Adaptability, Extensibility, and Flexibility in Real-Time Operating Systems: the Georgia Tech DRTOS

Pramote Kuacharoen, Tankut Akgul, Professor Vincent Mooney and Professor Vijay K. Madisetti

{pramote,tankut,mooney,vkm}@ece.gatech.edu

School of Electrical and Computer Engineering
Georgia Institute of Technology

This research is funded by the State of Georgia under the Yamacraw initiative and by NSF.

July 31, 2001
Outline

- Introduction
- Related Work
- Technical Approach
- Experiments and Results
- Conclusion
Related Work

- **SPIN (University of Washington) 1996**
  - A general purpose operating system that provides extensibility, safety and good performance.
  - SPIN OS consists of a set of extension services and core system services that execute within the kernel’s virtual address space.
  - Extensions can be loaded into kernel at any time. Once loaded, they integrate themselves into the existing infrastructure and provide system service specific to the application that require them.
  - User space and kernel space are kept separate.
  - Single processor.
  - The core system services cannot be changed.
Exokernel (MIT) 1997

- General purpose operating system.
- Exokernel’s sole function is to allocate, de-allocate, and multiplex physical resources in a secure way (very good protection is provided).
- The lower level interface allows flexible user-level implementations of traditionally rigidly defined OS services.
- Single processor.
- The core kernel code cannot be changed.
Why is it important to be able to dynamically change the core?

- Unsafe or not convenient to reboot
- Adaptability and Flexibility
  - Example: interrupt handling
    - Case 1: very fast handling of interrupts (always stop current interrupt)
    - Case 2: non-interruption of a particular interrupt
  - There may be no way to predict all the additional cases which could come up
Embedded Systems

Application

RTOS

Hardware

Priority-Based Scheduler

Earliest-Deadline Scheduler

Core

Time

Task

CPU

Memory

I/O

Application 1

Task1 Task2 Task3

Application 2

Task1 Task2 Task3

Priority-Based Scheduler

Earliest-Deadline Scheduler
### Module Installation

<table>
<thead>
<tr>
<th>Kernel Module</th>
<th>Core Module</th>
<th>Task Manager Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Executable code</td>
<td>Executable code</td>
<td></td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module variables</td>
<td>Module variables</td>
<td></td>
</tr>
<tr>
<td><strong>APIs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System APIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pCore</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pTime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pTask Manager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pScheduler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>APIs</td>
<td></td>
</tr>
</tbody>
</table>

**Create Task**
DRTOS Technical Approach (continued)

APIs Invocation

**Core Module**
- Executable code
- Module variables
- System APIs
  - pCore
  - pTime
  - pTask Manager
  - pScheduler
- APIs

**Scheduler Module**
- Executable code
- Module variables
- System APIs
- APIs

**Task Manager Module**
- Executable code
- Module variables
  - pSystem APIs
- APIs
  - createTask

Georgia Institute of Technology
Updating Scheduler module

- Load a priority-based scheduler
- Link the priority-based scheduler to the core module
- Unlink the round robin scheduler
- The round robin scheduler can be deleted from the memory
Function pointers are used for APIs

Invocation of a scheduler API

- `System *pSys = (System *) pTaskData->pSystem;`
- `SchedulerMethod *scheduler = (SchedulerMethod *) pSys->scheduler;`
- `scheduler->schedule(pNewTask, TASK_READY);`

Updating Core module

- Either (1) System variable must either be in the same location as before
- Or (2) each module must be notified when Core module is updated (all modules’ System variables must point to the new location for the System variable).
Updating Loader module

- The current loader module is called to update to the new loader module.
- The `initModule()` function of the new loader module is invoked.
- The return address from the `initModule()` function must be adjusted to the location which calls the update API of the old loader module by clearing the stack to ignore the call from the old loader module.
- The old loader module can be deleted from the memory.
Updating Loader module

Core Module
- Executable code
- Module variables
- System APIs
  - Core
  - Time
  - Task Manager
  - Scheduler
  - Loader
- APIs

New Loader
- Executable code
- Module variables
- System APIs
  - initModule()
  - loadModule()

OS Stack
- After adjusting the return address
- Top of Stack
Experiment 1

Simulation Environment

VCS → Seamless CVE → XRAY

OFDM Transmitter

\[ \begin{align*}
X(m) & \rightarrow \text{Serial to Parallel} \rightarrow X(n) \\
& \rightarrow \text{IDFT & Cyclic Prefix}
\end{align*} \]

MPC750 → Kernel Modules
MPC750 → Memory
MPC750 → Interrupt Service Routines

Interrupt

MPC750

\[ \begin{align*}
\text{MPC750}^* & \rightarrow \text{Dynamic RTOS Services} \\
& \rightarrow \text{Kernel Switching Code}
\end{align*} \]

*: Processor Running Kernel Management Process

Storage Device
Initially running VUI code
- uses round-robin scheduler

Want to change to OFDM code
- install new I/O code
- install new priority scheduler code

OFDM Code Size: 1600 lines of code

Time to load (VUI still operational): 4 kbytes x 2 cycles/byte

Time to switch to new DRTOS code: 60 cycles
Experiment 2

Simulation Environment

- MBX860
- JTAG
- CrossView

Switching between schedulers

- Priority-based
- Round Robin
Three tasks
- DRTOS uses priority-based scheduler
Change to round robin scheduler
- install round robin scheduler code
- migrate tasks from the previous scheduler
Round Robin Code Size: 200 lines of code
Switching Time: $60 + 8n$ assembly instructions
\(n\) = number of tasks currently in the system, the scheduler needs to poll each task to get its handle
Priority-based scheduler

Round robin scheduler
Conclusion

- Existing real-time operating systems not fully dynamic
- The needs of a new real-time operating system architecture to support emerging applications
- Our approach: the Georgia Tech DRTOS
- Initial experiments and results