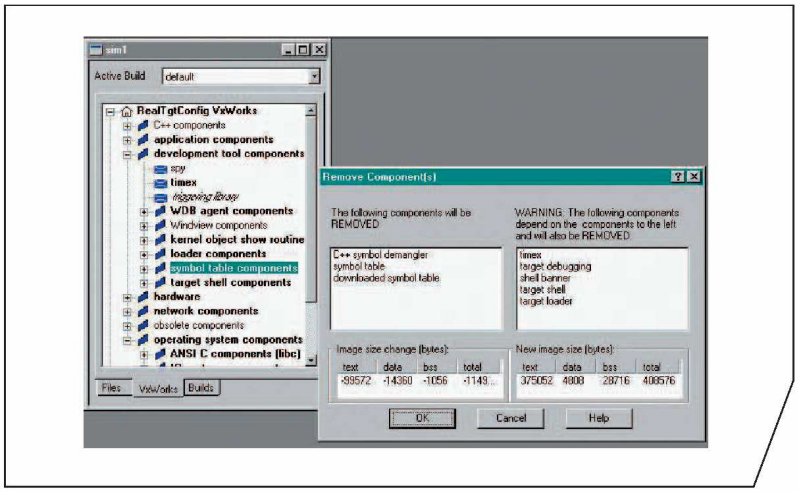
Embedded operating systems
- Requirement: Configurability -

- **Configurability**
  No single RTOS will fit all needs, no overhead for unused functions tolerated ☞ configurability needed.
  - simplest form: remove unused functions (by linker ?).
  - Conditional compilation (using #if and #ifdef commands).
  - Dynamic data might be replaced by static data.
  - Advanced compile-time evaluation useful.
  - Object-orientation could lead to a derivation subclasses.

  Verification a potential problem of systems with a large number of derived OSs:
  Each derived OS must be tested thoroughly;
  potential problem for eCos (open source RTOS from Red Hat), including 100 to 200 configuration points [Takada, 01].

Example: Configuration of VxWorks



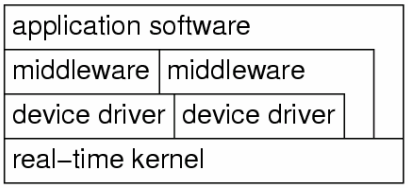
Automatic dependency analysis and size calculations allow users to quickly custom-tailor the VxWORKS operating system.

© Windriver

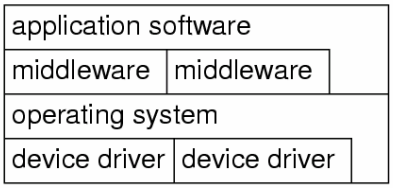
Embedded operating systems
-Requirement: Disc and network handled by tasks-

- **Disc and network handled by tasks instead of integrated drivers**
  Many ES without disc, a keyboard, a screen or a mouse.
  Effectively no device that needs to be supported by all versions of the OS, except maybe the system timer.
  Relatively **slow** discs & networks can be handled by tasks.

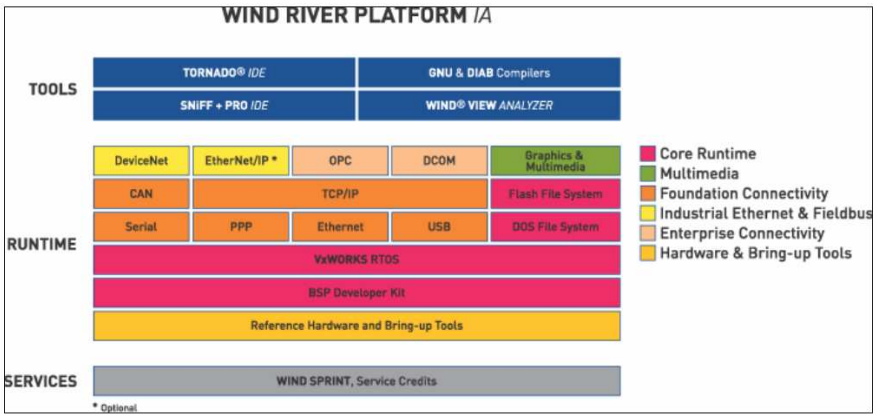
RTOS



Standard OS



Example: WindRiver Platform Industrial Automation



© Windriver

Embedded operating systems
- Requirement: Protection is optional-

- **Protection mechanisms not always necessary:**
  ES typically designed for a single purpose,
  untested programs rarely loaded, SW considered reliable.
  (However, protection mechanisms may be needed for safety and security reasons).

No desire to implement I/O instructions as privileged instructions and tasks can be allowed to do their own I/O.

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

```
load register,switch
```

instead of OS call.

Embedded operating systems
- Requirement: 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 necessary and
  - –since efficient control over a variety of devices is required,

  it is possible to let interrupts directly start or stop tasks (by storing the tasks start address in the interrupt table).
  More efficient than going through OS services.
  However, composability suffers: if a specific task is connected to some interrupt, it may be difficult to add another task which also needs to be started by an event.

Embedded operating systems
- Requirement: Real-time capability-

- Many embedded systems are real-time (RT) systems and, hence, the OS used in these systems must be **real-time operating systems (RTOSes).**

Real-time operating systems
- Real-time OS (1) -

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 deterministic:
   - unlike standard Java,
   - short times during which interrupts are disabled,
   - contiguous files to avoid unpredictable head movements.

[Takada, 2001]

Real-time operating systems
- Real-time OS (2) -

2. OS must manage the timing and scheduling

- OS possibly has to be aware of task deadlines; (unless scheduling is done off-line).
- OS must provide precise time services with high resolution.

[Takada, 2001]

Real-time operating systems
- Real-time OS (3) -

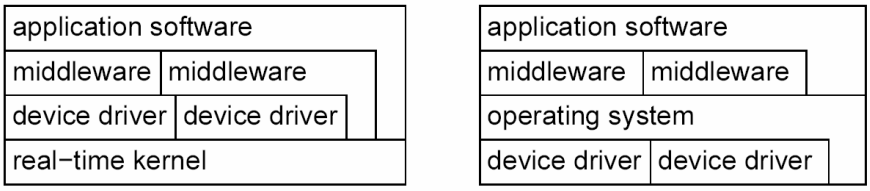
3. The OS must be fast

Practically important.

[Takada, 2001]

RTOS-Kernels

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



Distinction between
- general RTOSes and RTOSes 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 according to R. Gupta
1. Fast proprietary kernels

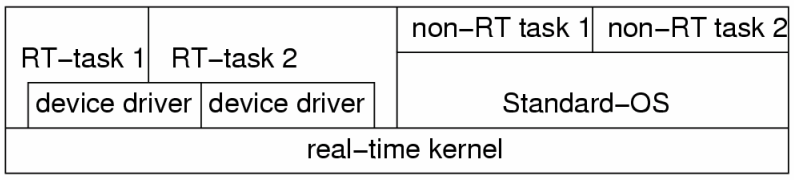
- **Fast proprietary kernels**
  For complex systems, these kernels are inadequate, because they are designed to be fast, rather than to be predictable in every respect

  [R. Gupta, UCI/UCSD]

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

Classes of RTOSes according to R. Gupta
2. Real-time extensions to standard OSs

- **Real-time extensions to standard OSes:**
  Attempt to exploit comfortable main stream OSes.
  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

Classes of RTOSes according to R. Gupta
3. Research systems trying to avoid limitations

Market

- **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.

Competition between

- traditional vendors (e.g. Wind River Systems) and
- Embedded Windows XP and Windows CE

Middleware

1. Real-time data bases
2. Access to remote objects

Real-time data bases (1)

Goal: store and retrieve persistent information

Transaction= sequence of read and write operations

Changes not final until they are committed

Requested ("ACID") properties of transactions

1. **Atomic:** state information as if transaction is either completed or had no effect at all.
2. **Consistent:** Set of values retrieved from several accesses to the data base must be possible in the world modeled.
3. **Isolation:** No user should see intermediate states of transactions
4. **Durability:** results of transactions should be persistent.

Real-time data bases (2)

Problems with implementing real-time data bases:

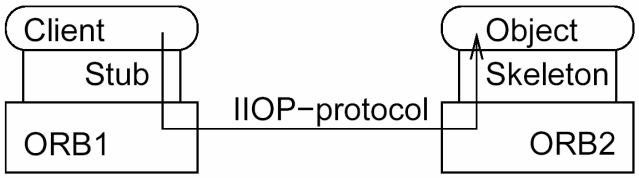
1. transactions may be aborted various times before they are finally committed.
2. For hard discs, the access times to discs are hardly predictable.

Possible solutions:

1. Main memory data bases
2. Relax ACID requirements

Access to remote objects

Software packages for access to remote objects;
Example:
CORBA (Common Object Request Broker Architecture).
Information sent to Object Request Broker (ORB) via local stub.
ORB determines location to be accessed and sends information via the IIOP I/O protocol.



Access times not predictable.

Real-time (RT-) CORBA

A very essential feature of RT-CORBA is to provide

- *end-to-end predictability of timeliness in a fixed priority system*.
- This involves *respecting thread priorities between client and server for resolving resource contention*,
- and bounding the latencies of operation invocations.
- Thread priorities might not be respected when threads obtain mutually exclusive access to resources (priority inversion).
- RT-CORBA includes provisions for bounding the time during which such priority inversion can happen.