technische universität dortmund

fakultät für informatik

Embedded and Real-time Operating Systems

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

Anas Toma

LS 12, TU Dortmund

October 18, 2018

Outline

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

Outline

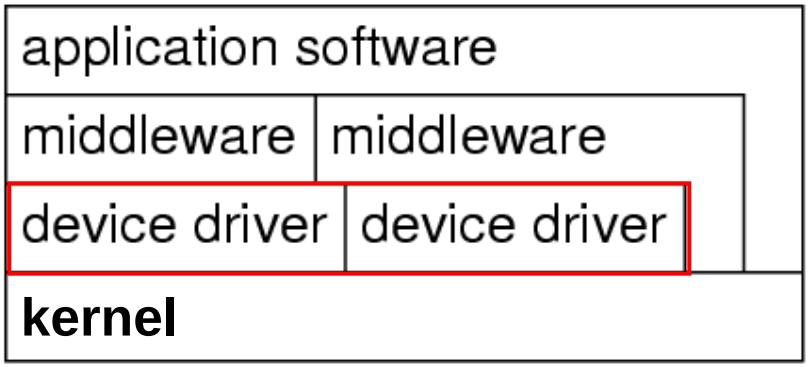
- Embedded operating systems
  - Characteristics
- Real-time operating systems (RTOS)
  - Definition and requirement
  - Kernels
  - Classes

Embedded operating systems

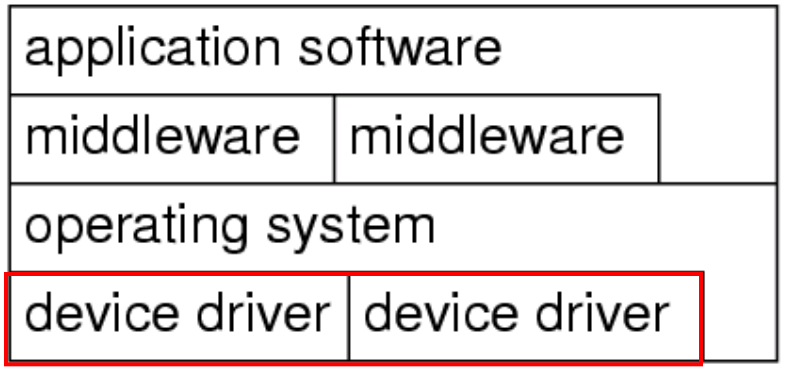
- Characteristics: Disk and network handled by tasks -

- 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.**

Embedded OS



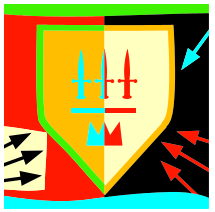
Standard OS



Embedded operating systems
- Characteristics: Protection is optional-

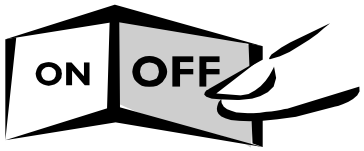
- Protection mechanisms not always necessary:
  ES typically designed for a single purpose, untested programs rarely loaded, SW considered reliable.

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



  Example: Let `switch` **be the address of some switch Simply use**

```
load register,switch
```



  instead of OS call.

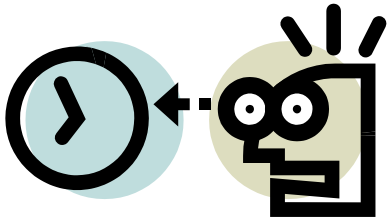
- However, protection mechanisms may be needed for safety and security reasons.

Embedded operating systems
- Characteristics: Interrupts not restricted to OS -

- Interrupts can be employed by any process
  - Embedded programs can be considered to be tested
  - Protection is not always necessary
  - 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 -

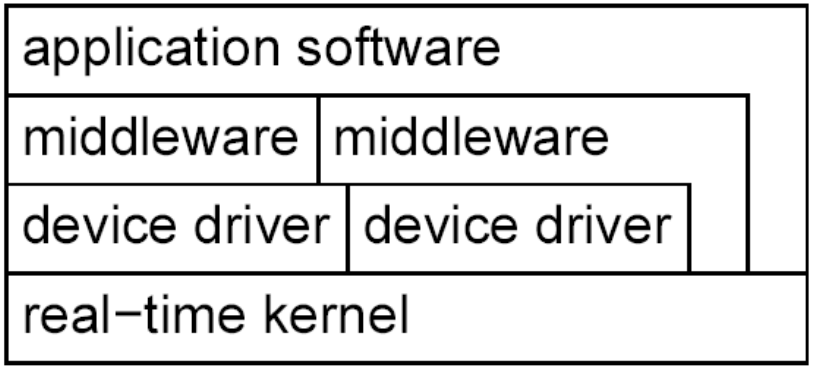
2. OS should manage the **timing and scheduling**
   - OS possibly has to be aware of task deadlines;
     (unless scheduling is done off-line).

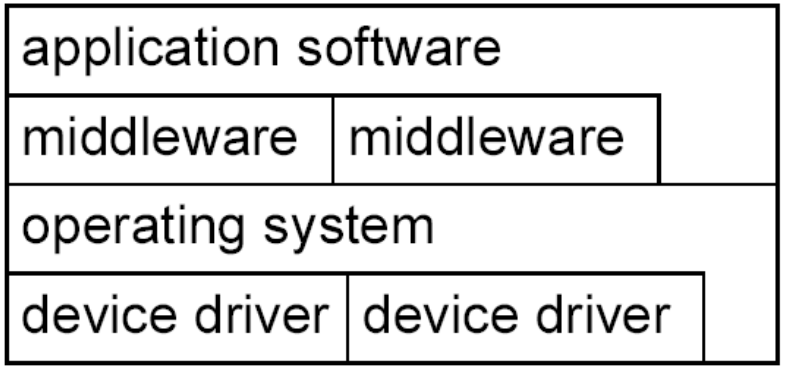
[Takada, 2001]

RTOS-Kernels

Distinction between

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





Distinction between

- **General** RTOSs and RTOSs for **specific domains**

Functionality of RTOS-Kernels

Includes

- processor management,
- memory management,
- and timer management;

} **resource management**

- task management (resume, wait etc),
- inter-task communication and synchronization.

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

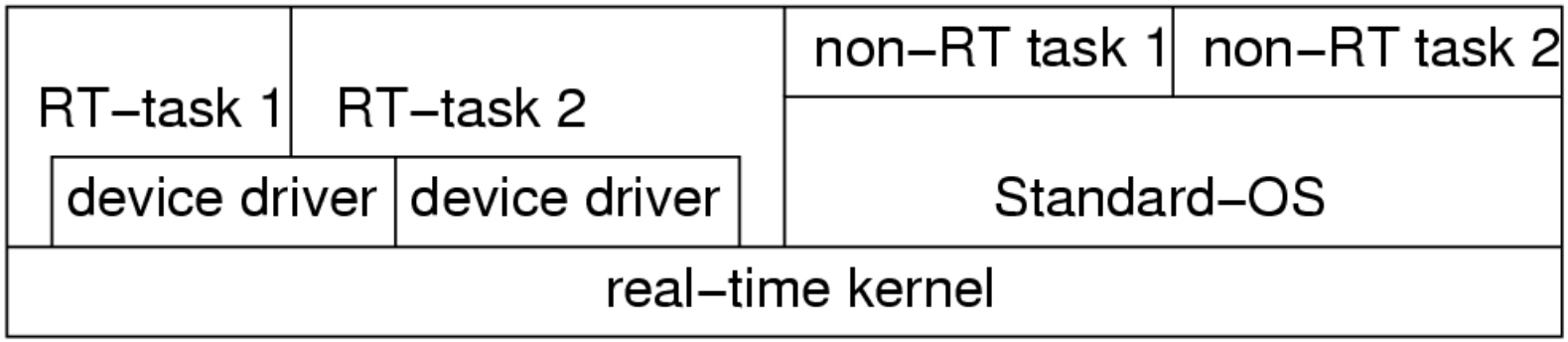
[R. Gupta, UCI/UCSD]

Examples include

QNX, PDOS, VCOS, VTRX32, VxWORKS.

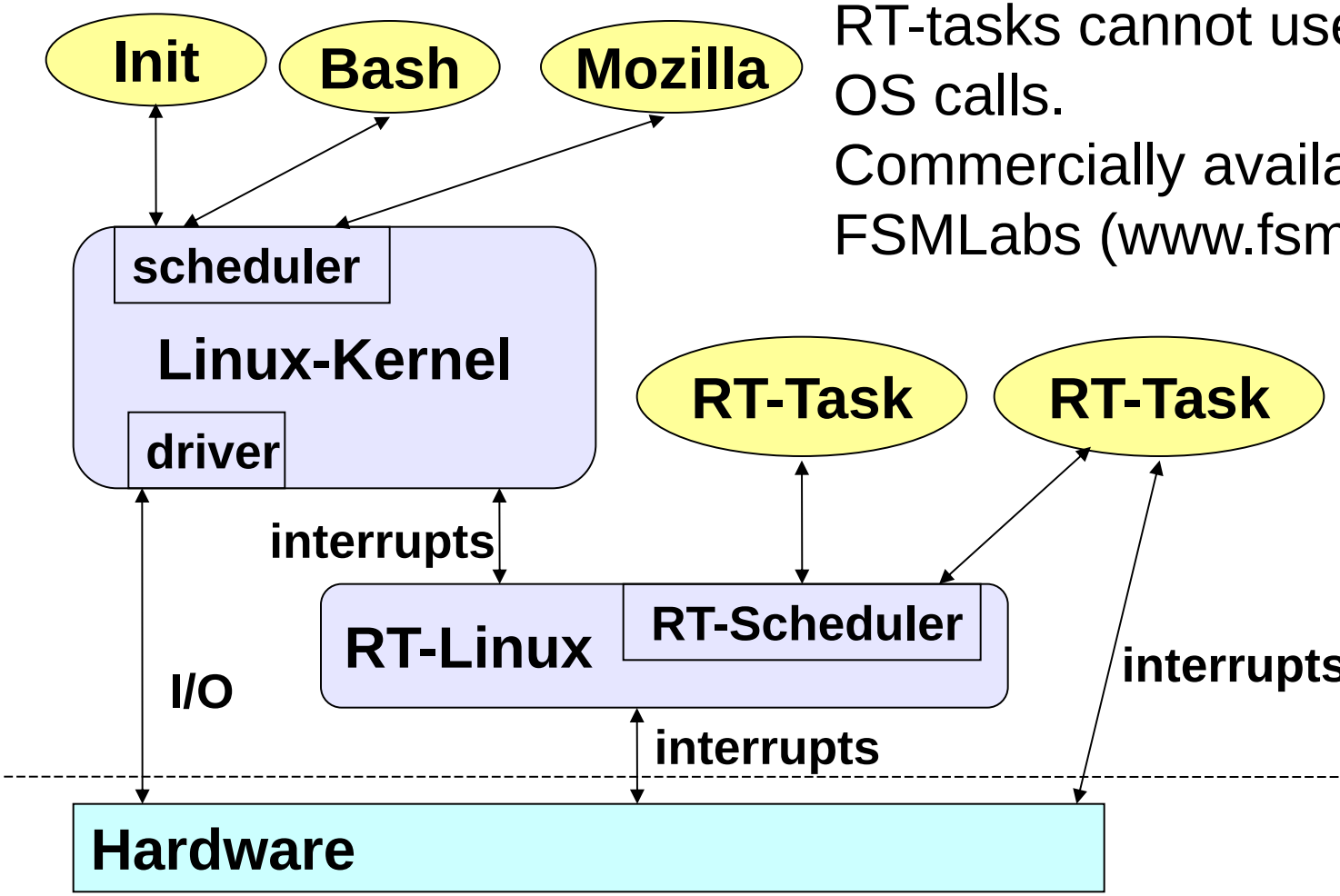
Classes of RTOSs:
2. RT extensions to standard OSs

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



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

Example: RT-Linux



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

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