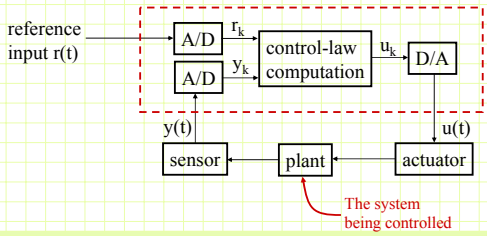
Διαχείριση Πόρων σε Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Real-Time Systems

- A real-time system is a system whose specification includes both logical and temporal correctness requirements.
  - ✓ Logical Correctness: system produces correct outputs.
  - ✓ Temporal Correctness: system produces outputs at the right time.
    - o It is not enough to say that "brakes were applied"
    - o You want to be able to say "brakes were applied at the right time"
- Main Characteristics of a Real-Time System
  - ✓ Event-driven, reactive (computations respond to external event).
  - ✓ High cost of failure.
  - ✓ Concurrency/multiprocessing.
  - ✓ Stand-alone (without operator) /continuous operation.
  - ✓ Reliability/fault-tolerance requirements.
  - ✓ Predictable behavior.

Example Real-Time System

Example: A simple one-sensor, one-actuator control system.



Example Real-Time System (cont'd)

Pseudo-code for this system:

```
set timer to interrupt periodically with period T;
at each timer interrupt do
    do analog-to-digital conversion to get y;
    compute control output u using y and r;
    output u and do digital-to-analog conversion;
end do
```

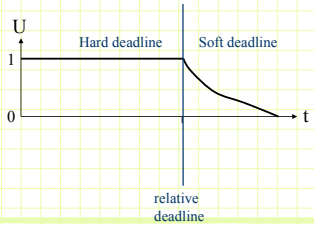
T is called the **sampling period**. *T* is a key design choice. *T*ypical range for *T*: seconds to milliseconds.

Hard vs. Soft Real Time Systems

- A **hard deadline** *must* be met.
  - ✓ If ***any*** hard deadline is ***ever*** missed, then the system is ***incorrect***.
  - ✓ A means for ***validating*** that deadlines are met is required.
- A **soft deadline** may *occasionally* be missed.
  - ✓ **Question:** How to define "occasionally"?
- **Hard real-time system:** A real-time system in which ***all*** deadlines are hard.
  - ✓ **Examples:** Nuclear power plant control, flight control.
- **Soft real-time system:** A real-time system in which **some** deadlines are soft.
  - ✓ **Examples:** Telephone switches, multimedia applications.

Ορισμός του "Occasionally"

- Use probabilistic requirements.
  - ✓ e.g., "99% of deadlines will be met".
- Define a "usefulness" function:
  - ✓ validation is trickier here.



What's Important in Real-Time

- Metrics for real-time systems differ from those for time-sharing systems.

	Time-Sharing Systems	Real-Time Systems
Capacity	High throughput	Schedulability *1
Responsiveness	Fast average response	Ensured worst-case response
Overload	Fairness	Stability *2

- ✓ *1: schedulability is the ability of tasks to meet all hard deadlines
- ✓ *2: stability in overload means the system meets critical deadlines even if all deadlines cannot be met

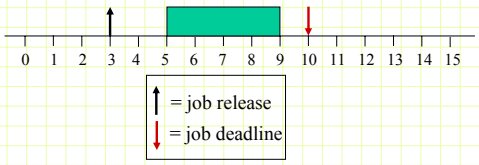
Common Misconceptions about Real-Time Systems (Stankovic '88)

- Advances in hardware will take care of real-time requirements.
  - ✓ "buy a faster processor" argument…
- Real-time computing is equivalent to fast computing
  - ✓ No, it means PREDICTABLE computing
- Real-time programming is assembly coding
  - ✓ Not only, we would like to automate Real Time system design, instead of relying on clever hand-crafted code
- "Real time" is performance engineering
  - ✓ No, in RT computing, timeliness is almost always more important than raw performance…
- It is not meaningful to talk about guaranteeing real-time performance when things can fail
  - ✓ Though things may fail, we certainly do not want the operating system to fail!

Some Definitions

- **Task,** T_i **:** an instance of a program at execution.
- **Job,** J_i**:** an instance of a task (appearing each time the task gets the CPU).
- Jobs require **resources** to execute.
  - ✓ **Example resources:** CPU, memory, disk, network, critical section,...
- **Release time of a job,** r_i**:** The time instant the job becomes ready to execute.
  - ✓ Sometimes, instead of r_i, release time is specified to range over $[r_i^-, r_i^+]$. This range is called **release-time jitter**.
- **Absolute Deadline of a job,** d_i**:** The time instant by which the job must complete execution.
- **Relative deadline of a job,** D_i**:** "Absolute Deadline – Release time".
- **Response time of a job:** "Completion time – Release time".
  (Note completion time is in general different than absolute deadline)
- **Execution time of a job,** e_i**:** time needed by a job to finish
  - ✓ Sometimes, instead of e_i, execution time is specified to range over $[e_i^-, e_i^+]$.
  - ✓ **Note:** It can be difficult to get a precise estimate of e_i

Example

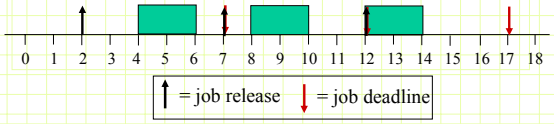


- Job is released at time 3.
- Its (absolute) deadline is at time 10.
- Its relative deadline is 7.
- Its response time is 6.

Periodic, Sporadic, Aperiodic Tasks

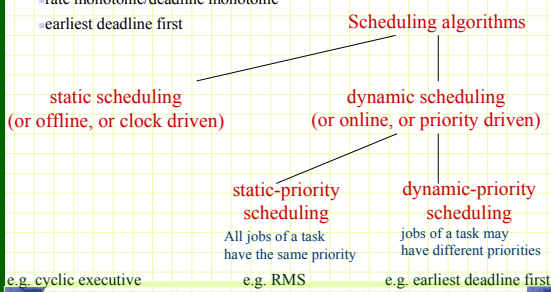
- Periodic task:
  - ✓ A task T_i released periodically.
  - ✓ The task period p_i is the interval between job releases.
- Sporadic and Aperiodic tasks**:** both released at arbitrary times.
  - ✓ **Sporadic:** Has a hard deadline.
  - ✓ **Aperiodic:** Has no deadline or a soft deadline.

A periodic task T_i with $r_i = 2$, $p_i = 5$, $e_i = 2$, $D_i = 5$ executes like this:



Scheduling Algorithms

- CPU scheduling policy: a rule to select task to run next
  - cyclic executive
  - rate monotonic/deadline monotonic
  - earliest deadline first



Rate Monotonic Scheduling / Analysis (RMS/RMA)

- RMS: Priorities of periodic tasks are based on their rates: highest rate gets highest priority.
- RMS has strong theoretical basis:
  - ✓ optimal fixed scheduling policy (when deadlines are at end of period)
  - ✓ analytic formulas to check schedulability
- RMA: Rate-monotonic analysis is a set of mathematical techniques for analyzing sets of real-time tasks.
- Basic theory applies only to independent, periodic tasks, but has been extended to address:
  - ✓ priority inversion
  - ✓ task interactions
  - ✓ aperiodic tasks
- Must distinguish between scheduling and analysis
  - ✓ rate monotonic scheduling forms the basis for rate monotonic analysis
  - ✓ any scheduling approach may be used, but all real-time systems should be analyzed for timing

Periodic Tasks - Concepts and Definitions

- Periodic task τ_i
  - ✓ initiated (released) at fixed intervals
  - ✓ must finish before start of next cycle (deadline=end of period)

$$U_i = \frac{C_i}{T_i}$$

- Task's CPU utilization:
  - ✓ C_i = worst-case compute time (execution time) for task τ_i
  - ✓ T_i = period of task τ_i
- CPU utilization for a set of tasks

$$U = U_1 + U_2 + \ldots + U_n$$

- U <= 1, otherwise overload

Schedulability: Utilization Bound (UB) Test

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1)$$

- Utilization bound (UB) test: a set of n independent periodic tasks scheduled by the rate monotonic algorithm will always meet their deadlines, for all task phasings, if

U(1) = 1.0 U(4) = 0.756 U(7) = 0.728
U(2) = 0.828 U(5) = 0.743 U(8) = 0.724
U(3) = 0.779 U(6) = 0.734 U(9) = 0.720
U(0) = ∞
U(∞) = ln2 = 0.693 =>
for large number of tasks, we have schedulability if utilisation is <= 69%

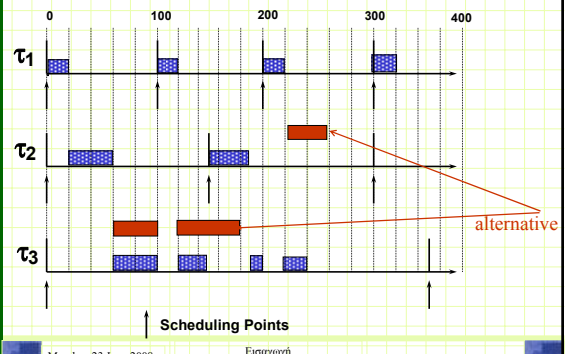
- For ***harmonic*** task sets, the utilization bound is U(n)=1.00 for all n.
- A task set is ***harmonic*** iff the periods of each pair of tasks in the task set are multiple of one-another.

Example: UB Test

	C	T	U
Task τ_1:	20	100	0.200
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

- Total utilization is .200 + .267 + .286 = .753 < U(3) = .779
- The periodic tasks in the sample problem are schedulable according to the UB test
- NOTE: we order tasks in descending priority (or rate) order.

Timeline for Sample Problem



Towards a More Precise Test

- UB test has three possible outcomes:

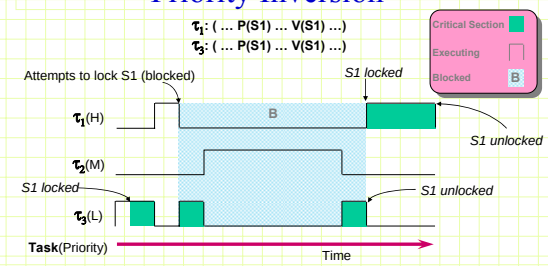
0 < U <= U(n) → Success (schedulable)

U(n) < U < 1.00 → Inconclusive (cannot decide) maybe schedulable, use another, precise test

1.00 < U → Overload (not schedulable)

- UB test is conservative.
- A more precise test ('RT' test) can be applied.

Priority Inversion



- Problem: **Unbounded priority inversion**: while blocking the high priority task, the low priority task is pre-empted by medium priority tasks for an unbounded number of times. The high priority task never finishes!
- Solution: Priority Inheritance: when a low priority task blocks a higher priority task, it inherits the priority of that task.