The Actor Model applied to the Raspberry Pi and the Embedded Domain

The Erlang Embedded Project

Omer Kilic  ||  @OmerK
omer@erlang-solutions.com
Outline

• Current state of Embedded Systems
• Overview of Erlang and the Actor Model
• Modelling and developing systems using Erlang
• The Erlang Embedded Project
• Future Explorations
• Q & A
An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs.

- Infinite Wisdom of Wikipedia
Embedded Systems (II)

- Specific and limited set of functions
- Designed for a particular application

- General purpose
- Can be used for pretty much any computing needs
Current Challenges

• Complex SoC platforms
• “Internet of Things”
  – Connected and distributed systems
• Multicore and/or heterogeneous devices
• Time to market constraints
  – The Kickstarter Era
  – Rapid prototyping
  – Maker Culture
Samsung Exynos Reference System
Internet of Things
Internet of Fridges?
Distributed Bovine Networks?

These things are not just smartphones and tablets.

They're everything.

A Dutch startup, Sparked, is using wireless sensors on cattle.

So that when one is sick or pregnant, it sends a message to the farmer. Each cow transmits 200 mb of data per year.
Exciting times
Embedded Systems

• Bare Metal
  – No underlying OS or high level abstractions

• RTOS
  – Minimal interrupt and switching latency, scheduling guarantees, minimal jitter

• Embedded Linux
  – Slimmed down Linux with hardware interfaces
RTOS Concepts

• Notion of “tasks”

• OS-supervised interprocess messaging
  – Shared memory

• Mutexes/Semaphores/Locks

• Scheduling
  – Pre-emptive: event driven
  – Round-robin: time multiplexed
Embedded Linux

• Not a new concept, increased popularity due to abundant supply of cheap boards
  – Raspberry Pi, Beagleboard/Beaglebone, Gumstix et al.

• Familiar set of tools for software developers, new territory for embedded engineers
  – No direct mapping for RTOS concepts, especially tasks

• Complex device driver framework
  – Here be dragons
The four languages most often reported as the primary language for embedded projects for the years 2005 to 2012, along with linear trendlines.

Source: http://embedded.com/electronics-blogs/programming-pointers/4372180/Unexpected-trends
• Knowledge Transfer Partnership between Erlang Solutions and University of Kent
  – Aim of the project: Bring the benefits of concurrent systems development using Erlang to the field of embedded systems; through investigation, analysis, software development and evaluation.
Erlang? (I)

{functional, declarative, concurrent, parallel, garbage-collected, soft real-time, fault-tolerant, robust, portable, distributed hot code loading}
Erlang? (II)

• First version developed in 1986

• Battle-tested at Ericsson and many other companies
  – Originally designed for Embedded Systems!

• Implements the Actor model
  – Support for concurrency and distributed systems out of the box

• Easy to create robust systems
High Availability/Reliability

- Designed for systems with high availability constraints
  - Nine 9s availability
- Simple and consistent error recovery and supervision hierarchies
- Built in fault-tolerance
  - Isolation provided by Actors
- Support for dynamic reconfiguration
  - Hot code loading
Actor Model

- Proposed in 1973 by Hewitt, Bishop and Steiger
  - “Universal primitives for concurrent computation”
- Building blocks for modular, distributed and concurrent systems
- No shared-state, self-contained and atomic
- Implemented in a variety of programming languages
Actor Model

- Asynchronous message passing
  - Messages kept in a mailbox and processed in the order they are received in

- Upon receiving messages, actors can:
  - Make local decisions and change internal state
  - Spawn new actors
  - Send messages to other actors
Limitations of the Actor Model

• No notion of inheritance or general hierarchy
  – Specific to language and library implementation

• Asynchronous message passing can be problematic for certain applications
  – Ordering of messages received from multiple processes
  – Abstract definition may lead to inconsistency in larger systems
    • Fine/Coarse Grain argument
Creating an Actor

\[ \text{spawn(math, fact, [5])} \]

\[ \text{math:fact(5)} \]
Communication

Pid1 \rightarrow \{\text{Pid1, msg}\} \rightarrow \text{Pid2}

\text{Pid2} ! \{\text{msg, data}\}
Bidirectional Links

link(Pid2)
Propagating Exit Signals

{'EXIT', PidA, Reason}

{'EXIT', PidB, Reason}

PidA

PidB

PidC
Trapping Exits

PidA \rightarrow \{ 'EXIT', PidA, Reason \} \rightarrow PidB

process_flag(trap_exit, true)
Supervision Hierarchies

- Let it Fail!
  - Abstract error handling away from the modules
Fine Grain Abstraction

• Advantages
  – Application code becomes simpler
  – Concise and shorter modules
  – Testing becomes easier
  – Code re-use (potentially) increases

• Disadvantage
  – Architecting fine grain systems is difficult
Erlang, the Maestro

(flickr/dereckesanches)
Why Raspberry Pi?

- The Raspberry Pi Foundation is a UK registered charity.
- Mission statement: "...to promote the study of computer science and related topics, especially at school level, and to put the fun back into learning computing."

Future Engineers/Programmers!

(flickr/lebeus)
Raspberry Pi Peripherals

- GPIO
- UART
- I2C
- I2S
- SPI
- PWM
- DSI
- CSI-2
External Interfaces in Erlang

• Facilities to interface the Erlang runtime to the outside world
• Used for device drivers and kernel abstractions in the embedded domain
Accessing hardware

• Peripherals are memory mapped

• Access via /dev/mem
  – Faster, needs root, potentially dangerous!

• Use kernel modules/sysfs
  – Slower, doesn’t need root, easier, relatively safer
**GPIO Interface (I)**

\[ \text{init}(\text{Pin}, \text{Direction}) \rightarrow \]

\[
\{ \text{ok, FdExport} \} = \text{file:open}("/sys/class/gpio/export", [write]), \\
\text{file:write}(\text{FdExport}, \text{integer_to_list}(\text{Pin})), \\
\text{file:close}(\text{FdExport}), \\
\]

\[
\{ \text{ok, FdPinDir} \} = \text{file:open}("/sys/class/gpio/gpio" ++ \text{integer_to_list}(\text{Pin}) ++ "/direction", [write]), \\
\text{case Direction of} \\
\quad \text{in} \rightarrow \text{file:write}(\text{FdPinDir}, "in"); \\
\quad \text{out} \rightarrow \text{file:write}(\text{FdPinDir}, "out") \\
\text{end}, \\
\text{file:close}(\text{FdPinDir}), \\
\]

\[
\{ \text{ok, FdPinVal} \} = \text{file:open}("/sys/class/gpio/gpio" ++ \text{integer_to_list}(\text{Pin}) ++ "/value", [read, write]), \\
\text{FdPinVal}. \\
\]
**GPIO Interface (II)**

```erlang
write(Fd, Val) ->
    file:position(Fd, 0),
    file:write(Fd, integer_to_list(Val)).
```

```erlang
read(Fd) ->
    file:position(Fd, 0),
    {ok, Val} = file:read(Fd, 1),
    Val.
```

```erlang
release(Pin) ->
    {ok, FdUnexport} = file:open("/sys/class/gpio/unexport", [write]),
    file:write(FdUnexport, integer_to_list(Pin)),
    file:close(FdUnexport).
```
Example: GPIO

PidA → Pin17 → PidB → Pin17 → PidC

{state, high}

{state, low}

{state, high}
Example: GPIO

- PidA
- PidB
- PidC
- Pin17

Diagonal lines indicate transitions between states.

{state, high} from PidA to GPIO Proxy
{state, low} from PidB to GPIO Proxy
{state, high} from GPIO Proxy to Pin17
GPIO Proxy

• Replaces ‘locks’ in traditional sense of embedded design
  – Access control/mutual exclusion
• Can be used to implement safety constraints
  – Toggling rate, sequence detection, direction control, etc.
Concurrency Demo

```erlang
-module(led).
-export([start/1, stop/1, loop/2]).

start(Pin) ->
  Fd = gpio:init(Pin, out),
  Pid = spawn(?MODULE, loop, [Fd, Pin]),
  Pid.

stop(Pid) ->
  Pid ! stop.

loop(Fd, Pin) ->
  receive
    on ->
      file:write(Fd, "1"),
      loop(Fd, Pin);
    off ->
      file:write(Fd, "0"),
      loop(Fd, Pin);
    {blink, Delay} ->
      file:write(Fd, "1"),
      timer:sleep(Delay),
      file:write(Fd, "0"),
      timer:sleep(Delay),
      self() ! {blink, Delay},
      loop(Fd, Pin);
    stop ->
      file:close(Fd),
      gpio:release(Pin),
      ok
  end.
```

10/04/2013  ACCU 2013 Conference  Slide 41 of 49
Universal Peripheral/Component Modules
Universal Peripheral/Component Modules

An Example:

Temperature Sensor with I2C Interface

Sensor API

I2C Bus Driver

Linux sysfs

Sensor API

I2C Bus Driver

BSD sysctl

Sensor API

I2C Bus Driver

mmap()'ed register voodoo
Hardware Projects – Ponte
Hardware Projects – Demo Board
Hardware Simulator
Future Explorations

Parallella:

The Epiphany™ Multicore Solution

Coprocessor to ARM/Intel Host

C/C++/OpenCL Programmable

Scales to 1000s of cores on a chip
Thank you

• http://erlang-embedded.com
• embedded@erlang-solutions.com
• @ErlangEmbedded

“The world is concurrent.
Things in the world don't share data.
Things communicate with messages.
Things fail.

-- Joe Armstrong
Father of Erlang