



**Sharif University of Technology**  
**Department of Computer Science and Engineering**

**Lec. 4.2:**  
**Embedded System Software**



M. Ansari

Spring 2025

According to Peter Marwedel's Lectures

# Increasing design complexity + Stringent time-to-market requirements ➡ Reuse of components

---

Reuse requires knowledge from previous designs to be made available in the form of **intellectual property** (IP, for **SW & HW**).



- HW
- ➡ ■ Operating systems
- Middleware (Communication, data bases, ...)
- ....

# Embedded operating systems

## - Characteristics: Configurability -

---

### Configurability

No overhead for unused functions tolerated,  
no single OS fits all needs, ☞ configurability needed.

- Object-orientation could lead to a of derivation subclasses.
- Aspect-oriented programming
- Conditional compilation (using `#if` and `#ifdef` commands).
- Advanced compile-time evaluation useful.
- Linker-time optimization (removal of unused functions)

Dynamic data might be replaced by static data.



# 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 operating systems

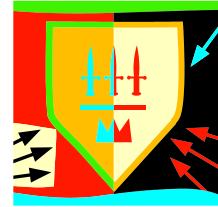
## - Characteristics: Protection is optional-

---

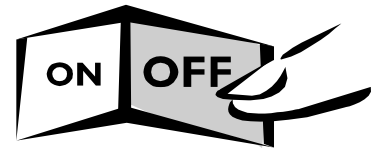
### Protection mechanisms not always necessary:

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.



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

For standard OS: serious source of unreliability.

Since

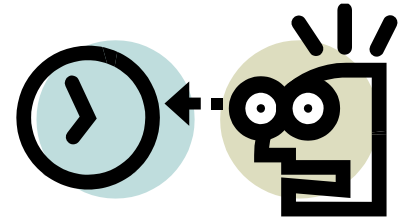
- embedded programs can be considered to be tested,
- since protection is not always necessary and
- since efficient control over a variety of devices is required,
- it is possible to let interrupts directly start or stop SW (by storing the start address in the interrupt table).
- More efficient than going through OS services.
- Reduced composability: if SW is connected to an interrupt, it may be difficult to add more SW which also needs to be started by an event.

# 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)**.



# RT operating systems - Definition and requirement 1: predictability -

---

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

The following are the three 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:

- short times during which interrupts are disabled,
- (for hard disks:) contiguous files to avoid unpredictable head movements.

[Takada, 2001]



# Real-time operating systems 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).
- Frequently, the OS should provide precise time services with high resolution.

[Takada, 2001]

# Real-time operating systems requirement 3: Speed

---

## 3. The OS must be fast

Practically important.



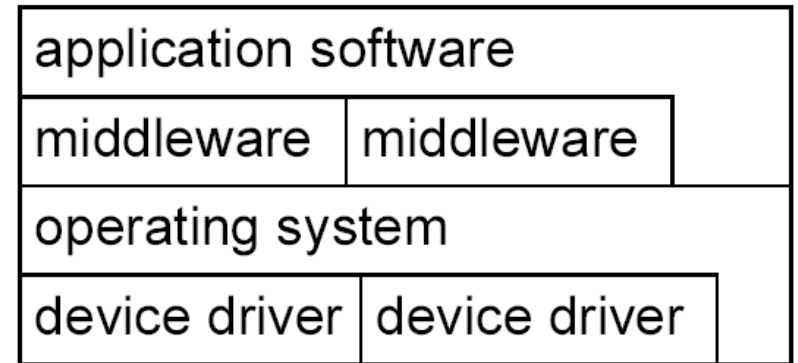
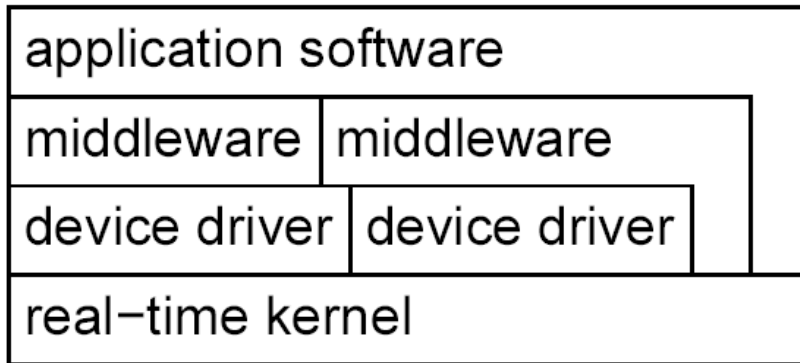
[Takada, 2001]

# RTOS-Kernels

---

## Distinction between

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



## Distinction between

- general RTOSs and RTOSs for specific domains,
- standard APIs (e.g. POSIX RT-Extension of Unix, ITRON, OSEK) or proprietary APIs.

# 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 RTOSes:

## 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, VxWORKS, VxWORKS.

# Classes of RTOSs:

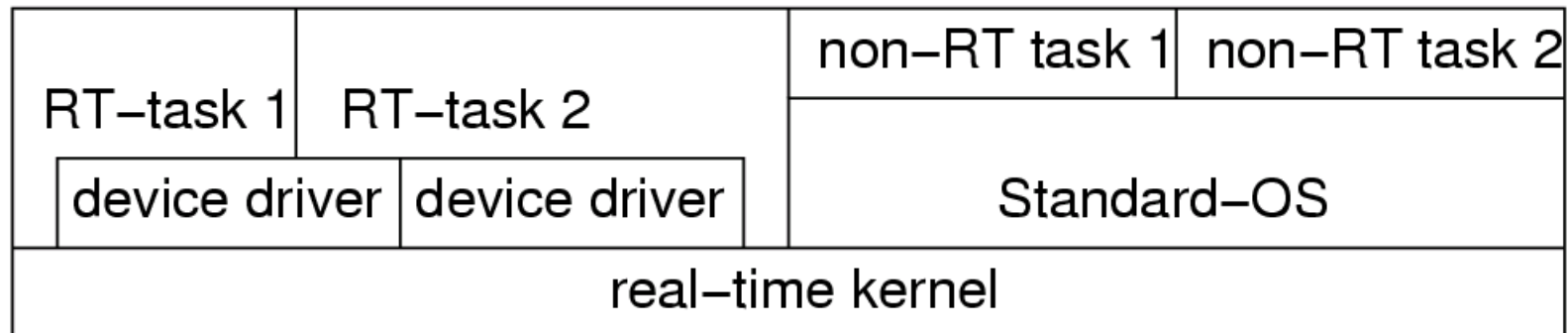
## 2. RT extensions to standard OSs

---

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

# Evaluation

---

According to Gupta, trying to use a version of a standard OS:

*not the correct approach because too many basic and inappropriate underlying assumptions still exist such as **optimizing for the average case** (rather than the worst case), ... **ignoring most if not all semantic information**, and **independent CPU scheduling and resource allocation**.*

Dependences between tasks not frequent for most applications of std. OSs & therefore frequently ignored.

Situation different for ES since dependences between tasks are quite common.

# 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
- support for continuous media
- quality of service (QoS) control.



# Summary

---

- General requirements for embedded operating systems
  - Configurability
  - Disk and network handled by tasks.
  - Protection is optional.
  - Interrupts not restricted to OS
  - Real-time Capability
- General properties of real-time operating systems
  - Predictability
  - Managing time
  - Speed