When do People understand and Admire You?

Einstein to Chaplin:
"What I most admire about your art....
you don't say a word
and yet
the rest of the world understands you"

Chaplin answered :
It’s true, but your glory is even greater..
"The whole world admires you,
even though they don't
understand a word of what you say...!"

The problem of multi program scheduling on a single processor

Characteristics peculiar to the program functions that need guaranteed service.

We will use J1, J2, • • • , Jm to denote m periodic tasks, with their request periods being T1, T2, ... , Tm and their run-times being C1, C2, ... , Cm, respectively.

The *request rate* of a task is defined to be the reciprocal of its request period.

The problem of multi program scheduling on a single processor : Assumptions

1. The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
2. Deadlines consist of run-ability constraints only--i.e, each task must be completed before the next request for it occurs.
3. The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.
4. Run-time for each task is constant for that task and does not vary with time. Run-time here refers to the time which is taken by a processor to execute the task without interruption.

Rate-Monotonic priority Assignment

Tasks with higher request rates will have higher priorities. Such an assignment of priorities is known as the *rate-monotonic priority assignment.*

Rate-Monotonic Scheduling: Static or Fixed Priority Scheduling

Priorities are assigned by rank order of the period, with the process with the shortest period being assigned the highest priority.

It is a pre-emptive scheme. While executing a process if a higher priority process needs attention there will be context switching.

The Deadline Driven Scheduling Algorithm or Earliest Deadline First Scheduling

• Priorities are assigned to tasks according to the deadlines of their current requests.

• A task will be assigned the highest priority if the deadline of its current request is the nearest, and will be assigned the lowest priority if the deadline of its current request is the furthest.

• Priorities will be recalculated at every completion of a process.

•At any instant, the task with the highest priority and yet unfulfilled request will be executed.

An Example: EDF

J1, J2 and J3 with execution times 1,1,2 and periods 3,4 and 5 respectively.

Fixed priority : C1 C2 C3 C1 C2 C3 C1 C3 C2 C1 C3 C3 C1 C2 C3 C1
Time slot : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

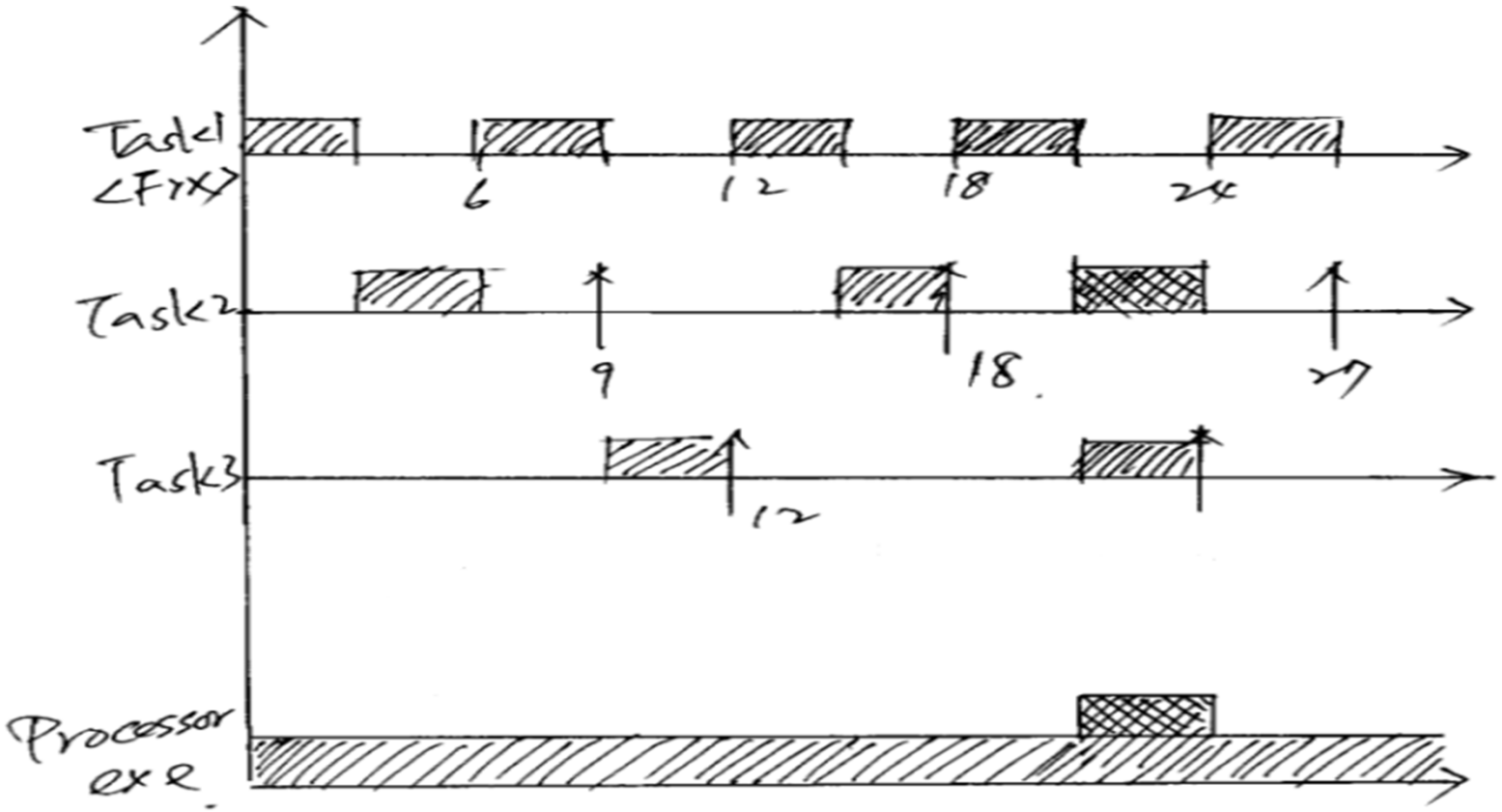
Rate monotonic scheduling is not possible?

Deadline driven scheduling C2 C3
C1 C2 C3 C1 C2C1 C3 C1 C1
EDF : C1 C2 C3 C3 C1 C2 C1 C3 C3 C1 C2 C3 C3 C1 C2 C1
Time slot : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Another Example: Comments

T_1=6 T_2=9 T_3=12

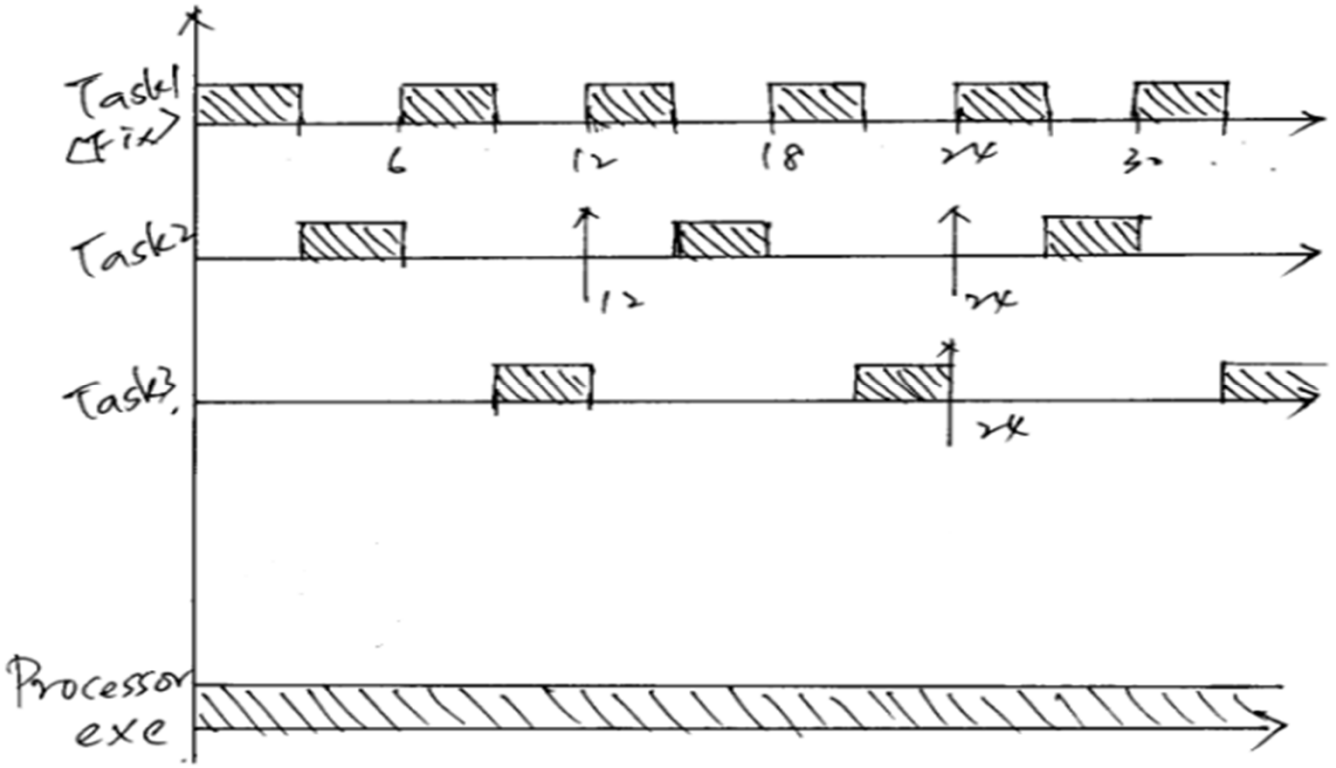
C_1=3 C_2=3 C_3=3 U>1



Another Example: Comments

T_1=6 T_2=12 T_3=24

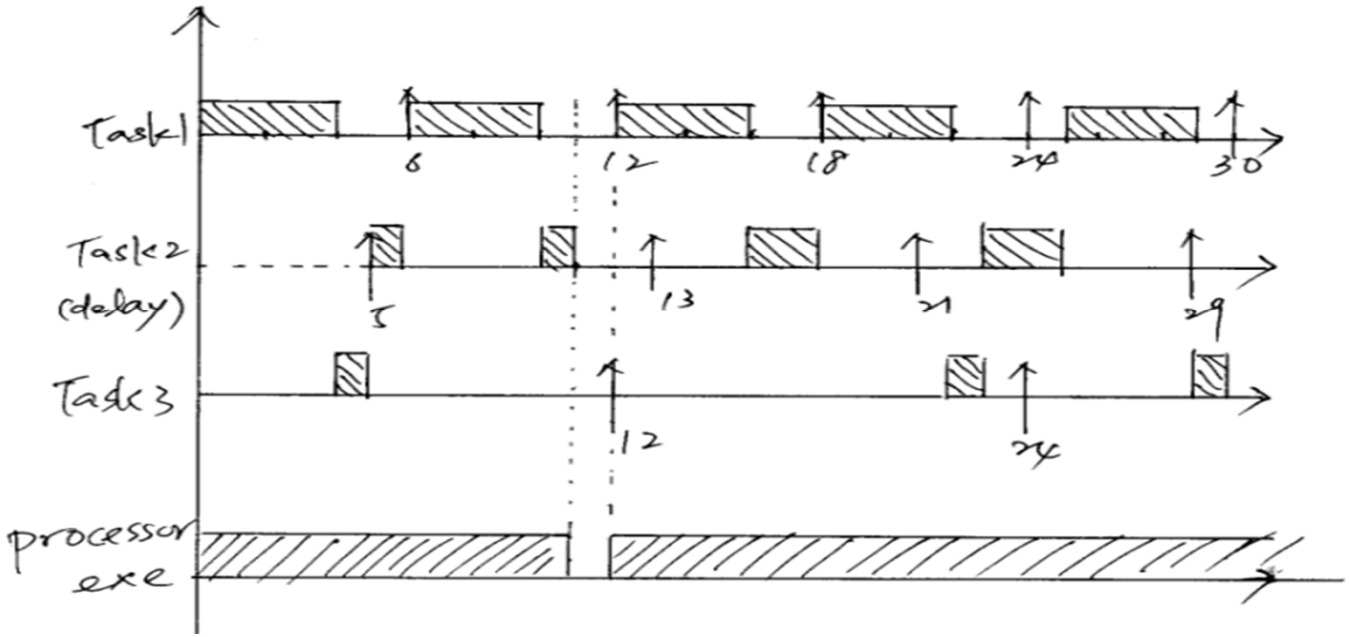
C_1=3 C_2=3 C_3=3 U=1



Another Example: Comments

$T_1=6 \quad T_2=8 \quad T_3=12$

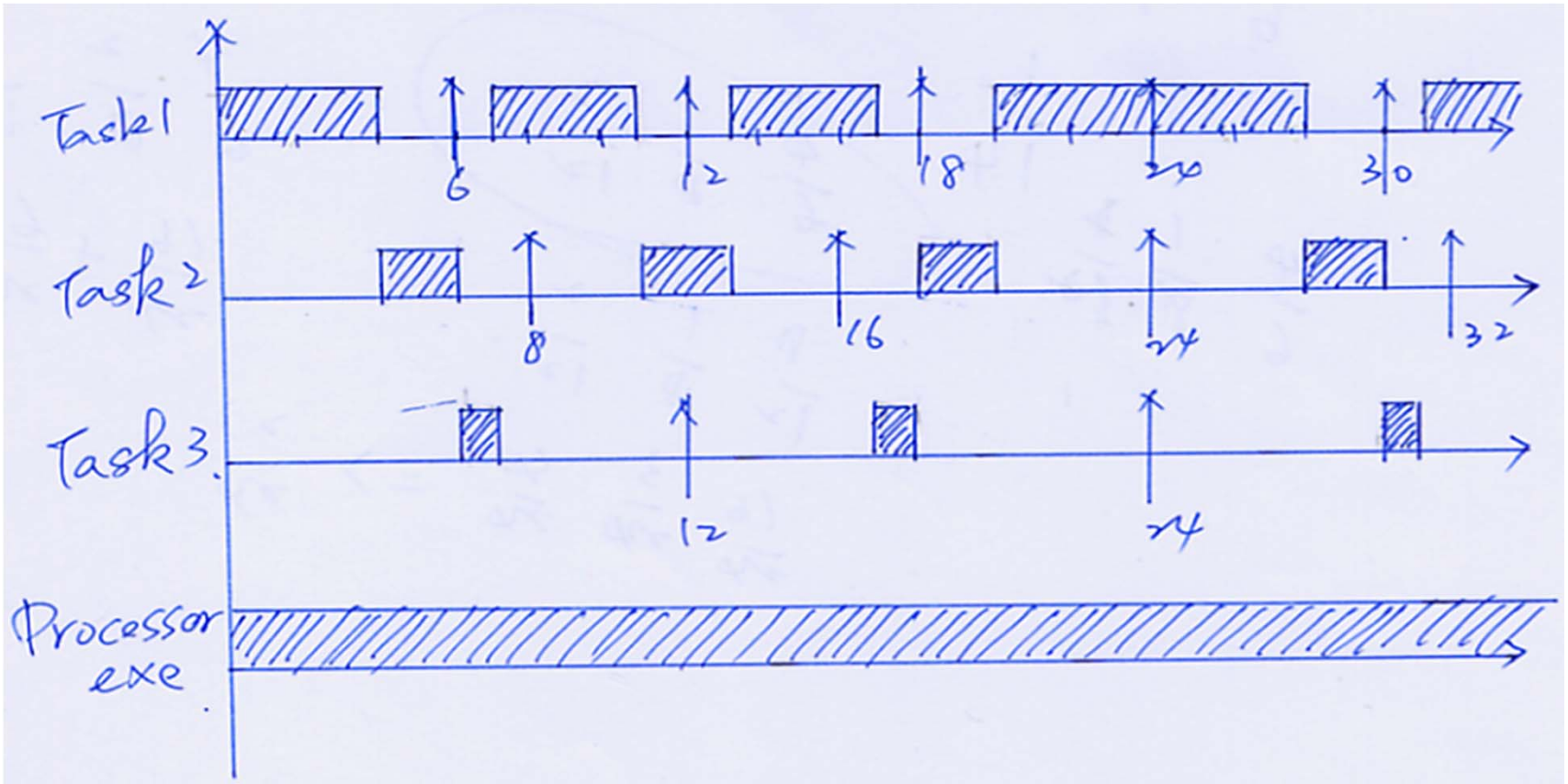
$C_1=4 \quad C_2=2 \quad C_3=1 \qquad U=1$



Another Example: Comments

- $T_1=6$ $T_2=8$ $T_3=12$

  $C_1=4$ $C_2=2$ $C_3=1$ $U=1$



Rate Monotonic Analysis

. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment C. L. LIU *Project MAC, Massachusetts Institute of Technology* AND JAMES W. LAYLAND *Jet Propulsion Laboratory, California Institute of Technology*

The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

A Rule for Static Scheduling problem : By Liu and Layman

The *deadline* of a request for a task is defined to be the time of the next request for the same task.

Overflow: For a set of tasks scheduled according to some scheduling algorithm, we say that 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.

A Rule for Static Scheduling problem

We define the *response time* of a request for a certain task to be the time span between the request and the end of the response to that request.

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

A *critical time zone* for a task is the time interval between a critical instant and the end of the response to the corresponding request of the task.

A Nice Result about critical instant

Result 1. *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

Proof

Let J1, J2, • • • , Jm denote a set of priority-ordered tasks (jobs) with Jm being the task with the lowest priority.

Consider a particular request for Jm that occurs at t1. Suppose that between t1 and t1+Tm , the time at which the subsequent request of Jm occurs, requests for task Ji i <m, occur at t2, t2+Ti, t2 t2+2Ti, • • • , t2+kTi as illustrated in Figure 1.

Clearly, the pre-emption of Jm by Ji, will cause a certain amount of delay in the completion of the request for Jm that occurred at t1, unless the request for Jm is completed before t2.

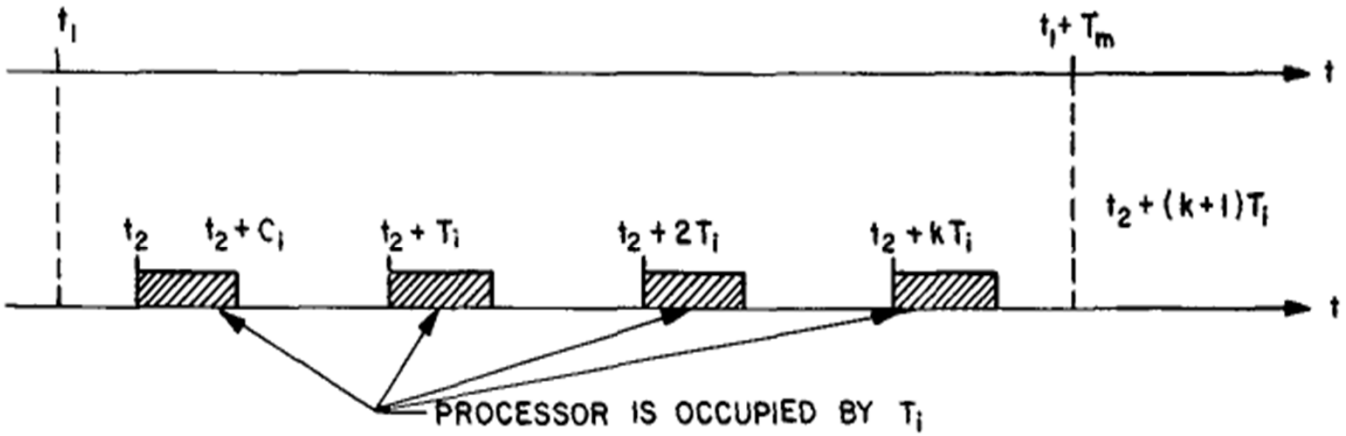


FIG. 1. Execution of τ_i between requests for τ_m

Proof

Moreover, from Figure 1 we see immediately that advancing the request time t2 will not speed up the completion of Jm. The completion time of Jm is either unchanged or delayed by such advancement.

Consequently, the delay in the completion of Jm is largest when t2 coincides with t1.

Repeating the argument for all Ji, i = 2, • • • , m -- 1, we prove the theorem.

Feasibility Test for a given priority assignment

How to check that a given priority assignment will yield a feasible scheduling algorithm?

If the requests for all tasks at their critical instants are fulfilled before their respective deadlines, then the scheduling algorithm is feasible.

Check for Feasibility

J1, J2 and J3 with execution times 1,1,2 and periods 3,4 and 5 respectively.

Fixed priority : C1 C2 C3 C1 C2 C3 C1 C3 C2 C1 C3 C3 C1 C2 C3 C1
Time slot : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Rate monotonic scheduling is not possible?

J1 at its critical instant gets completed before its deadline

J2 at its critical instant can be completed before its deadline of 4

J3 at its critical instant cannot be completed before its deadline of 5

Usefulness of Critical Time Instants : Higher priority tasks must be taken up first

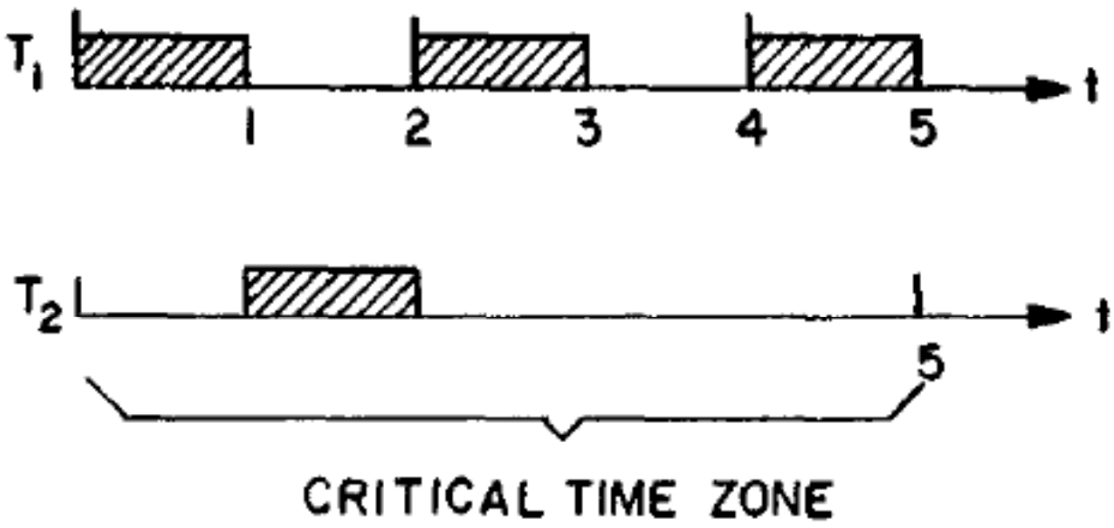
If the requests for all tasks at their critical instants are fulfilled before their respective deadlines, then the scheduling algorithm is feasible.

Result 1*. A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

An Example: Schedules for two tasks

Consider a set of two tasks J1 and J2 with T1 = 2, T2 = 5, and C1 = 1, C2 = 1.

If we let J1 be the higher priority task, then from Figure 2 (a) we see that such a priority assignment is feasible.

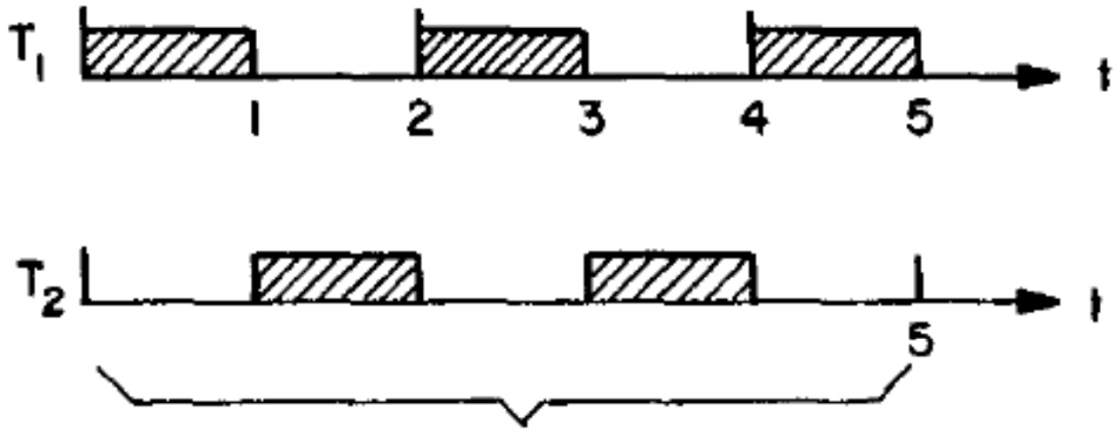


(a)

An Example: Schedules for two tasks

Consider a set of two tasks J1 and J2 with T1 = 2, T2 = 5, and C1 = 1, C2 = 1.

If we let J1 be the higher priority task. The value of C2 can be increased at most to 2 but not further as illustrated in Figure 2 (b).

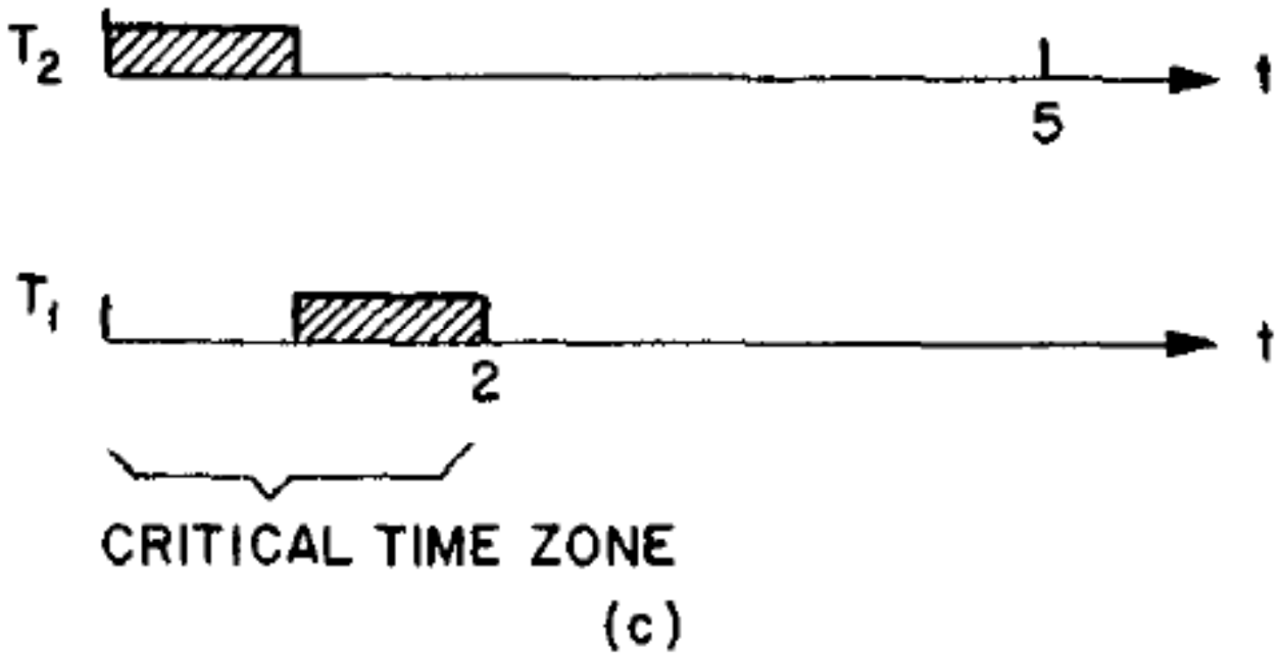


(b)

An Example: Schedules for two tasks

Consider a set of two tasks J1 and J2 with T1 = 2, T2 = 5, and C1 = 1, C2 = 1.

If we let J2 be the higher priority task, then neither of the values of C1 and C2 can be increased beyond 1 as shown in Figure 2(c)



An Interesting Idea

Consider the case of scheduling two tasks J1 and J2. Let T1 and T2 be the request periods of the tasks, with T1 < T2. If we let J1 be the higher priority task, then,

$$(T2/T1)C1+C2 < T2 \qquad (1)$$

(T2/T1) denotes floor of T2/T1, the largest integer <= T2/T1

If we let J2 be the higher priority task, then, the following inequality must be satisfied:

$$C1 + C2 < = T1 \qquad (2)$$

Do you see that (2) implies (1) but not the other way around?

(T2/T1)C1 + (T2/T1)C2 <= (T2/T1)T1 <= T2

An Interesting Result : *rate-monotonic priority assignment*

Whenever T1 < T2 and C1, C2 are such that the task schedule is feasible with J2 at higher priority than J1, it is also feasible with J1 at higher priority than J2, but the opposite is not true.

Thus we should assign higher priority to J1 and lower priority to J2.

Hence, more generally, it seems that a "reasonable" rule of priority assignment is to assign priorities to tasks according to their request rates, independent of their run-times.

An Interesting Result : *rate-monotonic priority assignment*

THEOREM 2. *If a feasible priority assignment exists for some task set, the rate monotonic priority assignment is feasible for that task set.*

An Interesting Result : *rate-monotonic priority assignment*

PROOF. Let Jl, J2, … Jm be a set of m tasks with a certain feasible priority assignment.

Let Ji and Jk be two tasks of adjacent priorities in such an assignment with Ji being the higher priority one.

Suppose that Ti > Tk.

Let us interchange the priorities of Ji and Jk.

It is not difficult to see that the resultant priority assignment is still feasible.

Since the rate-monotonic priority assignment can be obtained from any priority ordering by a sequence of pair wise priority reorderings as above

QED : Quite Easily Done!!

Achievable Processor Utilization

To determine a least upper bound to processor utilization in fixed priority systems.

Definition: *utilization factor* is the fraction of processor time spent in the execution of the task set.

The utilization factor is equal to one minus the fraction of idle processor time.

Since *Ck/ Tk* is the fraction of processor time spent in executing task Jk , for m tasks, the utilization factor is given by

$$U = \sum_{i=1}^{m} Ci / Ti$$

Achievable Processor Utilization

Although the processor utilization factor can be improved by increasing the values of the Ci or by decreasing the values of the Ti it is upper bounded by the requirement that all tasks satisfy their deadlines at their critical instants.

$$U = \sum_{i=1}^{m} Ci / Ti$$

How large the processor utilization factor can be?

Corresponding to a priority assignment, 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.

For a given fixed priority scheduling algorithm, the *least upper bound* of the utilization factor is the minimum of the utilization factors over all sets of tasks that fully utilize the processor.

$$U = \sum_{i=1}^{m} Ci / Ti$$

How large the processor utilization factor can be?

For all task sets whose processor utilization factor is below this bound, there exists a fixed priority assignment which is feasible.

Therefore, for all task sets whose processor utilization factor is below this bound, Rate monotonic assignment is feasible.

$$U = \sum_{i=1}^{m} Ci / Ti$$

How large the processor utilization factor can be?

Since the rate-monotonic priority assignment is optimum, the utilization factor achieved by the rate-monontonic priority assignment for a given task set is greater than or equal to the utilization factor for any other priority assignment for that task set.

Thus, the least upper bound to be determined is the infimum of the utilization factors corresponding to the rate-monotonic priority assignment over all possible request periods and run-times for the tasks.

$$U = \sum_{i=1}^{m} Ci / Ti$$

Achievable Processor Utilization

Corresponding to a priority assignment, 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.

Achievable Processor Utilization for Two Tasks

THEOREM 3. *For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is*

$$U = 2(2^{1/2} - 1)$$

Processor Utilization with Two Tasks

PROOF. Let J1 and J2 be two tasks with their periods being T1 and T2 and their run-times being C1 and C2, respectively. Assume that T2 > T1. According to the rate-monotonic priority assignment, J1 has higher priority than J2.

In a critical time zone of J2, there are [T2/T1] requests for J1. [x] denotes ceiling of x (smallest integer greater than or equal to x.

Let us now adjust C2 to fully utilize the available processor time within the critical time zone. Two cases occur:

Processor Utilization with Two Tasks

Case 1. The run-time C_1 is short enough that all requests for τ_1 within the critical time zone of T_2 are completed before the second τ_2 request. That is,

$$C_1 \leq T_2 - T_1 \lfloor T_2/T_1 \rfloor .$$

Thus, the largest possible value of C_2 is

$$C_2 = T_2 - C_1 \lceil T_2/T_1 \rceil .$$

The corresponding processor utilization factor is

$$U = 1 + C_1[(1/T_1) - (1/T_2) \lceil T_2/T_1 \rceil].$$

In this case, the processor utilization factor U is monotonically decreasing in C_1 .

Processor Utilization with Two Tasks

Case 2. The execution of the $\lceil T_2/T_1 \rceil$th request for t_1 overlaps the second request for τ_2. In this case

$$C_1 \geq T_2 - T_1 \lfloor T_2/T_1 \rfloor .$$

It follows that the largest possible value of C_2 is

$$C_2 = -C_1 \lfloor T_2/T_1 \rfloor + T_1 \lfloor T_2/T_1 \rfloor$$

and the corresponding utilization factor is

$$U = (T_1/T_2) \lfloor T_2/T_1 \rfloor + C_1[(1/T_1) - (1/T_2) \lfloor T_2/T_1 \rfloor].$$

In this case, U is monotonically increasing in C_1.

Processor Utilization with Two Tasks : Final Observation

The minimum of U clearly occurs at the boundary between these two cases. That is, for

$$C_1 = T_2 - T_1 \lfloor T_2/T_1 \rfloor$$

we have

$$U = 1 - (T_1/T_2)[\lceil T_2/T_1 \rceil - (T_2/T_1)][(T_2/T_1) - \lfloor T_2/T_1 \rfloor]. \qquad (3)$$

For notational convenience,[3] let $I = \lfloor T_2/T_1 \rfloor$ and $f = \{T_2/T_1\}$. Equation (3) can be written as

$$U = 1 - f(1 - f)/(I + f).$$

Since U is monotonic increasing with I, minimum U occurs at the smallest possible value of I, namely, $I = 1$. Minimizing U over f, we determine that at $f = 2^{\frac{1}{2}} - 1$, U attains its minimum value which is

$$U = 2(2^{\frac{1}{2}} - 1) \simeq 0.83.$$

[3] $\{T_1/T_2\}$ denotes $(T_2/T_1) - \lfloor T_2/T_1 \rfloor$, i.e. the fractional part of T_2/T_1.

Processor Utilization with m Tasks

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

For m = 3 U is about .78
For large m U is near ln 2 or .699

Feasibility Test For Rate Monotonic Scheduling : Sufficient Condition

A set of ***n*** periodic tasks scheduled by the rate-monotonic algorithm can always meet their deadlines if

$$\frac{C_1}{T_1} + \cdots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1)$$

Feasibility Test For Rate Monotonic Scheduling

A set of ***n*** periodic tasks scheduled by the rate-monotonic algorithm will meet all their deadlines for all tasks if and only if

$$\forall i,\ 1 \le i \le n, \qquad \min_{(k,l)\in R_i} \sum_{j=1}^{i} C_j \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil$$

$$= \min_{(k,l)\in R_i} \sum_{j=1}^{i} U_j \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \le 1$$

where C_j, T_j, and U_j are the execution time, period, and utilization of task τ_j, respectively, and $R_i = \{(k, l) | 1 \le k \le i, l = 1, \cdots, \lfloor T_i/T_k \rfloor\}$.

Lehoczky, Sha, and Ding 1989

Feasibility test for Earliest Deadline First Scheduling

For a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if

$$(\frac{C_1}{T_1}) + (\frac{C_2}{T_2}) + ...(\frac{C_m}{T_m}) \leq 1$$

References

1. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment C. L. LIU *Project MAC, Massachusetts Institute of Technology* AND JAMES W. LAYLAND *Jet Propulsion Laboratory, California Institute of Technology*

2. Computers as Components, Principles of Embedded Computing System Design, Chapter 6, Wayne Wolf

3. Priority Inheritance Protocols: An Approach to Real-Time Synchronization, Lui Sha, R. Rajkumar, John P. Lehoczky, IEEE TRANSACTIONS ON COMPUTERS, VOL. **39,** NO. 9, SEPTEMBER **1990**