Execution architecture concepts

by Gerrit Muller  Buskerud University College

e-mail: gaudisite@gmail.com

www.gaudisite.nl

Abstract
The execution architecture determines largely the realtime and performance behavior of a system. Hard real time is characterized as ”missing a deadline” will result in system failure, while soft real time will result ”only” in dissatisfaction.

An incremental design approach is described. Concepts such as latency, response time and throughput are illustrated. Design considerations and recommendations are given such as separation of concerns, understandability and granularity. The use of budgets for design and feedback is discussed.
Execution Architecture

dead lines
timing, throughput
requirements

process

interrupt
handlers

execution architecture
issues:
concurrency
scheduling
synchronisation
mutual exclusion
priorities
granularity

functional
model

receive

display

store

demux

process

Map

execution architecture

repository
structure

Applications
play
zap
list

UI toolkit
menu

processing
list

foundation
classes
debuf
list

hardware

CPU

DSP

RAM

tuner

drive

version: 1.1
March 6, 2013
CVexecutionArchitecture
Fuzzy customer view on real time

- hard real time
  - disastrous failure
  - loss of information
  - loss of functionality
  - limited throughput
  - loss of eye hand coordination

- soft real time
  - dissatisfaction
  - waiting time

- human safety
- device safety

Execution architecture concepts
3    Gerrit Muller
Smartening requirements

Limited set of hard real time cases

Precise form of the distribution is not important.

Be aware of systematic effects

No exception allowed
Worst case must fit

Well defined set of performance critical cases

Typical within desired time, limited exceptions allowed.

Exceptions may not result in functional failure
Latency

Connection latency

Perceived delay

Speak
Listen

blablabla

Telephone

Long distance connection

reaction

time

Execution architecture concepts

version: 1.1
March 6, 2013
EAClatency
Response Time

total response time

zap repetition

visual feedback
open for next response
new channel

time

visual feedback time

P+
P-
remote control

new channel

zap
Throughput

throughput:
+ processing steps/frame
+ frames/second
+ concurrent streams
Gross versus Nett

bus bandwidth, processor load [memory usage]
useful macroscopic views, be aware of microscopic behavior

<table>
<thead>
<tr>
<th>margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>loss = not schedulable</td>
</tr>
<tr>
<td>overhead</td>
</tr>
<tr>
<td>bus, OS, scheduling</td>
</tr>
<tr>
<td>function 4</td>
</tr>
<tr>
<td>function 3</td>
</tr>
<tr>
<td>function 2</td>
</tr>
<tr>
<td>function 1</td>
</tr>
</tbody>
</table>

depends on design
depends strongly on granularity

application overhead is still in this "nett" number
Design recommendations separation of concerns

- Decoupling
- Queues or buffers
- Minimal shared resources
- Process as unit of execution
- Clear single demarcation between hard and soft
- Minimize influence
- Performance
- Separation
- Cost
- Manage tension explicit

Execution architecture concepts

version: 1.1
March 6, 2013
EACseparation
Design recommendations understandability

- Hard real-time systems should be explainable with a few A4 diagrams.
- Overview is based on understanding many (critical) details.
- Complex reality; many details, many relations.
- Simulation: additional means if declared indispensable this is often a symptom of poor models.
- Limited use of tasks, threads, priorities.
- Reasoning must be possible.
- To combine or not to combine?

Simple is better.
Granularity considerations

<table>
<thead>
<tr>
<th>unit of buffering</th>
<th>== or &lt;&gt;</th>
<th>unit of synchronization</th>
<th>== or &lt;&gt;</th>
<th>unit of processing</th>
<th>== or &lt;&gt;</th>
<th>unit of I/O</th>
</tr>
</thead>
</table>

- video frame
- video line
- pixel

**Fine grain:**
- flexible
- high overhead

**Coarse grain:**
- rigid
- low overhead
Design patterns

synchronous
- safety critical, reliable, subsystems
- very low overhead
- predictable
- understandable
- works best in total separation
- does not work for multiple rhythms

thread based
- Asynchronous applications and services
- separation of timing concerns
- sharing of resources (no wait)
- poor understanding of concurrency
- danger of high overhead

timer based
- regular rhythm;
  - e.g. monitor HW status, update time, status display
- low "tunable" overhead
- understandable
- fast rhythms significant overhead

interrupt based
- I/O and HW events
  - data available, display frame sync
- separation of timing concerns
- definition of interrupts determines:
  - overhead, understandability
Synchronous design

**Execution architecture concepts**

13 Gerrit Muller

version: 1.1
March 6, 2013
EACsynchronousDesign

**double buffer:**
full decoupling of calculation and execution
Actual timing on logarithmic scale

- FO4 inverter delay: \(10^{-12}\) (ps)
- Cycle 2 GHz CPU
- Pure context switch
- Zero message transfer
- Application level message exchange
- Application level function response
- 100 Hz TV frame
- Human eye
- Eye-hand co-ordination
- Human reaction time
- Human 1st irritation threshold
- Human 2nd irritation threshold

From low to high level processing times

- From low to high level storage/network

Execution architecture concepts

14 Gerrit Muller

version: 1.1
March 6, 2013 RVTimeAxis
Typical micro benchmarks for timing aspects

<table>
<thead>
<tr>
<th>Category</th>
<th>Infrequent Operations, Often Time-Intensive</th>
<th>Often Repeated Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>start session</td>
<td>perform transaction</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td>query</td>
</tr>
<tr>
<td><strong>network, I/O</strong></td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td><strong>high level construction</strong></td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope other context</td>
</tr>
<tr>
<td><strong>low level construction</strong></td>
<td>object creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td><strong>basic programming</strong></td>
<td>memory allocation</td>
<td>function call</td>
</tr>
<tr>
<td></td>
<td>memory free</td>
<td>loop overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></td>
<td>power up, power down</td>
<td>cache flush</td>
</tr>
<tr>
<td></td>
<td>boot</td>
<td>low level data transfer</td>
</tr>
</tbody>
</table>
The transfer time as function of blocksize

- Time
- Block size
- Worst case
- Optimal block-size
- Overhead rate $^{-1}$

Execution architecture concepts

16 Gerrit Muller

version: 1.1
March 6, 2013
RVparametrizedTransferRate
### Example of a memory budget

<table>
<thead>
<tr>
<th>memory budget in Mbytes</th>
<th>code</th>
<th>obj data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared code</td>
<td>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</td>
<td>0.3</td>
<td>1.2</td>
<td>9.0</td>
<td>10.5</td>
</tr>
<tr>
<td>optical storage server</td>
<td>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.3</td>
<td>0.2</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td><strong>application SW total</strong></td>
<td>13.4</td>
<td>12.6</td>
<td>35.0</td>
<td>61.0</td>
</tr>
<tr>
<td>UNIX Solaris 2.x</td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td>file cache</td>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td></td>
<td></td>
<td></td>
<td>74.0</td>
</tr>
</tbody>
</table>
Complicating factors and measures

<table>
<thead>
<tr>
<th>Complications</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>considered margin</td>
</tr>
<tr>
<td>bus allocation</td>
<td>explicit behavior</td>
</tr>
<tr>
<td>memory management</td>
<td>architecture rules</td>
</tr>
<tr>
<td>garbage collection</td>
<td>monitoring, logging</td>
</tr>
<tr>
<td>memory (buffer, storage)</td>
<td>pool management</td>
</tr>
<tr>
<td>fragmentation</td>
<td>feedback to architect</td>
</tr>
<tr>
<td>non preemptable OS activities</td>
<td>flipover simulation</td>
</tr>
<tr>
<td>&quot;hidden&quot; dependencies (ie [dead]locks)</td>
<td></td>
</tr>
<tr>
<td>systematic &quot;coincidences&quot;, avalanche triggers</td>
<td></td>
</tr>
<tr>
<td>instable response, performance</td>
<td></td>
</tr>
</tbody>
</table>