Real-Time Systems

C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM, 1973

Contents

- Background and Assumptions
- Terminologies: Revisited
- Rate-monotonic Priority Assignment
- Least Upper Bound (LUB) of Processor Utilization
- Some Results on Rate Monotonic Algorithm

Assumptions

- (A1) The requests for all tasks for which hard deadlines exist are periodic.
  - *Only for periodic task set*
- (A2) Deadlines consist of run-ability constraints only.
  - *Each task must be completed before its next invocation.*
- (A3) The tasks are independent to each other.
  - *No precedence relation*
- (A4) Run-time for each task is constant for that task and does not vary with time.
  - *Assumes worst-case execution time known a priori.*
- (A5) Any non-periodic tasks in the system are special.
  - *Soft deadline; background jobs*

Rate Monotonic RT Scheduling

- Basic rate monotonic scheduling (RMS)
  - Rate monotonic priority
    - The higher rate, the higher priority
  - Schedulability guaranteed if utilization rate is below a certain limit
  - For feasible schedules
    - $f_i = 1/T_i$: frequency (=rate)
    - C_i : execution time

$$\sum_{i=1}^{n} c_i f_i \leq 1$$

Rate Monotonic RT Scheduling

- If the total utilization rate has least upper bound (LUB) of $n(2^{1/n} - 1)$ where n = #tasks, there exists a feasible rate monotonic schedule. That is,

$$\sum_{i=1}^{n} c_i f_i \leq n(2^{\frac{1}{n}} - 1) = U(n)$$

  - $U(1) = 1.0$, $U(2) = 0.828$, $U(3) = 0.779$, $U(\infty) = \ln 2 = 0.693$
  - Only sufficient condition
- Priority inversion problem

Properties of RM Scheduling

- On-line
- Preemptive
- Priority-driven
- Static (Fixed-priority)

Terminologies: Revisted

- Task set $\tau = \{ \tau_1, \tau_2, \ldots, \tau_n \}$
  - A task is characterized by (T_i, C_i)
- Deadline of a request
  - The time of the next request for the same task
- An *overflow* occurs at time t if t is the deadline of an unfulfilled request
- For a given set of tasks, a scheduling algorithm is *feasible* if the tasks are scheduled so that no overflow ever occurs.
- Response time of a request for a certain task
  - The time span between the request and the end of the response to that request

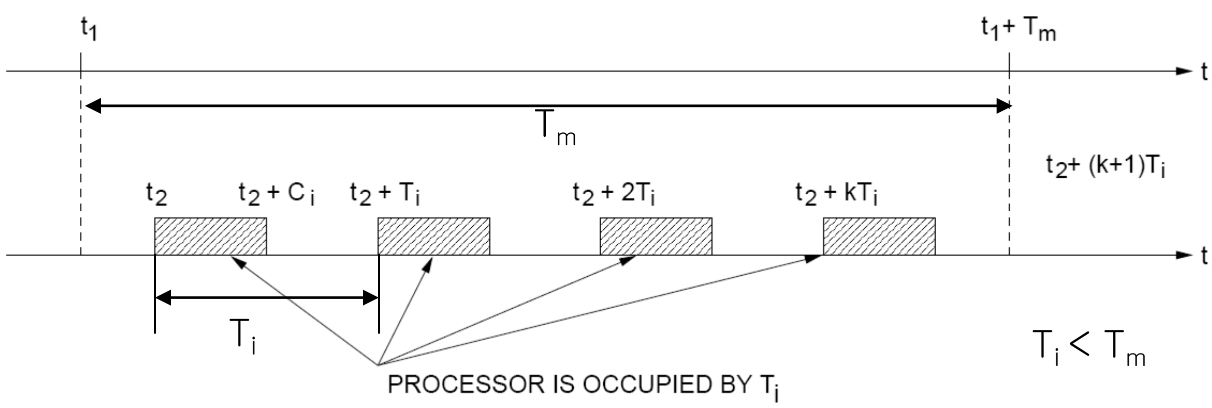
Critical Instant

Definition

A *critical instant* for a task is an instant at which a request for that task will have the largest response time.

Theorem 1.

A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.



Priority Assignment

- From the proof of Theorem 1,
  - A simple direct calculation can determine whether or not a given priority assignment will yield a feasible scheduling algorithm.
  - If the request for all tasks at their critical instants are fulfilled before their deadlines, then the scheduling algorithm is feasible.

Priority Assignment (cont'd)

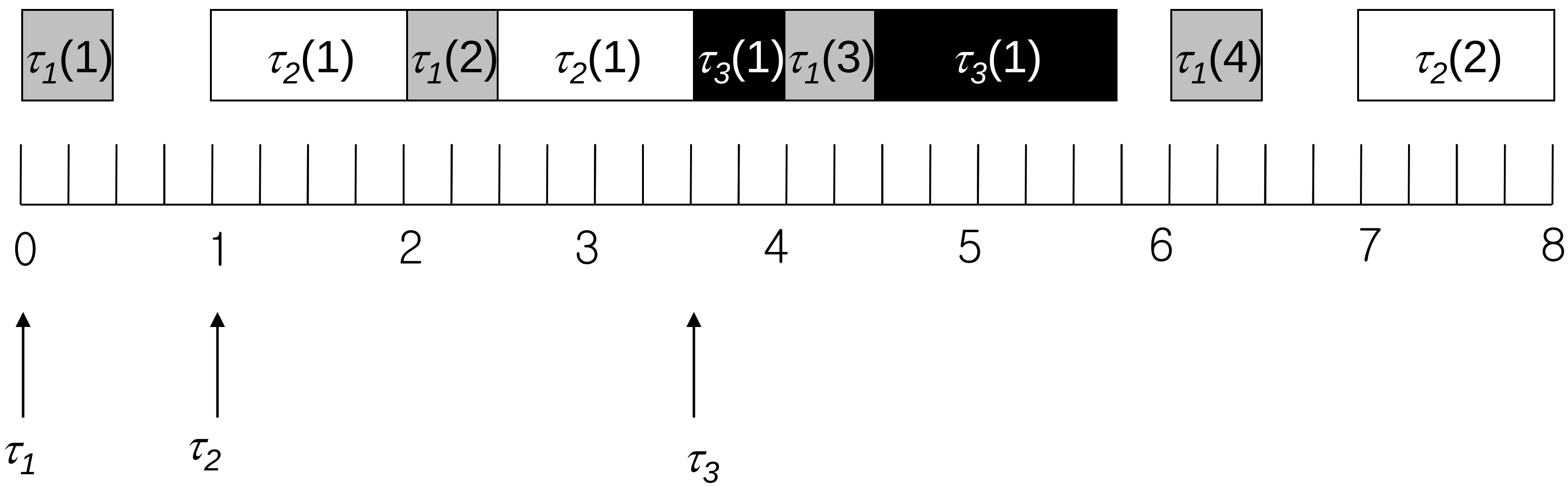
- "Reasonable" rule of priority assignment
  - Assign priorities to tasks according to their request rates, independent of their execution times.
- Rate monotonic (RM) priority assignment
  - Tasks with higher request rates (shorter period) will have higher priorities.

Theorem 2. Optimality of RM Priority Assignment

If a feasible priority assignment exists for some task set, the rate-monotonic priority is feasible for that task set.

Scheduling Example

- Three tasks τ_1=(2,0.5), τ_2=(6,2), τ_3=(10,1.75)



LUB of Processor Utilization

- Utilization factor U
  - The fraction of processor time spent in the execution of the task set (= 1-(the fraction of idle processor time))
- Full utilization of processor
  - A set of tasks is said to "*fully utilize*" the processor if the priority assignment is feasible for the set and if an increase in the run-time of any of the tasks in the set will make the priority assignment infeasible
- LUB (least upper bound) of utilization factor
  - The minimum of utilization factor over all sets of tasks that fully utilize the processor
  - For all task sets whose utilization factor is below this bound, there exists a fixed priority assignment which is feasible

A schedule is said to be ***feasible*** if all tasks can be completed according to the set of specified constraints
A set of tasks is said to be ***schedulable*** if there exists at least one algorithm that can produce a feasible schedule

Assumptions of LUB

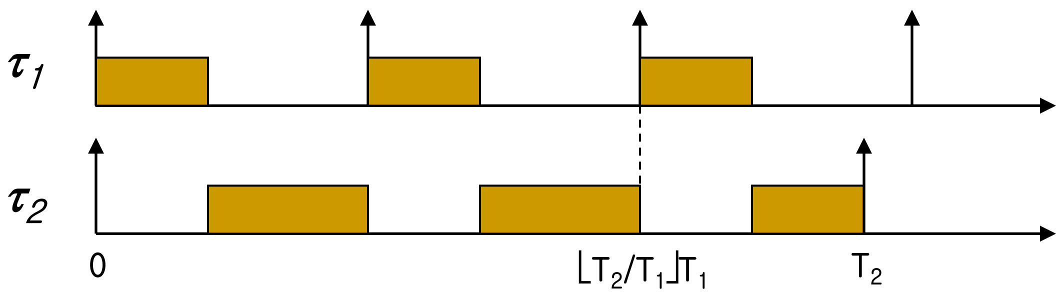
- The processor always executes the highest-priority task.
- Task priorities are assigned according to rate monotonic policy.
- Tasks do not synchronize with each other.
- Each task’s deadline is at the end of its period.
- Tasks do not suspend themselves in the middle of computations.
- No interrupts, No context switching overhead

LUB of Processor Utilization (cont'd)

Theorem 3. LUB for a set of two tasks

For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is $U=2(2^{1/2}-1)$.

Proof: (a) When $C_1 < T_2 - \lfloor T_2/T_1 \rfloor T_1$



$$C_2 = T_2 - C_1 \left\lceil \frac{T_2}{T_1} \right\rceil$$

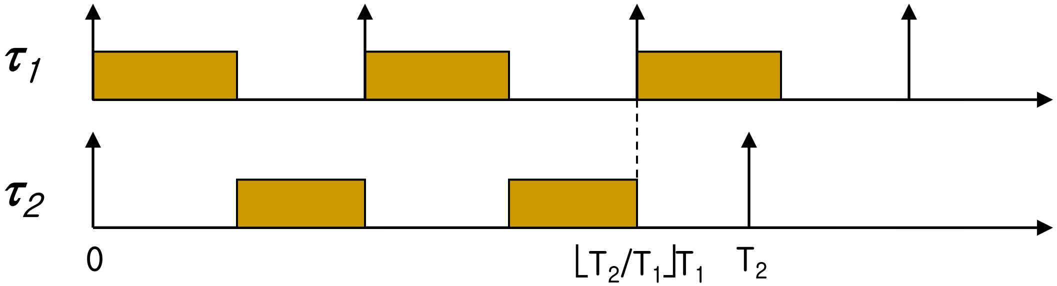
$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + 1 - \frac{C_1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil$$

Since $$C_1 \left(\frac{1}{T_1} - \frac{1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil \right) \le 0$$

The processor utilization is monotonically decreasing in C_1

LUB of Processor Utilization (cont'd)

(b) When $C_1 \geq T_2 - \lfloor T_2/T_1 \rfloor T_1$



$$C_2 = (T_1 - C_1)\left\lfloor \frac{T_2}{T_1} \right\rfloor$$

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2}$$

$$= \frac{C_1}{T_1} + \left\lfloor \frac{T_2}{T_1} \right\rfloor \left(\frac{T_1}{T_2} - \frac{C_1}{T_2} \right)$$

$$= \frac{T_1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor + C_1\left(\frac{1}{T_1} - \frac{1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor \right)$$

Since $\frac{1}{T_1} - \frac{1}{T_2}\left\lfloor \frac{T_2}{T_1} \right\rfloor \geq 0$

The processor utilization is monotonically increasing in C_1

LUB of Processor Utilization (cont'd)

Proof (cont'd):

In case (a), U is monotonically decreasing in C_1.
In case (b), U is monotonically increasing in C_1.
$\therefore$ The minimum value of U occurs at $C_1 = T_2 - T_1 \lfloor T_2/T_1 \rfloor$.

Let $I = \lfloor T_2/T_1 \rfloor$ and $T_2/T_1 = I + f\ (0 \le f < 1)$ $\ll\gg$ $\left\lceil \frac{T_2}{T_1} \right\rceil = \begin{cases} I & \text{if } f = 0 \\ I + 1 & \text{otherwise} \end{cases}$

$$U = \frac{C_1}{T_1} + 1 - \frac{C_1}{T_2}\left\lceil \frac{T_2}{T_1} \right\rceil, \text{ where } C_1 = T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

$$U = \begin{cases} 1 & \text{if } f = 0 \\ \dfrac{I + f^2}{I + f} & \text{otherwise} \end{cases}$$

LUB of Processor Utilization (cont'd)

Proof (cont'd):

- Minimum U ?

$$U = \frac{1+f^2}{1+f}\ ,\ \text{where } I = 1$$

$$\frac{dU}{df} = \frac{f^2+2f-1}{(1+f)^2} = 0 \Rightarrow \begin{array}{l} \cancel{f_1 = -1-\sqrt{2}} \\ f_2 = -1+\sqrt{2} \end{array} \Rightarrow U = 2(\sqrt{2}-1)\ ,\ \text{where } f = -1+\sqrt{2}$$

$\boxed{\text{About 83 \%}}$

LUB of Processor Utilization (cont'd)

Theorem 4.

For a set of n *tasks with fixed priority assignment, and the restriction that the ratio between any two request periods is less than* 2*, the least upper bound to the processor utilization factor is* $U=n(2^{1/n}-1)$.

Proof: From the restriction and the proof of theorem 3, we intend to show:

$$C_i = T_{i+1} - T_{i,} \text{ and } C_n = 2T_1 - T_n \quad (T_n < 2T_1)$$

(1) When $C_1 > T_2 - T_1$

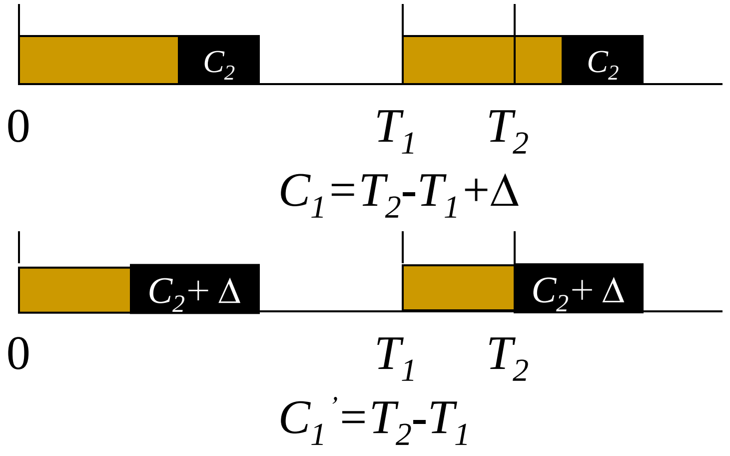
Suppose that $C_1 = T_2 - T_1 + \Delta,\ \Delta > 0.$

Let $C_1^{'} = T_2 - T_1, C_2^{'} = C_2 + \Delta$, and $C_i^{'} = C_i$ where $3 \le i \le n$

$C_1^{'}, C_2^{'}, \ldots, C_n^{'}$ also fully utilize the processor.

Let $U^{'}$ denote the corresponding utilization factor,

$$U - U^{'} = \frac{\Delta}{T_1} - \frac{\Delta}{T_2} > 0$$



LUB of Processor Utilization (cont'd)

Proof (Cont'd):

(b) When $C_1 < T_2 - T_1$

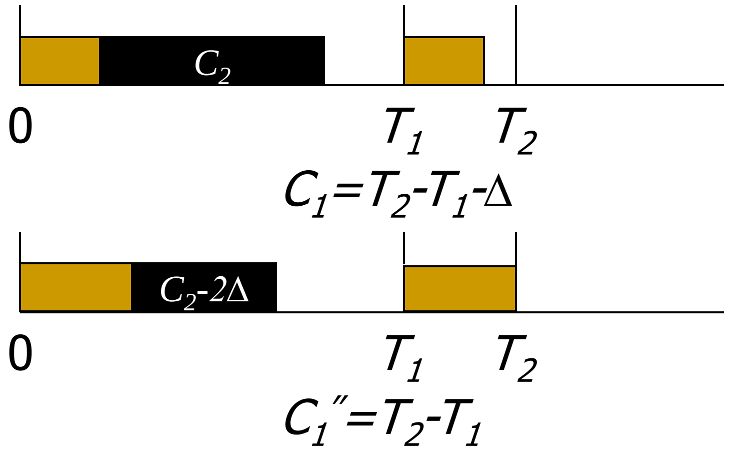
Suppose that $C_1 = T_2 - T_1 - \Delta,\ \Delta > 0.$

Let $C_1'' = T_2 - T_1, C_2'' = C_2 - 2\Delta,$ and $C_i'' = C_i$ where $3 \le i \le n$

$C_1'', C_2'', \ldots, C_n''$ also fully utilize the processor.

Let U'' denote the corresponding utilization factor,

$$U\text{-}U'' = -\frac{\Delta}{T_1} + \frac{2\Delta}{T_2} > 0$$

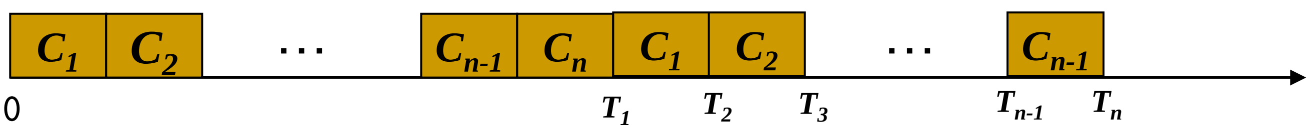


LUB of Processor Utilization (cont'd)

Proof (Cont'd):

From (1) and (2), we get $C_1 = T_2 - T_1$ if U is the minimum utilization factor.

Similarly, we get
$$\begin{cases} C_2 = T_3 - T_2 \\ \qquad \dots \\ C_{n-1} = T_n - T_{n-1} \\ C_n = T_n - 2\underbrace{(C_1 + \dots + C_{n-1})}_{T_n - T_1} = 2T_1 - T_n \end{cases}$$



$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{T_2 - T_1}{T_1} + \dots + \frac{T_n - T_{n-1}}{T_{n-1}} + \frac{2T_1 - T_n}{T_n}$$

Let $R_i = \dfrac{T_{i+1}}{T_i}$,

$$U = \sum_{i=1}^{n-1} R_i + \frac{2}{R_1 R_2 \dots R_{n-1}} - n \quad (\because\ R_1 R_2 \dots R_{n-1} = \frac{T_n}{T_1})$$

LUB of Processor Utilization (cont'd)

Proof (Cont'd):

For $U = \sum_{i=1}^{n-1} R_i + \frac{2}{R_1 R_2 \ldots R_{n-1}} - n$, to minimize U over $R_k, k = 1, \ldots, n-1$

we have: $$\frac{\partial U}{\partial R_k} = 1 - \frac{2}{R_k R_1 \ldots R_{n-1}} = 0$$

Defining $P = R_1 R_2 \ldots R_{n-1}$, $R_k P = 2$ for $\forall k, 1 \le k \le n-1$

$$\therefore R_1 = R_2 = \ldots = R_{n-1} = 2^{1/n}$$

It follows that,

$$U = (n-1)2^{1/n} + \frac{2}{2^{(1-1/n)}} - n = n(2^{1/n} - 1)$$

- For large n, $U \approx \ln 2 = 0.693$

LUB of Processor Utilization (cont'd)

Theorem 5. LUB for a set of *n* tasks
For a set of n *tasks with fixed priority assignment, the least upper bound to the processor utilization factor is* $U=n(2^{1/n}-1)$.

Proof: For details, refer the paper.

If a set of tasks fully utilizes the processor and for some i, $i < m$

$T_m/T_i \geq 2$

then, construct another set of tasks that will fully utilize the processor,

$T_m/T_i < 2$

➔ Show the utilization of the new task that is less than the original one.

➔ Hence, we need only consider tasks sets in which the ratio between any two periods is less than 2.

Sample Problem

- Sample Problem: Applying UB Test

	C	T	U
Task 1	20	100	0.200
Task 2	40	150	0.267
Task 3	100	350	0.286

U(1) = 1.0, U(2) = 0.828, U(3) = 0.779, U(∞) = *ln*2=0.693
only sufficient condition

Total utilization:

0.200 + 0.267 + 0.286 = 0.753

< U(3) = 0.779

The periodic tasks in the sample problem are schedulable according to the UB test.

Exercise

- Exercise: Applying the UB Test Given:

Task	C	T	U
τ_1	1	4	
τ_2	2	6	
τ_3	1	10	

a. What is utilization for each task?

b. Is the task set schedulable?

c. Draw the timeline.

d. What is the total utilization if $C_3=2$?

Limitations

- Feasibility test based on utilization bound is sufficient to guarantee the feasibility of any task set, but it is not necessary.
  - UB test has three possible outcomes.
    - $0 \le U \le LUB \rightarrow$ success
    - $LUB < U \le 1.00 \rightarrow$ inconclusive
    - $1.00 < U \rightarrow$ overloaded
  - UB test is conservative
  - A more precise test can be applied.
  - E.g) Harmonic task set (U=1)

Completion Time Test

Theorem 6. Completion Time Test

For a set of independent, periodic tasks, if each task meets its first deadline, with worst-case task phasing, the deadline will always be met.

- ➔ Completion Time Test
  - ➔ Let W_i = completion time of task i. W_i may be computed by the following iterative formula:

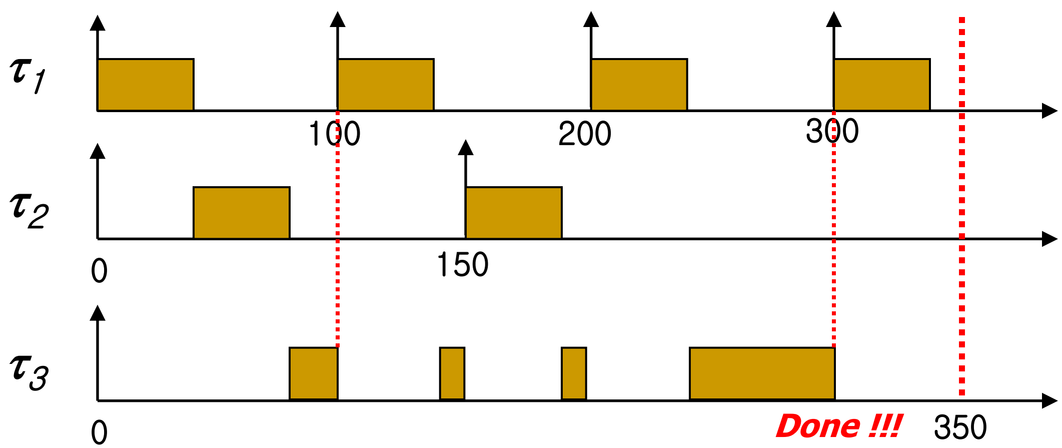
$$W_i(n+1) = C_i + \sum_{j<i} \left\lceil \frac{W_i(n)}{T_j} \right\rceil C_j \quad where\ W_i(0) = 0$$

  - ➔ Task i is schedulable if its completion time is before its deadline. That is, $W_i \le T_i$

Example

- Task τ_1: $C_1=20$, $T_1=100$, $U_1=0.2$
- Task τ_2: $C_2=40$, $T_2=150$, $U_2=0.267$
- Task τ_3: $C_3=100$, $T_3=350$, $U_3=0.286$

- Total utilization: $0.753 \le U_{lub}(3) = 3(2^{1/3}-1) = 0.779$
  - 24.7% of the CPU is usable for lower priority background computation.
- Suppose that C_1 is increased to 40,
  - The total utilization is increased to $0.953 > U_{lub}(3)$.



Example

- Applying CT Test
  - Taking the sample problem, we increase the compute time of T_1 from 20 to 40. Is the task set still schedulable?
  - Utilization of first two tasks: $0.667 < U(2) = 0.828$
    - first two tasks are schedulable by utilization bound test
  - Utilization of all three tasks: $0.953 > U(3) = 0.779$
    - utilization bound test is inconclusive
    - need to apply completion time test

Example

- Use CT test to determine if task 3 meets its first deadline

$$W_3(1) = C_3 + \sum_{j<3} \left\lceil \frac{0}{T_j} \right\rceil C_j = C_3 = 100$$

$$W_3(2) = C_3 + \sum_{j<3} \left\lceil \frac{100}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{100}{100} \right\rceil (20) + \left\lceil \frac{100}{150} \right\rceil (40) = 160$$

$$W_3(3) = 100 + \left\lceil \frac{160}{100} \right\rceil (20) + \left\lceil \frac{180}{150} \right\rceil (40) = 220$$

Example

$$W_3(3) = 220$$

$$W_3(4) = 100 + \left\lceil \frac{220}{100} \right\rceil (20) + \left\lceil \frac{260}{150} \right\rceil (40) = 240$$

$$W_3(4) = 100 + \left\lceil \frac{240}{100} \right\rceil (20) + \left\lceil \frac{300}{150} \right\rceil (40) = 240 \Rightarrow Done!$$

$$W_3 = 240 < T_3 = 350$$

Task 3 is schedulable using CT test.

Example

- Exercise: Applying CT Test

Task τ_1: $C_1 = 1$ $T_1 = 4$
Task τ_2: $C_2 = 1$ $T_2 = 4$
Task τ_3: $C_3 = 1$ $T_3 = 4$

a. Apply UB test.
b. Draw timeline.
c. Apply CT test.

Summary

- Summary
  - Utilization bound test is simple but conservative
  - Completion time test is more exact but also more complicated
  - To this point, UB and CT tests share the same limitations.
    - all tasks run on a single processor
    - all tasks periodic and noninteracting
    - deadlines always at the end of the period
    - no interrupts
    - rate monotonic priorities assigned
    - zero context switch overhead
    - tasks do not suspend themselves

Real-Time Systems

J. Lehoczky et al, "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior", In *Proceedings of the IEEE Real-Time Systems Symposium*, 1989

Motivation

- Utilization bound feasibility test is simple but
  - The feasibility condition is sufficient but not necessary (pessimistic).
  - The average case behavior is substantially better than the worst case behavior.

Problem Formulation

- Task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$
  - A task is characterized by (T_i, C_i, I_i)
    - T_i: period, C_i: computation time, I_i: Phasing (offset)
    - Each task instance is initiated (released) at times $I_i + kT_i$, $k \geq 0$.
    - The deadline of $(k+1)^{st}$ instance is $I_i + (k+1)T_i$.
- Basic idea
  - For a set of independent, periodic tasks, if each task meets its first deadline, with the worst-case task phasings, the deadline will always be met.
    - Worst-case task phasing = critical instant ($I_i = 0$ for $1 \leq i \leq n$)

Necessary and Sufficient conditions

- Assume that the task phrasings are all zero (i.e., the first iteration of each task is released at time zero)
- $T_1 < T_2 < T_3 < \ldots.\ T_i$
- τ_1 can be feasibly scheduled when $C_1 \leq T_1$
- If we can find some t in $[0, T_2]$ that satisfy the following condition

$$t \geq \left\lceil \frac{t}{T_1} \right\rceil C_1 + C_2$$

  τ_2 can be feasibly scheduled.
- Generally, τ_n can be feasibly scheduled using RM iff we can find some t in $[0, T_n]$ that satisfy the following condition

$$t \geq \sum_{j=1}^{n} C_j \left\lceil \frac{t}{T_j} \right\rceil$$

Problem Formulation

$$\text{Defining} \quad W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil$$

$$L_i(t) = \frac{W_i(t)}{t}, L_i = \min_{\{0 \le t \le T_i\}} L_i(t),\; L = \max_{\{1 \le i \le n\}} L_i$$

- Theorem 1.
  - A task τ_i is schedulable using RM if and only if $L_i \le 1$.
  - The entire task set is schedulable using RM if and only if $L \le 1$.

→ Should we check the condition at every t ?

  - The right-hand side of the above equation has jumps only at multiple of T_j
  - Practically, we only need to compute $W_i(t)$ at the arrival times of tasks with higher priority than τ_i before the deadline of τ_i and the deadline of τ_j .

Problem Formulation

- *$W_i(t)$* is constant, except at a finite number of points when tasks are released. We only need to compute *$W_i(t)$* at the scheduling points

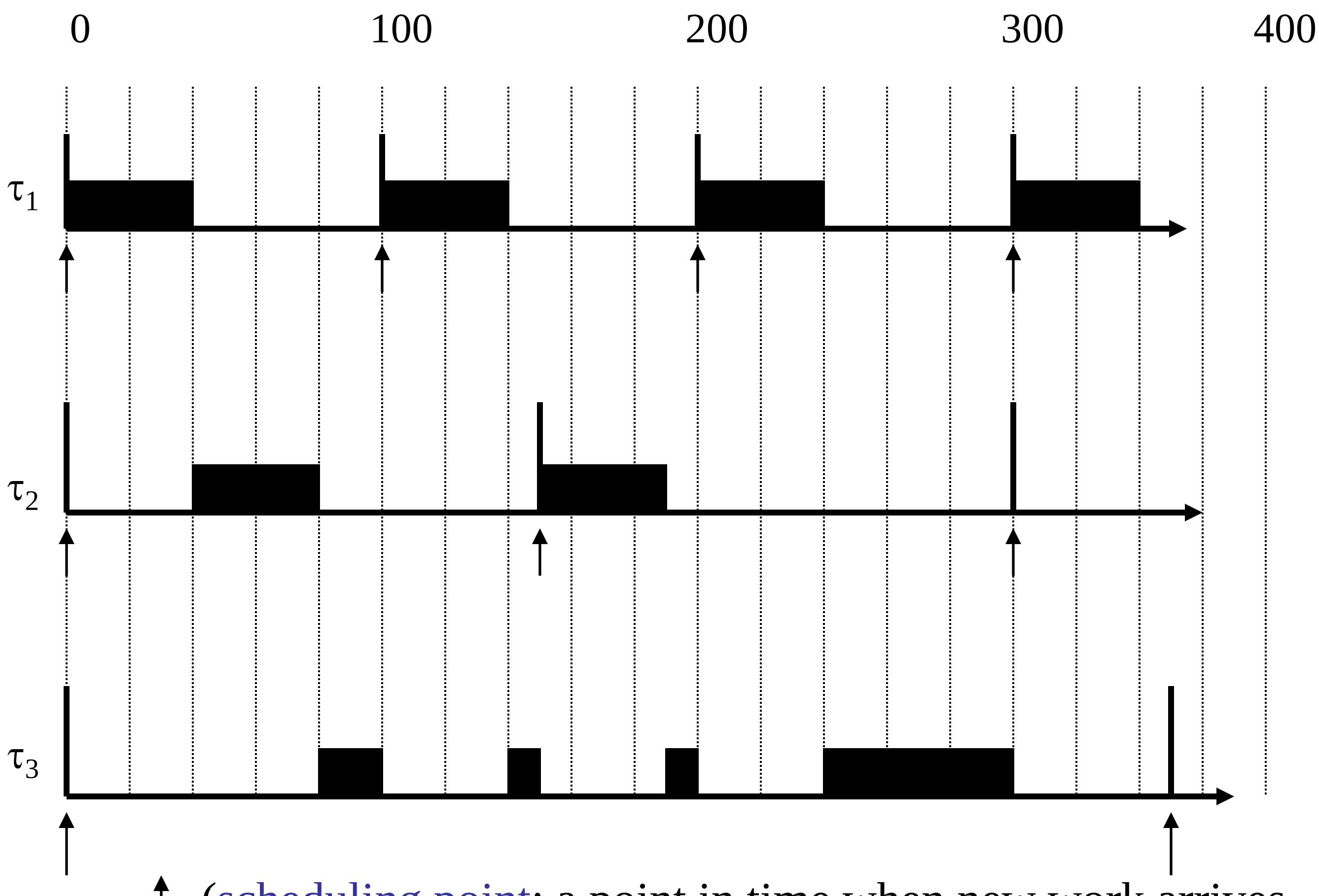
Rate Monotonic Scheduling Points

$$S_i = \{kT_j \mid j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i / T_j \rfloor\}$$

- Theorem 2.
  - A task τ_i is schedulable using RM if and only if

$$L_i = \min_{\{t \in S_i\}} \frac{W_i(t)}{t} \le 1$$

Example : Using Schedulability Points



(scheduling point: a point in time when new work arrives
worst-case phasing: all tasks are ready to execute at t=0)

Example Revisited

- Task τ_1: $C_1=40$, $T_1=100$, $U_1=0.4$
- Task τ_2: $C_2=40$, $T_2=150$, $U_2=0.267$
- Task τ_3: $C_3=100$, $T_3=350$, $U_3=0.286$

- With the schedulability test proposed in the Theorem 2,
  - For task τ_3, i=3 and S_3={100, 150, 200, 300, 350}

	$C_1+C_2+C_3 \le T_1$	$40+40+100 > 100$
or	$2C_1+C_2+C_3 \le T_2$	$80+40+100 > 150$
or	$2C_1+2C_2+C_3 \le 2T_1$	$80+80+100 > 200$
or	$3C_1+2C_2+C_3 \le 2T_2$	$120+80+100 = 300$
or	$4C_1+3C_2+C_3 \le T_3$	$160+120+100 > 350$

Problem

i	C_i	T_i	i	C_i	T_i
1	20	100	3	80	210
2	30	150	4	100	400

1. L_i?
2. RM schedulable condition of τ_1?
3. RM schedulable condition of τ_2?
4. RM schedulable condition of τ_3?
5. RM schedulable condition of τ_4?

More Results on RM

- A stochastic analysis for a randomly generated set of periodic tasks scheduled by RM has shown that
  - The average scheduling bound is usually much better than worst case behavior.

Reading List

- Least Slack Time Rate First: an Efficient Scheduling Algorithm for Pervasive Computing Environment