

Real Time Operating Systems and Middleware

Introduction to Real-Time Systems

Luca Abeni

`luca.abeni@unitn.it`

Credits: Luigi Palopoli, Giuseppe Lipari, Marco Di Natale, and Giorgio Buttazzo

Some Information

- Slides available from <http://www.disi.unitn.it/~abeni/RTOS>
- Interested students can have a look at:
 - *Giorgio Buttazzo*, “**HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications**”, Second Edition, Springer, 2005
- Exam: Written Exam
 - 4 questions, 30 minutes per question
 - Each answer gets a score from 0 to 30
 - **OPTIONAL** project.
- Prerequisites:
 - Programming skills: C, maybe C++
 - Knowledge about Operating Systems

Prerequisites

- You must know how to code in C (optionally C++)
 - This is not about knowing the C syntax...
 - It is about writing **good** and **clean** C code
 - If you do not feel confident enough with C, have a look at “**The C Programming Language**” by *Kerrigan* and *Ritchie*
 - Some notes about C programming are available on the course web site
- About Operating Systems:
 - “Sistemi Operativi I”, or similar exams
 - References: a good OS book (Stallings, ...)
 - How to use a shell, basic POSIX commands, make, how to compile, ...

Overview of the Course - 1

- Real-Time Systems: what are they?
 - Real-Time Computing, Temporal Constraints
 - Definitions and task model
 - Real-Time scheduling
- Notes about real-time programming, RT-POSIX, pthreads, ...
- Real-Time Scheduling
 - Fixed Priority scheduling, RM, DM
 - EDF and dynamic priorities
 - Resource Sharing (Priority Inversion, etc...)

Overview of the Course - 2

- Operating System structure
 - Notes about traditional kernel structures
 - Sources of kernel latencies
 - Some approaches to real-time kernels:
 - dual kernel approach
 - interrupt pipes
 - microkernels
 - monolithic kernels and RT

Real-Time Operating Systems

- Real-Time operating system (RTOS): OS providing support to Real-Time applications
- Real-Time application: the correctness depends not only on the output values, but also on the time when such values are produced
- Operating System:
 - Set of computer programs
 - Interface between applications and hardware
 - Control the execution of application programs
 - Manage the hardware and software resources
- OS as...
 - A Service Provider for user programs (interface...)
 - A Resource Manager

Operating System as a Resource Manager

- **Process** Management
- **Memory** Management
- **File** Management
 - VFS
 - File System
- Networking
- Device Drivers
- Graphical Interface

Resources must be managed so that real-time applications are served properly

Operating System Services

- Services (Kernel Space):
 - Process Synchronisation, Inter-Process Communication (IPC)
 - Process / Thread Scheduling
 - I / O
 - Virtual Memory

Specialised API?

- Resource Manager (device drivers...)
 - Interrupt Handling
 - Device Management
 - ...

OS Structure?

Typical Applications

- Air traffic control
- Flight control systems
- Robotics
- Telecommunication systems
- Nuclear / chemical power plants
- Multimedia systems
- Automotive systems

Real-Time Systems: What???

- Real-Time application: the time when a result is produced matters
 - a correct result produced too late is equivalent to a wrong result, or to no result
 - characterised by **temporal constraints** that have to be respected
- Example: mobile vehicle with a software module that
 1. Detects obstacles
 2. Computes a new trajectory to avoid them
 3. Computes the commands for engine, brakes, ...
 4. Sends the commands

Real-Time Systems: What???

- If the commands are correctly computed, but are not sent in time...
- ...The vehicle crashes into the obstacle before receiving the commands!
- Examples of temporal constraints:
 - must react to external events in a predictable time
 - must repeat a given activity at a precise rate
 - must end an activity before a specified time
- Temporal constraints are modelled using the concept of *deadline*

Real-Time & Determinism

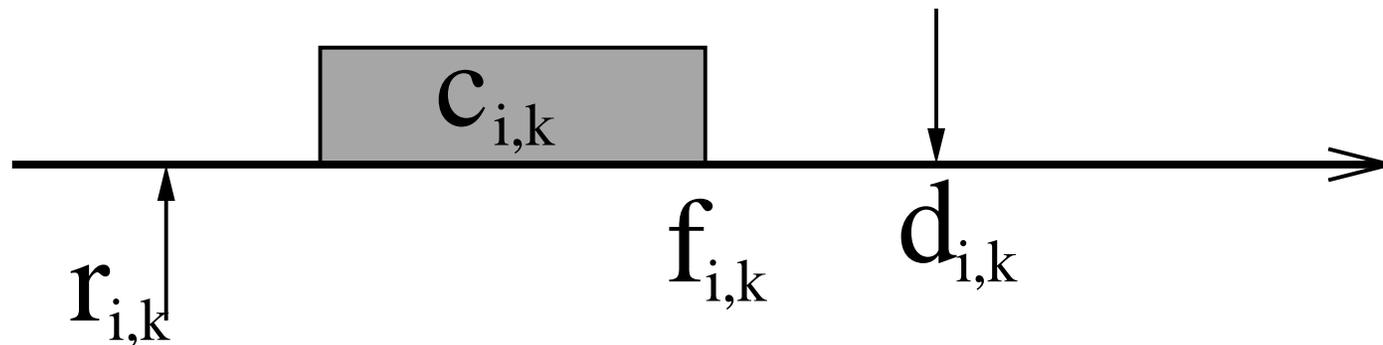
- A Real-Time system is not just a “fast system” ...
- ... Because speed is always relative to a specific environment.
- Running faster is good, but does not guarantee a correct behaviour
 - It must be possible to *prove* that temporal constraints are **always respected**
 - ... \Rightarrow worst-case analysis
- In general, fast systems tend to minimise the average response time of a task set ...
- ... While a real-time system **must** guarantee the timing behaviour of RT tasks!

Processes, Threads, and Tasks

- Algorithm → logical procedure used to solve a problem
- Program → formal description of an algorithm, using a *programming language*
- Process → instance of a program (program in execution)
 - Thread → flow of execution
 - Task → process or thread
- A task can be seen as a sequence of actions ...
- ... and a deadline must be associated to each activity!
 - Some kind of formal model is needed to identify activities and associate deadlines to them

Mathematical model of a task - 1

- Real-Time task τ_i : stream of jobs (or instances) $J_{i,k}$
- Each job $J_{i,k} = (r_{i,k}, c_{i,k}, d_{i,k})$:
 - Arrives at time $r_{i,k}$ (activation time)
 - Executes for a time $c_{i,k}$
 - Finishes at time $f_{i,k}$
 - Should finish within an **absolute deadline** $d_{i,k}$



Mathematical model of a task - 2

- Summing up, a job is an abstraction used to associate deadlines (temporal constraints) to activities
 - $r_{i,k}$ is the time when job $J_{i,k}$ is *activated* (by an external event, a timer, an explicit activation, etc...)
 - $c_{i,k}$ is the time needed by job $J_{i,k}$ to complete
 - $d_{i,k}$ is the absolute time instant by which job $J_{i,k}$ must complete
 - job $J_{i,k}$ respects its deadline if $f_{i,k} \leq d_{i,k}$
- the response time of job $J_{i,k}$ is $\rho_{i,k} = f_{i,k} - r_{i,k}$

Periodic Tasks

A periodic task $\tau_i = (C_i, D_i, T_i)$ is a stream of jobs $J_{i,k}$, with

$$\begin{aligned}r_{i,k+1} &= r_{i,k} + T_i \\d_{i,k} &= r_{i,k} + D_i \\C_i &= \max_k \{c_{i,k}\}\end{aligned}$$

- T_i is the task *period*;
- D_i is the task *relative deadline*;
- C_i is the task worst-case execution time (WCET);
- R_i is the worst-case response time:
 $R_i = \max_k \{\rho_{i,k}\} = \max_k \{f_{i,k} - r_{i,k}\};$
 - for the task to be correctly scheduled, it must be
 $R_i \leq D_i$.

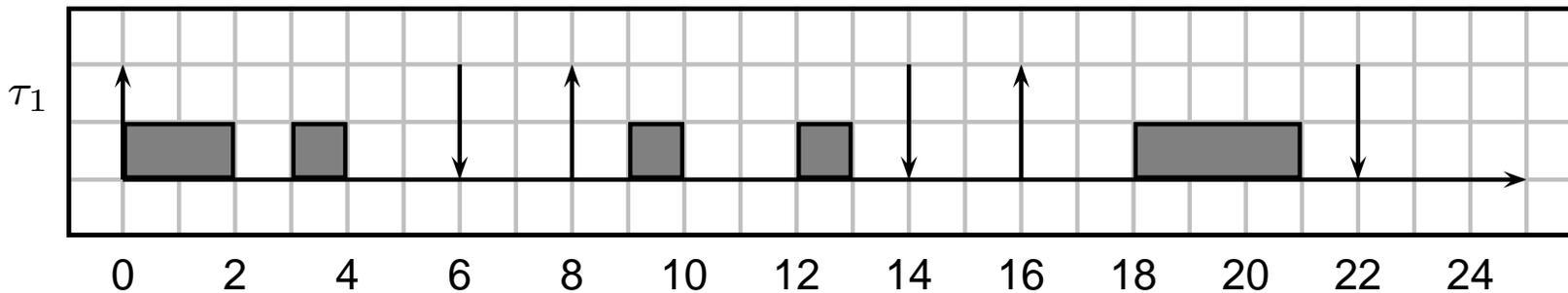
Periodic Task Model

- A periodic task has a regular structure (cycle):
 - activate periodically (period T_i)
 - execute a computation
 - suspend waiting for the next period

```
void *PeriodicTask(void *arg)
{
    <initialization>;
    <start periodic timer, period = T>;
    while (cond) {
        <read sensors>;
        <update outputs>;
        <update state variables>;
        <wait next activation>;
    }
}
```

Graphical representation

Tasks are graphically represented by using a GANNT chart. For example, the following picture shows a schedule of a periodic task $\tau_1 = (3, 6, 8)$ (with $WCET_1 = 3$, $D_1 = 6$, $P_1 = 8$)



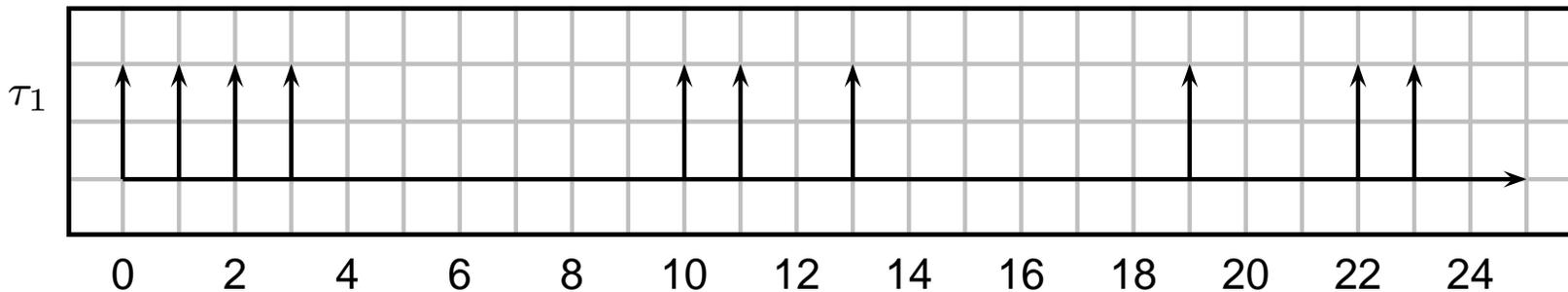
Notice that, while job $J_{1,1}$ and $J_{1,3}$ execute for 3 units of time (WCET), job $J_{1,2}$ executes for only 2 units of time.

Aperiodic Tasks

- *Aperiodic* tasks are not characterised by periodic arrivals:
 - A minimum interarrival time between activations does not exist
 - Sometimes, aperiodic tasks do not have a particular structure
- Aperiodic tasks can model:
 - Tasks that respond to events that occur rarely. Example: a mode change.
 - Tasks that respond to events that happen with an irregular structure. Example: bursts of packets arriving from the network.

Aperiodic Tasks - Example

The following example shows a possible arrival pattern for an aperiodic task τ_1



Notice that arrivals might be bursty, and there is not a minimum time between them.

Sporadic tasks

- *Sporadic* tasks are aperiodic tasks characterised by a *minimum interarrival time* between jobs.
- In this sense, they are similar to periodic tasks, but...
 - Periodic task \Rightarrow activated by a **periodic timer**
 - Sporadic task \Rightarrow activated by an **external event** (for example, the arrival of a packet from the network)

```
void *SporadicTask(void *)
{
    <initialization>;
    while (cond) {
        <computation>;
        <wait event>;
    }
}
```

Mathematical model of a sporadic task

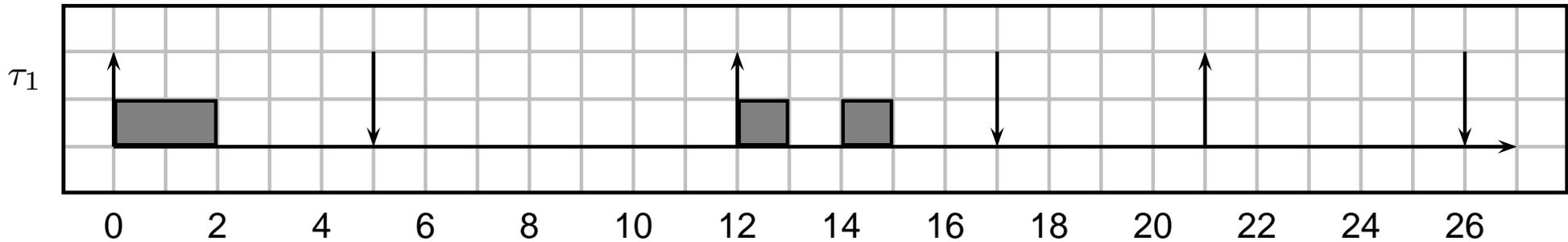
Similar to a periodic task: a sporadic task $\tau_i = (C_i, D_i, T_i)$ is a stream of jobs $J_{i,k}$, with

$$\begin{aligned}r_{i,k+1} &\geq r_{i,k} + T_i \\d_{i,k} &= r_{i,k} + D_i \\C_i &= \max_k \{c_{i,k}\}\end{aligned}$$

- T_i is the task *minimum interarrival time* (MIT);
- D_i is the task *relative deadline*;
- C_i is the task *worst-case execution time* (WCET).
- Again, for the task to be correctly scheduled it must be $R_i \leq D_i$.

Graphical representation

The following example, shows a possible schedule of a sporadic task $\tau_1 = (2, 5, 9)$.



Notice that

$$r_{1,2} = 12 > r_{1,1} + T_1 = 9$$

$$r_{1,3} = 21 = r_{1,2} + T_1 = 21$$

Task Criticality - 1

- A deadline is said to be *hard* if a deadline miss causes a critical failure in the system
- A task is said to be a *hard real-time task* if all its deadlines are hard
 - All the task's deadlines have to be guaranteed ($\forall j, \rho_{i,j} \leq D_i \Rightarrow R_i \leq D_i$) a priori, before starting the task
- Examples:
 - The controller of a mobile robot, must detect obstacles and react within a time dependent on the robot speed, otherwise the robot will crash into the obstacles.

Task Criticality - 2

- A deadline is said to be *soft* if a deadline miss causes a degradation in the *Quality of Service*, but is not a catastrophic event
- A task is said to be a *soft real-time task* if it has soft deadlines
 - Some deadlines can be missed without compromising the correctness of the system...
 - ... But the number of missed deadlines must be kept under control, because the “quality” of the results depend on the number of missed deadlines

Soft Real-Time Requirements - 1

- Characterising a soft real-time task can be difficult...
 - What's the tradeoff between “non compromising the system correctness” and not considering missed deadlines?
 - Some way to express the QoS experienced by a (soft) real-time task is needed
- The QoS of a soft task can be expressed in different ways. Examples are:
 - no more than X consecutive deadlines can be missed
 - no more than X deadlines in an interval of time T can be missed

Soft Real-Time Requirements - 2

- Other examples of soft real-time constraints:
 - the *deadline miss probability* must be less than a specified value
 - $P\{f_{i,j} > d_{i,j}\} \leq R_{max}$
 - the *deadline miss ratio* can be used instead

$$\frac{\text{number of missed deadlines}}{\text{total number of deadlines}} \leq R_{max}$$

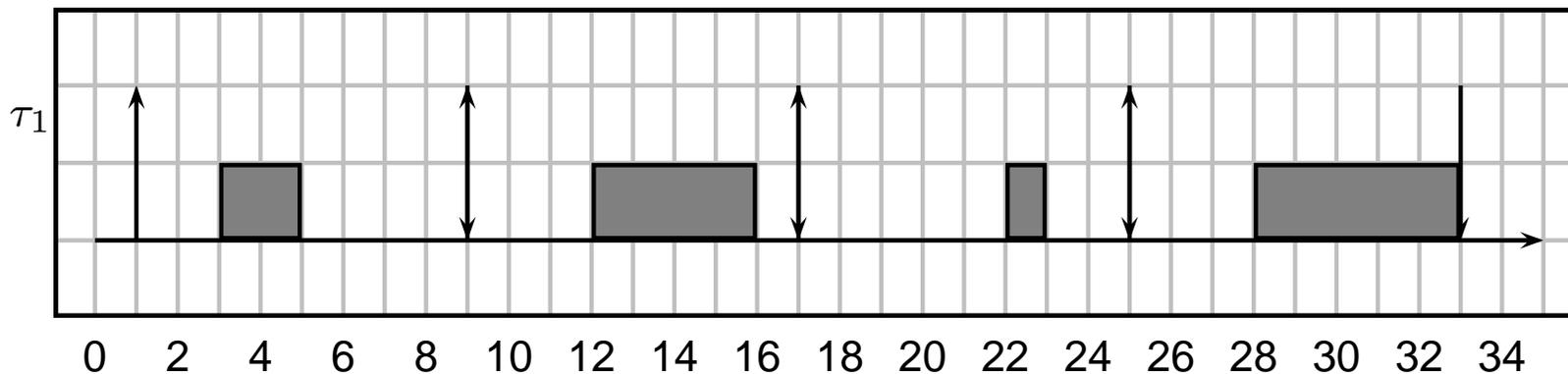
- the maximum *tardiness* must be less than a specified value
 - $\frac{R_i}{D_i} < L$
 - ...

Example of Soft Real-Time

- Audio / Video player:
 - fps: 25 \Rightarrow frame period: $40ms$
 - if a frame is played *a little bit* too late, the user might even be unable to notice any degradation in the QoS...
 - ...but *skipped frames* can be disturbing
 - missing **a lot** of frames by $5ms$ can be better than missing **only few** frames by $40ms$!
- In some robotic systems, some actuations can be delayed with little consequences on the control quality
- In any case, soft real-time does not mean no guarantee on deadlines...

Job Execution Times

- Tasks can have variable execution times between different jobs
- Execution times might depend on different factors:
 - Input data
 - Hardware issues (cache effects, pipeline stalls, etc...)
 - The internal state of the task
 - ...



Variable Execution Times: Video Player

Distribution of the job execution times for a video player
(frame decoding times for an MPEG video)

