Real-Time & Embedded Operating Systems

VO Embedded Systems Engineering

Benedikt Huber

WS 2010/11

Overview

- Real-Time Systems (Review)
- OS and RTOS
- RTOS Classification
- Linux as RTOS
- Programming Considerations
- Summary

Computer system classification

Transformational systems compute output values from input values, then stop.

- *numerical computations, compiler*

Interactive systems constantly interact with their environment. The system delivers a service to the user.

- *operating systems, databases*

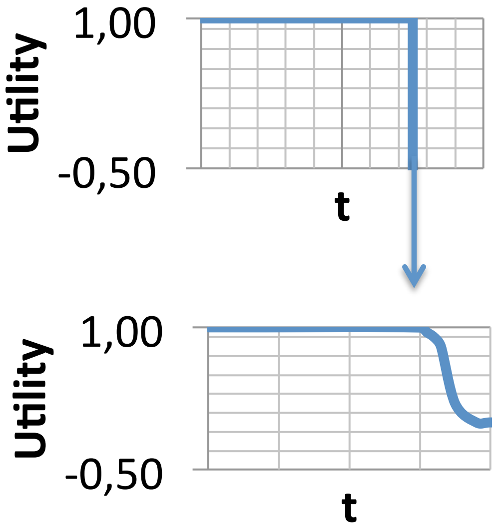
Reactive systems continuously react to stimuli from the environment. Reaction time is dictated by the environment.

- *signal processors, process controllers*

Real-Time Systems

In a real-time computer system **the correctness of the system behavior depends** not only on the logical results of the computations, but also **on the physical instant** at which these results are produced.

- **Hard Real-Time**
  deadline miss can have catastrophic results
- **Soft Real-Time**
  result has utility after the deadline



Schedulability Analysis

In a hard real-time system, we need to a high confidence that no deadline will be missed during operation

- Offline Scheduling
  - Timing analysis is much easier without preemption
- Offline Scheduling
  - Static Priority Scheduling (e.g., RMS)
  - Dynamic Priority Scheduling (e.g., EDF)
- For both: Need to know timing characteristics of runtime system (OS) and WCET of all tasks

Determinism & Predictability

A system behaves deterministic if the same sequence of input always produces the same sequence of output

- Key difference between reactive and interactive systems

→ Compare e.g. a process controller and a compiler

- Predictability
  - Is our (timing) model a precise characterisation of the system?
  - Can we predict e.g. context switch delay, execution times?
  - Temporal behavior of many non-RT systems in unpredictable

Embedded Software market grows

- Compare the **Average Annual Growth Rate** (AAGR) of the market for **embedded software (16%)** with the estimated growth rates of GDP (about 2%) shows the crucial relevance of the Embedded Systems.
- **A strong increase in the value of Embedded Systems is expected** Examples are telecommunications, logistics, automation, or automotive.
- **Further “softwareization” is predicted:** estimated AAGR 2004 - 2009
  - 16% for embedded software,
  - 14.2% for embedded hardware (integrated circuits, IC), and
  - 10% for embedded boards
- **Increased number of lines of code per functionality** (in aircraft systems from 10 to 105 between 1970 and 2007)

Market data for 2004 on tools and services

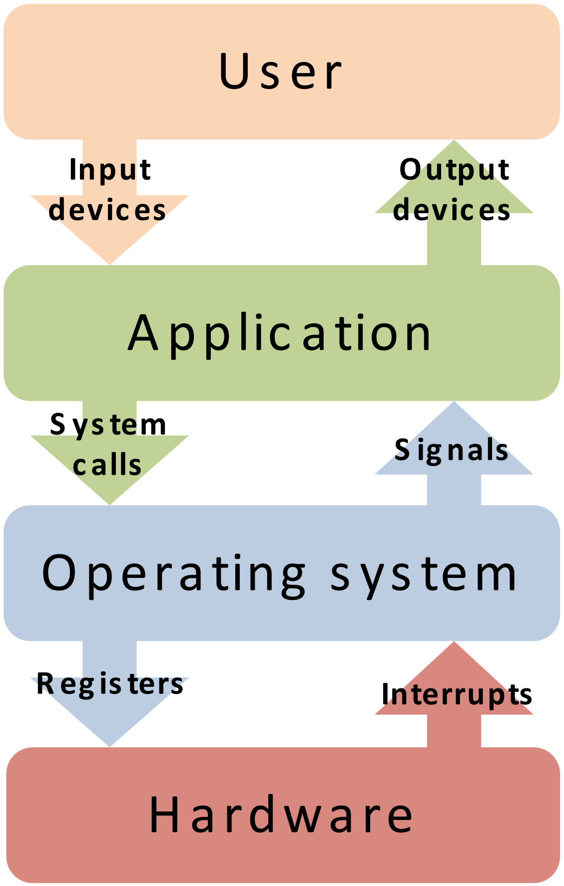
Characteristics Market (2004)	Market volume in million €	Market growth	Largest consuming region and its share	Largest consuming sector and its share
Embedded OSs, bundled tools, related services	712.6	20.9%	Americas 52.7%	Consumer Electronics 41.7%
SW development tools, related services	195.1	1.8%	Americas 48.2%	n.a.
Design automation tools, related services	275.6	n.a.	Americas n.a.	Military / Aerospace, n.a.
Test automation tools, related services	65.7	19.8%	Americas 50.8%	Military / Aerospace, 27.6%
	1,249.0	17.0% (weighted average)		

OS and RTOS

Operating Systems

Operating System (OS):

- abstracts from hardware
- provides access to I/O,
- memory management,
- sharing of the resources of the computer
- provides system calls to access low level functions



Embedded system is a hardware/software artifact.

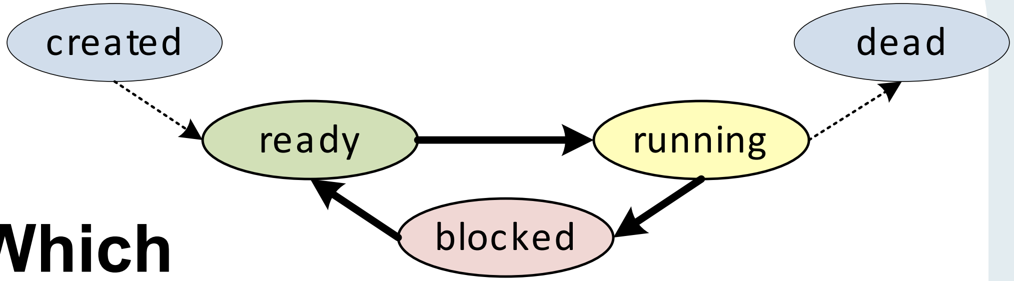
Common OS Services

- Task management
- Scheduling and Timers
- Memory Management
- Memory Protection and Error Handling
- Hardware Abstraction and I/O interface
- Inter Process Communication
  - Synchronization (mutual exclusion)
  - Message passing
  - Shared Memory

Real-Time Operating Systems (RTOS)

- Real-Time (RT) requirements for OS + features for timing constraints
  - guaranteed max. execution time of system calls
  - guaranteed OS response time to external events
  - guaranteed max. execution time of OS functions (ISRs, drivers, context switches, ...)
  - **Determinism and Predictability**
- Efficiency
  - Fast context switch
  - Minimize intervals during which the interrupts are disabled

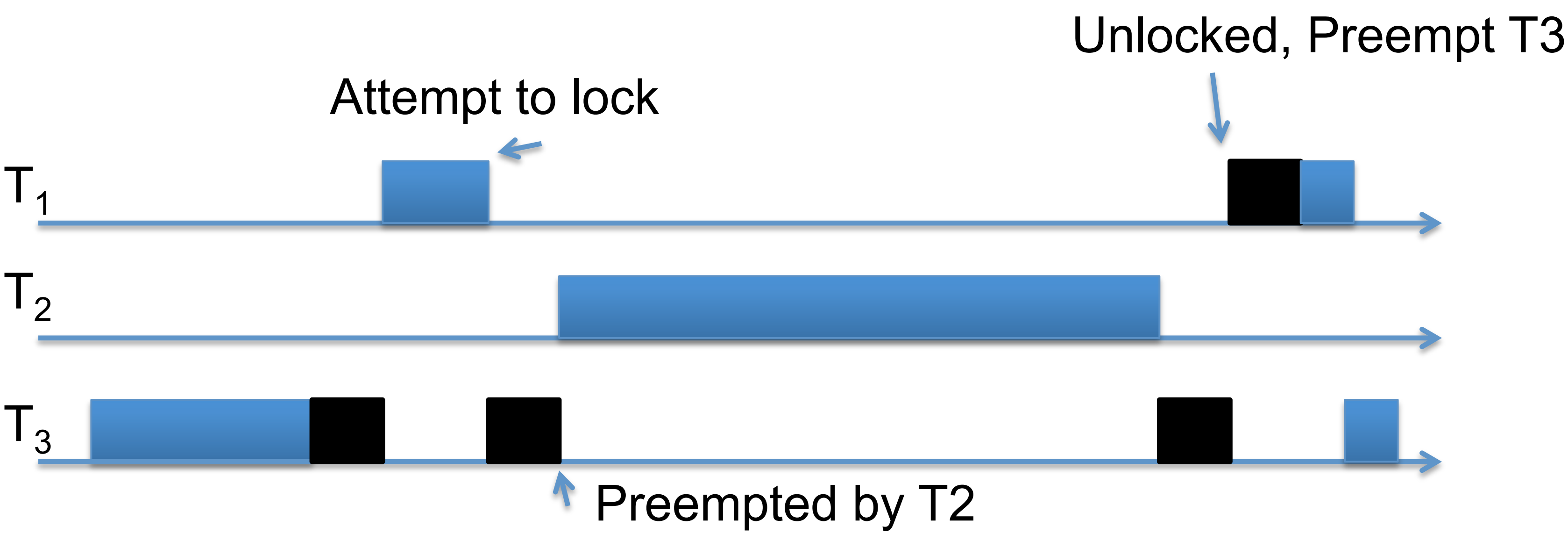
Scheduling



- Scheduling Decision: **Which task in ready state can execute?**
- Scheduler uses a scheduling strategy
  - fixed priorities, dynamic priorities, round-robin (time-slicing), rate monotonic, cooperative
- How does scheduler get CPU?
  - called by application
  - system timer interrupt

Mutual Exclusion

- Realized using e.g. semaphores
  - Increases response times
- For Real-Time Scheduling, protocol to avoid priority inversion due to locking is needed



Is Scheduler Necessary?

No, you can meet deadlines without any RTOS (generate offline schedule, implement it):

- inefficient (regularly poll infrequent events, reserve max. time for interrupts)
- long schedule cycle time (least common multiple of all task periods) or unnecessary overhead (shorten task periods)
- hard to change and maintain (use tools!)

```
T1: C1=1, P1=3
T2: C2=1, P2=6
T3: C3=2, P3=6
for(;;) {
  T1();
  T2();
  wait(1);
  T1();
  T3(); }
```

RTOS taxonomy and architecture

RTOS taxonomy and architecture (1)

Small, fast, proprietary kernels

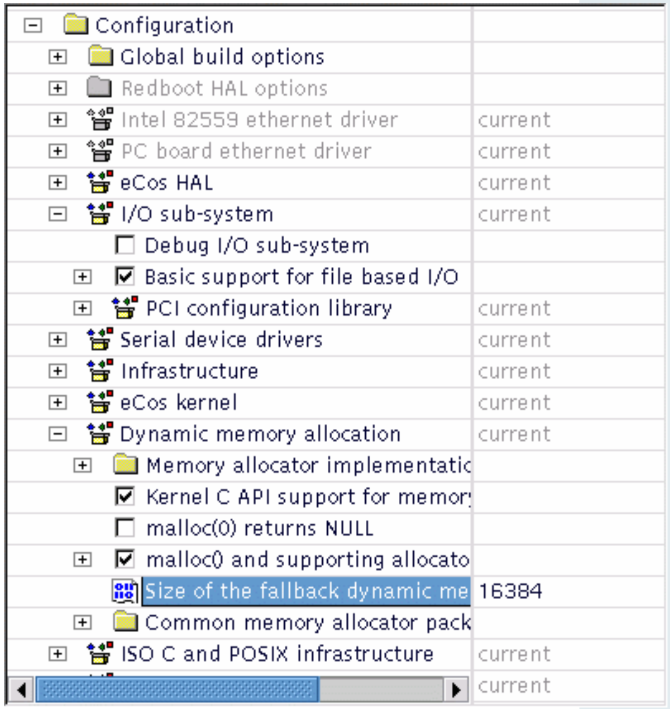
- Highly specialized to specific applications

Real-Time extensions to commercial operating systems

(RT-Linux, RT-Posix, RT-MACH, RT-WinNT)

- Compliant kernels
  Existing OS is modified such that non-rtos binaries run without modification
- Dual kernels
  Thin RT-kernel stay below the native OS
- Core kernel modification
  Changes are made in the core of OS
- The resource kernel approach
  Kernel is extended to provide support for resource reservation

RTOS taxonomy and architecture (2)



Component based kernels

- OS components can be selectively included to compose an RTOS
- Selection depends on the target and application
- Construction of OS through composition
  - eCos
  - PURE - embedded applications
  - MMLite - dynamic reconfiguration of components

RTOS taxonomy and architecture (3)

QoS based kernels

- Used for soft real-time systems

Research kernels

- Developed at university research projects to study research aspects of RTOS
- Examples: MARS (time triggered, distributed), SPRING (admission control, reservation), HARTOS (distributed communication)

Partioning RTOS

- Hard real-time / mixed criticality systems
- Spatial Isolation: Memory protection
- Temporal Isolation: Hierarchical Scheduling (Pike OS)
  - Static scheduling of partitions
  - Dynamic scheduling of tasks within partition
  - Background partition

Microkernels

- Kernel restricted to necessary minimum
  - IPC, Resource Allocation, but not scheduling, I/O …
  - Improves maintainability, predictability, security
- Hypervisor - Mikrokernel
  - Isolates hard real-time tasks
  - Virtual Environment for other paravirtualized RTOSs
- Example: OKL4
  - Used in e.g. Mobile Phones for the Android Platform

Designing an RTOS application

Design Decisions

- How many tasks?
- Poll events or use interrupts?
- If interrupt, handle event in ISR or in a task?

Number of Tasks

Few tasks:

- small OS overhead (not much scheduling)
- not much inter-task communication
- easy to understand complete system

Many tasks:

- good and consequent functional partitioning
- modularization – easy to maintain/change modules
- high parallelism

Programming Considerations

- Keep program short
- Keep memory usage down
- Be careful not to overwrite system memory (buffer overflow)
- Consider power consumption
  - use CPU as seldom as possible (no busy wait, no idle loop!)
  - on a distributed system, try to minimize consumption
    (but take care not to completely drain one node...)

Event Handling: Polling vs. Interrupts

Polling:

- easier to understand
- regular, hence easier to estimate worst case times
- under the control of the scheduler

Interrupt:

- only called when event occurs
- always called as soon as possible
- good performance in average case (resp. interrupt rate)

Event Handling: in ISR or in Task?

In ISR:

+ fastest possible reaction
– long ISR
– not all system calls allowed in ISR (e.g., I/O access)

In Task:

+ considers execution time needs of other tasks
– handling time harder to compute

Selecting an OS

- Supported processors?
- Memory requirements (OS + appl <= Target memory; don't forget RAM requirements)
- Features (scheduling strategies, IPC mechanisms, ...)
- Execution time (if real-time requirements)
- Support ! (hotline, documentation, sources?, ...)
- Check newsgroups & magazines for reports

Advantages of having a RTOS

- RTOS provide an abstraction (process model)
- Structuring in tasks improves
  - Modularity
  - Maintainability
  - Reusability
- Facilitates temporal and spatial partitioning of application
- Enables power management strategies
- Reduces effort for certification

Linux as RTOS

Linux as real-time operating system (1)

- **Predictability of real-time task execution**
  - Pre-2.6 scheduling algorithm was O(n), so it could delay a even a high priority task‘s start time.
    (Since 2.6 O(1) scheduler introduced: time to schedule is both fixed and deterministic regardless of the number of active tasks)
  - Kernel cannot always be preempted. Sometimes it requires exclusive access to resources and internal data structures in order to maintain their consistency
  - Interrupts from hardware may delay a task

Linux as real-time operating system (2)

- **Virtual memory management** introduces indeterministic delays (jitter). Real-Time applications may not use the virtual memory
- **Clock granularity** of 1ms insufficient for most real-time apps (e.g., a typical control loop task executes with 100 Hz = every 10ms)

RT-POSIX

- Real-Time Extensions to Portable Operating System Interface (POSIX)
- Real-Time Scheduling: SCHED_FIFO, SCHED_RR
- Virtual Memory Locking
- High-Precision Timers
- Real-Time Threads (Globally or Locally Scheduled)
- Mutexes with Protocols avoiding Priority Inversion
- Profiles to support small embedded devices

The Real-Time Preemption Patch

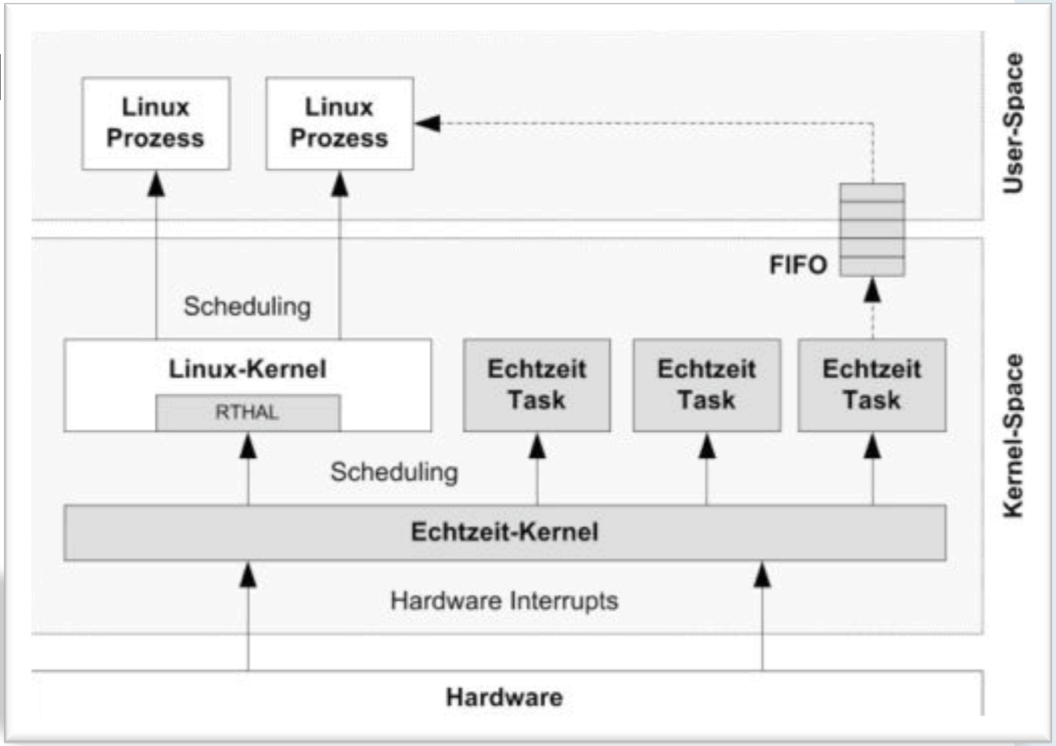
- **Makes Linux better suited to real-time systems**
- **Real-Time Scheduling** In the RTPreempt, RT-Posix fixed priority scheduling. There is also an EDF patch for Linux.
- **Critical sections in kernel are preemptable** They now use real-time mutexes, including a priority inheritance protocol
- **Interrupt handlers are interruptable kernel threads**
- **High-precision timer support**
- Not suitable for safety-critical hard real-time

[SCHED_DEADLINE, http://www.evidence.eu.com/content/view/313/390/]

Linux as real-time operating system (3)

Solutions to run hard-real time tasks and Linux:

- Real Time Linux (**RTLinux**)
- Real-Time Application Interface (**RTAI**)
- **Xenomai**



RTLinux & RTAI

- **Dual-kernel approach**, two schedulers are in operation
- Supports hard real-time (deterministic) operation through **interrupt control** between the hardware and the operating system. (Adeos Event Pipeline)
- **Linux runs as its lowest-priority process**
- **Real-time applications specifically written** for the non-Linux kernel using an associated real-time API
- Can **exchange data with Linux** applications (LXRT)

RTLinux & RTAI

- **Tasks are executed inside kernel memory space**, which prevents threads to be swapped-out and also the number of TLB misses is reduced.
- **Threads are executed in processor supervisor mode** (i.e. ring level 0 in i386 arch), have full access to the underlying hardware.
- Since the RTOS and the application are linked together in a "single" execution space, **system call mechanism is implemented by means of a simple function call** (default is software interrupt which produces higher overhead).

Xenomai

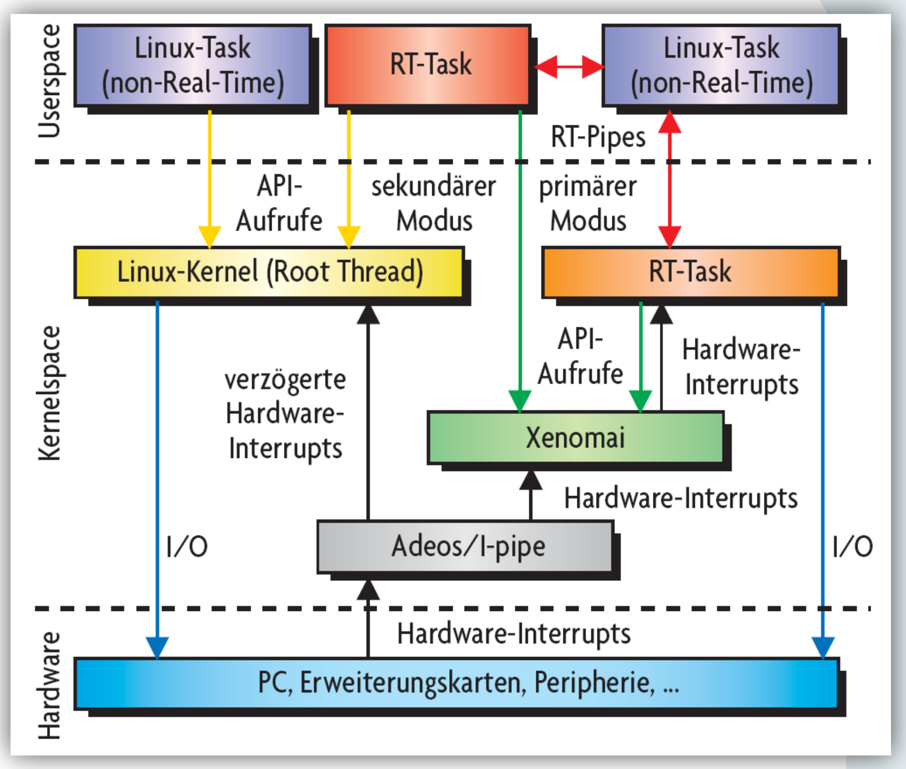
RTAI spin-off, designed for

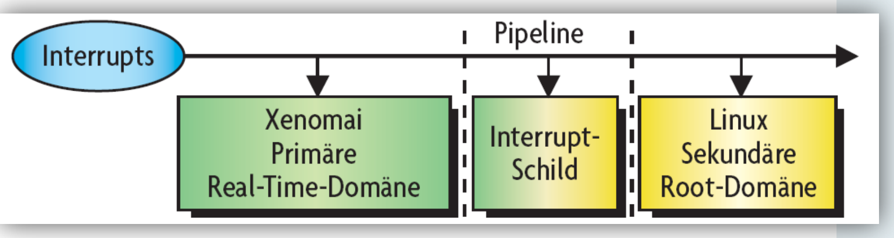
- extensibility
- portability
- maintainability

Task can switch transparently

- Primary mode (hard RT)
- Secondary mode (soft RT)

Skins can emulate RTOS APIs





ENDE

Danke für die Aufmerksamkeit!