Erlang Language Seminar Outline

Slide #1: Title - Erlang programming language

Purpose of the seminar: To expose students to new ideas and new ways of thinking. Mainly to dispel this idea that advanced topics and different paradigms in computer science are theoretical only and not really useful in the real world. The idea of using the right technology for a specific problem.

Slide #2: Why this seminar topic?

- Wired magazine article “Why WhatsApp Only Needs 50 Engineers for Its 900M Users” - September 15th, 2015 - the article talks about how whatsapp uses Erlang to achieve its goals. I had never heard of erlang, so i was curious to know what it was and why it was being used.
- Erlang was developed in 1986...not exactly a new idea - so we come to this topic of reusing old ideas yet again. most of the things we have learned about in this class are not new ideas, in fact they have been around for decades.
- Talk about why ideas or technologies sometimes need to “marinate” before they become mainstream. OOP and parallel computing weren’t always mainstream. Example of Android and discuss some obstacles like IP patents
- Learning to program in Erlang is not the purpose - same as learning Ada is not the purpose of 3P93. i feel that exposure to new ideas and new ways of thinking is the main purpose.

Origin of Erlang: It’s purpose and what problem it was meant to solve. Overview of the things it’s good at.

Slide #3: What is Erlang?

- Initially a proprietary language, developed in 1986 within Ericsson for telephony applications (telecom hardware multinational), has since been open-sourced
- The main purpose was telecommunications where applications generally need to be up and available non-stop
- Highly available systems discussed later. Erlang is nine 9’s pic?
- A multi-paradigm language: namely OOP and functional

Slide #4: nine 9’s pic

Brief review of functional languages: Very very brief, just the idea of functions as a central paradigm and code that doesn't have any side effects, which allows for easy proofs. Maybe some examples of how this tends to simplify algorithms. There is a whole course dedicated to this topic so I will not go into too much detail...for now just the idea that it requires a different way of thinking about a problem. (in comparison with imperative procedural languages)

Slide #5: Functional programming (quick review)

- Central building block is the "function"
- Variables are immutable - assigned only once per function call
- Functions don't create side effects (most of the time)
- Lends itself to proofs
- Tail recursion used for iteration (compiler optimized, no stack)
- Pattern matching & list comprehension

Other courses dedicated to these topics

Slide #6: Programming example slides for factorial and quicksort (talk about quicksort being highly parallelizable and the fact that # of comparisons and total time are not necessarily proportional when performing a parallelized operation)

Concurrency: Massive concurrency using very lightweight processes as building blocks (benchmarked at 20 million on modest hardware). Supervision Tree, workers and supervisor processes. Generic and Specific parts of process code. No shared memory and communication via asynchronous message passing.

Slide #7: Concurrency

- Central building block for Erlang is the "process"
- Don't confuse with operating system processes or threads
- Very lightweight: can be stateless and nothing is shared between them
- Communication via message passing: messages are queued and received if they match a desired pattern
- A process can represent an object
- Behaviors: functionality is similar to inheritance and interfaces
- The code can have Generic (behavior module) and Specific (callback module) parts - module is a compilation unit

Fault tolerance: Important for highly available non-stop applications. High availability design principles. Clean handling of error state by sending message to supervisor and exiting. Usually another identical process picks up where the first left off. The user will never see an error, but the supervisors will handle it.

Slide #8: Fault Tolerance

- Fail early and often! Concurrency used as main method to handle errors.
- High availability design principles:
    a. Elimination of single points of failure - many concurrent processes facilitate this
    b. Reliable crossover - processes neatly exit and send a message to a controlling process upon error/crash/failure
- Commonly achieved with worker-supervisor design pattern (can form a supervision tree) - the workers might be servers in a client-server pattern
- If these principles are implemented correctly, the user will never see an error, but the supervisor will always handle it when it does occur
- Allows one to focus only on the things that need to be done and ignore externalities (bad inputs or whatever) since processes can ignore messages they don't understand and can cry to their supervisor if something unexpected happens (factorial function supervisor - simple example)

Code hot swapping: How different modules of code are kept on the system and used. Hot code loading design principles. OTP (Open Telecom Platform).

Slide #9: Code hot swapping!

- Open Telecom Platform (OTP): includes a lot of libraries,tools, behaviors and design patterns for Erlang. Systems built using their design principles allow for a range of interesting behaviors.
- Code can have an "old" and "new" module loaded into memory
- Processes can run both and only switch to the "new" version after a callback to that module
- Reset the Torus anyone?

Distributed Systems: Erlang nodes running many processes some local and some not, all communicating. Achieves high level of concurrency and fault tolerance.

Slide #10: Putting it all together...Distributed Systems

- You can run many “nodes” locally or over a network
- A node can be one or many processes
- Allows for highly responsive, concurrent, fault tolerant systems

Who uses Erlong and why? Are functional languages useful? Design patterns? Right tool for the job?

Slide #11: Who uses Erlang and why?

- Amazon, Yahoo, Facebook, WhatsApp, T-Mobile, Goldman Sachs...to name a few
- The right tool for the job
- Focus on design patterns: The tools you use to solve a problem change depending on the problem, but design patterns are always useful