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 concepts

version: 1.1
October 20, 2017
CV_executionArchitecture

Execution Architecture

dead lines
timing, throughput
requirements

other architecture
views

functional
model
receive
demux
display
process
store

hardware
CPU
DSP
RAM
tuner
drive

repository
structure
Applications
play
zap
list
UI toolkit
menu
processing
DCT
foundation
classes
queue
list
hardware
abstraction
interrupt
handlers

execution architecture
issues:
concurrency
scheduling
synchronisation
mutual exclusion
priorities
granularity

Map

process
task
thread
Fuzzy customer view on real time

- hard real time
  - disastrous failure
  - human device safety
  - loss of information
  - loss of eye hand coordination

- soft real time
  - dissatisfaction
  - device safety
  - loss of functionality
  - limited throughput
  - waiting time

Execution architecture concepts
version: 1.1
October 20, 2017
EAChardVsSoft
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

- Limited set of hard real time cases
  - Time: 20 ms
  - Frequency: 20 ms

- Well defined set of performance critical cases
  - Time: 200 ms, 500 ms
  - Frequency: 90%
Latency

connection latency

perceived delay

connection latency

speak
listen

speak
listen

bla bla bla

reaction

Long distance connection

Execution architecture concepts
Gerrit Muller
version: 1.1
October 20, 2017
EAClatency
Response Time

- New channel
- P+
- P-
- Remote control
- Zap
- Total response time
- Zap repetition
- Visual feedback
- Open for next response
- New channel
- Visual feedback time
throughput:
+ processing steps/frame
+ frames/second
+ concurrent streams
bus bandwidth, processor load [memory usage]
useful macroscopic views, be aware of microscopic behavior

- margin
- loss = not schedulable
- overhead: bus, OS, scheduling
- function 4
- function 3
- function 2
- function 1

depends on design
depends strongly on granularity

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

- Soft Real Time
- Hard Real Time

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

Execution architecture concepts
Gerrit Muller

version: 1.1
October 20, 2017
EACseparation
Design recommendations understandability

- Complex reality: many details, many relations
- Limited use of tasks, threads, priorities
- Hard real time systems should be explainable with a few A4 diagrams
- Overview is based on understanding many (critical) details
- Simple is better
- Reasoning must be possible
- Simulation: additional means
- If declared indispensable, this is often a symptom of poor models
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;
- low "tunable" overhead
- understandable
- fast rhythms significant overhead

**interrupt based**
- I/O and HW events
- separation of timing concerns
- definition of interrupts determines:
  - overhead, understandability
Synchronous design

**Execution architecture concepts**

**version:** 1.1  
**October 20, 2017**

**double buffer:** full decoupling of calculation and execution
Typical micro benchmarks for timing aspects

<table>
<thead>
<tr>
<th></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 query</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td></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</td>
</tr>
<tr>
<td><strong>low level construction</strong></td>
<td>object creation</td>
<td>other context</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

- optimal block-size
- worst case
- \( t_{\text{overhead}} \)
- \( \text{rate}^{-1} \)
## Example of a memory budget

<table>
<thead>
<tr>
<th></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</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</td>
<td>0.8</td>
</tr>
<tr>
<td>application SW total</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>total</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) fragmentation</td>
<td>pool management</td>
</tr>
<tr>
<td>non preemptable OS activities</td>
<td>feedback to architect</td>
</tr>
<tr>
<td>&quot;hidden&quot; dependencies (ie [dead]locks)</td>
<td>flipover simulation</td>
</tr>
<tr>
<td>systematic &quot;coincidences&quot;, avalanche</td>
<td></td>
</tr>
<tr>
<td>triggers</td>
<td></td>
</tr>
<tr>
<td>instable response, performance</td>
<td></td>
</tr>
</tbody>
</table>