Embedded Systems 21

REVIEW: Scheduling idea

1. Divide the time line into time slices such that each period of each process is divided into an integral number of time slices.
   Slice length $T = \text{GCD}(T_1, \ldots, T_n)$.
2. Within each time slice, allocate processor time in proportion to the utilization $U_i = \frac{C_i}{T_i}$ originating from the various tasks.
   Processing time per slice $r_i = TU_i = T\frac{C_i}{T_i}$.
   Hence, each task runs $\frac{T_i}{T} r_i = \frac{T_i}{T} T \frac{C_i}{T_i} = C_i$ time units within its period.
3. Allocate r_i according to the following algorithm
   (a) Look for the first processor proc_j that has free capacity in its time slices.
   (b) Allocate that portion of r_i to proc_j that proc_j can accommodate.
   (c) If all of r_i has been allocated then proceed with the next task (goto step a).
   (d) Otherwise allocate the remainder of r_i to proc_{j+1}.
   proc_{j+1} has enough spare capacity as it has not previously been used and $r_i \leq T$ due to $U_i \leq 1$. Furthermore, due to $r_i \leq T$, we don't generate temporal overlap between the two partial runs of task i.

Example (2 processors)

	τ_1	τ_2	τ_3
T_i	4	8	6
C_i	2	8	3

$$U = \frac{2}{4} + \frac{8}{8} + \frac{3}{6} = 2$$

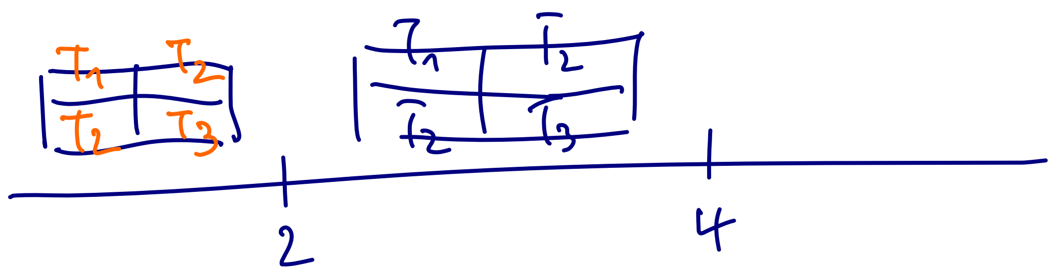
$$T = \gcd(4, 8, 6) = 2$$

In each slice,

τ_1 has $2 \cdot \frac{2}{4} = 1$ unit

τ_2 has $2 \cdot \frac{8}{8} = 2$ units

τ_3 has $2 \cdot \frac{3}{6} = 1$ unit



Scheduling idea

This scheme works if

- the load isn't too high:

$$U = \sum_{i \in M} \frac{C_i}{T_i} \leq n$$

  and

- the time slices allocated have integral length:

$$r_i = TU_i = T\frac{C_i}{T_i} \in \mathbb{N} \text{ for each } i \in M$$

Rescheduling fractional parts

- Let $X_i = T*C_i/T_i - \lfloor T*C_i/T_i \rfloor$

- In each period,
  allocate in Xi * Ti/T slices: $\lfloor T*C_i/T_i \rfloor + 1$ units
  and in all other slices: $\lfloor T*C_i/T_i \rfloor$ units

- This can be done without allowing any task to miss its deadline: use EDF!

Example (2 processors)

	τ_1	τ_2	τ_3
T_i	4	6	4
C_i	2	4	3

$$U = \frac{2}{4} + \frac{4}{6} + \frac{3}{4} = \frac{23}{12} < 2$$

$$T = \gcd(4, 6, 4) = 2$$

In each slice

τ_1 has $\lfloor 2 \cdot \frac{2}{4} \rfloor = 1$ time unit

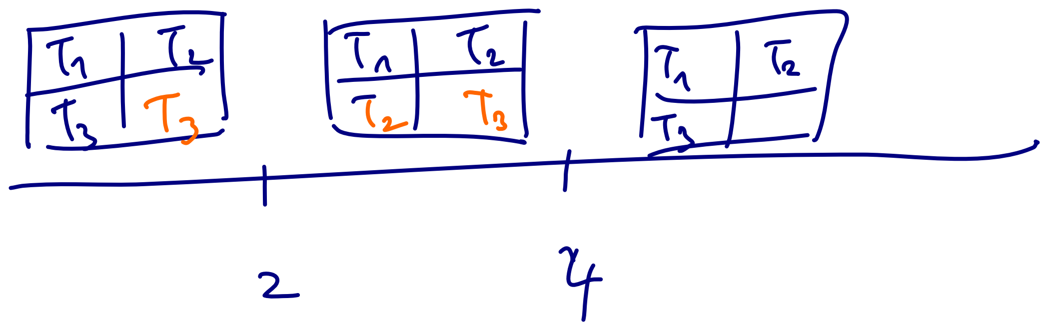
τ_2 has $\lfloor 2 \cdot \frac{4}{6} \rfloor = \lfloor \frac{8}{6} \rfloor = 1$ time unit

τ_3 has $\lfloor 2 \cdot \frac{3}{4} \rfloor = \lfloor \frac{6}{4} \rfloor = 1$ time unit

τ_2, τ_3 have fractional parts

$\Rightarrow$ τ_2 needs 1 extra unit every 3 slices

τ_3 needs 1 extra unit every 2 slices


T1
T2
T3
T3
T1
T2
T2
T3
T1
T2
T3
2
4

Extension: Task migration time

Theorem: A **necessary** and **sufficient** condition for scheduling periodic tasks on n processors is $U \leq n$, if the task migration time is one unit.

Extension: Task migration time

Lemma: If $U \leq n$, then **within each time slice** the tasks can meet the migration time requirement without missing deadlines, if the task migration time is one unit.

- Sort tasks according to non-increasing computation times
- If computation block = T → allocate a processor exclusively
- If computation block < T:
  - Allocate completely on one processor if possible; no migration
  - Allocate a part of computation at the end of $proc_i$, rest at the beginning of $proc_{i+1}$ → gap of at least 1

Extension: Task migration time

Lemma: If $U \leq n$, then **between time slices** the tasks can meet the migration time requirement without missing deadlines, if the task migration time is one unit.

- For each slice, sort tasks according to non-increasing computation blocks
- If computation block = T → find processor that executes the task at the end of the previous slice → no migration
- If no such processor exists → assign it to some left-on processor at the end (migration time already accounted for in previous slice)

- If computation block < T

  → find processor j that executed the task at the end of the previous slice

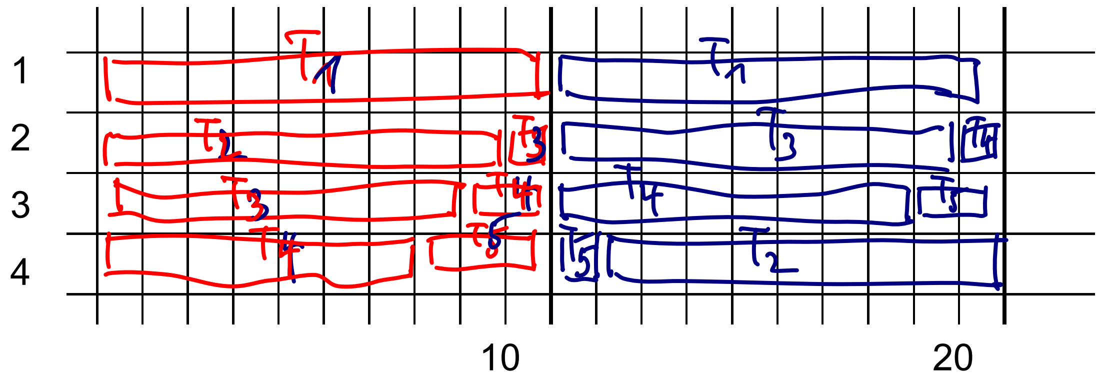
  Assign as much as possible to current processor;

  If insufficient, use j from the beginning (no migration at the beginning, >= 1 with gap within the slice)
- If no such processor exists → assign task later (migration time already accounted for in the previous slice)

Example (4 processors)

	Computation block
τ_1	10
τ_2	9
τ_3	9
τ_4	9
τ_5	3

T=10



Extension: Task migration time

Theorem: Let T=gcd(T_1, …, T_m) and let R be the task migration time. A **sufficient condition** for scheduling the m periodic tasks is that $U \leq n \cdot (T-R+1)/T$.

- Schedule as before, but only use T-R+1 units of slice
- If migration time is too short
  => shift task(s) to the right end of schedule

Example (4 processors)

i	*Computation block*
τ_1	10
τ_2	9
τ_3	9
τ_4	9
τ_5	3

T=12,
R=3