communication)
- Distributed Systems: based on Chapters 1–4, 7, 14–18 of [Coulouris&al ]
  - Overview, Networks, Distributed Time and State, Agreement, Transactions, Replication
- practicals (supervised tutorial/laboratories): 8 (formal) and (3 × 0.5) informal
  - register NOW via http://cs.anu.edu.au/streams
  - important to come prepared! will contain examinable material!
- assignments: 2
- more details on the course schedule page

COMP2310 Lecture 1: Introduction 2013 2 2

COMP2310 Lecture 1: Introduction 2013 3 3

COMP2310 Lecture 1: Introduction 2013 4 4
Course Assessment

- see the assessment web page
- Assignments: due weeks 7 (18%) and 12 (12%)
  - note plagiarism issues and unacceptable vs acceptable collaboration:
    producing a solution with the aid of another's solution is cheating!
    working in close groups throughout is dangerous and is unfair to others!
- Practical Session Quizzes and Exercises: 10%
  - best 5 of 2% marks in 8 sessions
  - note: no extensions! (you should plan to attend all)
- Mid-Semester Exam: week 8 (Concurrency)
- Final Exam: 50%
  - 3 hours, 1 page A4 notes (both sides, printed or hand written), no calculator
  - former exam papers available from course web page;
    note that not all questions are still applicable

Mastering Concurrent & Distributed Systems

- Describe the concepts involved in the construction of CDS.
- “Speak the language” of concurrent modelling, design and programming.
  - explain the ideas & issues; explain and analyze key algorithms in CDS
  - be fluent with 1 modelling language and 2 programming languages
- Select appropriate modelling techniques and mechanisms and apply them to the solution of problems in CDS
  - model problems in FSP and analyze for required properties
- Select the appropriate programming language and environment for the task at hand.
  - understand the strengths and weaknesses of approaches and languages
- Write scalable, concurrent modules which show the intended and predictable behaviors.
  - includes monitors!
- Analyze and debug small to medium scale concurrent programs.
  - design & implement non-trivial solutions in Java and C

Approach for Concurrent & Distributed Systems

- emphasis is on general principles, design (modelling - new!), and systematic implementation
  - modelling includes Finite State Processes, with structure diagrams
    \[
    \text{BSEMA(Init=0)} = \text{BSEMA}[\text{Init}],
    \text{BSEMA}[v:\text{Int}] = (\text{when } v < \text{Max} \text{ up } \Rightarrow \text{BSEMA}[v+1] \\
    \text{when } v > 0 \text{ down } \Rightarrow \text{BSEMA}[v-1]).
    \]
- emphasis is on contemporary practice rather than classical theory
- CDS are part of our everyday life but reliable computer implementation requires sound understanding of various complex and subtle concepts!
- a serious approach is needed for succeeding in this course:
  - engagement throughout is essential!
    - especially coming prepared, attending all tutorial / laboratories and completing the (main) exercises
    - don’t be deceived by the terseness of FSP;
      it can be subtle and takes time to master!
    - need to use the ltsatool (& study resulting LTS diagrams)
    - reading the textbooks, particularly [Magee&Kramer]) is needed for follow-up

Introduction to Concurrency

Overview:

- what is a concurrent program?
- concurrent and distributed software
- why concurrent programming?
- why is it important to get it right?
- (Magee & Kramer’s) Cruise Control System
- models and modeling the Cruise Control System
- programming practice in Java
- objective of the concurrency module; and the book!

(the reminder of these slides including graphics are a modified version of the slides from Ch1 of Magee & Kramer: Concurrency)
What is a Concurrent Program?

- A sequential program has a single thread of control.
- A concurrent program has multiple threads of control allowing it to:
  - Perform multiple computations in parallel
  - Control multiple external activities which occur at the same time

The proper co-ordination of these threads and maintaining the integrity of shared data form the major challenges of concurrent programming!

Why Use Concurrent Programming?

- Performance gain from multiprocessing hardware
  - Parallelism: now the only way to gain speed!
- Increased application throughput via hiding of latency
  - An I/O call need only block one thread (web & database servers)
  - A cache miss need only block one thread (on-chip multiprocessors)
- Increased application responsiveness
  - High priority thread for user requests
- Reduce code complexity for inherently concurrent applications
  - I.e. Programs which interact with the environment, control multiple activities and handle multiple events
  - Permits separation of concerns (more appropriate structure)
  - Avoids a potentially exponential increase in complexity!

Why is it Important to get Concurrent Programming Right?

- Concurrency is widespread but error prone
  - Inherently more complex, has new types of problems (deadlocks, race hazards and other non-deterministic behavior)
- Notorious examples of failures due to faulty concurrent codes
  - Therac-25 (late 80's): A computerized radiation therapy machine concurrent programming errors contributed to accidents causing deaths and serious injuries
  - Undisciplined use of threads
  - Mars Rover (2004–10)
    - Resource contention from interaction between (otherwise independent) concurrent tasks caused periodic software resets reducing availability for exploration
Magee & Kramer’s Cruise Control System

- when the car ignition is on and the on button is pressed, the current speed is recorded and the system is enabled: it maintains the speed of the car at the recorded setting
- pressing the brake, accelerator or off button disables the system. Pressing resume re-enables the system

Is the system safe?
Would testing be sufficient to discover all errors?

Modelling the Cruise Control System

- we use the LTSA Animator to step through the system’s actions (≡ events)
- LTS of the process that monitors speed
- later, we will explain how to construct such models such and perform animation and verification

Models

- a model is a simplified representation of the real world
- engineers use models to gain confidence in the adequacy and validity of a proposed design
- we focus on an aspect of interest – concurrency
  - we use model animation to visualize behavior
  - permits mechanical verification of properties (safety & progress)
- models for concurrent systems are described using state machines
  - called Labelled Transition Systems (LTS)
  - these are described textually as finite state processes (FSP)
  - and displayed and analyzed by the LTSA analysis tool

Concurrent Programming in Java

- Java:
  - is widely available, generally accepted and portable
  - provides sound set of concurrency features
  - the threads API that we need for concurrency is compact and (relatively) simple
  - however, the challenge of correct concurrent reasoning remains!!!
- hence Java will be used for the illustrative examples and exercises
- later, we will explain how to construct Java programs for the Cruise Control System (from its model)
- “toy” problems are also used as they exemplify particular aspects of concurrent programming problems!
Concurrency Module Objectives

● to provide a sound understanding of the concepts, models and practice involved in designing concurrent software
● the emphasis on principles and concepts provides a thorough understanding of both the problems and the solution techniques
  ■ modeling provides insight into concurrent behavior and aids reasoning about particular designs
  ■ concurrent programming in Java provides the programming practice and experience
● the text is highly recommended: it will improve/expedite your understanding of concurrency!