19Z604 Embedded Systems

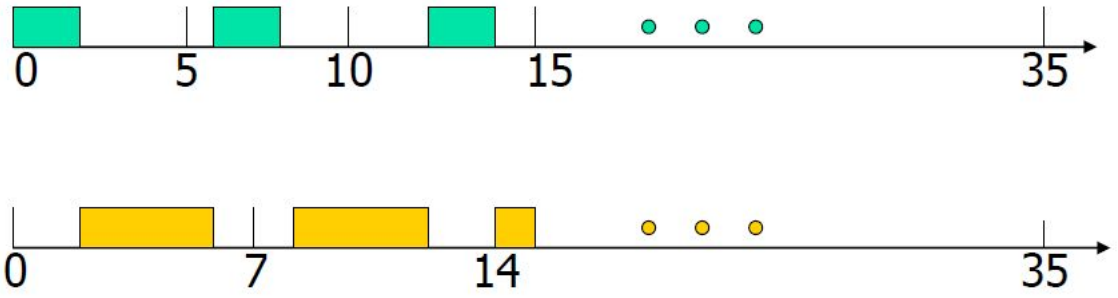
Dr.N.Arulanand,
Professor,
Dept of CSE,
PSG College of Technology

Earliest Deadline First (EDF)

- Dynamic priority scheduling
- Whenever a new task arrive, **sort the ready queue** so that the task closest to the end of its period assigned the highest priority
- **Preempt the running task** if it is not placed in the first of the queue in the last sorting

Earliest Deadline First (EDF)

- Task set: {(2,5),(4,7)}
- U = 2/5 + 4/7= 34/35 ~ 0.97 (schedulable!)



Schedulability Criteria for EDF

$$\sum_{i=1}^{n} e_i / p_i = \sum_{i=1}^{n} u_i \leq 1$$

Example: the Car Controller

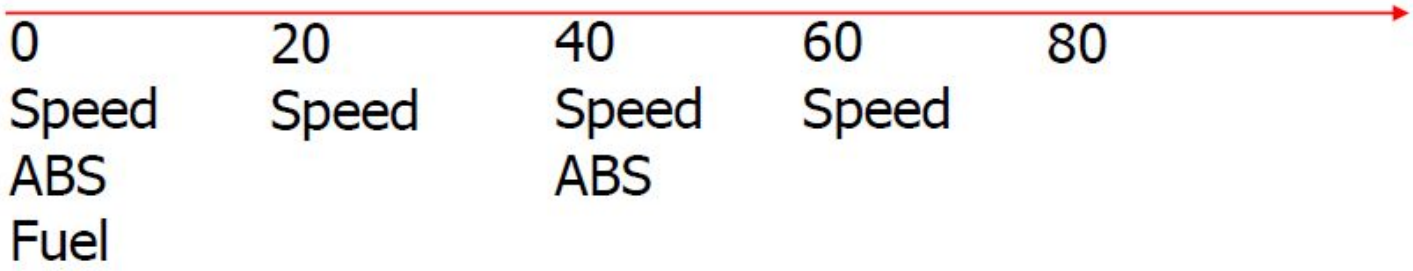
Activities of a car control system. Let

1. C= worst case execution time
2. T= (sampling) period
3. D= deadline

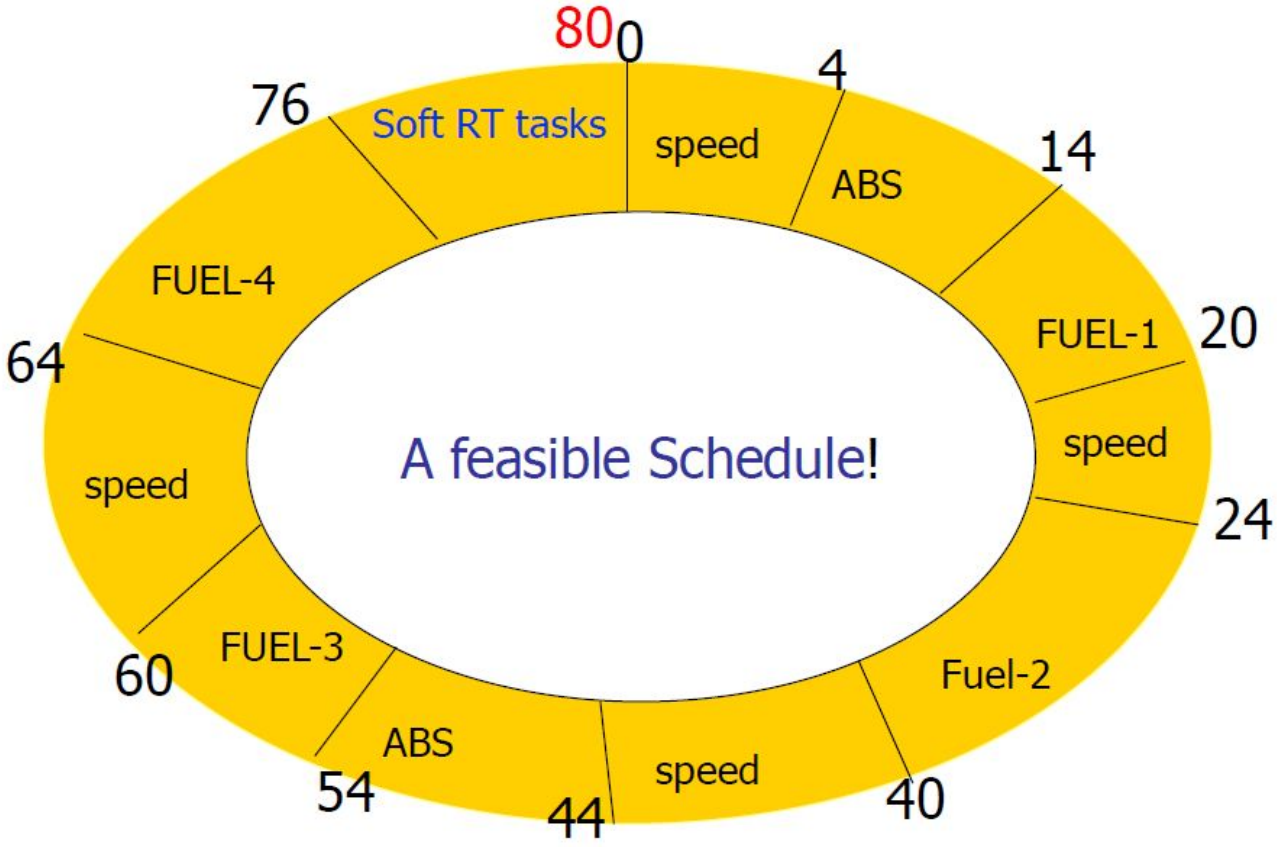
- Speed measurment: C=4ms, T=20ms, D=20ms
- ABS control: C=10ms,T=40ms, D=40ms
- Fuel injection: C=40ms,T=80ms, D=80ms
- Other software with soft deadlines e.g audio, air condition etc

The car controller: static cyclic scheduling

- The shortest repeating cycle = 80ms
- All task instances within the cycle:



The car controller:
time table constructed with EDF



EDF

- Exercise 1: Consider the following periodic tasks. Determine whether the task set is schedulable.

  T1 = (e1 = 35ms, p1 = 80ms)

  T2 = (e2 = 25 ms, p2 = 50ms)

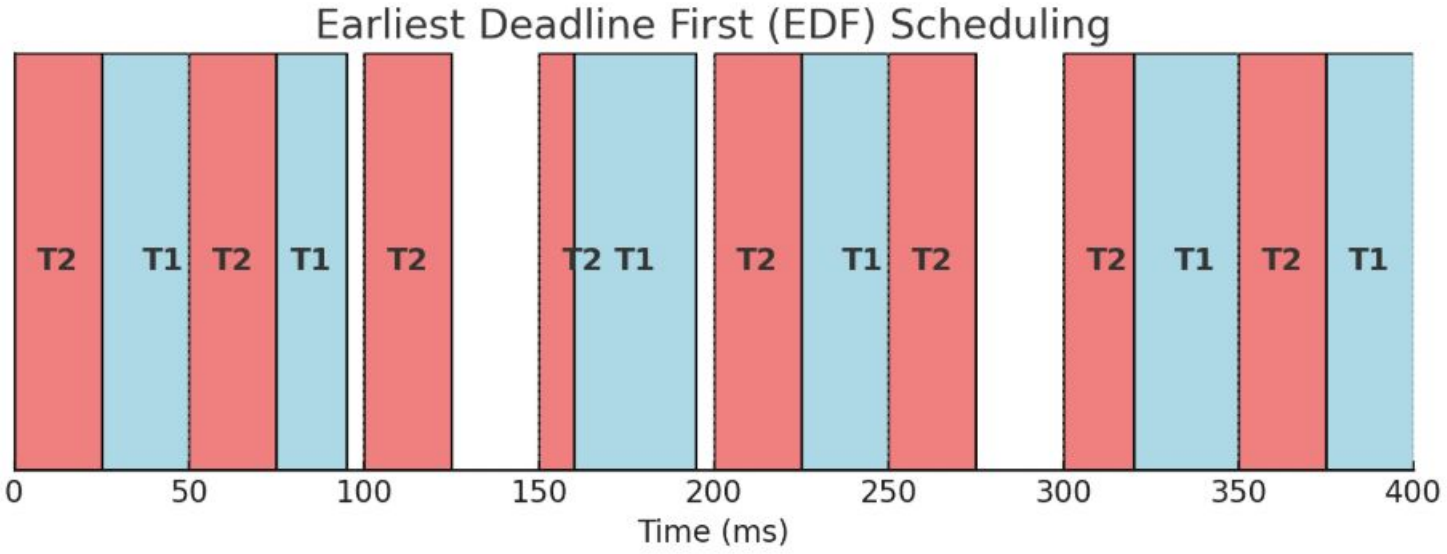
EDF

Determine whether the task set is schedulable ? Draw the chart.

T1 = (e1 = 35ms, p1 = 80ms)

T2 = (e2 = 25 ms, p2 = 50ms)

Schedulability Test: U = 35/80 + 25/50 < 1



EDF with period and deadline differ

- If deadline is given use the formula for Schedulability Test

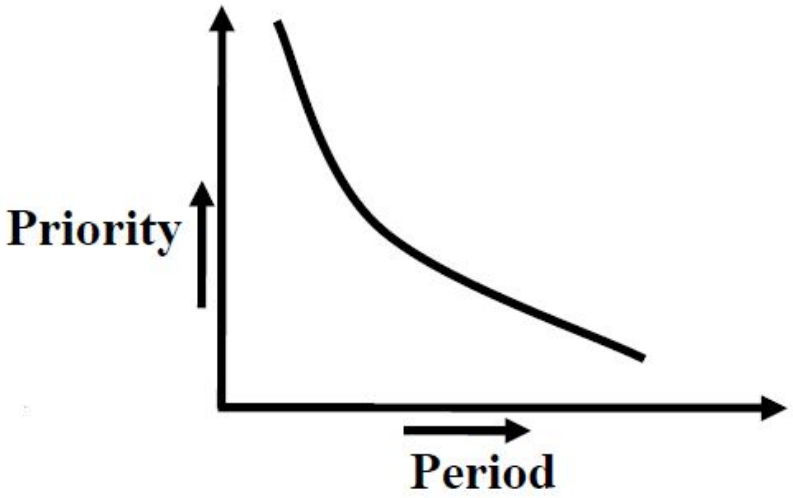
$$\sum_{i=1}^{n} e_i / min(p_i, d_i) \leq 1$$

Pros and Cons of EDF

- Simple EDF algorithm; it works for all types of tasks: periodic or non periodic
- It is simple and works nicely in theory (+)
- Simple schedulability test: U <= 1 (+)
- Optimal (+)
- Best CPU utilization (+)
- Difficult to implement in practice. It is not very often adopted due to the dynamic priority-assignment (expensive to sort the ready queue on-line), which has nothing to do with the periods of tasks. Note that Any task could get the highest priority (-)
- **Non stable**: if any task instance fails to meet its deadline, the system is not predictable, any instance of any task may fail (-)

RMS (Rate Monotonic Scheduling)

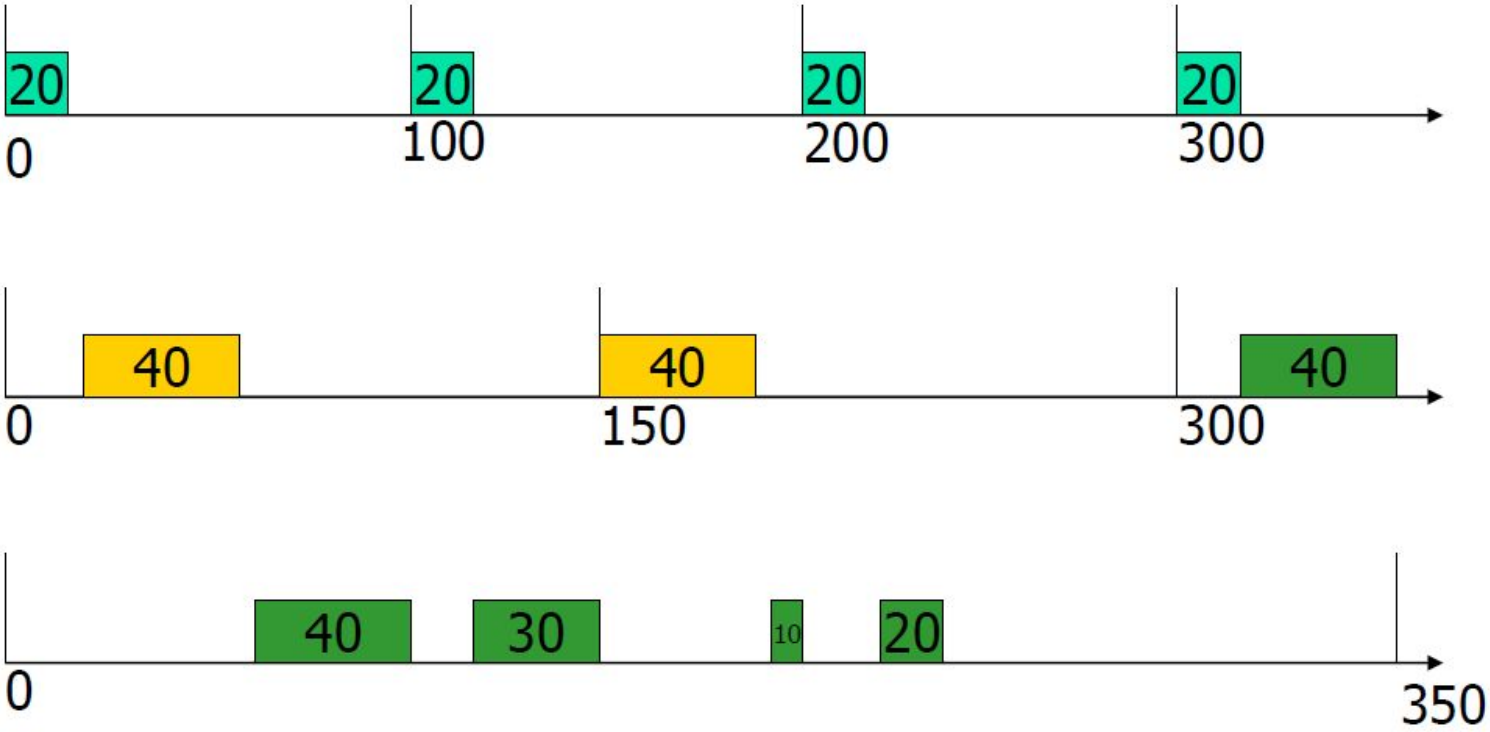
- a.k.a Rate Monotonic Algorithm (RMA)
- Fixed Priority scheduling
  - Priority is based on time-period
  - Lower the period, higher the priority



RMA Scheduling

Example: RM Scheduling

{(20,100),(40,150),(100,350)}



Schedulability test for RMA

- Necessary Condition
  - A set of periodic real-time tasks would not be RMA schedulable unless they satisfy the following necessary condition:

$$\sum_{i=1}^{n} e_i / p_i = \sum_{i=1}^{n} u_i \leq 1$$

- Sufficient Condition: Liu and Layland's condition

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

Utilization Bound Test (U_B)

Substituting n = 1

$$\sum_{i=1}^{1} u_i \le 1(2^{1/1} - 1) \text{ or } \sum_{i=1}^{1} u_i \le 1$$

Similarly for n = 2, we get,

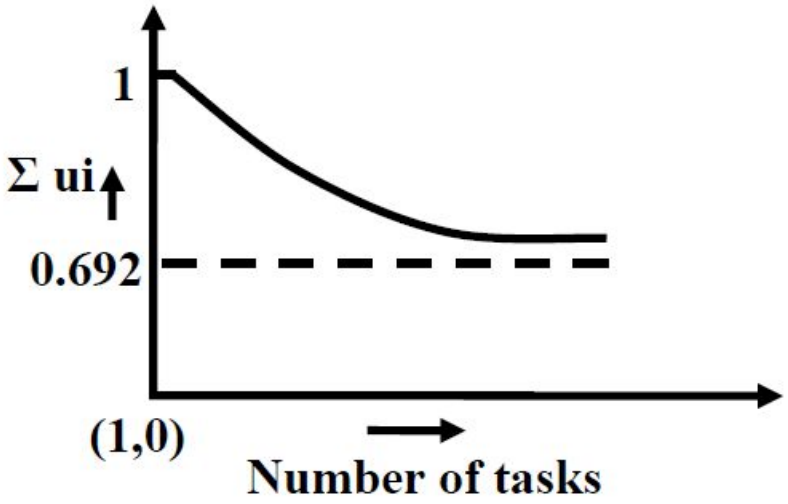
$$\sum_{i=1}^{2} u_i \le 2(2^{1/2} - 1) \text{ or } \sum_{i=1}^{2} u_i \le 0.828$$

For n = 3, we get,

$$\sum_{i=1}^{3} u_i \le 3(2^{1/3} - 1) \text{ or } \sum_{i=1}^{3} u_i \le 0.78$$

For $n \to \infty$, we get,

$$\sum_{i=1}^{\infty} u_i \le 3(2^{1/\infty} - 1) \text{ or } \sum_{i=1}^{\infty} u_i \le \infty.0$$



RMA

UB test is only sufficient, not necessay!

- Let $U = \Sigma\, Ci/Ti$ and $B(n) = n*(2^{1/n}-1)$
- Three possible outcomes:
  - $0 <= U <= B(n)$: schedulable
  - $B(n) < U <= 1$: no conclusion
  - $1 < U$: overload

Exercise

Example: applying UB Test

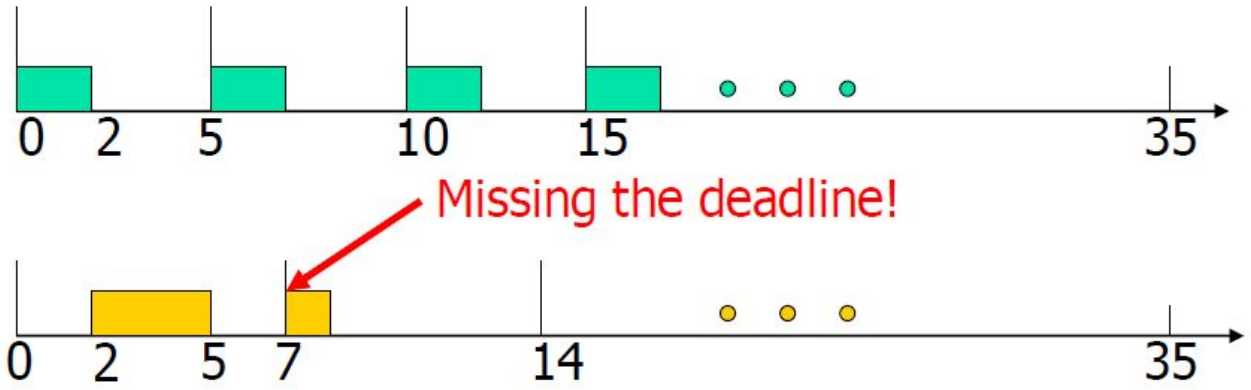
	C	T (D=T)	C/T
Task 1	20	100	0.200
Task 2	40	150	0.267
Task 3	100	350	0.286

Total utilization: $U=0.2+0.267+0.286=0.753<B(3)=0.779$!
The task set is schedulable

RMA

Example

- Task set: T1=(2,5), T2=(4,7)
- U = 2/5 + 4/7= 34/35 ~ 0.97 (schedulable?)
- RMS priority assignment: Pr(T1)=1, Pr(T2)=2



Example: UB test is sufficient, not necessary

- Assume a task set: {(1,3),(1,5),(1,6),(2,10)}
- CPU utilization U= 1/3+1/5+1/6+2/10=0.899
- The utilization bound B(4)=0.756, U>B(4)
- But the task set is schedulable anyway!

Why Lehoczky's Test ?

Example 6: Check whether the following set of three periodic real-time tasks is schedulable under RMA on a uniprocessor: $T_1 = (e_1=20, p_1=100)$, $T_2 = (e_2=30, p_2=150)$, $T_3 = (e_3=90, p_3=200)$.

Solution: Let us first compute the total CPU utilization due to the given task set:

$$\sum_{i=1}^{3} u_i = 20/100 + 30/150 + 90/200 = 0.7$$

Now checking for Liu and Layland criterion:

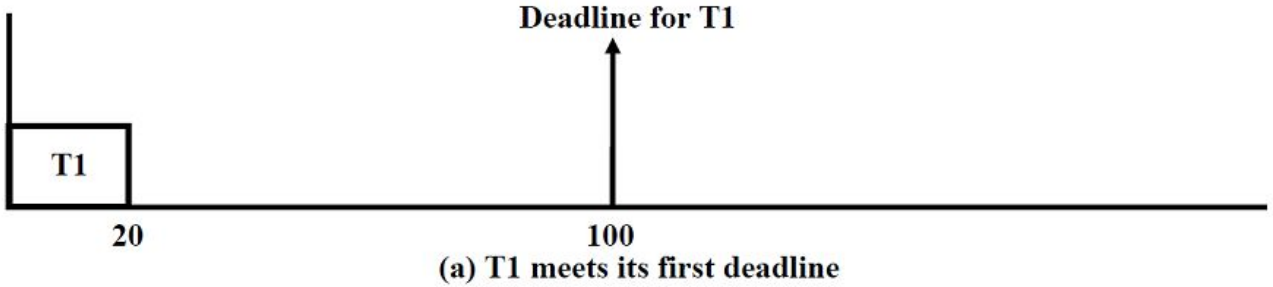
$$\sum_{i=1}^{3} u_i \leq 0.78$$

Since 0.85 is not ≤ 0.78, the task set is not RMA-schedulable.

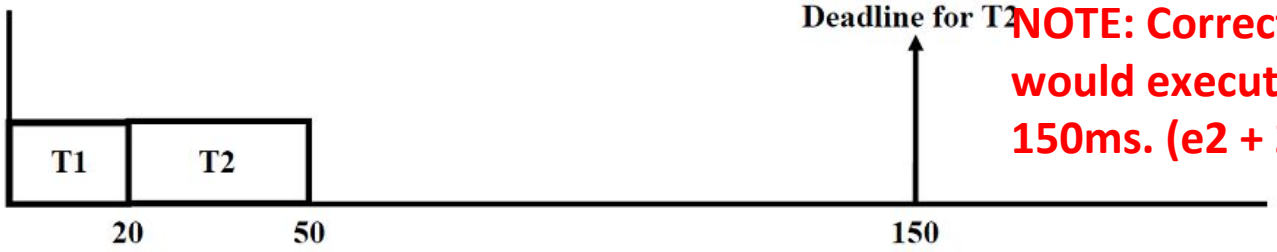
Liu and Layland test (Expr. 2.10) is pessimistic in the following sense.

If a task set passes the Liu and Layland test, then it is guaranteed to be RMA schedulable. On the other hand, even if a task set fails the Liu and Layland test, it may still be RMA schedulable.

Lehoczky's Criterion for Tasks

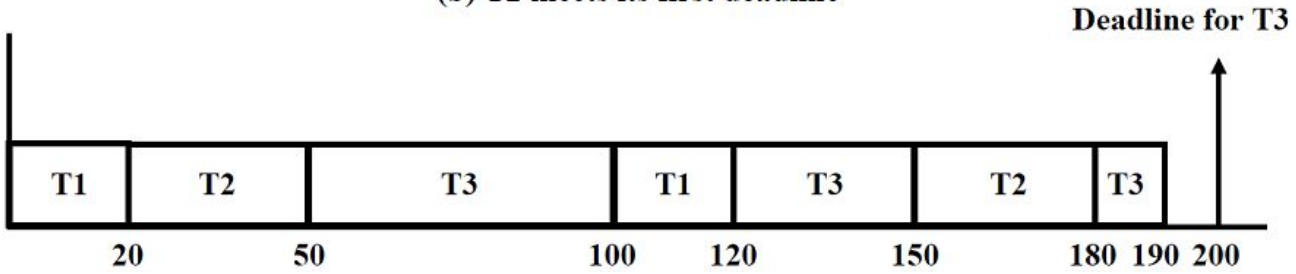


(a) T1 meets its first deadline



(b) T2 meets its first deadline

NOTE: Correction in fig (b). T1 would execute twice within 150ms. (e2 + 2e1 < p2)



For the task T_2: T_1 is its higher priority task and considering 0 phasing, it would occur once before the deadline of T_2. Therefore, $(e_1 + e_2) < p_2$ holds, since 20 + 30 = 50 msec < 150 msec. Therefore, T_2 meets its first deadline.

For the task T_3: $(2e_1 + 2e_2 + e_3) < p_3$ holds, since 2*20 + 2*30 + 90 = 190msec < 200 msec

Lehoczky's Test

Apply the below formula, when period and deadline are same

$$e_i + \sum_{k=1}^{j-1} \lceil p_i / p_k \rceil * e_k \leq p_i$$

Apply the below formula, when period and deadline are different

$$e_i + \sum_{k=1}^{j-1} \lceil d_i / p_k \rceil * e_k \leq d_i$$

RMA Transient Overload Handling

- RMA possess good transient overload handling.
- Good transient overload condition means that when a lower priority task doesn't complete within its planned complete time, the higher priority task doesn't miss the deadline.

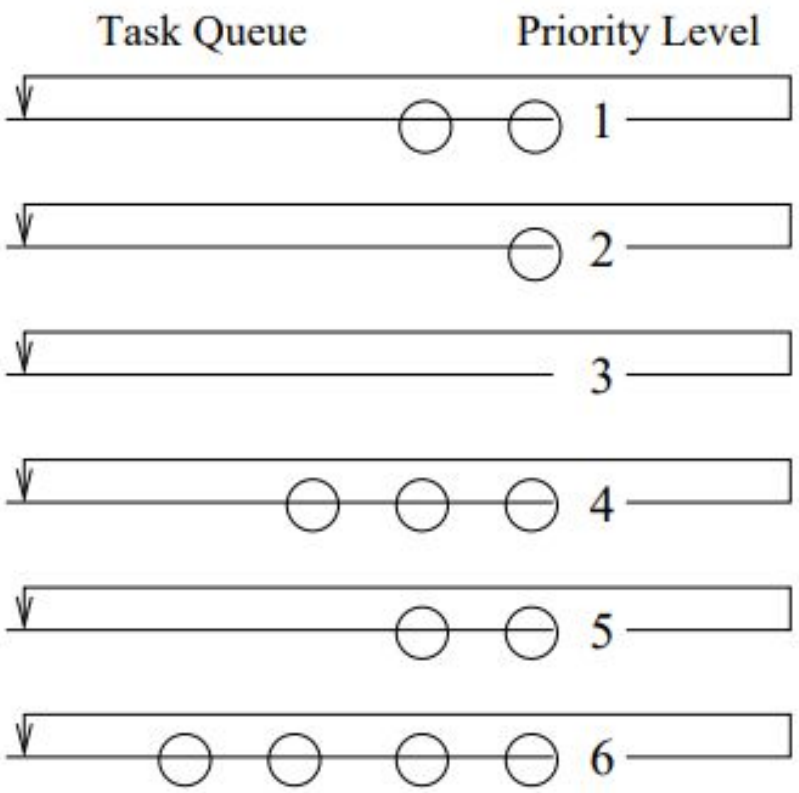


Figure 15: Multi-Level Feedback Queue

Deadline Monotonic Algorithm (DMA)

- RMA no longer remains an optimal scheduling algorithm for the periodic real-time tasks, when task deadlines and periods differ (i.e. $d^i \neq p^i$)
- DMA is essentially a variant of RMA and assigns priorities to tasks based on their deadlines, rather than assigning priorities

DMA

- **Example 1: Is the following task set schedulable by DMA? Also check whether it is schedulable using RMA.**
  $T^1 = (e^1=10, p^1=50, d^1=35)$
  $T^2 = (e^2=15, p^2=100, d^1=20)$
  $T^3 = (e^3=20, p^3=200, d^1=200)$ [time in msec].

- Under DMA, the priority ordering of the tasks is as follows: $pr(T^2) > pr(T^1) > pr(T^3)$.
- Checking for T^2:
  15 msec < 20 msec.
  Hence, T^2 *will meet its first deadline.*
- Checking for T^1:
  - (15 + 10) < 35

  Hence T^1 *will meet its first deadline.*
- Checking for T^3:
  - (20 + 30 + 40) < 200

  Therefore, T^3 *will meet its deadline.*

Therefore, the given task set is schedulable under DMA but not under RMA

Context Switching Overhead

So far, while determining schedulability of a task set, we had ignored the overheads incurred on account of context switching. Let us now investigate the effect of context switching overhead on schedulability of tasks under RMA.

For simplicity we can assume that context switching time is constant, and equals 'c' milliseconds where 'c' is a constant. From this, it follows that the net effect of context switches is to increase the execution time e_i of each task T_i to at most $e_i + 2*c$. It is therefore clear that in order to take context switching time into consideration, in all schedulability computations, we need to replace e_i by $e_i + 2*c$ for each T_i.

Example 11: Check whether the following set of periodic real-time tasks is schedulable under RMA on a uniprocessor: $T_1 = (e_1=20, p_1=100)$, $T_2 = (e_2=30, p_2=150)$, $T_3 = (e_3=90, p_3=200)$. Assume that context switching overhead does not exceed 1 msec, and is to be taken into account in schedulability computations.

Solution: The net effect of context switches is to increase the execution time of each task by two context switching times. Therefore, the utilization due to the task set is:

$$\sum_{i=1}^{3} u_i = 22/100 + 32/150 + 92/200 = 0.89$$

Since $\sum_{i=1}^{3} u_i > 0.78$, the task set is not RMA schedulable according to the Liu and Layland test.

Let us try Lehoczky's test. The tasks are already ordered in descending order of their priorities.

Checking for task T_1:

$$22 < 100$$

The condition is satisfied; therefore T_1 meets its first deadline.

Checking for task T_2:

$$(22*2) + 32 < 150$$

The condition is satisfied; therefore T_2 meets its first deadline.

Checking for task T_3:

$$(22*2) + (32*2) + 90 < 200.$$

The condition is satisfied; therefore T_3 meets its first deadline.

Therefore, the task set can be feasibly scheduled under RMA even when context switching overhead is taken into consideration.

Self Suspension

A task might cause its self-suspension, when it performs its input/output operations or when it waits for some events/conditions to occur. When a task self suspends itself, the operating system removes it from the ready queue, places it in the blocked queue, and takes up the next eligible task for scheduling. Thus, self-suspension introduces an additional scheduling point, which we

Let us now determine the effect of self-suspension on the schedulability of a task set. Let us consider a set of periodic real-time tasks $\{T_1, T_2, \ldots, T_n\}$, which have been arranged in the increasing order of their priorities (or decreasing order of their periods). Let the worst case self-suspension time of a task T_i is b_i. Let the delay that the task T_i might incur due to its own self-suspension and the self-suspension of all higher priority tasks be bt_i. Then, bt_i can be expressed as:

$$bt_i = b_i + \sum_{k=1}^{i-1} min(e_k, b_k)$$

$$e_i + bt_i + \sum_{k=1}^{i-1} \lceil p_i / p_k \rceil * e_k \leq p_i$$

Example 14: Consider the following set of periodic real-time tasks: $T_1 = (e_1=10, p_1=50)$, $T_2 = (e_2=25, p_2=150)$, $T_3 = (e_3=50, p_3=200)$ [all in msec]. Assume that the self-suspension times of T_1, T_2, and T_3 are 3 msec, 3 msec, and 5 msec, respectively. Determine whether the tasks would meet their respective deadlines, if scheduled using RMA.

Solution: The tasks are already ordered in descending order of their priorities. By using the generalized Lehoczky's condition given by Expr. 3.9, we get:
For T_1 to be schedulable:

$$(10 + 3) < 50$$

Therefore T_1 would meet its first deadline.
For T_2 to be schedulable:

$$(25 + 6 + 10*3) < 150$$

Therefore, T_2 meets its first deadline.
For T_3 to be schedulable:

$$(50 + 11 + (10*4 + 25*2)) < 200$$

This inequality is also satisfied. Therefore, T_3 would also meet its first deadline.
It can therefore be concluded that the given task set is schedulable under RMA even when self-suspension of tasks is considered.

Self Suspension with Context Switching Overhead

each task suffers at most two context switches – one context switch when it starts and another when it completes. It is easy to realize that any time when a task self-suspends, it causes at most two additional context switches.

Let us denote the maximum context switch time as c. The effect of a single self-suspension of tasks is to effectively increase the execution time of each task T_i in the worst case from e_i to $(e_i + 4*c)$.

Consider the following set of three real-time periodic tasks.

Task	**Start Time** mSec	**Processing Time** mSec	**Period** mSec	**Deadline** mSec
T_1	20	25	150	100
T_2	40	10	50	50
T_3	60	50	200	200

a. Check whether the three given tasks are schedulable under RMA. Show all intermediate steps in your computation.

b. Assuming that each context switch incurs an overhead of 1 msec, determine whether the tasks are schedulable under RMA. Also, determine the average context switching overhead per unit of task execution.

c. Assume that T1, T2, and T3 self-suspend for 10 msec, 20 msec, and 15 msec respectively. Determine whether the task set remains schedulable under RMA. The context switching overhead of 1 msec should be considered in your result. You can assume that each task undergoes self-suspension only once during each of its execution.

d. Assuming that T1 and T2 are assigned the same priority value, determine the additional delay in response time that T2 would incur compared to the case when they are assigned distinct priorities. Ignore the self-suspension times and the context switch overhead for this part of the question.