Concurrent Programming

K. V. S. Prasad
Dept of Computer Science
Chalmers University
January – March 2015

Teaching Team

- K. V. S. Prasad
- Behrouz Talebi
- Raul Pardo Jimenez
- Anton Ekblad
- John Camilleri

Website

- http://www.cse.chalmers.se/edu/course/TDA383
- Should be reachable from student portal
  - Search on ”concurrent”
  - Go to their course plan
  - From there to our home page

Contact

- Join the Google group
- From you to us: mail Google group
  - Or via your course rep (next slide)
- From us to you
  - Via Google group if one person or small group
  - News section of Course web page otherwise

Course representatives

- Randomly chosen by admin

- TKDAT bennhage@student.chalmers.se DENNIS BENNHAGE
  TKDAT linujoh@student.chalmers.se LINUS JOHANSSON
  TKITE simlindk@student.chalmers.se SIMON LINDKVIST
  TKDAT rahnf@student.chalmers.se FREDRIK RAHN
  MPCSN kims@student.chalmers.se KIM STRANDBERG

- Expecting two more from GU
- Plan to meet after Monday lecture, weeks 2, 4, 6.

Practicalities

- An average of two lectures per week: for schedule, see
  - http://www.cse.chalmers.se/edu/course/TDA383/time_inf.html
- Rough guidelines (marks out of 100):
  - Pass = >40 points, Grade 4 = >60p, Grade 5 = >80p
  - To pass, must pass all labs and exam separately
- Written Exam 68 points (4 hours, closed book)
- Four programming assignments (labs) – 32 points
  - To be done in pairs
  - See schedule for submission deadlines and marks
  - Supervision available at announced times

Textbook

- M. Ben-Ari, ”Principles of Concurrent and Distributed Programming”, 2nd ed

  Addison-Wesley 2006

Central to your study. Exam based entirely on Chaps 1 through 9 of book.

Student wiki for problems from the book, and for past exams.

Other resources

- Old slides (both mine and Alejandro Russo's)
- Ben-Ari's slides with reference to the text
- Language resources – Java, Erlang
- Gregory R. Andrews
  - *Foundations of Multithreaded, Parallel, and Distributed Programming*
    - Recommended reading
- Joe Armstrong
  - *Programming in Erlang*
    - Recommended reading

Programming Languages

- For labs
  - Java (labs 1 and 2), Erlang (labs 3 and 4)
  - Erlang untyped functional language with asynchronous channels
  - Tutorials on Erlang next week
    - GET STARTED NOW WITH ERLANG EXAMPLES
- For lectures and exam
  - Ben-Ari's pseudo code
    - Can use Java+Erlang in exam, BUT WITH CARE
  - Spin/Promela as teaching aid (ignore if you wish)
- All but Erlang supported by Ben-Ari's textbook

Course always in transition!

- We now use Java and Erlang
  - Only as implementation languages in the labs
- Orally graded labs newish
  - Worked well last term
- Good text book
  - But we sadly still have no machine-aided proofs officially in course
- For discussion
  - pseudo-code as in book

What we did today

- Ideas from other sciences, music and cinema
- Correctness, semantics, dangers, debugging …
- Example: Unit Record Equipment (mention)

Parallelism in nature

- Everywhere!
  - The world is a parallel place
    - Physics, chemistry, biology, economics, medicine, history, football, tennis, ….
      - 10 million agents to simulate spread of infection
      - Simulate patient at various levels
        - » Cannot predict what will happen, but can show what might
  - And in art
    - Music, cinema
- Programming may be the only field where only one thing happens at a time
  - Was never really true (interrupts, etc.)
    - But education still 30 years out of date

Music

- Parallel
  - Time holds everything together ("real time" in CS)
    - What is held together?
      - Threads (themes, motifs)
        - » Can be logical or physical (which instrument, which hand)
    - Things that happen in time are called "events" in CS
    - The themes and motifs are called "processes"
  - Synchronisation is everywhere
  - Harmony and counterpoint are music's version of "coordination"

Cinema

- Concurrent (potentially parallel)
  - There is only one screen
    - So stories go on (or pause) off screen
  - There are cuts
    - within a scene (punctuation in a story)
    - and intercuts between scenes (“meanwhile”, …)
  - The priest’s voice provides a time-stamp.
    - Without it, the other scenes could be “meanwhile”, but not necessarily at the same instant
  - With the trains, synchronisation is visual or audible (phone)

Death by concurrency

- The presence of death in those film clips was not incidental – it was intended
- Concurrent systems are often embedded (in cars, planes, medical equipment, train signals)
  - Get them wrong and you too can kill
    - Not just in your video games, but for real
- Train crash in NE India (see website)
- Therac radiation therapy machine (see website)

Debugging doesn't work

- Concurrent systems are non-deterministic
  - Don't know who speaks first
  - Don't know who arrives first at a meeting
- So cannot re-run
  - So cannot set break points, backup and find bugs
- Then what do we do?
  - Use model checkers or proof checkers
  - They check spec versus implementation

Semantics

- What do you want the system to do?
- How do you know it does it?
- How do you even say these things?
  - Various kinds of logic
- Build the right system (Validate the spec)
- Build it right (verify that system meets spec)