Other Topics Relevant for Real-Time Systems

Real-Time Operative Systems Course

Paulo Pedreiras, DETI/UA/IT

December 14, 2021

Last lecture

Aperiodic task scheduling

- Joint execution of periodic and sporadic tasks
- Use of aperiodic task servers
  - Fixed-priority aperiodic task servers
  - Dynamic-priority aperiodic task servers

Agenda for today

Other Topics Relevant for Real-Time Operative Systems

- Non-preemptive scheduling
- Practical aspects related with the implementation of applications on a RTOS
  - Cost of tick handler
  - Cost of context switching
  - Measuring of WCET
  - Cost of ISR
  - Impact of Release Jitter

Non-preemptive scheduling

- Non preemptive scheduling consists in executing the jobs until completion, without allowing its suspension for the execution of higher priority jobs
- Main characteristics/ **advantages** :
  - Very simple to implement, as it is not necessary to save the intermediate job's state.
  - Stack size much lower (equal to the stack size of the task with higher requirements + interrupt service routines)
  - No need for any synchronization protocol to access shared resources, since tasks execute inherently with mutual exclusion

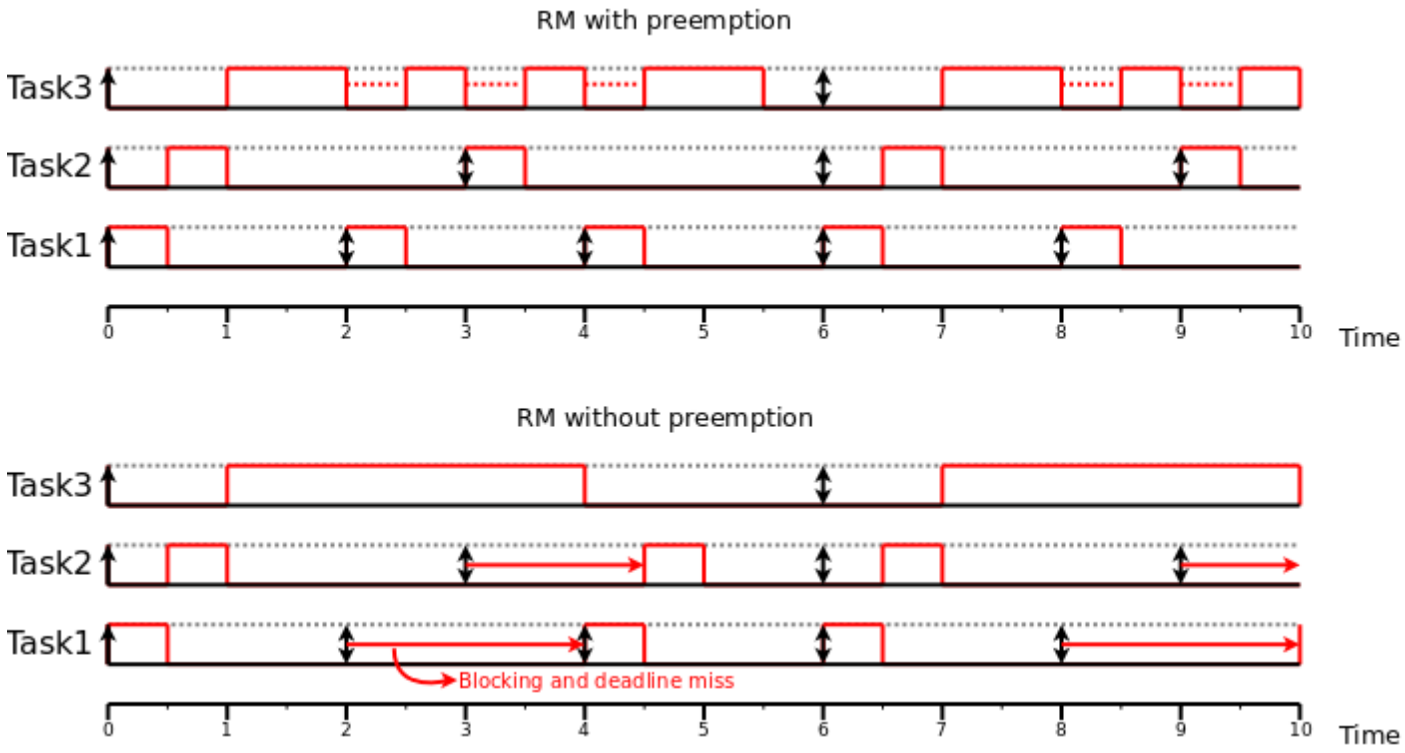
Non-preemptive scheduling

Main characteristics/ **disadvantages** :

- **Penalizes the system schedulability** , mainly when there are tasks with long execution times.
- This penalization may be excessive when, simultaneously, the system has tasks with high activation rates (short periods).
- The penalization can be seen as a blocking on the access of a shared resource, in the case the CPU.

Non-preemptive scheduling

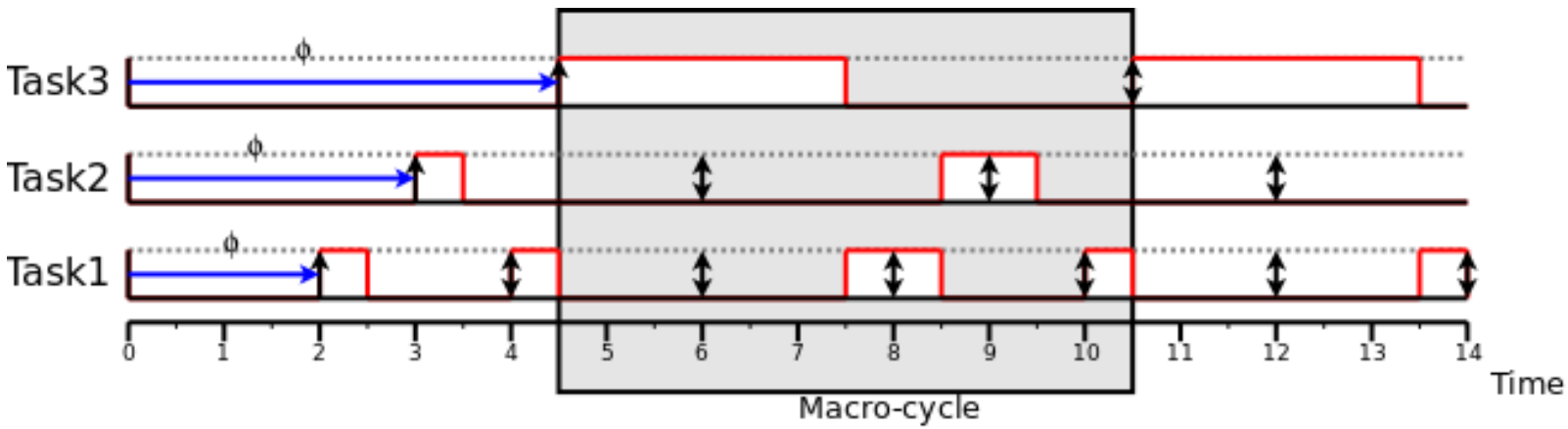
τ	C	T
1	0.5	2
2	0.5	3
3	3	6



Non-preemptive scheduling

- Sometimes the use of offsets may turn a system schedulable.
- However, finding these offset is usually is very complex ...

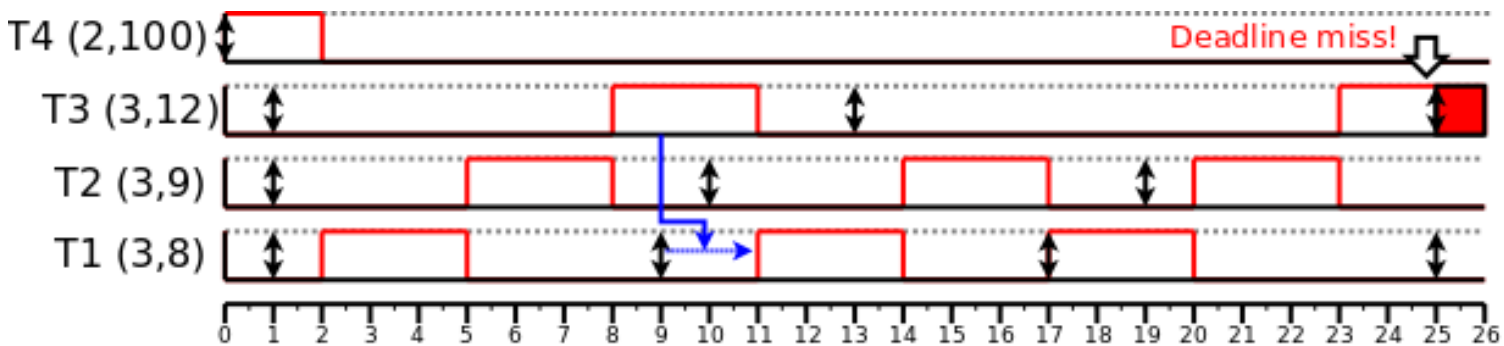
τ	C	T	ϕ
1	0.5	2	2
2	0.5	3	3
3	3	6	4.5



Non-preemptive scheduling

Computation of the Rwc_i with fixed priorities:

- The feasibility analysis must be adapted and becomes more complex.
- In non-preemptive scheduling **the longest response time does not necessarily coincide with the first job after the critical instant** .
  - **Self-pushing phenomenon** : high priority jobs activated during the first instance of τ_i are pushed ahead to the following jobs of τ_i, which may experience higher interference



Non-preemptive scheduling

- Therefore the response time analysis must be performed until the processor finishes executing all tasks with priority greater than or equal to P_i: **Level-i Active Period (L_i)**

$$B_i = \max_{j:P_j<P_i} \{C_j - \delta\}, \delta = \textit{clock resolution}$$

$$L_i(0) = B_i + C_i$$

$$L_i(s) = B_i + \sum_{h:P_h \geq P_i} \left\lceil \frac{L_i(s-1)}{T_h} \right\rceil \cdot C_h$$

L_i is the smallest value for which $L_i(s) = L_i(s-1)$

- For a generic task τ_i we must compute the response time for all jobs $\tau_{i,k}$, where $k = 1, \ldots K_i$ and

$$K_i = \left\lceil \frac{L_i}{T_i} \right\rceil$$

Non-preemptive scheduling

The starting times of the relevant jobs can be computed as follows:

$$\boxed{s_{i,k}^{(0)} = B_i + \sum_{h:P_h>P_i} C_h}$$

$$\boxed{s_{i,k}^{(l)} = B_i + (k-1)\cdot C_i + \sum_{h:P_h>P_i} \left(\left\lfloor \frac{s_{i,k}^{(l-1)}}{T_h} \right\rfloor + 1 \right) \cdot C_h}$$

Since once started a job cannot be preempted, the finishing time is

$$\boxed{f_{i,k} = s_{i,k} + C_i}$$

And finally the worst-case response time is

$$\boxed{Rwc_i = \max_{k\in[1,K_i]} \{f_{i,k} - (k-1)\cdot T_i\}}$$

Non-preemptive scheduling

Exercise:

Given the task set below, compute the worst-case response time, without preemption, of each tasks. Is the task set schedulable without preemption? And with preemption?

τ_i	C_i	$T_i (= D_i)$
1	1	6
2	3	8
3	5	18

Other issues of practical importance

- When developing real applications, there are several aspects that must be taken into account, as they have impact on system schedulability.
- Examples include:
  - The processing cost of internal mechanisms (e.g. **tick handler**)
  - The overhead due to **context switching**
  - **Interrupt Service Routines**
  - Deviations on the tasks activation instants (**release jitter**)

System tick cost

Evaluating the computational cost of the system tick

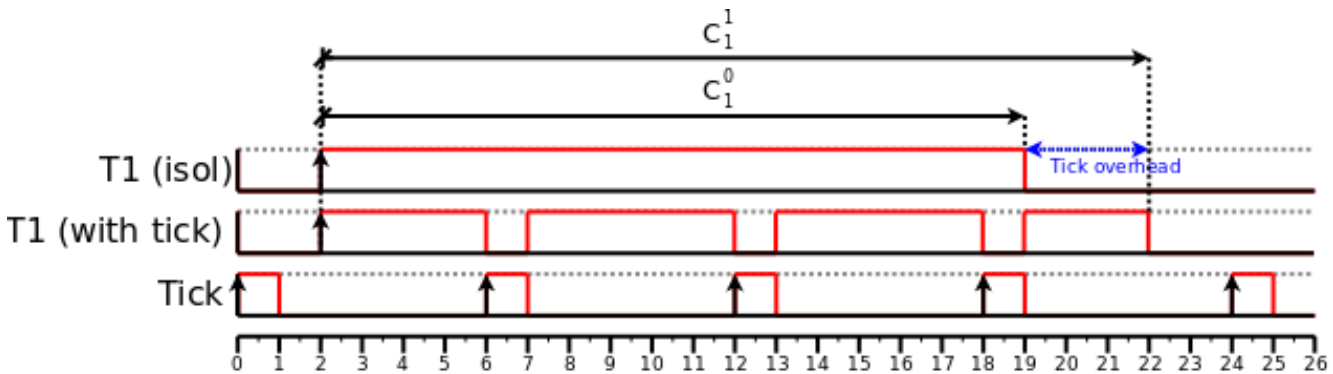
- The service to the system tick uses CPU time (overhead), which is **taken from the tasks' execution** .
- It is the highest priority activity on the system and can be modeled by a **periodic task** .
- The respective overhead (σ) may have a substantial impact on the system, as it is a part of the CPU availability that is not available to the application tasks.

System tick cost

Evaluating the computational cost of the system tick

- Can be measured either directly or via the timed execution of a long function, executed with and without tick interrupts (period T_{tick}) and measuring the difference on the execution times (C_1^0 and C_1^1 respectively).
- In this case the average value for σ is as follows:

$$\sigma = \frac{C_1^1 - C_1^0}{\left\lceil \frac{C_1^1}{T_{Tick}} \right\rceil}$$

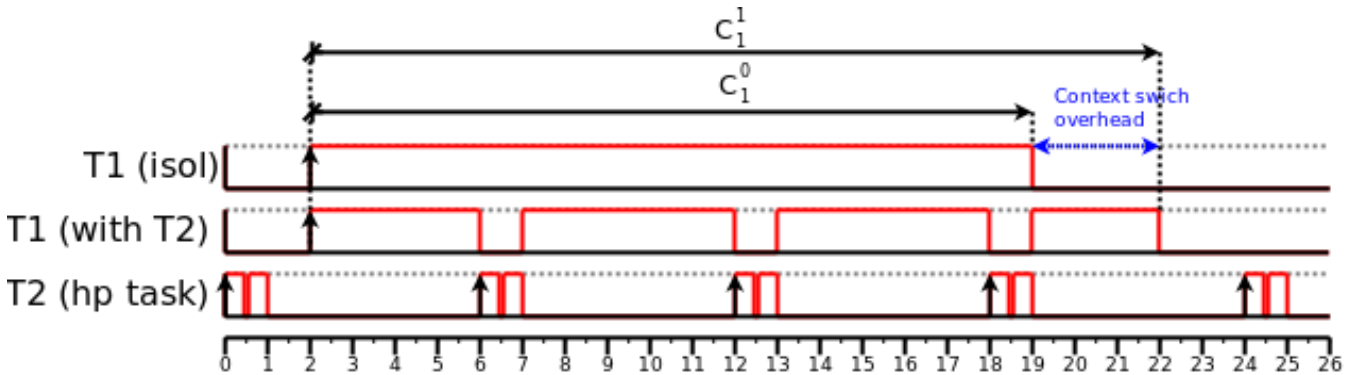


Context switch costs

Evaluating the cost of context switches

- Context switches also require CPU time to save and restore the tasks' context.
- A simple way of measuring this overhead (δ) consists in using two tasks, a long one (τ_1) and another one with higher priority (τ_2), quicker period (T_2) and empty (no code). Then it is only required measuring the execution time of the first task alone (C_1^0) and together with the second one (C_1^1).
- In this case, the average value for δ is:

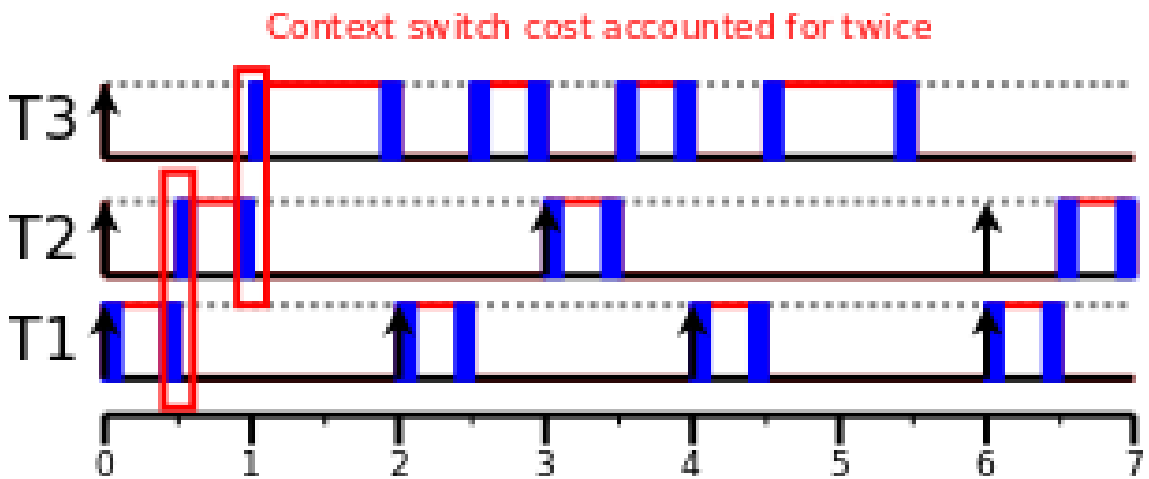
$$\delta = \frac{C_1^1 - C_1^0}{\left\lceil \frac{C_1^1}{T_2} \right\rceil}$$



Context switch costs

Evaluating the cost of context switches (cont.)

- A simple (but pessimistic) way of taking into account the overhead due to context switching (δ) consists in adding that time to the execution time of the tasks. This way it is taken into account not only the context switching overhead due to the task itself as well as the one relative to all context switches that may occur.
- Simple but pessimistic, as the overhead is take into account twice



Task's WCET

Evaluating the task's execution time

- Can be made via source code analysis, to determine the **longest execution path** , according with the input data.
  - Then the corresponding object code is analyzed to determine the required number of CPU cycles
- Note that the **execution time of a task may vary** from instance to instance, according with the input data or internal state, due to presence of conditionals and cycles, etc.

Task's WCET

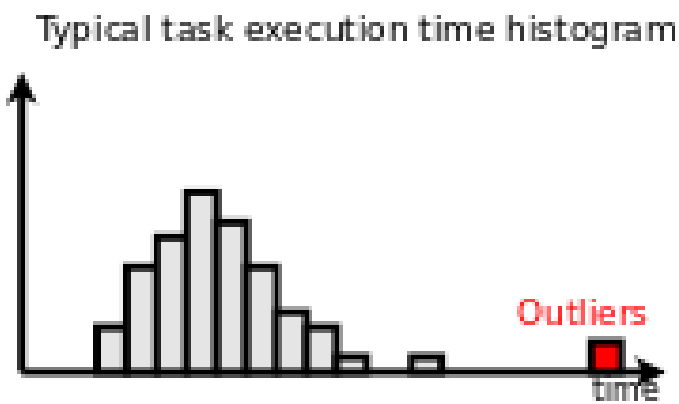
Evaluating the task's execution time (cont.)

- It is also possible execute tasks in isolation and in a controlled fashion, feeding them with **adequate input data** and measuring its execution time on the target platform.
  - This experimental method requires extreme care to make sure that the **longest execution paths are reached** , a necessary condition to obtain an upper bound on the execution time!
- **Modern processors** use features like pipelines and caches (data and/or instructions) that improve dramatically the average execution time but that present an **increased gap between the average and the worst-case scenarios** .
  - For these cases are used specific analysis that try to reduce the pessimism, e.g. by bounding the maximum number of cache misses and pipeline flushes, according with the particular instruction sequences.

Task's WCET

Evaluating the task's execution time (cont.)

- Nowadays there is an growing interest on stochastic analysis of the execution times and respective impact in terms of interference.
- The basic idea consists in determining the distribution of the probability of the execution times and use an estimate that covers a given target (e.g. 99% of the instances).
- In many cases (mainly when the worst case is infrequent and much worst than the average case) this technique allows reducing drastically the impact of the gap between the average execution time and the WCET (higher efficiency)



Cost of ISR's

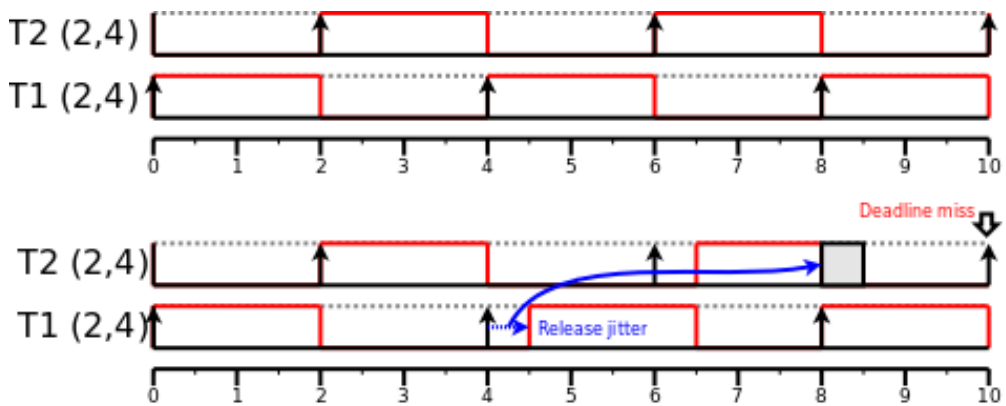
Impact of Interrupt Service Routines

- Generally, the **Interrupt Service Routines** (ISR) execute with an **higher priority** level than all other system tasks.
- Therefore, on a fixed priority system, the respective impact can be taken directly into account by **including these ISR as tasks** in the schedulability analysis.
- In systems with dynamic priorities the situation is much more complex (e.g. how to assign deadlines?). In these cases it is usually considered that the time windows in which such ISR execute are not available for normal tasks execution. This can be taken into account in the CPU load analysis.

Impact of release jitter

Impact of the variations on the tasks' activation instants

- Tasks may suffer deviations on the respective activation instants, e.g. when a task is activated by the completion of another one, by an external interrupt or by the reception of a message on a communication port. In such cases the real time lapse between consecutive activations may vary with respect to the predicted values – **release jitter**
- The existence of release jitter must be taken into account in the schedulability analysis, as in such cases the tasks can **execute during time instants different from the assumed ones** .



Impact of release jitter

Schedulability tests including the impact of Relese Jitter:

- The presence of release jitter can be modeled by the anticipation of the activation instants of the following task instances.

Computing Rwc_i with release jitter (J_k) for preemptive systems with fixed priorities:

$$\forall i, Rwc_i = I_i + C_i, \text{ with } I_i = \sum_{k \in hp(i)} \left\lceil \frac{Rwc_i + J_k}{T_k} \right\rceil \cdot C_k$$

Solved iteratively, as usual:

$$Rwc_i(0) = \left(\sum_{k \in hp(i)} C_k \right) + C_i$$

$$Rwc_i(m+1) = \left(\sum_{k \in hp(i)} \left\lceil \frac{Rwc_i(m) + J_k}{T_k} \right\rceil \cdot C_k \right) + C_i$$

Summary

Other real-time scheduling issues

- Non-preemptive scheduling
- Practical aspects related with the implementation of real-time
  - Cost of tick handler
  - Cost of context switching
  - Measuring of WCET
  - Cost of ISR
  - Impact of Release Jitter