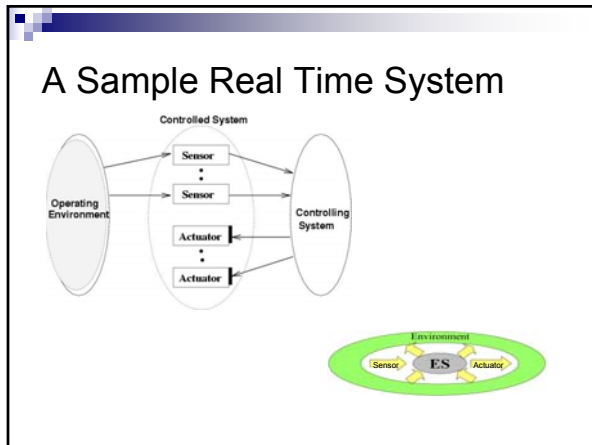


Real-time OS

- ## Introduction
- What is a Real-Time System?
 - correctness of the system depends not only on the logical results, but also on the **time** in which the results are produced.
 - works in a reactive and time-constrained environment
 - Examples
 - Real-time temperature control of a chemical reactor
 - Space mission control system
 - Nuclear power generator system
 - Many safety-critical systems



- ## A Sample Real Time System
- **Mission:** Reaching the destination safely.
 - Controlled System: Car.
 - Operating environment: Road conditions.
 - Controlling System
 - Human driver: Sensors - Eyes and Ears of the driver.
 - Computer: Sensors - Cameras, Infrared receiver, and Laser telemeter.
- Volvo XC90
Some 35+ computers
7 networks
-

- ## A Sample Real Time System
- **Controls:** Accelerator, Steering wheel, Break-pedal.
 - **Actuators:** Wheels, Engines, and Brakes.
 - **Critical tasks:** Steering and braking
 - **Non-critical tasks:** Turning on radio

- ## A Sample Real Time System
- **Performance** is not an absolute one. It measures the goodness of the outcome relative to the best outcome possible under a given circumstance.
 - **Cost of fulfilling the mission** → Efficient solution.
 - **Reliability of the driver** → Fault-tolerance is a must.

Key Properties

Real-time systems

- Timeliness & Concurrency
- Reliability
- Reactivity
- QoS

Embedded systems

- Timeliness & Concurrency
- Dedicated (not "general purpose")
- Liveness (Non-terminating programs)
- Reliability
- QoS

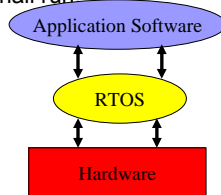
What is What???

What is the relation between the terms "Real-Time System" and "Embedded system"?

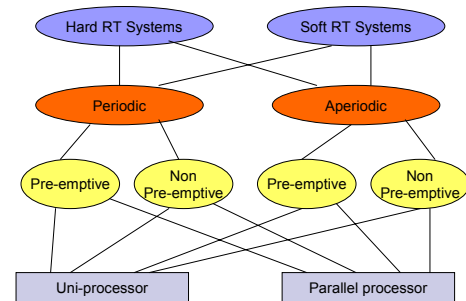
1. All Embedded Systems are Real-Time Systems
2. All Real-Time Systems are Embedded Systems
3. **Some** Real-Time Systems are Embedded Systems, and **some** Embedded Systems are Real-Time Systems
4. In real-time I'd like to embed myself in a nice warm bed (aka. I don't know the answer)

RTOS Kernel

- RTOS Kernel provides an Abstraction layer that hides from application software the hardware details of the processor / set of processors upon which the application software shall run



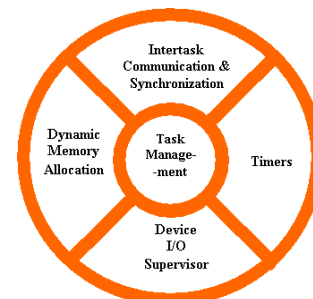
Real Time Systems



What is the difference?

- Hard Real Time system -> guarantee deadlines
 - To guarantee deadline, need to know worst case execution times
 - Predictability: need to know if deadlines may be missed
- Soft Real Time system -> try to meet deadlines
 - If a deadline is missed, there is a penalty
 - Provides statistical guarantees (probabilistic analysis)
 - Need to know the statistical distribution of execution times

RTOS Kernel Functions



Task Management

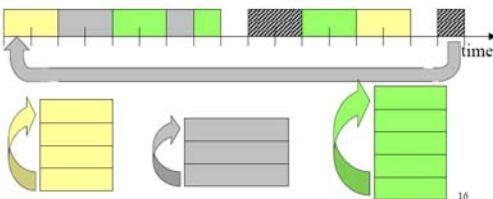
- Set of services used to allow application software developers to design their software as a number of separate chunks of software each handling a distinct topic, a distinct goal, and sometimes its own real-time deadline.
- Main service offered is Task Scheduling
 - controls the execution of application software tasks
 - can make them run in a very timely and responsive fashion.

Task Scheduling

- Scheduling is:
 - The process of selecting which task to execute at what time
- In real-time systems:
 - goal of scheduling is to make all tasks meet their deadline
- Goal in desktop systems:
 - fairness, responsiveness, throughput, etc...

Off-line Scheduling

- Guaranteeing deadlines is difficult
- Idea: construct schedule before the system is executed



Off-line scheduling (cont)

- Terminology:
 - Schedule is constructed before run-time
 - **Off-line**
 - Schedule stored in table
 - **Table driven**
 - Cyclically repeated while system runs
 - **Static cyclic scheduling**
 - Progress in the schedule is governed by progress of time



Off-line scheduling (cont II)

- Traditional technique in real-time systems
 - Used for 30+ years.
- Today used in
 - small and simple systems, and in
 - safety critical systems



Off-line scheduling (cont III)

- Pros
 - No OS overhead
 - Very simple OS-mechanism
 - Easy to verify that deadlines are met
 - "Proof-by-construction"
 - Predictable execution
 - Ease system testing
 - Ease system verification
- Cons
 - Inflexible
 - Maybe difficult to find schedule
 - May waste resources
 - Blank spaces in schedule
 - Often need to put in tasks too often in schedule (polling)
 - May have poor responsiveness
 - Difficult trade-off between excessive polling and responsiveness

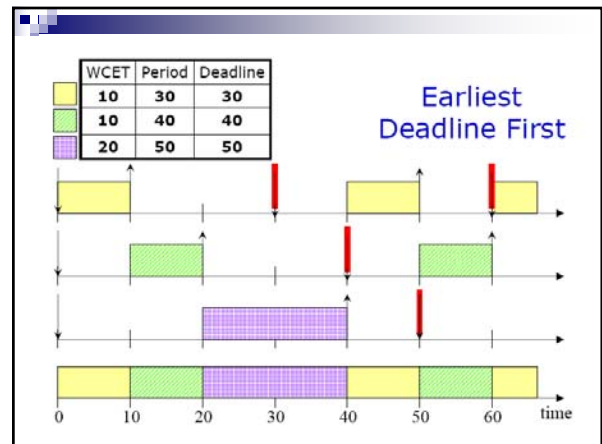
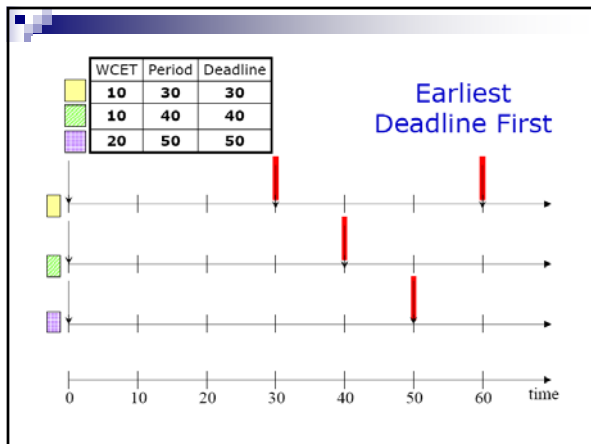
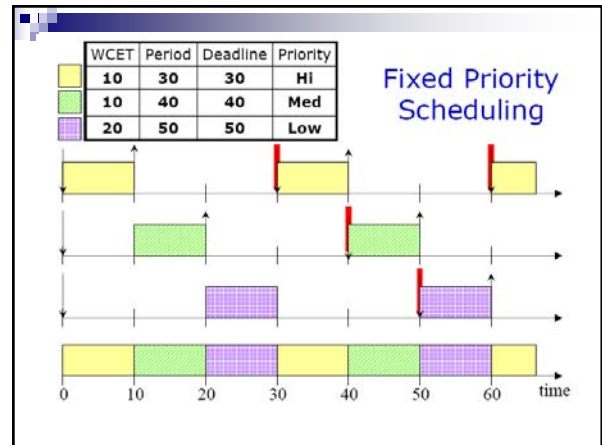
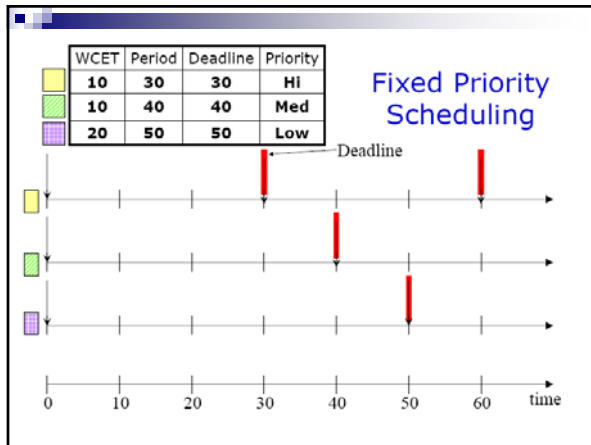
On-line Scheduling

- OS decide which task to run
- Terminology
 - Schedule is constructed during run-time
 - **On-line**
 - Schedule may change at any time
 - **Dynamic scheduling**
 - Scheduling decisions are based on what events takes place in the system
 - **Event-triggered scheduling**

Progress of time is not an event, but the firing of a timer interrupt is.

On-line scheduling (cont)

- Pros:
 - Easy to add/remove tasks and update system
 - No waist-full polling is needed
 - Quick response-times (for important function)
- Cons:
 - Each time systems execute, a different execution pattern may occur
 - Difficult to do system testing
 - Difficult to do system verification
 - Larger overhead in OS and memory-usage
- Issue:
 - How to guarantee that deadlines are meet?



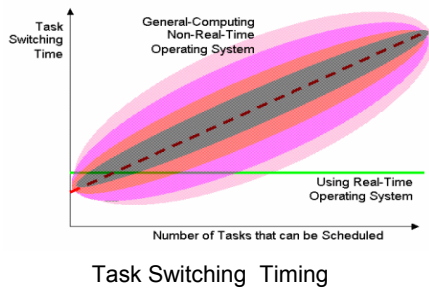
FPS vs EDF

- EDF can (often) handle higher loads and still meet deadlines
- However, if EDF misses a deadline the system may become instable (more misses)
- In FPS, if a deadline is missed its always the lowest priority task that misses (stable behavior)
- Support for FPS is available in "all" Real-Time OSes
- Support for EDF is only available in research OSes
- In both EDF and FPS method to guarantee that deadlines are meet are available
 - but today not practically useful in commercial OSes

Task Switch

- A Non Real time operating system might do task switching only at timer tick times.
- Even with preemptive schedulers a large array of tasks is searched before a task switch.
- A Real time OS shall use Incrementally arranged tables to save on time.

Task Switch

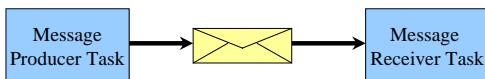


Intertask Communication & Synchronization

- These services makes it possible to pass information from one task to another without information ever being damaged.
- Makes it possible for tasks to coordinate & productively cooperate with each other.

Inter-Task communication & Synchronization

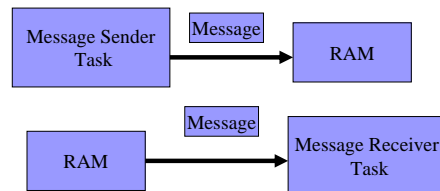
- The most important communication b/w tasks in an OS is the passing of data from one task to another.



- If messages are sent more quickly than they can be handled, the OS provides message queues for holding the messages until they can be processed.

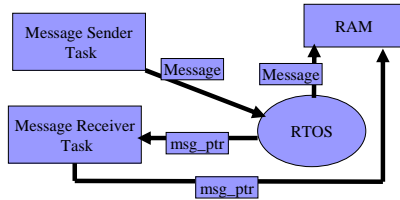
Message passing in OS

- Most General Purpose OS actually copy messages twice as they transfer them from task to task via a message queue.



Message passing in RTOS

- In RTOS, the OS copies a pointer to the message, delivers the pointer to the message-receiver task, and then deletes the copy of the pointer with message-sender task.



Dynamic Memory Allocation in General Purpose OS

- Non-real-time operating systems offer memory allocation services from what is termed a **Heap**.
- Heaps suffer from a phenomenon called **External Memory Fragmentation**.
- Fragmentation problem is solved by **Garbage collection / Defragmentation**.
- Garbage collection algorithms are often wildly non-deterministic.

Dynamic Memory Allocation in RTOS

- RTOS does it by a mechanism known as Pools.
- Pools memory allocation mechanism allows application software to allocate chunks of memory of 4 to 8 different buffer sizes per pool.
- Pools avoid external memory fragmentation, by not permitting a buffer that is returned to the pool to be broken into smaller buffers in the future.
- When a buffer is returned the pool, it is put onto a **free buffer list** of buffers of its own size that are available for future re-use at their original buffer size

OS Used In Embedded Systems

- Non Real-time Embedded OS
 - Embedded Linux
- Real-time OS
 - Integrity
 - VxWorks (Wind River)
 - Linux
- Handheld/Mobile OS
 - Handhelds.org

