Real-Time Operating Systems Design and Implementation

(slides are based on Prof. Dr. Jian-Jia Chen)

Anas Toma

LS 12, TU Dortmund

October 11, 2018

Outline

Organization

- Instructor: Anas Toma, anas.toma@tu-dortmund.de
  - Office hours: Monday, 12:00-13:00. Please make an appointment by email
- Grading:
  - Oral exam
  - Achievement at least 50% of the total exercises points
- References
  - Textbooks:
    - Richard Barry, "Using the FreeRTOS Real Time Kernel - a Practical Guide," Real Time Engineers Ltd, 2011. ISBN: 978-1-4467-6274-5
    - Giorgio C. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer, Third Edition, 2011. ISBN: 978-1-4614-0675-4

Course Description

Time: Thursday, 10:15-11:45 and 12:15-13:45
(or a block session from 10:15 to 13:15)

Place: Room U08 (CILAB), OH16

Start: on 11.10.2018

- **Prerequisite**: Embedded systems or equivalent.
  - Desirable skills: Basic knowledge of Operating Systems and C Programming
- **Please register in the exercise**
  - <u>Maximum 18 students!</u>

Course Calendar and Topics

- Topics:
  - Organization and introduction
  - Real-time operating systems
  - Real-time tasks management
  - Drivers and libraries
  - Queues
  - Interrupts
  - Resource sharing
  - Kernel internals
  - Resource reservation servers
  - Schedulers
- Course calender: `https://ls12-www.cs.tu-dortmund.de/daes/lehre/courses/wintersemester-2018/real-time-operating-systems-design-and-implementation-html?format=html&lang=en`

Outline

Organization

Introduction to Real-Time Systems

FreeRTOS

Computing Systems

Desktops/Servers/Clusters

- Mainframe computing (60's-70's)
- Desktop computing and the Internet (80's-90's)
- Millions of units per year for purchase
- High performance computing
- Optimize for average response time

Embedded Systems

- ubiquitous/physical computing (00's-?)
- Billions of units per year for purchase
- Timing property is an important requirement to be satisfied.

Embedded Systems

Complex "best effort" systems

- Mobile telecommunications
- Consumer products, e.g., digital camera, digital video, etc.
- Good reactivity and good dependability

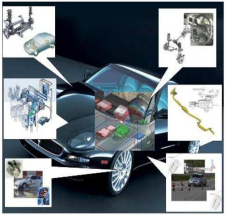





Critical control systems

- Automated aircraft landing systems
- Automotive control for gearing, ABS, airbag, etc.
- High reactivity and high dependability






Real-Time Systems

- Dual notations of correctness:
  - Logical correctness ("the results are correct")
    - Requires functional analysis
  - Temporal correctness ("the results are delivered in/on time")
    - Requires non-functional analysis
- High reactivity and high dependability are more important than performance

Examples for Real-Time Systems

- Chemical & Nuclear Power Plants
- Railway Switching Systems
- Flight Control Systems
- Space Mission Control
- Automotive Systems
- Robotics
- Telecommunications Systems
- Stock Market, Trading System,
- Information Access
- Multimedia Systems
- Virtual Reality
- ......

Hard Real-Time Systems

Catastrophic if some deadlines are missed

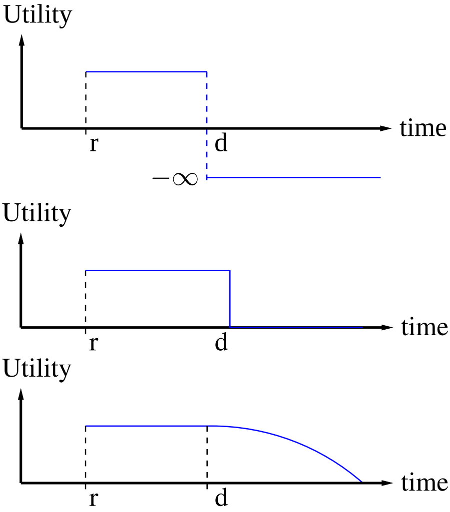
Firmed Real-Time Systems

The results are useless if the deadlines are missed

Soft Real-Time Systems

The results are not very useful if the deadlines are missed

Classifications of Real-Time Systems



Hard Real-Time Systems

Catastrophic if some deadlines are missed

Firmed Real-Time Systems

The results are useless if the deadlines are missed

Soft Real-Time Systems

The results are not very useful if the deadlines are missed

Fundamentals

- Algorithm:
  - It is the logical procedure to solve a certain problem
  - It is informally specified a sequence of elementary steps that an "execution machine" must follow to solve the problem
  - It is not necessarily (and usually not) expressed in a formal programming language
- Program:
  - It is the implementation of an algorithm in a programming language
  - It can be executed several times with different inputs
- Process:
  - An instance of a program that given a sequence of inputs produces a set of outputs

Operating System

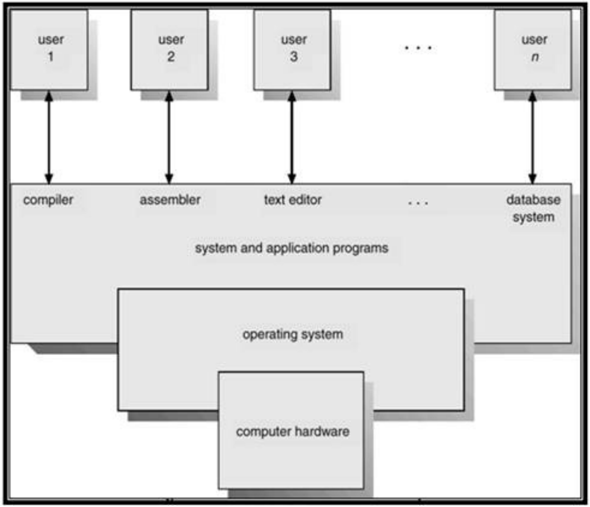
An operating system is a program that

- acts as an intermediary between a user of a computer and the computer hardware by providing interfaces
- provides an “abstraction” of the physical machine (for example, a file, a virtual page in memory, etc.)
- manages the access to the physical resources of a computing machine
- makes the computer system convenient to use
- executes user programs and makes solving user problems easier
- and more . . . . . .

Computer System Components

1. Hardware: provides basic computing resources (CPU, memory, I/O devices)
2. Operating system: controls and coordinates the use of the hardware among the various application programs for the various users
3. Applications programs: define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs)
4. Users (people, machines, other computers)

Abstract View of System Components



Abstraction

- Why abstraction?
  - Programming the HW directly has several drawbacks
    - It is difficult and error-prone
    - It is not portable
  - A simple example to read a text file from a hard disk and display on the screen
    - Without a proper interface, the whole process will need to involve many interactions with the hardware devices
- Application programming interface (API)
  - Provides a convenient and uniform way to access to one service so that
    - HW details are hidden to the high level programmer
    - One application does not depend on the HW
    - The programmer can concentrate on higher level tasks
  - Examples
    - read, write, open system calls in standard unix/linux OS.

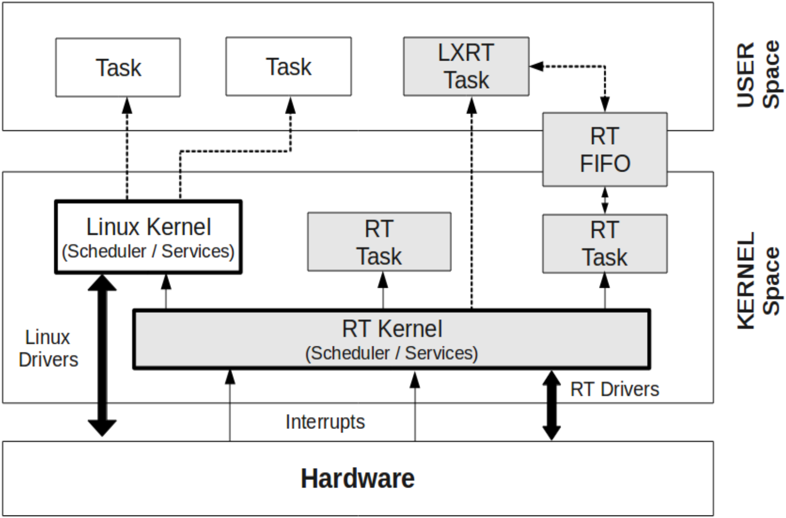
What is A Real-Time OS?

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
- Well-defined fixed-time constraints
  - The system allows access to sensitive resources with defined response times:
    - Maximum response times are required for hard real-time systems
    - Average response times could be okay for soft real-time systems
  - For example, context switch overhead
    - 10 ns
    - 10 μs
    - 10 s
- Taxonomy of RTOSes
  - RT extensions to commercial timesharing systems
  - Research kernels
  - Small, fast, proprietary kernels

RT Extensions

- A common approach is to extend Unix
  - Linux: RT-Linux, RTLinuxPro, RTAI, etc.
  - Posix: RT-POSIX
- Also done for Windows based on virtualization, e.g. RTOSWin
- Advantages
  - Richer environment, more functionality.
  - These systems use familiar interfaces, even standards.
- Disadvantages
  - Generally slower and less predictable.
  - Timers too coarse
  - Memory management has no bounded execution time
  - Intolerable overhead, excessive latency

An Example: RTAI



Research Real-Time Kernels

Many researchers built a new kernel for one of these reasons:

- Challenge basic assumptions made in timesharing OS
- Developing real-time process models
- Developing real-time synchronization primitives
- Developing solutions facilitating timing analysis
- Strong emphasis on predictability or fault tolerance
- Investigate the object-oriented approach
- Real-time multiprocessor/multicore support

Small, Fast, Proprietary Kernels

- Usually used for small embedded systems
- Typically specialized for one particular application
- Typically stripped down and optimized versions:
  - Fast context switch
  - Small size, limited functionality
  - Low interrupt latency
  - Fixed or variable sized partitions for memory management
- FreeRTOS, PICOS18, pSOS, MicroC OS, ...

Needs of Concurrency

Reasons for concurrency

- Functional
  - allow multiple users
  - perform many operations concurrently
- Performance
  - take advantage of blocking time
  - parallelism in multi-processor machines
- Expressive Power
  - many control application are inherently concurrent
  - concurrency support helps in expressing concurrency, making application development simpler

Multi-Tasking

- The execution entities (tasks, processes, threads, etc.) are competing from each other for shared resources
  - In FreeRTOS, we will only use “tasks” for the rest of the whole course.
- Scheduling decision policy is needed
  - When to schedule an entity?
  - Which entity to schedule?
  - How to schedule entities?
- The focus of this course is to implement and analyze scheduling algorithms for different settings in real-time systems

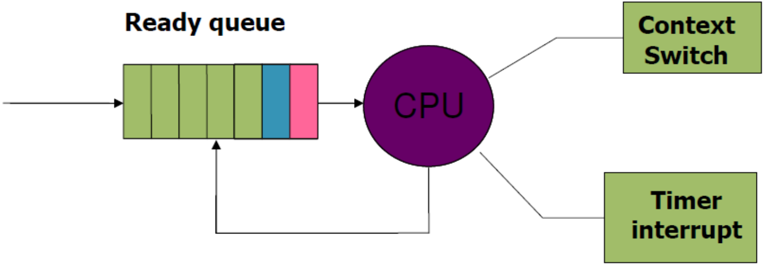
Context switch

- It happens when
  - A task instance has been "preempted" by another higher-priority task instance
  - The task instance blocks due to some condition
  - In time-sharing systems, the task instance has completed its "round" and it is the turn of some other task instance
- We must be able to restore the task later
  - Therefore we must save its state before switching to another task instance

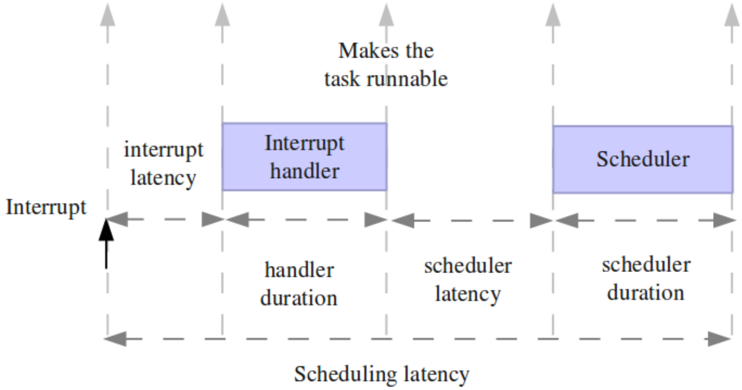
Definitions of Terms

- A task is a re-incurring entity which allows for multiple initializations of task instances
- A task instance (job) is a single execution unit for a re-incurring task

Time Sharing



Scheduling Latency



$$\text{latency} = \begin{array}{l}\text{interrupt latency} + \text{handler duration} + \\ \text{scheduler latency} + \text{scheduler duration}\end{array}$$

Outline

Organization

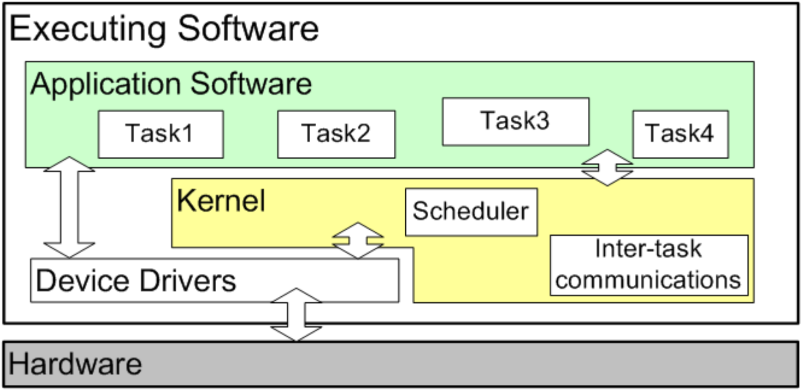
Introduction to Real-Time Systems

FreeRTOS

Introduction to FreeRTOS

- Cross platform OS
- Open source
- Modular design
- Light-weighted and micro-kernel approach
- features for powerful trace and stack overflow detections
- Commercial licensing options
- http://www.freertos.org

FreeRTOS


Executing Software
Application Software
Task1
Task2
Task3
Task4
Kernel
Scheduler
Inter-task communications
Device Drivers
Hardware

Key Components in FreeRTOS

- Task Management
    - To create an execution entity
    - Fixed-priority scheduling
- Queue Management
    - To create and manipulate real-time message queues for inter-task communications
- Interrupt Management
    - To allow different interrupt service routines (ISR)
    - Deferred interrupt scheme
- Resource and Memory Management
    - Critical sections and semaphores
    - Priority inversion and priority inheritance protocol
- Related topics
    - Bootstrap
    - Troubleshooting and debugging