YASA - A Framework for Validation, Test, and Analysis of Real-Time Scheduling Algorithms

Jan Blumenthal, Jens Hildebrandt, Frank Golatowski, and Dirk Timmermann
University of Rostock, Germany

5th Real-Time Linux Workshop
Valencia, 2003/11/13
Outline

• Preliminaries
• YASA Framework
  – Introduction & vision
  – Integration into target systems
  – Configuration & evaluation
• Conclusion
Design Goals in Real-Time Systems

Design Goals: - Proof of functionality
- Proof of timeliness

Our goal: Reduction of deadline misses
- Concentration on optimization of scheduling behavior
Scheduling

Task: Assignment of processes to CPU

Priority based:

Earliest Deadline First:

- Analysis required
- Exchange of schedulers would be favorable
Synchronization Protocols

**Semaphores:**
- Protect critical areas
- Typical tasks:
  - Data exchange between processes
  - Request for resources

**Synchronization protocols:**
- Prevent while entering critical sections
  - Priority inversion
  - Deadlocks
- Often tightly coupled with schedulers

Exchange of synchronization protocols required, too!
Current Systems

- Operating system and system scheduler are tightly coupled
- Dynamic schedulers or synchronization protocols are not supported
- Unknown scheduling behavior
Goal: Optimization of scheduling behavior in real-time operating systems

Features:

- Analysis of scheduling behavior
- Exchange of schedulers
- Support of additional static and dynamic synchronizations protocols
Design Flow Comparison

**Current flow:**
- Development → Execution
- Adjustments → Trial & Error
- Adjustments → Trial & Error

**YASA 2:**
- Development → Execution
- Optimization → Analysis
- Logfile

**Characteristics:**
- Unknown scheduling behavior
- No analysis
- Poor failure detection

**Characteristics:**
- Logged scheduling behavior
- Analysis of scheduling behavior
- Better failure detection
Design Flow in YASA

Graphical User Interface
- Processes
- Executives
- Scheduler

YASA - Workspace
- Environment settings
- Evaluation/analysis

Target System
- Project creation
- Execution
- Logfile

- Project management & configuration
- Source code generation
- Evaluation

- Compilation
- Execution
- Logfile creation
Project Cycle in YASA

Configuration phase
- Processes
- Semaphores
- Sync. protocols
- Scheduler
- Executive

Creation phase
- Process attributes
- Semaphore attributes
- Process sources
- Kernel sources
- Kernel patch
- YASA support functions

Execution phase
- Executable project
- Operating System Kernel
- Logfile
- Evaluation/visualization
Integration of YASA

- Encapsulate scheduling environment into Executives!
Design of an Executive

- Virtual scheduling environment
- Interface between scheduler and processes
- Support for platform independent schedulers
Simulator-Executive

- Execution of applications with synthetic work-load
- Virtual program flow
- Simulation of
  - Process states
  - Resource conditions
- Multiprocessing capable
- Platform independent
- Application fields:
  - Proof of theoretical analysis
  - Development and test of new schedulers
Project Cycle Using Simulator

Configuration phase
- Processes
- Mutexes
- Sync. protocols
- Scheduler
- Executive

Creation phase
- Process attributes
- Mutex attributes
- Process sources
- Simulator sources
- YASA support functions

Execution phase
- Executable project
- Simulator library
- Logfile

Evaluation/visualization

Particularities of the Simulator-Executive
RT-Linux Executive

- Adaptation layer to RT-Linux
- Execution of real applications
- Enhancements of POSIX-API

Changes by RT-Linux Executive
RT-Linux Coprocessor Executive

- Integration into OS by replacement / extension of several kernel functions
- **Parallel computation** (favorable for ELLF)

Executive not finished, yet!
Coprocessor – Internal Design

- task module 0
- task module 1
- task module n-1
- comparator/selector
- output format converter
- control logic

Connections:
- address decoder
- comparator/selector
- control logic
- output format converter

Signals:
- address
- data
- /cs
- /id
- /rd
- tck
- err_int
- sched_int
Development Environment
Scheduler

Properties:
- Replacement during runtime possible
- Implementation
  - Executive independent
  - Platform independent
  - No limitations in time resolution and process count

Available schedulers:

<table>
<thead>
<tr>
<th>Static schedulers</th>
<th>Dynamic schedulers</th>
<th>Cyclic schedulers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority based</td>
<td>Earliest Deadline First</td>
<td>Round Robin</td>
</tr>
<tr>
<td>Rate Monotonic</td>
<td>Least Laxity First</td>
<td>First Come – First Serve</td>
</tr>
<tr>
<td>Deadline Monotonic</td>
<td>Enhanced Least Laxity First</td>
<td></td>
</tr>
</tbody>
</table>
Synchronization Protocols

Properties:

- Synchronize operating resources
  - Reduce effect of priority inversion
  - Prevent deadlocks

Available synchronization protocols:

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority Inheritance Protocol (PIP)</td>
<td>Dynamic Priority Ceiling Protocol (DPCP)</td>
</tr>
<tr>
<td>Priority Ceiling Protocol (PCP)</td>
<td>Stack Resource Policy (SRP)</td>
</tr>
<tr>
<td>Ceiling Semaphore Protocol (CSP)</td>
<td></td>
</tr>
</tbody>
</table>
Synchronization Protocols II

- Decoupling synchronization protocols and schedulers
- Supporting **dynamic priority types**
- Combining synchronization protocols with several schedulers at runtime
  - Dynamic synchronization protocols with every scheduler
  - Static synchronization protocols with every static scheduler

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Type</th>
<th>Possible Synchronization Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority Based Scheduler</td>
<td>Static</td>
<td>• Priority Inheritance Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Priority Ceiling Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ceiling Semaphore Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Dynamic Priority Ceiling Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Stack Resource Policy</td>
</tr>
<tr>
<td>Earliest Deadline First</td>
<td>Dynamic</td>
<td>• Dynamic Priority Ceiling Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Stack Resource Policy</td>
</tr>
</tbody>
</table>
Dynamic Priority Types

- Comparison methods and data types in synchronization protocols and schedulers must match.
- Dynamic priority type:
  - depends on scheduler
  - controls comparison operation
  - affects data type size
- Access to data type through macros during compilation

### Possible Priority Types:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data type</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Static</td>
<td>int</td>
<td>$P_1 &gt; P_2$</td>
</tr>
<tr>
<td>Period</td>
<td>Static</td>
<td>YASA_TIME</td>
<td>$P_1 &lt; P_2$</td>
</tr>
<tr>
<td>Laxity</td>
<td>Dynamic</td>
<td>YASA_TIME</td>
<td>$P_1 &lt; P_2$</td>
</tr>
<tr>
<td>Deadline</td>
<td>Dynamic</td>
<td>YASA_TIME</td>
<td>$P_1 &lt; P_2$</td>
</tr>
<tr>
<td>Remaining runtime</td>
<td>Dynamic</td>
<td>YASA_TIME</td>
<td>$P_1 &gt; P_2$</td>
</tr>
</tbody>
</table>
Graphical User-Interface YASA

- Object-oriented programming
- Using QT class library (261 classes)
- Platform independent
- Utilize Design Patterns
- Multilingual (UNICODE)
- Evaluate application logfiles in terms of

graphics

Task diagram
Mutex diagram
Processor diagram

tables

Task table
Mutex table
Processor table
Welcome to YASA - the amazing scheduling analyzer

First hint: If you want to create a new project please click with the right mouse button on "Projects" and select add!

Parsing log file "~/webprojects/stud/Blu/yasa/projects/multiplemutex/workdir/RT/Linux_ELFF_Logfile"

Generating statistics.

Logfile read successfully.

Cleaning up environment "RT/Linux (ELFF)" of project "RT/Linux 1".

Changing directory to: "~/webprojects/stud/Blu/yasa/projects/multiplemutex/workdir/RT/Linux_ELFF"

bash -c "cleanup"
Graphical Evaluation I

Represent timing diagrams in
- Task order
- Mutex order
- Processor order
Graphical Evaluation II

Visualization in Environment order!
Conclusion

• Framework YASA
  – Scheduling analysis with different schedulers
  – Extends existing real-time operating systems
  – Platform independent schedulers and synchronization protocols
  – Simulation and execution of real programs
  – Graphical front-end for configuration and evaluation issues

• Homepage
  http://yasa.e-technik.uni-rostock.de
  http://sourceforge.net/projects/yasa
Thank you

Jan Blumenthal, Jens Hildebrandt, Frank Golatowski, and Dirk Timmermann
Universität Rostock

5th Real-Time Linux Workshop
Valencia, 2003/11/13