

## 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

The screenshot shows the 'RealTimeConfig VxWorks' application. On the left is a tree view of components including C++ components, application components, development tool components, RW, times, WDB agent components, Winview components, kernel object show routine, loader components, symbol table components, target shell components, hardware, network components, obsolete components, operating system components, and ANSI C components (libc). A 'Remove Components' dialog box is open in the foreground, listing components to be removed: C++ symbol demangler, symbol table, downloaded symbol table, times, target debugging, shell banner, target shell, and target loader. Below the lists are two tables showing image size changes in bytes.

Image size change (bytes):				New image size (bytes):			
text	data	bss	total	text	data	bss	total
-89572	-14360	-1056	-1149...	375052	4900	28716	408576

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-

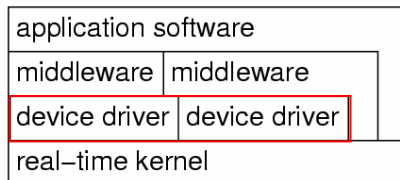
#### Disc & network handled by tasks instead of integrated drivers

Many ES without disc, a keyboard, a screen or a mouse.

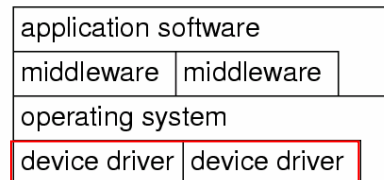
**Effectively no device that needs to be supported by all versions of the OS**, except maybe the system timer.

Relatively **slow** discs & networks can be handled by tasks.

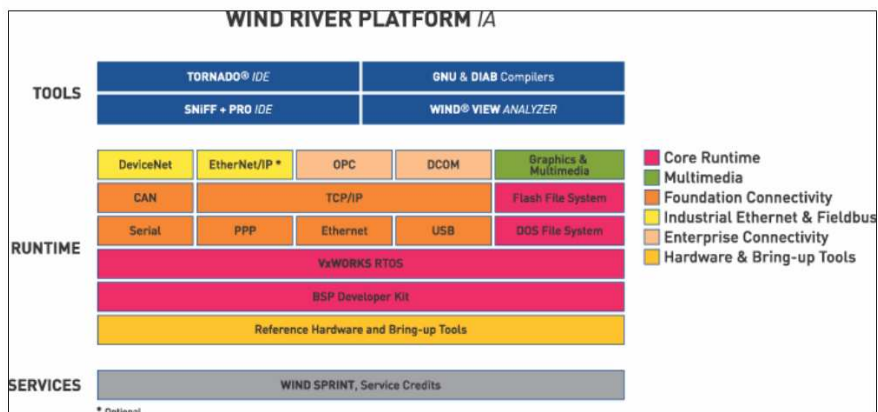
#### RTOS



#### Standard OS



## Example: WindRiver Platform Industrial Automation



## 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.

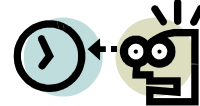
Since

- 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]



## Time services

Time plays a central role in “real-time” systems.

Actual time is described by real numbers.

Two discrete standards are used in real-time equipment:

- **International atomic time TAI**  
(french: *temps atomique internationale*)  
Free of any artificial artifacts.



- **Universal Time Coordinated (UTC)**  
UTC is defined by astronomical standards

UTC and TAI were identical on Jan. 1st, 1958.

In the meantime, 30 seconds had to be added.

Not without problems: New Year may start twice per night.



## Internal synchronization

Synchronization with one master clock

- Typically used in startup-phases

Distributed synchronization:

1. Collect information from neighbors
2. Compute correction value
3. Set correction value.



Precision of step 1 depends on how information is collected:

Application level: ~500  $\mu$ s to 5 ms

Operation system kernel: 10  $\mu$ s to 100  $\mu$ s

Communication hardware: < 10  $\mu$ s



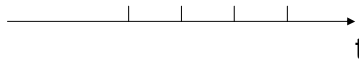
## Byzantine Error

Erroneous local clocks can have an impact on the computed local time.

Advanced algorithms are fault-tolerant with respect to Byzantine errors. Excluding  $k$  erroneous clocks is possible with  $3k+1$  clocks (largest and smallest values will be excluded).

Many publications in this area.

$k=1$



## External synchronization

External synchronization guarantees consistency with actual physical time.

Recent trend is to use GPS for ext. synchronization

GPS offers TAI and UTC time information.

Resolution is about 100 ns.



## Problems with external synchronization

Problematic from the perspective of fault tolerance:

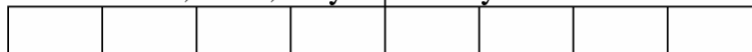
Erroneous values are copied to all stations.

Consequence: Accepting only small changes to local time.

Many time formats too restricted;

e.g.: NTP protocol includes only years up to 2036

Full seconds, UTC, 4 bytes    Binary fraction of second, 4 bytes



Range up the years 2036; 136 year wrap around cycle

For time services and global synchronization of clocks  
synchronization see Kopetz, 1997.



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

### 3. The OS must be fast

Practically important.

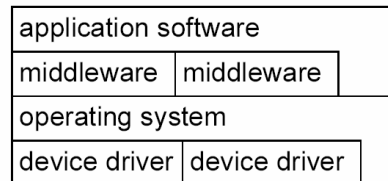
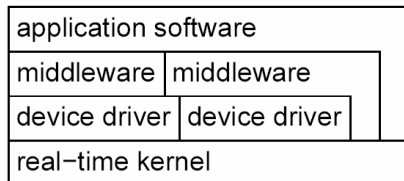
[Takada, 2001]



## 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, VxWORKS, VTRX32, VxWORKS.

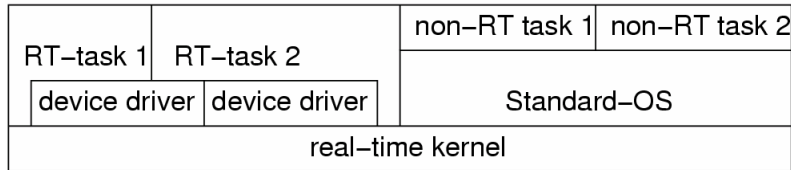


## Classes of RTOSes according to R. Gupta

### 2. Real-time extensions to standard OSs

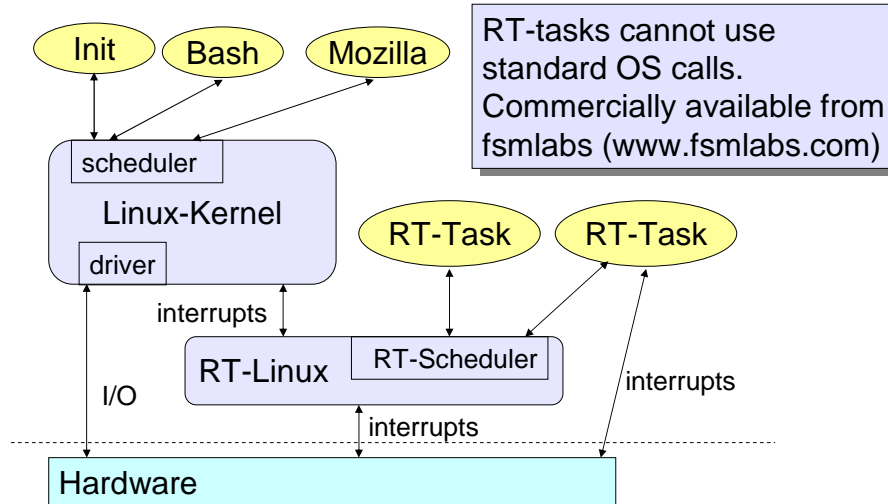
#### Real-time extensions to standard OSes:

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



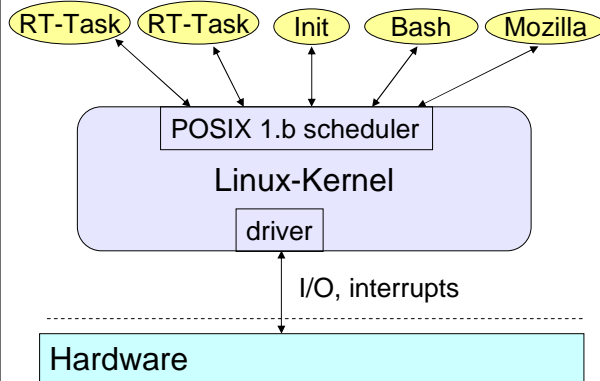
- + Crash of standard-OS does not affect RT-tasks;
- RT-tasks cannot use Standard-OS services;  
less comfortable than expected

## Example: RT-Linux



## Example: Posix 1.b RT-extensions to Linux

Standard scheduler can be replaced by POSIX scheduler implementing priorities for RT tasks



Special RT-calls and standard OS calls available.  
Easy programming, no guarantee for meeting deadline



## Evaluation (Gupta)

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**.*

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

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



## 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

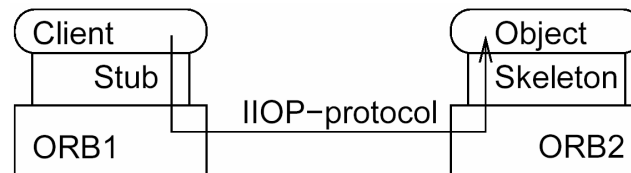
Software packages for access to remote objects;

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.



## Real-time COBRA - Thread priority management -

- RT-CORBA includes facilities for thread priority management.
- Priority independent of the priorities of the underlying OS, even though it is compatible with the RT-extensions of the POSIX standard for OSs [Harbour, 1993].
- The thread priority of clients can be propagated to the server side.
- Priority management for primitives for mutually exclusive access to resources. Priority inheritance protocol must be available in implementations of RT-CORBA.
- Pools of preexisting threads avoid the overhead of thread creation and thread-construction.



## Message passing interface (MPI)

- Message passing interface (MPI): alternative to CORBA
- MPI/RT: a real-time version of MPI [MPI/RT forum, 2001].
- MPI-RT does not cover issues such as thread creation and termination.
- MPI/RT is conceived as a potential layer between the operating system and standard (non real-time) MPI.



## Summary

---

- General requirements for embedded operating systems
  - Configurability, I/O, interrupts
- General properties of real-time operating systems
  - Predictability
  - Time services, synchronization
  - Classes of RTOSs, device driver embedding
- Middleware (briefly)
  - RT-data bases
  - Access to remote objects (RT-CORBA, RT-MPI)

