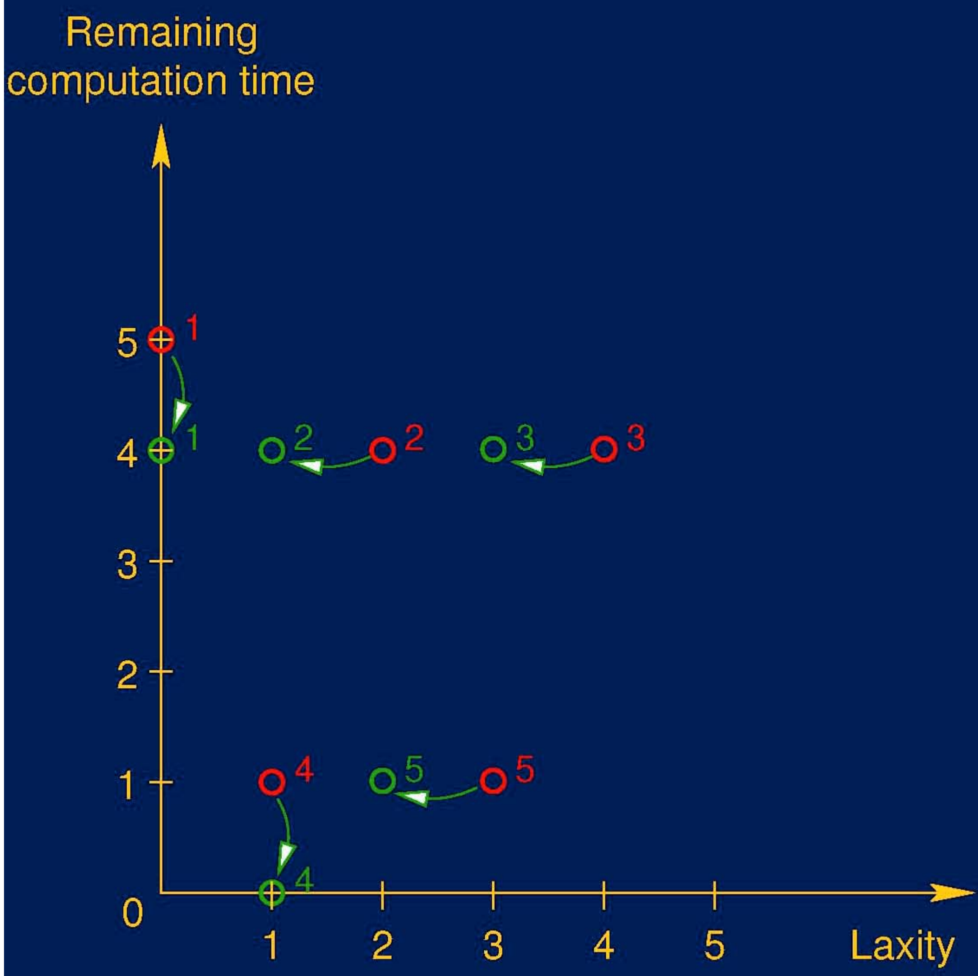
Embedded Systems 20

REVIEW: LLF (Least Laxity First)

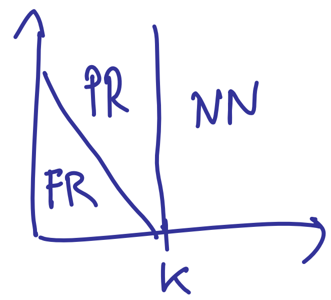


When tasks are released, they are inserted into the game board according to their WCET and laxity (= deadline – remain. comp. time).

In every time scheduling step / turn of the game:
– at most n nodes go down by 1
– the rest moves 1 to the left

LLF is optimal.

REVIEW: Schedulability



Within a set M of aperiodic tasks, we identify three classes with respect to the next k time units starting at time t:

1. Tasks that have to be fully run within the next k time units:

$$FR(t, k) = \{i \in M \mid D_i(t) \leq k\}$$

2. Tasks that have to be partially run within the next k time units:

$$PR(t, k) = \{i \in M \mid L_i(t) \leq k \wedge D_i(t) > k\}$$

3. Tasks that need not be run within the next k time units:

$$NN(t, k) = \{i \in M \mid L_i(t) > k\}$$

Surplus computing power

$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

Theorem: If all tasks are released at time 0, then SCP(0,k)≥0 for all k>0 is a **necessary and sufficient** condition for schedulability.

We show that no deadlines are missed using LLF.

Proof by induction on t:

1) $SCP(t,k) \geq 0 \quad \forall k$
$\Rightarrow$ # of token on the C-axis is $\leq n$

2) $SCP(t,k) \geq 0 \quad \forall k$
$\Rightarrow SCP(t+1,k) \geq 0 \quad \forall k.$

Ad 1) $SCP(t,1) \geq 0$

$$n \geq \overbrace{\sum_{i \in FR(t,1)} C_i(t) + \sum_{i \in PR(t,1)} (1 - L_i(t))}^{\#\text{ token on C axis}}$$

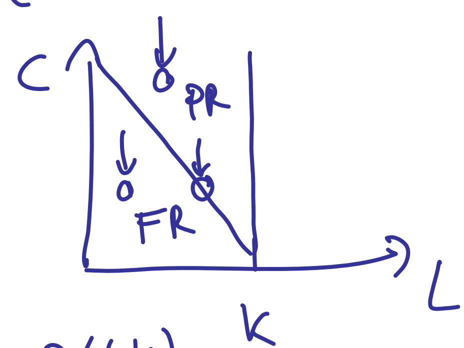
$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

Ad 2)

Claim: $\forall k \geq 0 \; \exists k'$ s.t.

$$SCP(t+1, k) \geq SCP(t, k')$$

Case 1: All token left of $L=k$ have arrived by vertical move



Pick $k' := k$: $\Delta SCP = SCP(t+1,k) - SCP(t,k)$

- Token in FR: contribute 1 to ΔSCP
- Token in PR: contributes 0 to ΔSCP
- Token that move from PR to FR, hitting $L+C=k$

$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

Contribution to ΔSCR : $-C_i(t+1) + (k - L_i(t))$

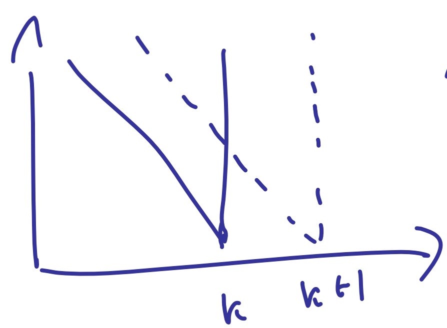
$$= -(k - L_i(t+1)) + (k - L_i(t)) = 0$$

$$= L_i(t)$$

- Tasks that moved from NN to line $L=k$:
  Contribution to ΔSCR : $-(k - L_i(t+1)) = 0$

Case 2: Some of the tasks left of $L=k$ have arrived by horizontal move

Pick $k' := k+1$



$$\Delta SCP = SCP(t+1,k) - SCP(t,k+1)$$

$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

By LLF, <u>n token</u> must have moved downward left of $L = k+1$.

Consider a token that moved downward

in PR: contribution to $\Delta SCP =$

$$- (k - L_i(t+1)) + (k+1 - L_i(t))$$

$$= 1$$

in FR: contribution to $\Delta SCP =$

$$- C_i(t+1) + C_i(t) = 1$$

$$SCP(t,k) = \underbrace{kn}_{\text{avail. computing power}} - \underbrace{\sum_{i \in FR(t,k)} C_i(t)}_{\text{needed for full runs}} - \underbrace{\sum_{i \in PR(t,k)} (k - L_i(t))}_{\text{needed for partial runs}}$$

Consider a token that moved horizontally:

LLF chooses token in FR first

$\Rightarrow$ token in PR:

Contribution to ΔSCP: $-(k - L_i(t+1)) + (k+1 - L_i(t))$

$L_i(t+1) = L_i(t)+1$

$= 0$

$\Delta SCP = k \cdot n - (k+1) \cdot n + n$

$= 0$

Periodic tasks

Theorem: A necessary and sufficient condition for the schedulability of periodic tasks is that $U \leq n$.

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 $r_i \le T$ due to $U_i \le 1$. Furthermore, due to $r_i \le T$, 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

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

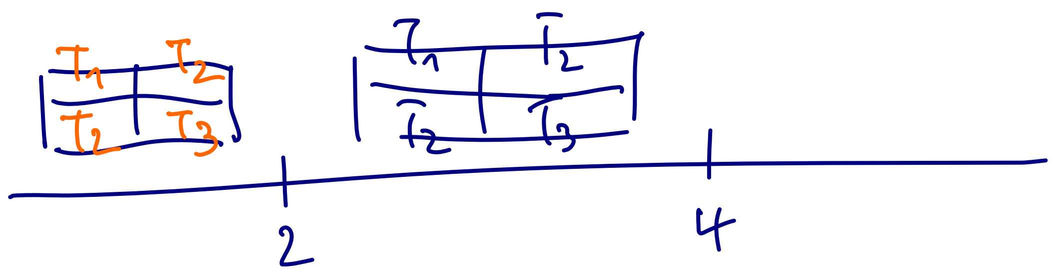
$$T = \gcd(4, 8, 6) = 2$$

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 $\left\lfloor 2 \cdot \frac{2}{4} \right\rfloor = 1$ time unit

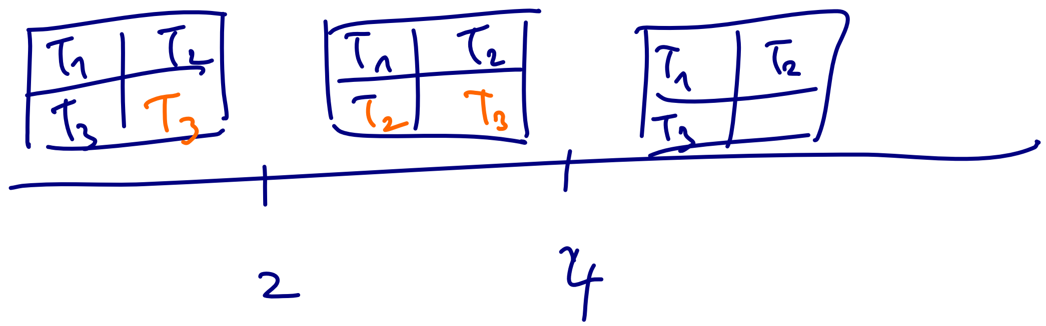
τ_2 has $\left\lfloor 2 \cdot \frac{4}{6} \right\rfloor = \left\lfloor \frac{8}{6} \right\rfloor = 1$ time unit

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