Embedded and Real-time Operating Systems

Dr. Kuan-Hsun Chen
LS 12, TU Dortmund
October 24, 2019

(slides are based on Prof. Dr. Chen and Prof. Dr. Marwedel)
Outline

- Embedded operating systems
  - Characteristics

- Real-time operating systems (RTOS)
  - Definition and requirement
  - Kernels
  - Classes
Outline

- Embedded operating systems
  - Characteristics

- Real-time operating systems (RTOS)
  - Definition and requirement
  - Kernels
  - Classes
Embedded operating systems

- Characteristics: Disk and network handled by tasks -

- Effectively no device needs to be supported by all variants of the OS, except maybe the system timer.
- Many ES without disk, a keyboard, a screen or a mouse.
- Disk & network handled by tasks instead of integrated drivers.

<table>
<thead>
<tr>
<th>Embedded OS</th>
<th>Standard OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>application software</td>
<td>application software</td>
</tr>
<tr>
<td>middleware</td>
<td>middleware</td>
</tr>
<tr>
<td>device driver</td>
<td>device driver</td>
</tr>
<tr>
<td>kernel</td>
<td>operating system</td>
</tr>
<tr>
<td>device driver</td>
<td>device driver</td>
</tr>
</tbody>
</table>
Embedded operating systems
- Characteristics: Protection is optional -

• Protection mechanisms not always necessary:
  ES typically designed for a single purpose, untested programs rarely loaded, SW considered reliable.

• Privileged I/O instructions not necessary and tasks can do their own I/O.

Example: Let switch be the address of some switch
Instead of OS call, simply use:

  load register,switch

• However, protection mechanisms may be needed for safety and security reasons.
Embedded operating systems
- Characteristics: Interrupts not restricted to OS -

• Interrupts can be employed by any process
  ▪ Embedded programs can be considered to be tested
  ▪ Since protection is not always necessary
  ▪ Since efficient control over a variety of devices is required
    ▪ More efficient than going through OS services
  ▪ It is possible to let interrupts directly start or stop SW

• For standard OS: serious source of unreliability
Embedded operating systems
- Characteristics: Real-time capability -

Many embedded systems are real-time (RT) systems and, hence, the OSs used in these systems must be real-time operating systems (RTOSs).
Outline

- Embedded operating systems
  - Characteristics

- Real-time operating systems (RTOS)
  - Definition and requirement
  - Kernels
  - Classes
RTOS - Definition and requirement 1: predictability -

Def.: (A) *real-time operating system is an operating system that supports the construction of real-time systems.*

Key requirements:

1. The timing behavior of the OS must be predictable. ∀ services of the OS: Upper bound on the execution time! RTOSs must be timing-predictable:
   - (for hard disks:) contiguous files to avoid unpredictable head movements.

   [Takada, 2001]
RTOS - requirement 2: Managing timing -

2. OS should manage the timing and scheduling
   - OS possibly has to be aware of task deadlines; (unless scheduling is done off-line).
   - OS should provide precise time services with high resolution.

[Takada, 2001]
RTOS-Kernels

Distinction between

- Real-time kernels and modified kernels of standard OSes.

<table>
<thead>
<tr>
<th>application software</th>
<th>application software</th>
</tr>
</thead>
<tbody>
<tr>
<td>middleware</td>
<td>middleware</td>
</tr>
<tr>
<td>device driver</td>
<td>device driver</td>
</tr>
<tr>
<td>real-time kernel</td>
<td></td>
</tr>
</tbody>
</table>

- General RTOSs and RTOSs for specific domains

Source: R. Gupta, UCSD
Functionality of RTOS-Kernels

Includes

- processor management,
- memory management,
- and timer management;
- task management (resume, wait etc),
- inter-task communication and synchronization.

resource management
Classes of RTOSs:
1. Fast proprietary kernels

For complex systems, these kernels are inadequate, because they are designed to be fast, rather than to be predictable in every respect

[R. Gupta, UCI/UCSD]

Examples include
QNX, PDOS, VCOS, VTRX32, VxWORKS.

Source: R. Gupta, UCSD
Classes of RTOSs:
2. RT extensions to standard OSs

Attempt to exploit comfortable main stream OS
RT-kernel running all RT-tasks
Standard-OS executed as one task

<table>
<thead>
<tr>
<th>RT–task 1</th>
<th>RT–task 2</th>
<th>non–RT task 1</th>
<th>non–RT task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>device driver</td>
<td>device driver</td>
<td>Standard–OS</td>
<td></td>
</tr>
</tbody>
</table>

real–time kernel

+ Crash of standard-OS does not affect RT-tasks;
– RT-tasks cannot use Standard-OS services; less comfortable than expected

Source: R. Gupta, UCSD
Example: RT-Linux

RT-tasks cannot use standard OS calls. Commercially available from FSMLabs (www.fsmlabs.com)

Dr. Kuan-Hsun Chen (LS 12, TU Dortmund)
Classes of RTOSs:
3. Research trying to avoid limitations

- Research systems trying to avoid limitations.
  - Include MARS, Spring, MARUTI, Arts, Hartos, DARK, and Melody

- Research issues [Takada, 2001]:
  - Low overhead memory protection
  - Temporal protection of computing resources
  - RTOSes for on-chip multiprocessors
  - Quality of service (QoS) control

Source: R. Gupta, UCSD