COSC441

Concurrent Programming

Richard A. O'Keefe

Plagiarism

- It's trying to claim the credit for other people's work.
- It's bad. Don't do it.
- The University doesn't tolerate it. Really don't do it.
- It's getting easier to detect all the time. Honestly, really do not do it.
- Quoting someone else with a proper citation is research; it's good; it gets you marks. So get credit for giving credit.

Class reps

- Every class should have a class rep.
- It's not hard to be a class rep. You just have to listen to other students' troubles and tell the lecturer (me), 400-level coordinator (me), or HoD there is a problem that needs to be addressed. There are a couple of brief meetings to go to, but don't wait.
- Send paper number, your full nam, your University e-mail address, and ID to Kaye.

General aims

- Understand what *concurrent programming* is
- Understand *shared memory* concurrent programming using C11 and POSIX threads
- Understand *threads* and *stacks.*
- Understand issues of concurrent memory access including *data races* and *tearing*
- Understand *critical regions, locks, conditions, monitors, semaphores,* and *barriers*.
- Understand *communication* including *bounded buffers* and *flow control.*

General aims 2

- Understand *hardware level* locking including *atomic updates, compare-and-swap*, and *load-locked/store-conditional*.
- Be aware of problems with locking, including *contention, convoying,* and *priority inversion.*
- Be aware of *lock-free* data structures and some reasons for using them.
- Be aware of *transactional memory* and some of its benefits and issues.

General aims 3

- Understand *(shared-nothing) message-passing* as an alternate concurrency model.
- Understand how *distribution* changes things.
- Understand some of the issues with *time* in concurrent and distributed programming, including *causality*, *happens-before,* and *vector clocks.*
- Be aware of some design patterns using concurrency.

Next week

- Next week we shall look at the classic *memory model* for programming languages like Fortran, C, Pascal, especially how procedure calls were mapped onto a *stack*
- We'll look briefly at the *cactus stack* model used by Burroughs Algol, Simula 67, and ML
- and the *thread* model used in POSIX and C11.
- We shall also look at the *memory hierarchy*

Week 3

- We'll look at what goes wrong with the classic memory model due to compiler optimisations (that assume single-threading) and concurrency
- We'll introduce the ideas of *atomic operations, critical regions,* and *locks.*
- This will give you enough to write simple concurrent programs.

What's happening today

- Overview.
- Distinction between parallel and concurrent.
- Getting to know you.
- A bit of history.

Parallel

- Computer has multiple computing units
- They are active at the same time
- Vector instructions like SPARC VIS, Power AltiVec, x86 MMX &c are an example.
- ```
  forall (i = 1, n)
     y(i) = dot_product(a(i,*), x)
  end forall
  ```
  we don't care about the order!
  ```

Concurrent

- The world has many things operating at the same time.
- We sometimes have to model this in a computer program.
- The most natural way is one modelled activity : one concurrent task.
- Concurrent activities *interact* and we have to model and manage those interactions.
- One processor can simulate concurrency.
- Programs can be concurrent *and* parallel.

Distributed

- A distributed system has multiple computing devices communicating through a network;
  - a cluster in one cabinet
  - a LAN in one building
  - a WAN across a city, country, or planet.

Distributed systems can simulate shared memory at heavy cost, because communication is slow. Distributed systems can fail in complex ways.

Getting to know you

- I need to know your programming background. There's no point in me saying "This is rather like Emerald except ..." if you don't know Emerald.
- I shall ask you to say what you already know about concurrency. I need to adjust my lectures.
- I would like to know what you need/expect from this paper.