Research Trends in Real-Time Computing for Embedded Systems

Giorgio Buttazzo

Presenter: Henrik Kragh-Hansen

October 18, 2007
## The Article

### Published
- ACM SIGBED Review (July 2006)
- Special issue on major international initiatives on real-time and embedded systems

### Contributions
- Presents major research trends
- Suggestions for improvements of real-time OS
- More predictable and adaptive to environmental changes
The Author

Giorgio Buttazzo

- Full Professor of Computer Engineering at the Scuola Superiore Sant’Anna of Pisa
- Ph.D. in Computer Engineering in 1991
- Electronic Engineering at the University of Pisa in 1985
- Founded and coordinated the RETIS Laboratory on real-time systems
- Main research interests: real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks.
Embedded Systems

Problems
- Resource constraints
- Real-time
- Dynamic environment
Evolution of Embedded Systems

The last 30 years

- Exponential growth in numbers and complexity
- Not followed by control software
- No good alternatives so far
- Increase of abstraction results in unacceptable memory and speed penalties
Outline

1. Embedded Systems
2. Current Approach
3. Future Directions
4. Conclusion
## Properties of Embedded Systems

### Limited resources
- Space, weight, and energy constraints
- Limited memory and computational power
- Cost constraints

### Real-time constraints
- React to external events within timing constraints
- Predictable execution must be ensured
- Off-line analysis of requirements

### Dynamic behavior
- Concurrent activities compete for shared resources
- Depends on sensory data inputs
- Workload varies a lot during execution
Real-Time Requirements

- Classic WCET is too restrictive
- Must adapt to available resources
- Adaption should be handled at lower layers
- Real-time scheduling and resource management
## Types of Real-Time Systems

### Control software
- Periodic tasks and sensor input
- Hard timing requirements

### Media processing
- High consumer of hardware resources
- Soft real-time aperiodic tasks with QoS

### Interaction software
- Complex and increasing in size
- Interactive-response requirement
Task priorities

Fixed priority
- Expresses the importance of a task
- Priority is specified at compile time
- Grouping tasks is difficult due to global priority

Dynamic priorities
- Changes the priority of a task on runtime
- Better usage of available resources
- Built on top of fixed priority scheduling
Operating Systems should be

**Reflective**
- Reflects the application characteristics
- Deadline, periodicity constraints, importance, QoS values, computation time etc.

**Resource aware**
- Partitioning the available resources

**Informative**
- Provide information of the current state of execution
- Adjust system parameters based on difference from expected behavior
Resource Reservation

- Programmer can control resources assigned to tasks
- Constant Bandwidth Server (CBS)
- Each task is guaranteed a bandwidth
- Deadlines are dynamically computed
- Overruns are isolated to specific tasks

![Diagram of three tasks running under a CPU reservation scheme.]

- \( \tau_1 \) to CBS_1, \( U_s = 0.2 \)
- \( \tau_2 \) to CBS_2, \( U_s = 0.5 \)
- \( \tau_3 \) to CBS_3, \( U_s = 0.3 \)

CPU

Ready queue
Efficient Resource Reclaiming

- Resources reserved to tasks must correspond to the amount needed
- Requires execution tests and code analysis, to estimate resource consumption
- Unused resources are scheduled to other tasks
- Quite efficient for small reservation errors
Integrating Real-Time and Control

Control theory
- When execution is very variable and unpredictable
- Estimates the current workload and tune parameters

Use cases
- Used in kernel to be more adaptive
- Used during the design of the system
Flexible Scheduling

- Requirements on tasks may change dynamically
- Algorithms might change during runtime
- Timing constraints other than deadlines and periods
- Scheduling algorithm must adapt changing parameters
- Trade-off between predictable performance and efficient use of resources
Hierarchical Scheduling

- Resources have a hierarchy of schedulers
- Tasks are hierarchically grouped
- Scheduling algorithm schedules the children of that node
- A root scheduler can guarantee resource reservation and temporal isolation
- Can simulate independent systems
## Overload Handling

<table>
<thead>
<tr>
<th>Selection of different QoS levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Selecting different algorithms</td>
</tr>
<tr>
<td>- Precision can be enhanced by increasing number of steps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adjustable timing constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Relax timing constraints</td>
</tr>
<tr>
<td>- Increase periods on periodic threads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admission control</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Add a filter to incoming requests</td>
</tr>
<tr>
<td>- Very drastic approach</td>
</tr>
</tbody>
</table>
Energy-Aware Scheduling

- Performance vs. energy consumption
- Scheduling algorithms must take voltage into account
- Currently algorithms focus on CPU
- Memory, disk, communication devices, and input/output
- Energy overhead of the scheduling algorithm?
Portability

- Standard programming interface
- Several operating system providers
- Promotes competition, increasing quality
- Current standards must be extended with more flexibility
Modeling

Currently use of models
- Only used in early phases of system design
- Requirements are difficult to track
- Validation done by experiments

What should models be used for?
- Support every step of the design process
- Support validation tools
Towards Component-based RTOS

Component based kernel
- The designer can configure the kernel
- Speeding up development process and efficiency
- The ability to replace the scheduling algorithm or resource management protocol
- The ability to combine different schedulers

Advantages
- Enhance the functionality of the kernel
- Tailor the kernel to the specific system
- Easier to integrate new research results
- Easier to port the kernel to other platforms
Next Generation of Embedded Systems

- Requires a new real-time operating system
- Execute efficiently on limited resources
- Higher level of abstraction
- Support explicit resource allocation and QoS functionality
- Adaptive to resource needs
New Research Areas

- Flexible scheduling services
- Protection - space and time
- Dynamicity
- Quality of Service
- Multiprocessor support
- Drivers
- Networks
- Modeling
## My Opinion

### Pros
- Many ideas/research areas
- Focus on different aspects
- Good structure

### Cons
- Existing programming languages and tools?
- No new stuff
What do we have today?

<table>
<thead>
<tr>
<th>Real-time Java (RTSJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fixed priority</td>
</tr>
<tr>
<td>• Portability</td>
</tr>
<tr>
<td>• Higher level of abstraction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIMES and Bandera</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Modeling tools</td>
</tr>
<tr>
<td>• Verifies design based on finite-state models</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>aJile and JOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>• CPUs executing Java byte code</td>
</tr>
</tbody>
</table>
Thank you for your Attention

Any questions?