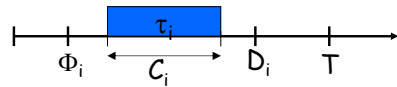


CMPE 117

Lectures:
Periodic Task Scheduling

© Luca de Alfaro, 2002-04

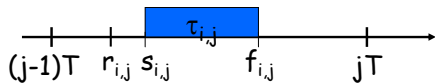
Periodic Task Scheduling



Terminology:

- τ_i : i-th task
- T_i : period of i-th task
- D_i : relative deadline of i-th task
- Φ_i : phase (relative start time) of i-th task
- C_i : computation time of i-th task

Periodic Task Scheduling



Terminology:

- $\tau_{i,j}$: instance j of task i
- $r_{i,j}$: release time of instance j of task i
- $s_{i,j}$: start time of instance j of task i
- $f_{i,j}$: finish time of instance j of task i

For example, $r_{i,j} = (j-1)T_i + \Phi_i$

Quality of schedule

- Response time $R_{i,j} = f_{i,j} - r_{i,j}$
- Critical instant: the time at which, if a task is released, it will have the largest response time.
- Absolute jitter:
 - Start: $\max_j (s_{i,j} - r_{i,j}) - \min_j (s_{i,j} - r_{i,j})$
 - Finish: $\max_j (f_{i,j} - r_{i,j}) - \min_j (f_{i,j} - r_{i,j})$

Assumptions

- A1: The tasks are periodic, period = T_i
 - A2: All instances have the same worst-case execution time C_i
 - A3: All instances have the same relative deadline $D_i = T_i$
 - A4: All tasks in the set Γ of tasks are independent
- (We will relax these conditions later)

More assumptions

- The tasks do not suspend themselves on I/O
- All overheads in the kernel are assumed to be 0.

Processor Utilization

- Processor utilization
$$U = \sum_{i=1}^n \frac{C_i}{T_i}$$
- For a set of periods Θ and an algorithm A , let $U_{ub}(\Theta, A)$ be the upper bound of the processor utilization factor for tasks with Θ to be schedulable by A .
- Let $U_{lub} = \min_{\Theta} U_{ub}(\Theta, A)$
- If $U < U_{lub}$, we know that the tasks are schedulable!

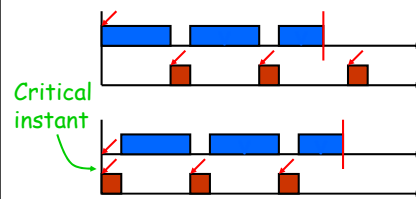
Task deadlines equal to the period

Rate Monotonic Scheduling

- Assign priority to process i proportional to the rate $1/T_i$
- Summary:

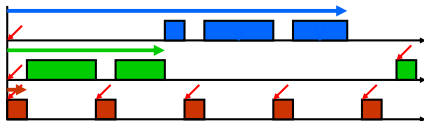
- Optimal with respect to all fixed-priority algorithms
- n tasks are guaranteed schedulable if
$$U \leq n(2^{1/n} - 1).$$
- For $n \rightarrow \infty$, $n(2^{1/n} - 1) \rightarrow \ln 2 \approx 0.69$

Critical Instant for RM



The critical instant occurs when all the tasks are released simultaneously.

Worst-Case Response Time for RM



The worst-case response time for a job occurs when it is released simultaneously with all higher-priority (shorter-period) jobs.

The worst-case response time can be computed either from a schedule (as above), or with an equation we will present later.

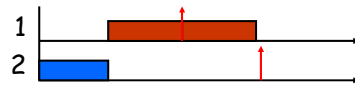
Schedulability test for RM

- **Approximate:** n tasks are guaranteed schedulable if
$$U \leq n(2^{1/n} - 1).$$
- **Precise:** n tasks are guaranteed schedulable iff, for all $1 \leq i \leq n$, we have $R_i \leq T_i$, where R_i is the worst-case response time.

RM is optimal among fixed priority schedulers

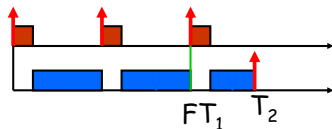
- We study the case of two processes, with $T_1 < T_2$.
- RM assigns $P_1 > P_2$.
- We will show this is better than $P_2 > P_1$.
- Clearly, if a generic algorithm uses a different order than RM, there must be at least two processes that are assigned priorities in reverse order wrt. RM.

Priority $P_2 > P_1$: (\neq RM)



- Maximum utilization: $C_1 + C_2 \leq T_1$

Priority $P_2 < P_1$: (= RM)



- Case 1: $F = \lfloor T_2 / T_1 \rfloor$, $C_1 < T_2 - F T_1$
- Schedulable if $(F+1)C_1 + C_2 \leq T_2$

Analysis

- By non-RM: $C_1 + C_2 \leq T_1$ (1)
- By RM: $(F+1)C_1 + C_2 \leq T_2$ (2)
- We show (1) implies (2): rewriting (1),

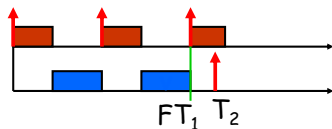
$$FC_1 + FC_2 \leq F T_1$$

$$FC_1 + C_2 \leq F T_1$$

$$(F+1)C_1 + C_2 \leq F T_1 + C_1$$

$$(F+1)C_1 + C_2 \leq F T_1 + C_1$$

Priority $P_2 < P_1$: (= RM)



- Case 2: $F = \lfloor T_2 / T_1 \rfloor$, $C_1 \geq T_2 - F T_1$
- Schedulable if $FC_1 + C_2 \leq F T_1$

Analysis

- By non-RM: $C_1 + C_2 \leq T_1$ (1)
- By RM: $FC_1 + C_2 \leq F T_1$ (3)
- We show (1) implies (3): rewriting (1),

$$FC_1 + FC_2 \leq F T_1$$

$$FC_1 + C_2 \leq FC_1 + FC_2 \leq F T_1$$

Least upper bound for RM

- n processes sharing CPU: $n(2^{1/n} - 1)$
- When $n \rightarrow \infty$: $\ln 2 \approx 0.69$
- Calculus check:

$$\begin{aligned} & n(2^{1/n} - 1) \\ &= n(e^{(\ln 2)/n} - 1) \\ &\approx n(1 + (\ln 2)/n - 1) \\ &\approx \ln 2 \text{ as } n \rightarrow \infty \end{aligned}$$

Earliest Deadline First

Theorem: A set $\{\tau_1, \dots, \tau_n\}$ of periodic tasks is schedulable with EDF iff

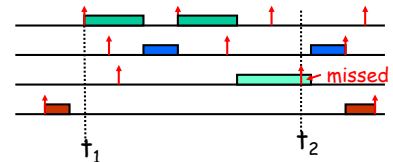
$$\sum_{i=1}^n \frac{C_i}{T_i} \leq 1.$$

Proof: only if: obvious.

Proof: if $\sum_{i=1}^n C_i/T_i \leq 1$ then schedulable

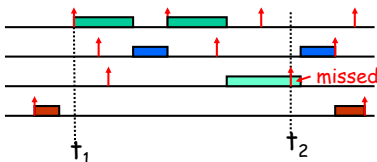
- We prove instead:
If not schedulable, then $\sum_{i=1}^n C_i/T_i > 1$

Proof: If not schedulable, then $\sum_{i=1}^n C_i/T_i > 1$



- Let t_2 be when the missed deadline occurs, and let $[t_1, t_2]$ be the longest interval of continuous utilization before t_2 , such that only instances with deadlines less than t_2 are executed in $[t_1, t_2]$.

Proof: If not schedulable, then $\sum_{i=1}^n C_i/T_i > 1$



$$\hat{C}_p(t_1, t_2) = \sum_{i=1}^n \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^n \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U.$$

- $t_2 - t_1 < C_p(t_1, t_2) \leq \hat{C}_p(t_1, t_2) \leq (t_2 - t_1)U$.
- $C_p(t_1, t_2)$: computation time requested between t_1 and t_2 with deadline $< t_2$.
- \hat{C}_p : upper bound for C_p .

Schedulability test for EDF

- **Precise**: n tasks are guaranteed schedulable iff $U \leq 1$.

Note: we cannot possibly do better than EDF! So why is RM used? Because it's simpler: we don't need to tell the processor the deadlines. It relies on simpler processor scheduling mechanisms.

Task deadlines shorter than the period

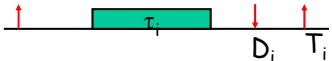
Periodic tasks with deadlines shorter than the period

Two approaches:

- Deadline monotonic (DM), derived from rate monotonic.
- EDF.

As usual, EDF is optimal.

Deadline Monotonic (DM)



- D_i : deadline of task i , relative to period: $D_i < T_i$
- Deadline monotonic algorithm: assign highest priority to the process with the smallest relative deadline D_i .

DM: Schedulability analysis

- Naive approach: reduce the problem to RM, and check

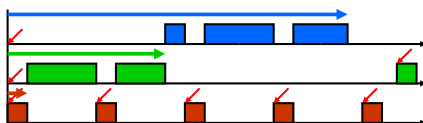
$$\sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

Note: used to be T_i

But we can do better!

DM: Schedulability Analysis

Schedulable iff $R_i \leq D_i$ for all tasks J_i , where R_i is the worst-case response time.



DM: Schedulability Analysis

How can we compute the worst-case response time?

Method 1: by drawing a schedule when all processes are released simultaneously.

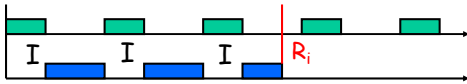


Computing the Response Time

We can write the worst-case response time as

$$R_i = C_i + I_i$$

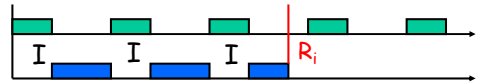
where R_i is the response time, and I_i is the interference to task i due to tasks having priorities higher than that of i .



Computing the response time

$$R_i = C_i + I_i \quad I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$



Computing the response time

$$R_i = C_i + I_i \quad I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

This equation in R_i has many fixpoints; we want the *least* fixpoint.

DM: Schedulability test

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- We want the least fixpoint.
- Think at it as $R_i = F(R_i)$.
- Note that F is monotonic and left-continuous.
- Solution:

$$R_i = \mu \times F(x) = \lim_{n \rightarrow \infty} F^n(0)$$

(compute by iteration)

DM: Schedulability test

- For all i , compute the response time $R_i = \mu \times F(x)$ by iteration.
- Check that $R_i \leq D_i$.

Note: when computing the k -th estimate R_i^k of R_i , we have that either $R_i^k = R_i^{k-1}$, or that $R_i^k - R_i^{k-1} \geq C_{\min}$, where $C_{\min} = \min_{1 \leq j \leq n} C_j$.

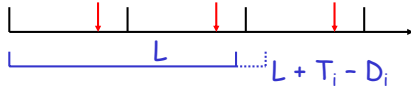
EDF, for deadlines \leq periods

- When the deadline is shorter than the period, we can also use EDF.
- As usual, EDF turns out to be better than DM (similarly to EDF vs. RM when the deadlines are equal to the periods). The drawback of EDF is that it needs priorities to be changed dynamically.

EDF, for deadlines \leq periods

- $C_p(0,L)$: the amount of processing that is requested in $[0,L]$, and that must be completed (has deadline) before L .

$$C_p(0, L) = \sum_{i=1}^n \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i$$



EDF Schedulability

Theorem: A set of periodic tasks is schedulable by EDF iff, for all $L \geq 0$:

$$L \geq C_p(0,L)$$

Proof: the theorem is proved by showing that the above is equivalent to the formulation based on utilization we showed before.

EDF schedulability test

- Note that we need to check that $L \geq C_p(0,L)$ only for values of L that:
 - correspond to some deadline (since that's where C_p may increase)
 - are smaller than the hyperperiod

$$H = \prod_{i=1}^n T_i$$
 - are no larger than the busy period (additional optimization, see Buttazzo)

EDF Schedulability: Example

- $J_1 : (C_1=2, T_1=6, D_1=5)$
- $J_2 : (C_2=2, T_2=8, D_2=4)$
- $J_3 : (C_3=4, T_3=12, D_3=8)$
- Hyperperiod: $H = \text{mcm}(6,8,12) = 24$.
- Deadlines $\leq H$:
 - 4 (J_2), 5 (J_1), 8 (J_3), 11 (J_1), 12 (J_2), 17 (J_1), 20 (J_2 and J_3), 23 (J_1).

EDF Schedulability: Example

- $C_p(0,4) = C_2 = 2 \leq 2$
- $C_p(0,5) = C_p(0,4) + C_1 = 4 \leq 5$
- $C_p(0,8) = C_p(0,5) + C_3 = 8 \leq 8$
- $C_p(0,11) = C_p(0,8) + C_1 = 10 \leq 11$
- $C_p(0,12) = C_p(0,11) + C_2 = 12 \leq 12$
- $C_p(0,17) = C_p(0,12) + C_1 = 14 \leq 17$
- $C_p(0,20) = C_p(0,17) + C_2 + C_3 = 20 \leq 20$
- $C_p(0,23) = C_p(0,20) + C_1 = 22 \leq 23$
- **Schedulable!**