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

Other architecture views

Functional model

Execution architecture

Hardware

Repository structure

Execution architecture issues:
- concurrency
- scheduling
- synchronisation
- mutual exclusion
- priorities
- granularity

Dead lines, timing, throughput requirements

Execution architecture concepts

Version: 1.1
June 5, 2018
Gerrit Muller
Fuzzy customer view on real time

- hard real time
- soft real time

Disastrous failure
Dissatisfaction

Human device safety
Loss of information

Limited throughput
Waiting time

Eye hand coordination
Loss of functionality
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

connection latency

speak

listen

bla bla bla

reaction

long distance connection
Response Time

- `P+` and `P-` for remote control
- `zap` and `new channel`
- `total response time` and `visual feedback time`
- `zap repetition` and `visual feedback`
throughput:
+ processing steps/frame
+ frames/second
+ concurrent streams
bus bandwidth, processor load [memory usage]
useful macroscopic views, be aware of microscopic behavior

- 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

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

Execution architecture concepts

Gerrit Muller

version: 1.1
June 5, 2018
EACseparation
Design recommendations understandability

- Hard real time systems should be explainable with a few A4 diagrams.
- Reasoning must be possible to combine or not to combine?
- Limited use of tasks, threads, priorities.
- Overview is based on understanding many (critical) details.
- Complex reality; many details, many relations.
- Simple is better.

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

HW: input for t_n, input for t_{n+1}, input for t_{n+2}, input for t_{n+3}
SW: calculate t_{n-1}, calculate t_n, calculate t_{n+1}, calculate t_{n+2}
HW: execute t_{n-2}, execute t_{n-1}, execute t_n, execute t_{n+1}

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

<table>
<thead>
<tr>
<th>Scale</th>
<th>Time Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(ps)</td>
<td>10^{-12}</td>
</tr>
<tr>
<td></td>
<td>(ns)</td>
<td>10^{-9}</td>
</tr>
<tr>
<td></td>
<td>(μs)</td>
<td>10^{-6}</td>
</tr>
<tr>
<td></td>
<td>(ms)</td>
<td>10^{-3}</td>
</tr>
<tr>
<td></td>
<td>(s)</td>
<td>1</td>
</tr>
</tbody>
</table>

- FO4 inverter delay
- Cycle 2 GHz CPU
- Pure context switch
- Zero message transfer
- Appl level message exchange
- Appl level function response
- Disk seek
- Appl level network message exchange
- Appl level function response
- From low to high level processing times

- Light travels 1 cm
- 100Hz video pixel time
- 100Hz video line
- 100 Hz TV frame
- Human eye
- Eye-hand co-ordination
- Human reaction time
- Human 1st irritation threshold
- Human 2nd irritation threshold

- Application needs

Execution architecture concepts
Gerrit Muller
## 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 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

- worst case
- optimal block-size

$t_{\text{overhead}}$
### 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</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>
</tbody>
</table>

| application SW total    | 13.4 | 12.6     | 35.0      | 61.0  |

| UNIX Solaris 2.x        |      |          |           | 10.0  |
| file cache              |      |          |           | 3.0   |

| total                   |      |          |           | 74.0  |
Complicating factors and measures

**Complications**
- cache
- bus allocation
- memory management
- garbage collection
- memory (buffer, storage) fragmentation
- non preemptable OS activities
- "hidden" dependencies (ie [dead]locks)
- systematic "coincidences", avalanche triggers
- instable response, performance

**Measures**
- considered margin
- explicit behavior
- architecture rules
- monitoring, logging
- pool management
- feedback to architect
- flipover simulation