DEADLINE MONOTONIC SCHEDULING

Neil C. Audsley[†]

September 1990

Department of Computer Science,
University of York,
Heslington,
York.
Y01 5DD
neil@uk.ac.york.minster

ABSTRACT

This paper investigates schedulability tests for mixtures of periodic and aperiodic processes. Following an introduction outlining the constraints associated with rate-monotonic scheduling new schedulability tests are presented for deadline-monotonic scheduling. These apply to collections of periodic processes which have periods not necessarily equal to their deadlines (as is the case for rate-monotonic scheduling). The introduction of aperiodic (sporadic) processes can be catered for in the rate-monotonic scheme by the use of sporadic servers. Restrictions on this method are investigated and a new model based on deadline-monotonic scheduling is described. This model has the property that any mixture of sporadic and periodic process deadlines can be guaranteed (subject to passing an appropriate schedulability test).

[†] This work is supported, in part, by the Information Engineering Advanced Technology Programme, Grant GR/F 35920/4/1/1214

Deadline-Monotonic Scheduling

Neil C. Audsley

Department of Computer Science,
University of York,
Heslington,
York.
Y01 5DD
neil@uk.ac.york.minster

1. INTRODUCTION

Hard real-time systems have been defined as those containing processes that have deadlines that cannot be missed [Bur89a]. Such deadlines have been termed hard: they must be met under all circumstances otherwise catastrophic system failure may result [Sta88a, Bur89a, Aud90a].

To meet hard deadlines implies constraints upon the way in which system resources are allocated at runtime. This includes both physical and logical resources. Conventionally, resource allocation is performed by scheduling algorithms whose purpose is to interleave the executions of processes in the system to achieve a pre-determined goal. For hard real-time systems the obvious goal is that no deadline is missed.

One scheduling method that has been proposed for hard real-time systems is the rate monotonic algorithm [Liu73a]. This is a static priority based algorithm for periodic processes in which the priority of a process is related to its period. Whilst this algorithm has several useful properties, including a schedulability test that is sufficient and necessary [Leh89a], the constraints that it imposes on the process system are severe: processes must be periodic, independent and have deadline equal to period.

Many papers have successively weakened the constraints imposed by the rate-monotonic algorithm and have provided associated schedulability tests. Reported work includes a test to allow aperiodic processes to be scheduled [Sha89a], and a test to schedule processes that synchronise using semaphores [Sha88a]. One constraint that has remained is that the deadline and period of a process must be equal.

The weakening of this latter constraint would benefit the application designer by providing a more flexible process model for implementing the system. The increased flexibility is seen by observation: processes with $deadline = period$ are expressible within a process model permitting $deadline \leq period$. For example, process systems whose timing characteristics are suitable for rate-monotonic scheduling would also be accepted by a scheduling scheme permitting deadlines and periods of a process to differ.

This paper relaxes this constraint and so transforms the rate-monotonic algorithm into the deadline-monotonic algorithm. Schedulability tests are developed which guarantee the deadlines of periodic processes. This approach is then shown to be applicable for guaranteeing the deadlines for arbitrary mixtures of periodic and sporadic processes.

The following sub-section gives a brief description of the symbols and terminology used in the remainder of the paper. Section 2 gives an overview of the rate-monotonic scheduling algorithm and associated schedulability tests. Section 3 introduces the deadline-monotonic scheduling algorithm. New schedulability constraints for the algorithm are developed. Section 4 outlines some previously proposed methods of guaranteeing sporadic process deadlines within the context of the rate-monotonic algorithm. The section then proposes a simpler method guaranteeing the deadlines of arbitrary mixtures of sporadic and periodic processes using the deadline-monotonic scheduling algorithm.

1.1. Notation

A process is periodic if it is released for execution in a periodic manner. When this is not the case, and a maximum release frequency can be defined, the process is termed sporadic. If no such maximum can be defined the process is termed aperiodic [Aud90a].

A process is given by τ_i, where i identifies the process. The subscript i is defined to be the priority of that process, where priorities are unique. Priorities are assigned numerically, taken from the interval $\left[1, n\right]$ where 1 is the highest priority and n (the number of processes in the system) the lowest.

The process τ_i has timing characteristics T_i, C_i and D_i. These refer to the value of the period, computation time and deadline of τ_i.

2. THE RATE-MONOTONIC SCHEDULING ALGORITHM

Rate-monotonic scheduling is a static priority based mechanism [Liu73a]. Priorities assigned to processes are inversely proportional to the length of the period. That is, the process with the shortest period is assigned the highest priority. Processes are executed in preemptive manner: at any time, the highest priority process with outstanding computation requirement is executed.

Amongst the class of static priority scheduling schemes, it has been shown that rate-monotonic priority assignment is optimal [Liu73a]. This implies that if a given static priority scheduling algorithm can schedule a process system, the rate-monotonic algorithm is also able to schedule that process system. In the case of the rate-monotonic scheduling algorithm, optimality implies the imposition of constraints upon the process system. These include:

- fixed set of processes;
- all processes are periodic;
- all processes have deadline equal to period;
- one instance of a process must be complete before subsequent instances are run;
- all processes have known worst-case execution times;
- no synchronisation is permitted between processes;
- all processes have an initial release at time 0.

The last of these constraints is fundamental in determining the schedulability of a given process system. When all processes are released simultaneously, we have the worst-case demand for the processor. The times at which all processes are released simultaneously are termed *critical instants*[Liu73a] (thus the first critical instant occurs at time 0). This leads to the observation that if all processes can meet their deadline in the executions starting at the critical instant, then all process deadlines will be met during the lifetime of the system.

Schedulability tests for the rate-monotonic algorithm are based upon the critical instant concept. In [Liu73a] the concept is developed into a schedulability test based upon process utilisations †. The test is given by:

$$\sum_{i=1}^{n} U_i \le n \left(2^{\frac{1}{n}} - 1 \right) \tag{1}$$

where the utilisation U_i of process τ_i is given by:

$$U_i = \frac{C_i}{T_i}$$

The utilisation converges on 69% for large n. Thus if a process set has utilisation of less than 69% it is guaranteed to be scheduled by the rate-monotonic algorithm. That is, all deadlines are guaranteed to be met.

Whilst test (1) is *sufficient*, it is also not *necessary*. That is, the test may indicate falsely that a process system is not schedulable. For example, consider two processes with the following periods and computational requirements:

$$C_1 = 2 \quad T_1 = 4$$

$$C_2 = 4 \quad T_2 = 8$$

For these processes, equation (1) evaluates to false, as the utilisation of two processes is 100%, greater than the allowable bound of 83%. However, when run, neither process will ever miss a deadline. Hence, the test is sufficient but not necessary.

A necessary and sufficient schedulability constraint has been found [Sha88a, Leh89a]. For a set of n processes, the schedulability test is given by ††:

$$\forall\, i : 1 \le i \le n, \ \min_{(k,l) \in R_i} \left(\sum_{j=1}^{i-1} \frac{C_j}{T_j} \frac{T_j}{l\,T_k} \left\lceil \frac{l\,T_k}{T_j} \right\rceil + \frac{C_i}{l\,T_k} \right) \le 1 \tag{2}$$

where

$$R_i = \left\{ (k, l) \mid 1 \le k \le i, l = 1, \ldots, \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$$

The equations take into account all possible process phasings.

2.1. Summary

In summary, schedulability tests are available for an optimal static priority scheduling scheme, the rate-monotonic scheduling algorithm, with processes limited by the following fundamental constraints:

- all processes are periodic;

† Utilisation is a measure of the ratio of required computation time to period of a process. The summation of these ratios over all processes yields the total processor utilisation.

†† Note: $\lceil x \rceil$ evaluates to the smallest integer $\ge x$; $\lfloor x \rfloor$ evaluates to the largest integer $\le x$

- all processes have period equal to deadline;
- no synchronisation is permitted between processes.

The first schedulability test (equation (1)) is sufficient and not necessary; the second test (equation (2)) is sufficient and necessary. One difference between the two schedulability tests lies in their computational complexities. The first test is of $O(n)$ in the number of processes. The second test is far more complicated: its complexity is data dependent. This is because the number of calculations required is entirely dependent on the values of the process periods. In the worst-case, the test can involve enumeration of the schedule for each process in the system, upto the period of that process. Hence, a trade-off exists between accuracy and computational complexity for these schedulability tests.

The following section removes the second of the three constraints (i.e. $period = deadline$). Section 4 then allows processes to be sporadic thus relaxing the first of the above constraints. The third constraint is beyond the scope of this paper but as was noted earlier, Sha *et al* have considered this [Sha88a, Sha90a, Sha87a].

3. DEADLINE MONOTONIC SCHEDULING

We begin by observing that the processes we wish to schedule are characterised by the following relationship:

$$computation \;\; time \;\leq\; deadline \;\leq\; period$$

Leung *et al*[Leu82a] have defined a priority assignment scheme that caters for processes with the above relationship. This is termed inverse-deadline or deadline monotonic priority assignment.

Deadline monotonic priority ordering is similar in concept to rate monotonic priority ordering. Priorities assigned to processes are inversely proportional to the length of the deadline [Leu82a]. Thus, the process with the shortest deadline is assigned the highest priority, the longest deadline process the lowest priority. This priority ordering defaults to a rate monotonic ordering when $period = deadline$.

Deadline monotonic priority assignment is an optimal static priority scheme for processes that share a critical instant. This is stated as Theorem 2.4 in [Leu82a]:

> "The inverse-deadline priority assignment is an optimal priority assignment for one processor."

To generate a schedulability constraint for deadline monotonic scheduling the behaviour of processes released at a critical instant is fundamental: if all processes are proved to meet their deadlines during executions beginning at a critical instant these processes will always meet their deadlines [Liu73a, Leu82a].

Using the results of Leung *et al* stated above as a foundation, new schedulability tests are now developed. Initially, two processes are considered, then we generalise to allow any number of processes.

3.1. Schedulability Of Two Processes

Consider two processes: τ_1 and τ_2. Process τ_1 has a higher priority than process τ_2 and so by deadline monotonic priority assignment,

$$D_1 \; < \; D_2$$

Consider the following case.

Case (i) : both processes are always released simultaneously.

This occurs if the following holds:

$$T_1 = T_2$$

This is illustrated in Figure 1.

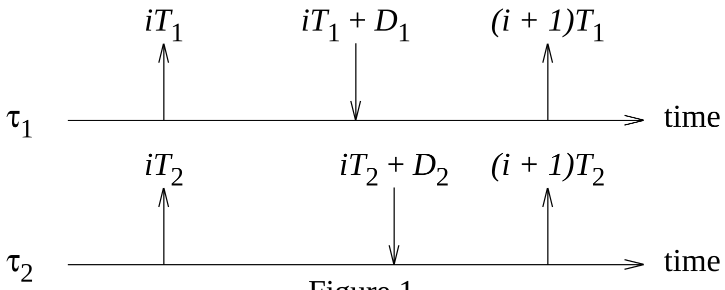


Figure 1.

Since τ_1 has the highest priority, it claims the processor whenever it has an outstanding computational requirement. This will occur for the first C_1 units of each period T_1. The schedulability of this system is given by:

(a) check schedulability of τ_1:

the deadline must be sufficiently large to contain the computation demand, i.e.

$$C_1 \leq D_1$$

hence, τ_1 is schedulable if

$$\frac{C_1}{D_1} \leq 1$$

(b) check schedulability of τ_2:

all higher priority processes (i.e. τ_1) have prior claim on the processor. Hence, in any interval $\left[iT_2 , (i+1)T_2 \right]$, τ_2 can utilise the processor in the interval $\left[iT_2 + C_{1,} \; iT_2 + D_2 \right]$. Therefore, τ_2 can have a maximum computation time defined by:

$$C_2 \leq D_2 - C_1$$

that is τ_2 is schedulable if

$$\frac{C_2}{D_2} + \frac{C_1}{D_2} \leq 1 \qquad (3)$$

■

The second term of equation (3) relates to the maximum time that τ_2 is prevented from executing by higher priority processes, in this case τ_1. This time is termed the *interference time*, I.

Definition 1: I_i is the interference that is encountered by τ_i between the release and deadline of any instance of τ_i. The interference is due to the execution demands of higher priority processes. The maximum interference on process τ_i occurs during a release of τ_i beginning at a critical instant (by definition of critical instant [Liu73a]).

■

Considering the processes in Case (i), I_2 equates to the time that τ_1 executes whilst τ_2 has outstanding computational requirement. Thus I_2 is equal to one computation of τ_1. That is:

$$I_2 = C_1$$

In Case (i) I_1 is zero, as τ_1 is the highest priority process:

$$I_1 = 0$$

The schedulability for Case (i) can be restated:

$$\forall i : 1 \le i \le 2 : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \tag{4}$$

where

$$I_1 = 0$$

$$I_2 = C_1$$

We now consider cases where the periods of the two processes are not equal.

Case (ii) : τ_2 is released many times before the second release of τ_1

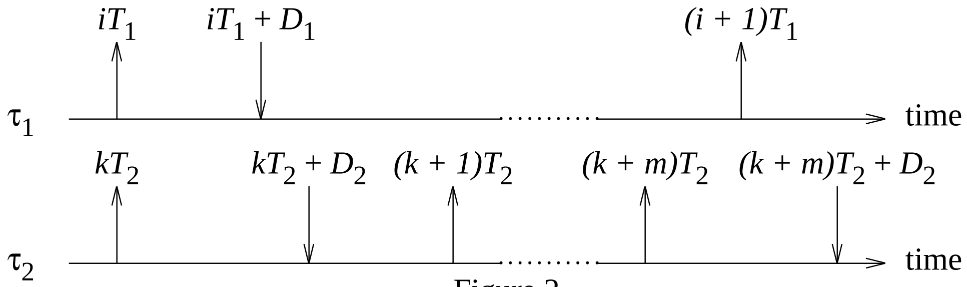


Figure 2.

In Figure 2, the maximum interference I_2 is equal to one computation time of τ_1. The schedulability equations (4) will hold for this case.

■

Case (iii) : τ_1 is released many times before the second release of τ_2

Consider Figure 3.

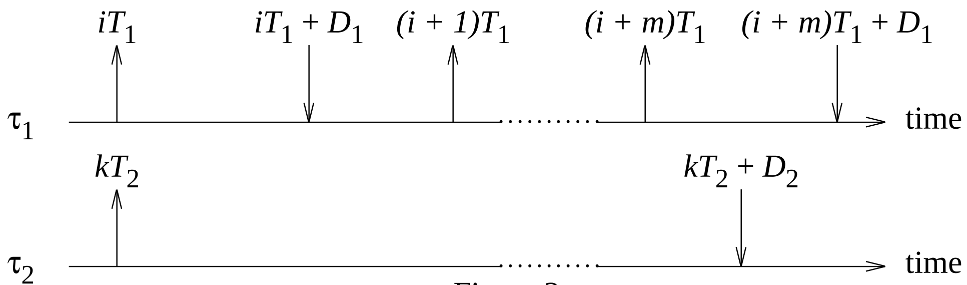


Figure 3.

Clearly, τ_2 is prevented from running by $i + m$ releases of process τ_1. The number of releases of τ_1 within the interval $\left[kT_{2,}\ kT_2 + D_2 \right]$ is given by:

$$\left\lceil \frac{D_2}{T_1} \right\rceil$$

Therefore, schedulability is expressed by:

$$\forall i : 1 \le i \le 2 : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \tag{5}$$

where

$$I_1 = 0$$

$$I_2 = \left\lceil \frac{D_2}{T_1} \right\rceil C_1$$

■

In equation (5), the value of I_2 above could be larger than the exact maximum interference. This is because I_2 includes computation time required by τ_1 for its $(i + m)^{th}$ release, some of which may occur after $(kT_2 + D_2)$. Since the value of I_2 is at least as great as the maximum interference, the test must hold since the it is based upon the exact maximum interference.

An example using schedulability equation (5) is now given.

Example

Consider the following process system.

$$\tau_1 : \ C_1 = 2,\ D_1 = 3,\ T_1 = 5$$

$$\tau_2 : \ C_2 = 6,\ D_2 = 10,\ T_2 = 15$$

The schedulability of the process system can be determined by equation (5).

(a) check process τ_1

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \le 1$$

$$\frac{2}{3} < 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \le 1$$

$$\frac{6}{10} + \frac{I_2}{10} \le 1$$

where

$$I_2 = \left\lceil \frac{10}{5} \right\rceil 2 = 4$$

substituting

$$\frac{6}{10} + \frac{4}{10} = 1$$

Hence τ_2 is schedulable. An example run of the system is given in Figure 4†.

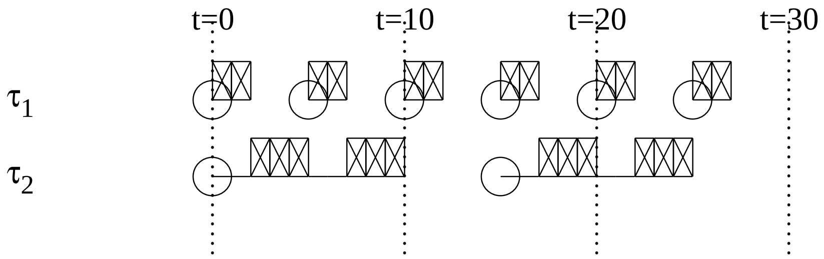


Figure 4.

The construction of I_2 is sufficient but not necessary as the following example shows. Consider the effect of increasing D_2 to 11. This should not affect the schedulability of the system.

(c) recheck process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \le 1$$

$$\frac{6}{11} + \frac{I_2}{11} \le 1$$

where

$$I_2 = \left\lceil \frac{11}{5} \right\rceil 2 = 6$$

substituting

$$\frac{6}{11} + \frac{6}{11} > 1$$

Hence τ_2 is now unschedulable by equation (5). However, the process system is schedulable (Figure 4 above).

† Simulation diagrams are discussed in Appendix 1.

■

The schedulability constraint in equation (5) is too strong due to the value of I_2. An exact expression for I_2 is now developed. Consider Figure 5.

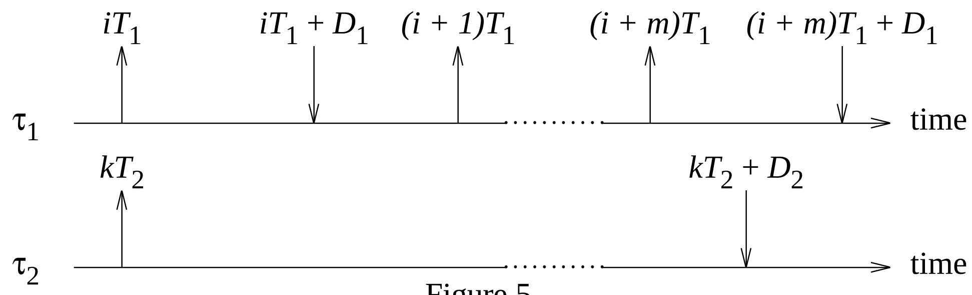


Figure 5.

A critical instant has occurred ($iT_1 = kT_2$) with the interference on τ_2 a maximum. We note that the interference consists of executions of τ_1 that have deadlines before D_2, and the execution of τ_1 that has a release before D_2 and a deadline after D_2. We can restate I_2 as

$$I_2 = b + k \qquad (6)$$

where b represents the interference due to complete executions of τ_1 and k the incomplete executions.

The number of complete executions in the interval $\left[kT_2, kT_2 + D_2 \right]$ is equal to the number of deadlines τ_1 has in this interval. The number is given by:

$$\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1$$

Hence, the interference due to complete executions is given by:

$$b = \left[\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right] C_1$$

The number of incomplete executions of τ_1 is given by the number of releases of τ_1 minus the number of deadlines of τ_1 in $\left[kT_2, kT_2 + D_2 \right]$. This evaluates to either 0 or 1. The number of releases is given by:

$$\left\lceil \frac{D_2}{T_1} \right\rceil$$

Note that if a release of τ_1 coincides with D_2, then it is deemed to occur fractionally after D_2. Hence the number of incomplete executions in $\left[kT_2, kT_2 + D_2 \right]$ is given by:

$$\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right)$$

The start of the incomplete execution is given by:

$$\left\lfloor \frac{D_2}{T_1} \right\rfloor T_1$$

Hence, the length of the interval utilised by the incomplete execution before D_2 is:

$$D_2 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_1$$

The maximum time τ_1 can use during the interval is given by the length of the interval. However, the interval may be longer than C_1. Therefore the maximum interference due to incomplete executions is given by:

$$k = \left[\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) \right] \times \min \left(C_1, D_2 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_1 \right)$$

Substituting b and k into equation (6) gives the following schedulability constraint:

$$\forall i : 1 \le i \le 2 : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \tag{7}$$

where

$$I_1 = 0$$

$$I_2 = \left[\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right] C_1 + \left[\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) \right] \times \min \left(C_1, D_2 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_1 \right)$$

Consider the following theorems which relate to the sufficient and necessary properties of equation (7).

Theorem 1: the schedulability test given in equation (7) is sufficient for two processes.

Proof :

The proof is by contradiction. We assume there is a process system that passes the test but is not schedulable and show that if the system is not schedulable then it must fail the test.

Consider a process system containing τ_1 and τ_2. Let process τ_1 pass the test. Now, let τ_2 pass the test, but not be schedulable. To pass the test, the following must hold:

$$\frac{C_2 + I_2}{D_2} \le 1 \tag{a}$$

For τ_2 not to be schedulable, it must miss its deadline during an instance of the process starting at the critical instant of all processes. At this point τ_2 suffers its maximum interference, I_2, due to the higher priority process. Therefore for τ_2 to miss its deadline and not be schedulable we have:

$$I_2 + C_2 > D_2$$

This gives

$$\frac{I_2}{D_2} + \frac{C_2}{D_2} > 1 \tag{b}$$

A clear contradiction exists between (a) and (b). Therefore, if τ_2 passes the test, it is schedulable.

■

The proof for Theorem 1 relies upon I_2 being exact (this is given by Theorem 2).

Theorem 1 will still hold if I_2 is greater than the exact value. This merely represents a worse than worse-case. Therefore, by implication of Theorem 1, the schedulability test given by equation (5) is also sufficient.

Theorem 2: the schedulability test is necessary if values of I_i are exact.

For process τ_2 to pass the schedulability test requires:

$$\frac{I'_2}{D_2} + \frac{C_2}{D_2} \le 1$$

where I'_2 represents the exact value of I_2. When comparing I'_2 and I_2 we have three cases:

(i) $I'_2 > I_2$ – this is clearly impossible as we know that I_2 is at least I'_2 from the above discussion.

(ii) $I'_2 < I_2$ – this occurs when we have made a pessimistic calculation for I_2. As I_2 increases, the computation time that could be guaranteed for τ_2 decreases since:

$$C_i \le D_2 - I_2$$

(iii) $I'_2 = I_2$ – this occurs when the calculation of I_2 is precise. The allowable computation time for τ_2 is maximised (by above inequality).

In summary, we have the greatest amount of time for τ_2 if I_2 is exact. Therefore, the schedulability test is necessary if I_2 is exact.

■

Therefore, the schedulability test given by equation (7) is necessary as the values for I_i are exact. By implication, the schedulability test given by equation (5) is also necessary if I_i is exact. However, I_i values in equation (7) are exact in more instances than in equation (5): the former will declare more process systems schedulable than the latter. The following example illustrates this point.

Example

We return to the process system that failed equation (5) but was illustrated to meet all deadlines.

$$\tau_1 : \; C_1 = 2, \; D_1 = 3, \; T_1 = 5$$

$$\tau_2 : \; C_2 = 6, \; D_2 = 11, \; T_2 = 15$$

The schedulability of the system can be determined by equation (7).

(a) check process τ_1:

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \le 1$$

$$\frac{2}{3} < 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \le 1$$

$$\frac{6}{11} + \frac{I_2}{11} \le 1$$

where

$$I_2 = \left(\left\lfloor \frac{11-3}{5} \right\rfloor + 1 \right) 2 +$$

$$\left(\left\lceil \frac{11}{5} \right\rceil - \left(\left\lfloor \frac{11-3}{5} \right\rfloor + 1 \right) \right) \times \min \left(2, 11 - \left\lfloor \frac{11}{5} \right\rfloor 5 \right)$$

$$I_2 = 4 + 1 = 5$$

substituting,

$$\frac{6}{11} + \frac{5}{11} = 1$$

Hence τ_2 is schedulable.

The system is schedulable by equation (7). A simulated run of the system was given in Figure 4 previously.

■

3.1.1. Summary

Noting the results stated in [Leu82a] that deadline monotonic priority assignment is optimal, two schedulability tests for two-process systems have been developed. The test in equation (5) is sufficient but not necessary, whilst the test in equation (7) is sufficient and necessary (and hence optimal).

One difference between the tests is that the former is of computational complexity $O(n)$ and the latter $O(n^2)$. A trade-off again exists between accuracy and computational complexity.

3.2. Schedulability Of Many Processes

The schedulability test given by equations (5) and (7) are now generalised for systems with arbitrary numbers of processes. Firstly, equation (5) is expanded. Consider Figure 6.

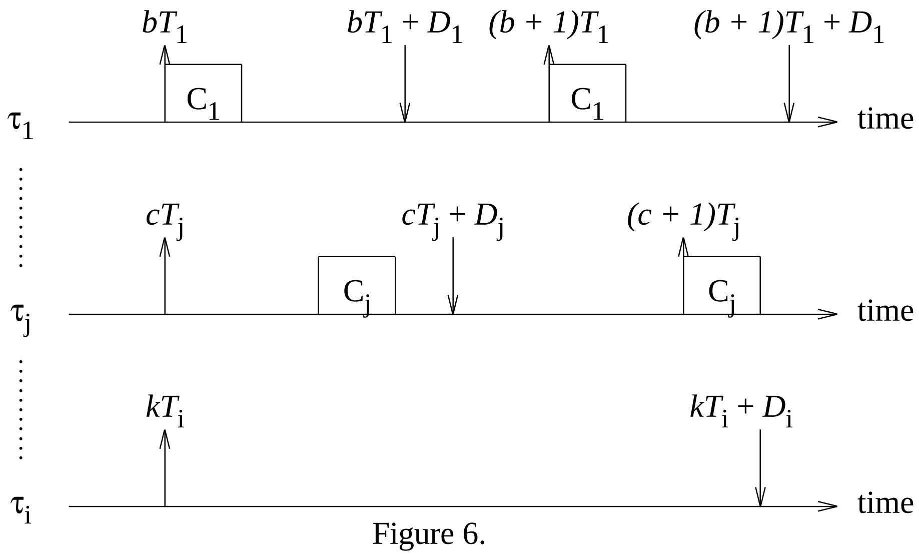


Figure 6.

The interference I_i that is inflicted upon process τ_i by all higher priority processes corresponds to the computation demands by those processes in the interval of time from the critical instant to the first deadline of τ_i.

The interference on τ_i by τ_j can be given by:

$$\left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

This may include part of an execution of τ_j that occurs after D_i. The total interference on τ_i can be expressed by:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Therefore to feasibly schedule all processes:

$$\forall i : 1 \le i \le n : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \qquad (8)$$

where

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Equation (8), like equation (5), is sufficient but not necessary. This is illustrated by the following example.

Example

Consider the following process system.

$$\tau_1 : C_1 = 2, D_1 = 3, T_1 = 5$$

$$\tau_2 : C_2 = 2, D_2 = 6, T_2 = 15$$

$$\tau_3 : C_3 = 4, D_3 = 10, T_3 = 20$$

The schedulability of the process system can be determined by equation (8).

(a) check process τ_1

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \leq 1$$

$$\frac{2}{3} < 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \leq 1$$

$$\frac{2}{6} + \frac{I_2}{6} \leq 1$$

where

$$I_2 = \left\lceil \frac{6}{5} \right\rceil 2 = 4$$

substituting

$$\frac{2}{6} + \frac{4}{6} = 1$$

Hence τ_2 is schedulable.

(c) check process τ_3

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} \leq 1$$

$$\frac{4}{10} + \frac{I_3}{10} \leq 1$$

where

$$I_3 = \left\lceil \frac{D_3}{T_1} \right\rceil C_1 + \left\lceil \frac{D_3}{T_2} \right\rceil C_2$$

$$I_3 = \left\lceil \frac{10}{5} \right\rceil 2 + \left\lceil \frac{10}{15} \right\rceil 2 = 6$$

substituting

$$\frac{4}{10} + \frac{6}{10} = 1$$

Hence τ_3 is schedulable.

Consider the effect of increasing D_3 to 11. This should not affect the schedulability of the system.

(d) recheck process τ_3

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} \leq 1$$

$$\frac{4}{11} + \frac{I_3}{11} \leq 1$$

where

$$I_3 = \left\lceil \frac{D_3}{T_1} \right\rceil C_1 + \left\lceil \frac{D_3}{T_2} \right\rceil C_2$$

$$I_3 = \left\lceil \frac{11}{5} \right\rceil 2 + \left\lceil \frac{11}{15} \right\rceil 2 = 8$$

substituting

$$\frac{4}{11} + \frac{8}{11} > 1$$

Hence τ_3 is unschedulable by equation (8).

The process system is unschedulable by equation (8). However, when the system is run all deadlines are met (see Figure 7).

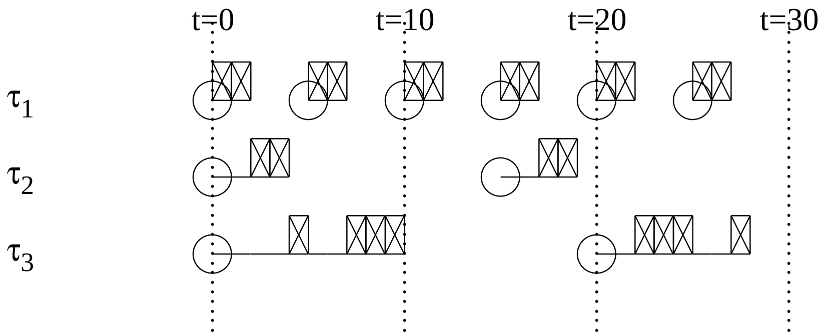


Figure 7.

■

In the above example, the process system is not schedulable by equation (8) because the values of I_i are greater than exact values. Each I_i can contain parts of executions that occur after D_i. This is similar to the drawback of equation (5) when a two-process system was being considered. To surmount this problem we generalise equation (7) for many processes. Consider Figure 8.

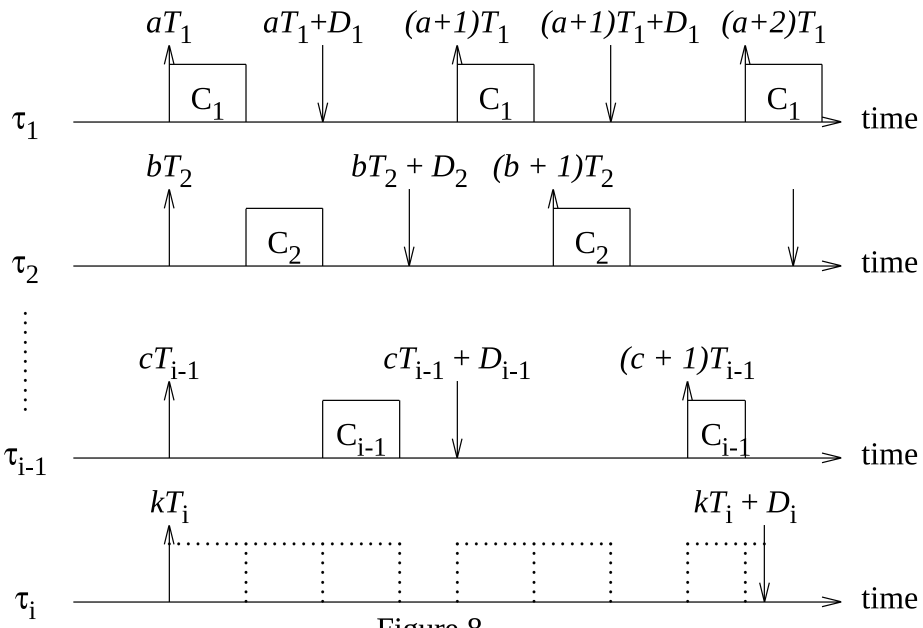


Figure 8.

From Figure 8, it can be seen that in the general case with n processes, I_i is equal to the interference of all the processes τ_1 to τ_{i-1} in the interval $\left[kT_i, kT_i + D_i \right]$. Thus, equation (7) can be rewritten to provide a schedulability test for an n process system:

$$\forall i : 1 \le i \le n : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \tag{9}$$

where

$$I_i = \sum_{j=1}^{i-1} \left[\left[\left\lfloor \frac{D_i - D_j}{T_j} \right\rfloor + 1 \right] C_j + \left[\left\lceil \frac{D_i}{T_j} \right\rceil - \left(\left\lfloor \frac{D_i - D_j}{T_j} \right\rfloor + 1 \right) \right] \times \min \left(C_j , D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right) \right]$$

L1

To show that the above constraint is more accurate than equation (8) consider the following example.

Example

We return to the process system that failed equation (8) but was shown to meet all deadlines (see Figure 7).

$$\tau_1 : C_1 = 2, D_1 = 3, T_1 = 5$$

$$\tau_2 : C_2 = 2, D_2 = 6, T_2 = 15$$

$$\tau_3 : C_3 = 4, D_3 = 11, T_3 = 20$$

(a) check process τ_1

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \le 1$$

$$\frac{2}{3} \le 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \le 1$$

$$\frac{2}{6} + \frac{I_2}{6} \le 1$$

where

$$I_2 = \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1\right) C_1 + \left(\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1\right)\right) \times \min\left(C_1, D_2 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_2\right)$$

$$I_2 = \left(\left\lfloor \frac{6-3}{5} \right\rfloor + 1\right) 2 + \left(\left\lceil \frac{6}{5} \right\rceil - \left(\left\lfloor \frac{6-3}{5} \right\rfloor + 1\right)\right) \times \min\left(2, 6 - \left\lfloor \frac{6}{5} \right\rfloor 5\right) = 3$$

substituting

$$\frac{2}{6} + \frac{3}{6} < 1$$

Hence τ_2 is schedulable.

(c) check process τ_3

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} \le 1$$

$$\frac{3}{11} + \frac{I_3}{11} \le 1$$

where

$$I_3 = \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1\right) C_1 + \left(\left\lceil \frac{D_3}{T_1} \right\rceil - \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1\right)\right) \times \min\left(C_1, D_3 - \left\lfloor \frac{D_3}{T_1} \right\rfloor T_1\right)$$
$$+ \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1\right) C_2 + \left(\left\lceil \frac{D_3}{T_2} \right\rceil - \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1\right)\right) \times \min\left(C_2, D_3 - \left\lfloor \frac{D_3}{T_2} \right\rfloor T_2\right)$$

$$I_3 = \left(\left\lfloor \frac{7}{5} \right\rfloor + 1\right) 2 + \left(\left\lceil \frac{11}{5} \right\rceil - 2\right) \times \min\left(2, 11 - \left\lfloor \frac{11}{5} \right\rfloor 5\right)$$
$$+ \left(\left\lfloor \frac{5}{10} \right\rfloor + 1\right) 2 + \left(\left\lceil \frac{11}{10} \right\rceil - 1\right) \times \min\left(2, 11 - \left\lfloor \frac{11}{10} \right\rfloor 10\right)$$

$$I_3 = 4 + 1 + 2 + 1 = 8$$

substituting

$$\frac{3}{11} + \frac{8}{11} = 1$$

Hence τ_3 is schedulable. An example run was given in Figure 7.

■

The expression for I_i in equation (9) is not exact. This is because the interference on τ_i due to incomplete executions of τ_1 to τ_{i-1} given by (9) is greater than or equal to the exact interference. Consider the interference on τ_i by incomplete executions of τ_1 and τ_{i-1} (Figure 8). Within I_i, allowance is made for τ_1 using all of $\left[(a+2)T_1, D_i\right]$, and for τ_{i-1} using all of $\left[(c+1)T_{i-1}, D_i\right]$. Since only one of these processes can execute at a time, I_i is greater than a precise value for the interference.

Consider the following theorems.

Theorem 3: the schedulability test given by equation (9) is sufficient.

Proof :

The proof follows from Theorem (1).

■

Theorem 4: the schedulability test given by equation (9) is necessary if values of I_i are exact.

Proof :

The proof follows from Theorem (2).

■

Theorems (3) and (4) show that both equations (9) and (8) (by implication) are sufficient and not necessary. When no executions of higher priority processes overlap the deadline of τ_i then I_i will be exact with both tests (8) and (9) being necessary. Indeed, if the I_i values in both equations (8) and (9) are exact, the two equations are equivalent. However, when executions do overlap the deadline of τ_i test (9) will pass more process systems than test (8) as it contains a more precise measurement of I_i.

To obtain an exact value for I_i under all cases requires the exact interleaving of all higher priority processes to be considered upto the deadline D_i. This could involve the enumeration of the schedule upto D_i with obvious computational expense. The following section outlines an alternative strategy for improving the schedulability constraint.

3.3. Unschedulability of Many Processes

The previous section developed a sufficient and not necessary test for the schedulability of a process system. We note that whilst this test identifies some of the schedulable process systems, a sufficient and not necessary unschedulability test will identify some of the unschedulable systems. This approach is illustrated by Figure 9.

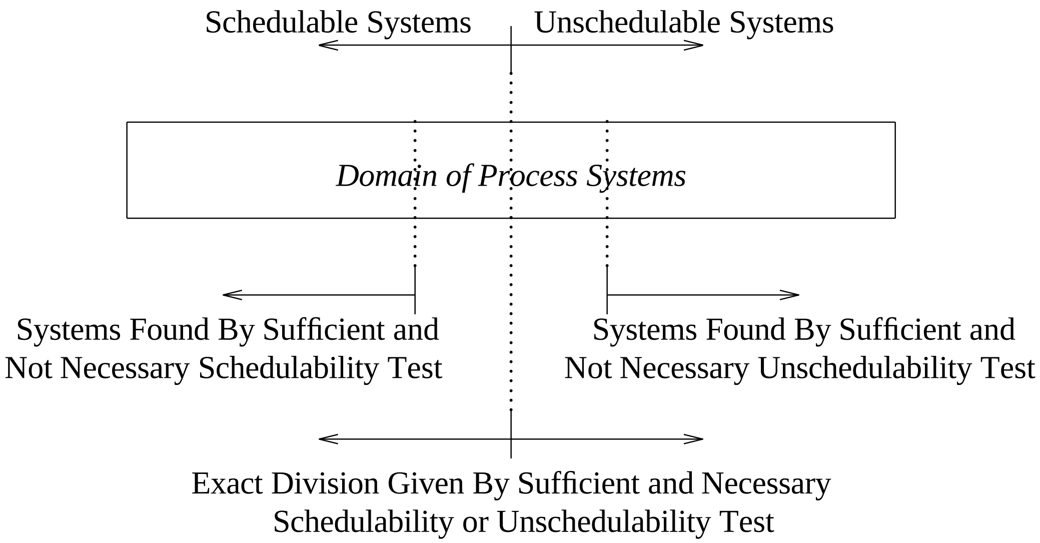

Schedulable Systems
Unschedulable Systems
Domain of Process Systems
Systems Found By Sufficient and Not Necessary Schedulability Test
Systems Found By Sufficient and Not Necessary Unschedulability Test
Exact Division Given By Sufficient and Necessary Schedulability or Unschedulability Test

Figure 9.

A sufficient and not necessary unschedulability test identifies some unschedulable process systems in the same manner as the test in the previous sub-section identifies schedulable systems. The combination of the two tests enables the identification of many schedulable and unschedulable process systems without resorting to a computationally expensive sufficient and necessary test. A sufficient and not necessary unschedulability test is now presented.

Consider the interference of higher priority processes upon τ_i. This is at a minimum when any incomplete executions of higher priority processes occur as late as possible. This maximises the time utilised by higher priority processes after D_i and minimises the time utilised before D_i.

Theorem 5 : I_i is at a minimum when incomplete executions of higher priority processes perform their execution as late as possible.

Proof :

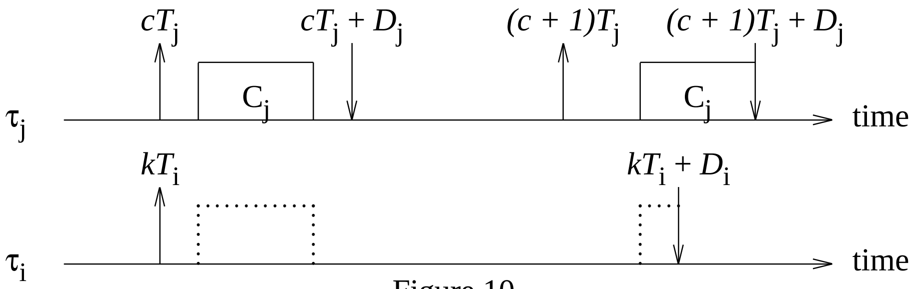


Figure 10.

Consider Figure 10. The execution of τ_j in $\left[\,(c+1)T_j, (c+1)T_j + D_j\,\right]$ is decreased by moving the execution towards the deadline of τ_j (which is after D_i). This movement decreases I_i. I_i will be at a minimum when the execution has been moved as close as possible to the deadline of τ_j.

■

Consider the schedulability of τ_i. When I_i is a minimum, we have the best possible scenario for scheduling τ_i. If τ_i cannot be scheduled with I_i a minimum, it cannot be scheduled with an exact I_i since this value is as least as large as the minimum value. Therefore, to show the unschedulability of a process system, it is sufficient to show the unschedulability of the system with minimum values of I_i.

An unschedulability test is now developed using minimum interference. In Figure 10, the interference on τ_i is the sum of complete executions of higher priority processes, and the parts of incomplete executions that must occur before $kT_i + D_i$. Let b be the complete executions and k the incomplete executions. The total interference is stated as:

$$I_i = b + k \tag{10}$$

Complete executions occur in the interval $\left[kT_i, kT_i + D_i \right]$ and are given by:

$$b = \sum_{j=1}^{i-1} \left(\left\lfloor \frac{D_i - D_j}{T_j} \right\rfloor + 1 \right) C_j$$

The incomplete executions number either 0 or 1 for each of the processes with a higher priority than τ_i. Hence, the interference due to incomplete executions can be stated as

$$k = \sum_{j=1}^{i-1} \left(\left\lceil \frac{D_i}{T_j} \right\rceil - \left(\left\lfloor \frac{D_j - T_i}{T_i} \right\rfloor + 1 \right) \right) \times \max\left(0, C_j - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j - D_j + D_i \right)$$

Substituting into equation (10) we generate an unschedulability test:

$$\forall i : 1 \le i \le n : \frac{C_i}{D_i} + \frac{I_i}{D_i} > 1 \tag{11}$$

where

$$I_i = \sum_{j=1}^{i-1} \left[\left(\left\lfloor \frac{D_i - D_j}{T_j} \right\rfloor + 1 \right) C_j + \left(\left\lceil \frac{D_i}{T_j} \right\rceil - \left(\left\lfloor \frac{D_j - T_i}{T_i} \right\rfloor + 1 \right) \right) \times \max\left(0, C_j - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j - D_j + D_i \right) \right]$$

We note that only one process need pass the unschedulability test for the process system to be unschedulable.

The converse of Theorem 3 proves equation (11) to be a sufficient condition for unschedulability. This follows from the observation that since I_i is a minimum (by Theorem 5), then by Theorem 5 if a process system cannot be scheduled with I_i less than an exact value, the process system cannot be scheduled with exact values of I_i. By Theorem 4, we note that equation (11) is a not necessary condition for unschedulability since the values used for I_i are less than or equal to the exact value for I_i.

Equations (9) and (11) can be used together. Consider a process system that fails equation (9). Since this is test is not necessary it does not prove the process system unschedulable. The same process system can be submitted to equation (10). If the system passes equation (10) we have determined the unschedulability of the process system. However, if the process system failed both schedulability and unschedulability tests we note that it could still be schedulable.

We illustrate the use combined use of equations (9) and (11) with the following example.

Example

Consider the following process system.

$$\tau_1 : \; C_1 = 4, \; D_1 = 6, \; T_1 = 10$$

$$\tau_2 : \; C_2 = 3, \; D_2 = 7, \; T_2 = 11$$

$$\tau_3 : \; C_3 = 7, \; D_3 = 13, \; T_3 = 20$$

We can show the unschedulability of the system by using equation (11).

(a) check process τ_1:

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} > 1$$

$$\frac{4}{6} < 1$$

Hence τ_1 fails the test and is therefore not unschedulable.

(b) check process τ_2:

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} > 1$$

$$\frac{3}{7} + \frac{I_2}{7} > 1$$

where

$$I_2 = \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) C_1 +$$

$$\left(\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) \right) \times \max \left(0, C_1 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_1 - D_1 + D_2 \right)$$

$$I_2 = 4 + 0 = 4$$

substituting

$$\frac{3}{7} + \frac{4}{7} = 7$$

Hence τ_2 fails the test and is therefore not unschedulable.

(c) check process τ_3:

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} > 1$$

$$\frac{7}{13} + \frac{I_3}{13} > 1$$

where

$$I_3 = \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1 \right) C_1$$

$$+ \left(\left\lceil \frac{D_3}{T_1} \right\rceil - \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1 \right) \right) \times \max \left(0, C_1 - \left\lfloor \frac{D_1}{T_3} \right. - D_3 + D_1 \right)$$

$$+ \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1 \right) C_2$$

$$+ \left(\left\lceil \frac{D_3}{T_2} \right\rceil - \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1 \right) \right) \times \max \left(0, C_2 - \left\lfloor \frac{D_2}{T_3} - D_3 + D_2 \right) \right.$$

$$I_3 = 4 + \max \left(0, -14 \right) + 3 + \max \left(0, -13 \right) = 7$$

substituting

$$\frac{7}{13} + \frac{7}{13} > 1$$

Therefore τ_3 passes the unschedulability test. The process system is therefore unschedulable. An example run of the system is given in Figure 11. Process τ_3 misses its deadline at time 13.

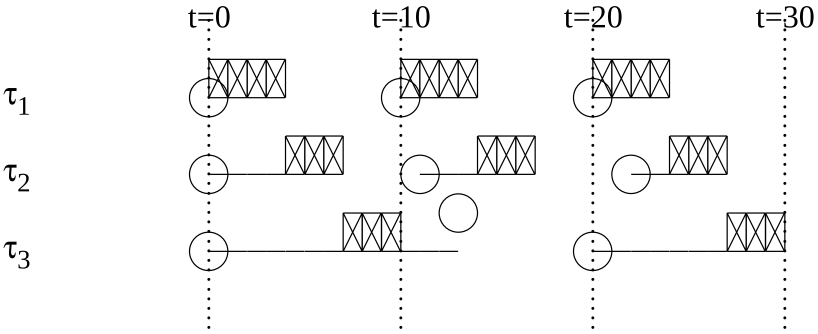


Figure 11.

We now reduce the computation time of process τ_3 to 5:

$$\tau_3 : \ C_3 = 5, \ D_3 = 13, \ T_3 = 20$$

By observation, we can see that τ_3 now fails the unschedulability test:

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} > 1$$

$$\frac{5}{13} + \frac{7}{13} < 1$$

Since the characteristics of the first two processes are identical, the process system as a whole fails the unschedulability test. However, the system is not necessarily unschedulable. Now we try to prove the process system schedulable using equation (9).

(a) check process τ_1

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \leq 1$$

$$\frac{4}{6} \leq 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \leq 1$$

$$\frac{3}{7} + \frac{I_2}{7} \le 1$$

where

$$I_2 = \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) C_1 + \left(\left\lceil \frac{D_2}{T_1} \right\rceil - \left(\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor + 1 \right) \right) \times \min \left(C_1, D_2 - \left\lfloor \frac{D_2}{T_1} \right\rfloor T_2 \right)$$

$$I_2 = 3$$

substituting

$$\frac{3}{7} + \frac{3}{7} < 1$$

Hence τ_2 is schedulable.

(c) check process τ_3

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} \le 1$$

$$\frac{5}{13} + \frac{I_3}{13} \le 1$$

where

$$I_3 = \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1 \right) C_1 + \left(\left\lceil \frac{D_3}{T_1} \right\rceil - \left(\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor + 1 \right) \right) \times \min \left(C_1, D_3 - \left\lfloor \frac{D_3}{T_1} \right\rfloor T_1 \right)$$

$$+ \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1 \right) C_2 + \left(\left\lceil \frac{D_3}{T_2} \right\rceil - \left(\left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor + 1 \right) \right) \times \min \left(C_2, D_3 - \left\lfloor \frac{D_3}{T_2} \right\rfloor T_2 \right)$$

$$I_3 = 4 + \min \left(4, 3 \right) + 3 + \min \left(3, 2 \right) = 12$$

substituting

$$\frac{5}{13} + \frac{12}{13} > 1$$

Hence τ_2 is not schedulable by equation (9).

In the above, we have shown the process system failing both the schedulability and unschedulability tests. Since both tests are sufficient and not necessary we have not decisively proved the process system schedulable or unschedulable.

■

The above example illustrated the combined use of unschedulability and schedulability tests. The first part of the example utilised the unschedulability test to prove the test unschedulable. Then, by decreasing the computation time of τ_3, the system fails the unschedulability test. However, after application of equation (9), the system was shown to fail the schedulability test also. Indeed, by examining the example we can see that when C_3 lies in $\left[1 \right]$ the system can be proved schedulable. When C_3 lies in $\left[7, \infty \right)$ the system can be proved unschedulable. When C_3 lies in $\left[2, 6 \right]$ we can not prove the system schedulable nor unschedulable. This requires a more powerful schedulability test. Such a test is presented in the next sub-section.

3.4. Exact Schedulability of Many Processes

The schedulability and unschedulability constraints for systems containing many processes, given by equations (9) and (11) respectively, are sufficient and not necessary in the general case. To form a sufficient and necessary schedulability test requires exact values for I_i (by Theorems 2 and 4). To achieve this, the schedule has to be evaluated so that the exact interleaving of higher priority process executions is known. This is costly if the entire interval between the critical instant and the deadline of process τ_i is evaluated as this would require the solution of D_i equations.

The number of equations can be reduced by observing that if τ_i meets its deadline at t'_i, where t'_i lies in $\left[0, D_i\right]$, we need not evaluate the equations in $\left(t'_i, D_i\right]$. Further reductions in the number of equations requiring solution can be made by considering the behaviour of the processes in the interval $\left[0, t'_i\right]$.

Consider the interaction of processes τ_1 to τ_{i-1} on process τ_i in the interval $\left[0, t'_i\right]$. For process τ_i to meet its deadline at D_i we require the following condition to be met:

$$\frac{I_i}{D_i} + \frac{C_i}{D_i} \le 1$$

We wish to consider only the points in D_i upto and including t'_i. Therefore, we need to refine the definition of interference on τ_i so that we can reason about the interval $\left[0, t'_i\right]$ rather than the single point in time D_i.

Definition 2: I_i^t is the interference that is encountered by τ_i between the release of τ_i and time t, where t lies in the interval $\left[0, D_i\right]$. This is equal to the quantity of work that is created by releases of higher priority processes in the interval between the release of τ_i and time t.

■

At t'_i the outstanding work due to higher priority processes must be 0 since τ_i can only execute if all higher priority processes have completed. Hence, the point in time at which τ_i actually meets its deadline is given by:

$$\frac{I_i^t}{t'_i} + \frac{C_i}{t'_i} = 1$$

Therefore, we can state the following condition for the schedulability of τ_i:

$$\exists\, t_{0 \le t \le D_i} : \frac{I_i^t}{t} + \frac{C_i}{t} = 1 \qquad (12)$$

where

$$I_i^t = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

We note that the definition of I_i^t includes parts of executions that may occur after t. However, since the outstanding workload of all processes is 0 at t'_i then when $t = t'_i$ the expression I_i^t is exact.

The above equations require a maximum of D_i calculations to be made to determine the schedulability of τ_i. For an n process system the maximum number of equations that need to be evaluated is:

$$1 + \sum_{i=2}^{n} D_i$$

The number of equations that need to be evaluated can be reduced. This is achieved by limiting the points in $\left[0, D_i\right]$ that are considered as possible solutions for t'_i. Consider the times within $\left[0, D_i\right]$ that τ_i could possibly meet its deadline. We note that I_i^t is monotonically increasing within the time interval $\left[0, D_i\right]$. The points in time that the interference increases occur when there is a release of a higher priority process. This is illustrated by Figure 12.

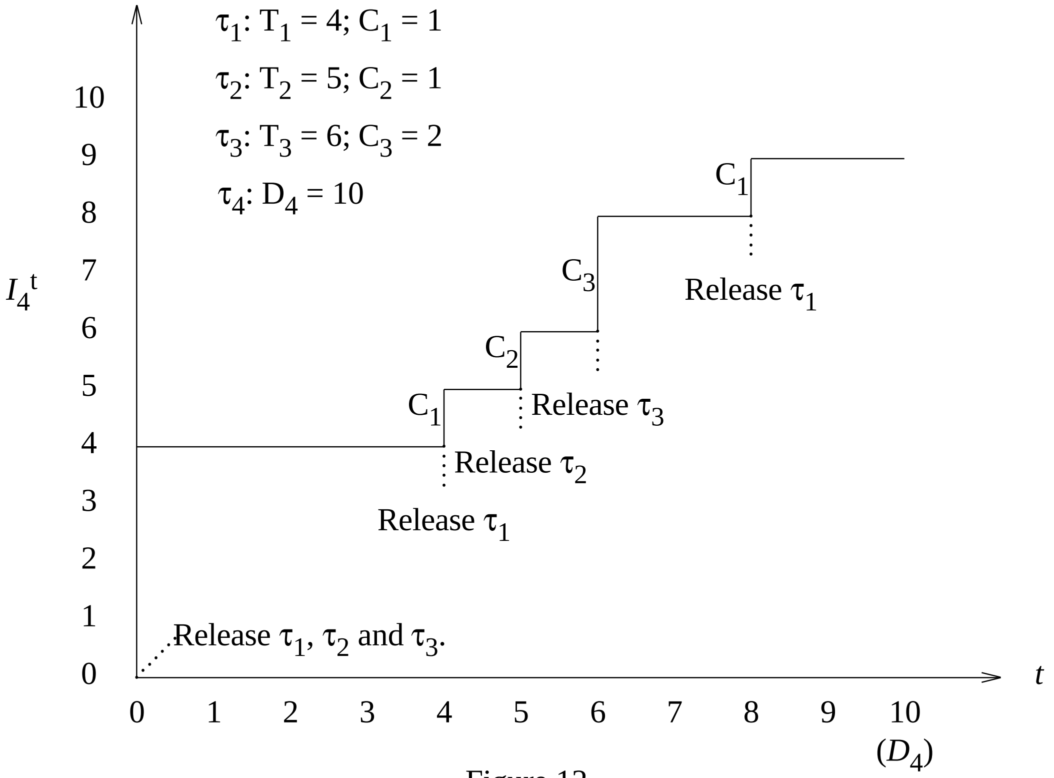


Figure 12.

In Figure 12, there are three processes with higher priority than τ_4. We see that as the higher priority processes are released, I_4^t increases monotonically with respect to t. The graph is stepped with plateaus representing intervals of time in which no higher priority processes are released. It is obvious that only one equation need be evaluated for each plateau as the interference does not change.

To maximise the time available for the execution of τ_i we choose to evaluate at the right-most point on the plateau. Therefore, one possible reduction in the number of equations to evaluate schedulability occurs by testing τ_i at all points in $\left[0, D_i\right]$ that correspond to a higher priority process release. Since as soon as one equation identifies the process system as schedulable we need test no further equations. Thus, the effect is to evaluate equations in $\left[0, t'_i\right]$.

The number of equations has been reduced in most cases. We note that no reduction will occur if for each point in time in $\left[0, D_i\right)$ a higher priority process is released with τ_i meeting its deadline at D_i.

The number of equations is reduced further by considering the computation times of the processes. Consider Figure 13.

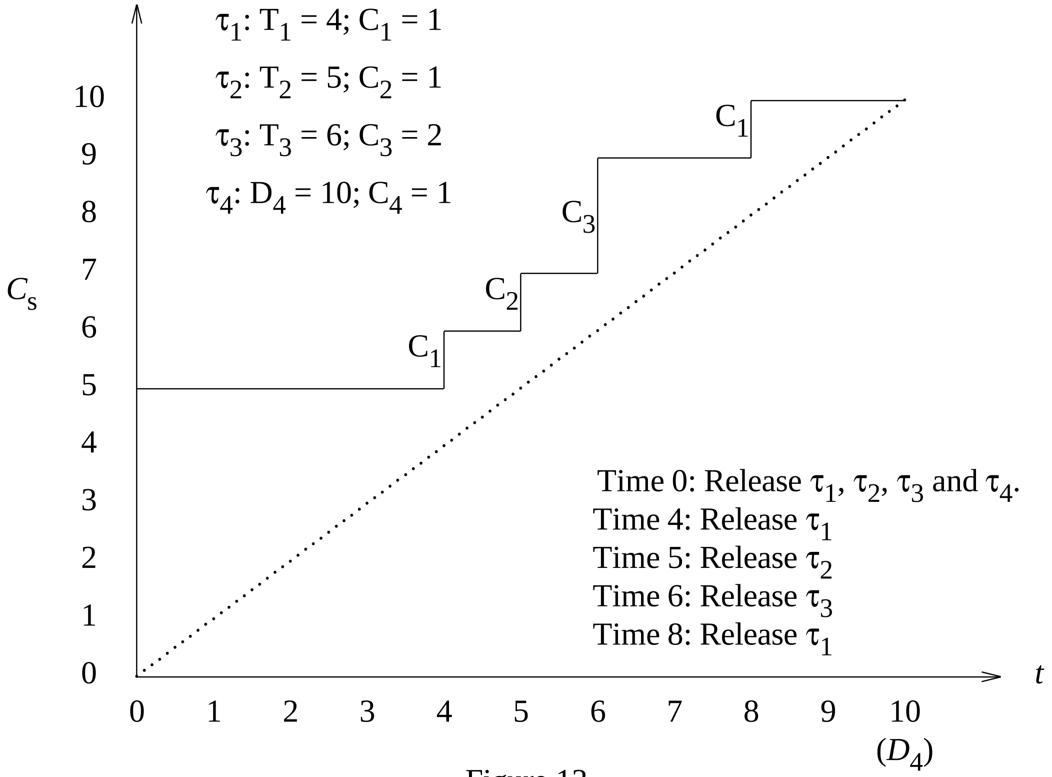


Figure 13.

In Figure 13 the total computation requirement of the system (C_s) is plotted against time. At the first point in time when the outstanding computation is equal to the time elapsed, we have found t'_4 (by equation (12)). In the above diagram this point in time coincides with the deadline of τ_4.

Considering Figure 13, there is no point in testing the schedulability of τ_i in the interval $\left[0, C_i\right)$. Also, since time 0 corresponds with a critical instant (a simultaneous release of all processes) the first point in time that τ_i could possibly complete is:

$$t_0 = \sum_{j=1}^{i} C_i$$

This gives a schedulability constraint of:

$$\frac{I_i^{t_0}}{t_0} + \frac{C_i}{t_0} \leq 1$$

Since the value of t_1 assumes that only one release of each process occurs in $\left[0, t_0\right)$, the constraint will fail if there have been any releases of higher priority processes within the interval $\left[0, t_0\right)$. The exact amount of work created by higher priority processes in this interval is given by:

$$I_i^{t_0}$$

The next point in time at which τ_i may complete execution is:

$$t_1 = I_i^{t_0} + C_i$$

This gives a schedulability constraint of:

$$\frac{I_i^{t_1}}{t_1} + \frac{C_i}{t_1} \le 1$$

Again, the constraint will fail if releases have occurred in the interval $\left[t_0, t_1\right)$. Thus, we can build a series of equations to express the schedulability of τ_i.

$$(i) \qquad \frac{I_i^{t_0}}{t_0} + \frac{C_i}{t_0} \le 1 \tag{13}$$

$$where \qquad t_0 = \sum_{j=1}^{i} C_j$$

$$(ii) \qquad \frac{I_i^{t_1}}{t_1} + \frac{C_i}{t_1} \le 1$$

$$where \qquad t_1 = I_i^{t_0} + C_i$$

$$(iii) \qquad \frac{I_i^{t_2}}{t_2} + \frac{C_i}{t_2} \le 1$$

$$where \qquad t_2 = I_i^{t_1} + C_i$$

$$\vdots$$

$$(q) \qquad \frac{I_i^{t_k}}{t_k} + \frac{C_i}{t_k} \le 1$$

$$where \qquad t_k = I_i^{t_{k-1}} + C_i$$

If any of the equations hold, τ_i is schedulable. The series of equations above is encapsulated by the following algorithm:

Algorithm

```
foreach $\tau_i$ do
    $t = \sum_{j=1}^{i} C_j$
    continue = TRUE
    while (continue) do
        if $\left( \frac{I_i^t}{t} + \frac{C_i}{t} \le 1 \right)$
            continue = FALSE      /* $\tau_i$ is schedulable */
        else
            $t = I_i^t + C_i$
        endif
        if ( $t > D_i$ )
            exit          /* $\tau_i$ is unschedulable */
        endif
    endwhile
endfor
```

■

The algorithm terminates as the following relation always holds.

$$t_i > t_{i-1}$$

When t_i is greater than D_i the algorithm terminates since τ_i is unschedulable. Thus we have a maximum number of steps of D_i. This is a worst-case measure.

The number of equations has been reduced from the method utilising plateaus in Figure 11. This is because we consider only the points in time where it is possible for τ_i to complete, rather than points in time that correspond to higher priority process releases.

An example use of the above algorithm is now given:

Example

We return to the process system which could not be proved schedulable nor unschedulable:

$$\tau_1 : C_1 = 4,\ D_1 = 6,\ T_1 = 10$$

$$\tau_2 : C_2 = 3,\ D_2 = 7,\ T_2 = 11$$

$$\tau_3 : C_3 = 5,\ D_3 = 13,\ T_3 = 20$$

Processes τ_1 and τ_2 were proved schedulable in the previous example so we confine attention to τ_3. We use the successive equations to show unschedulability.

(i)

$$\frac{I_3^{t_0}}{t_0} + \frac{C_3}{t_0} \le 1$$

where

$$t_0 \ = \ \sum_{j\,=\,1}^{3} C_i \ = \ 4+3+5 \ = \ 12$$

substituting

$$\frac{I_3^{12}}{12} \ + \ \frac{5}{12} \ \leq \ 1$$

where

$$I_3^{12} \ = \ \sum_{j=1}^{2} \left\lceil \frac{12}{T_j} \right\rceil \ C_j$$

$$= \ \left\lceil \frac{12}{10} \right\rceil 4 \ + \ \left\lceil \frac{12}{11} \right\rceil 3 \ = \ 14$$

substituting

$$\frac{14}{12} \ + \ \frac{5}{12} \ > \ 1$$

The process is unschedulable at time 12, so we proceed to the next equation.

(ii)

$$\frac{I_3^{t_1}}{t_1} \ + \ \frac{C_3}{t_1} \ \leq \ 1$$

where

$$t_1 \ = \ I_3^{t_0} \ + \ C_3 \ = \ I_3^{12} \ + \ 1 \ = \ 15$$

Since we now have $t_1 > D_3$ we terminate with τ_3 unschedulable.

We reduce the computation time of τ_3 to 3:

$$\tau_3 : \ C_3 = 3, \ D_3 = 13, \ T_3 = 20$$

We use the successive equations to show τ_3 schedulable.

(i)

$$\frac{I_3^{t_0}}{t_0} \ + \ \frac{C_3}{t_0} \ \leq \ 1$$

where

$$t_0 \ = \ \sum_{j\,=\,1}^{3} C_j \ = \ 4+3+3 \ = \ 10$$

substituting

$$\frac{I_3^{10}}{10} \ + \ \frac{5}{10} \ \leq \ 1$$

where

$$I_3^{10} \ = \ \sum_{j=1}^{2} \left\lceil \frac{1}{T_j} \right\rceil \ C_j$$

$$= \left\lceil \frac{10}{10} \right\rceil 4 + \left\lceil \frac{10}{11} \right\rceil 3 = 7$$

substituting

$$\frac{7}{10} + \frac{3}{10} = 1$$

Hence τ_3 is schedulable, meeting its deadline at time 10. An example run of the system is seen in Figure 14.

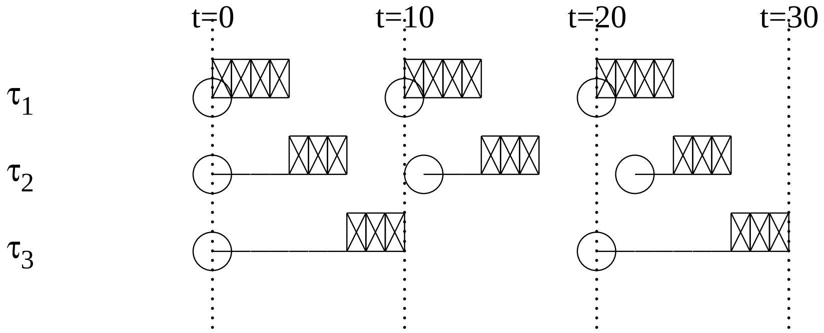


Figure 14.

■

The successive equations (13) have shown the process system to be schedulable. The solution of a single equation was required.

3.5. Summary

This section has introduced a number of schedulability and unschedulability tests for the deadline monotonic algorithm:

- a $O(n)$ schedulability test that is sufficient and not necessary;
- a $O(n^2)$ schedulability test that is sufficient and not necessary;
- a $O(n^2)$ unschedulability test that is sufficient and not necessary;
- a sufficient and necessary schedulability test that has data-dependent complexity.

The first test provides the coarsest level. The second and third tests combine to provide a finer grain measure of process systems that are definitely schedulable or definitely unschedulable. The sufficient and necessary test is able to differentiate schedulable and unschedulable systems to provide the finest level of test.

One constraint on the process systems is that they must have a critical instant. This is ensured as all processes have an initial release at time 0.

4. SCHEDULING SPORADIC PROCESSES

Non-periodic processes are those whose releases are not periodic in nature. Such processes can be subdivided into two categories [Aud90a]: aperiodic and sporadic. The difference between these categories lies in the nature of their release frequencies. Aperiodic processes are those whose release frequency is unbounded. In the extreme, this could lead to an arbitrarily large number of simultaneously active processes. Sporadic processes are those that have a maximum frequency such that only one instance of a particular sporadic process can be active at a time.

When a static scheduling algorithm is employed, it is difficult to introduce non-periodic process executions into the schedule: it is not known before the system is run when non-periodic processes will be released. More difficulties arise when attempting to guarantee the deadlines of those processes. It is clearly impossible to guarantee the deadlines of aperiodic processes as there could be an arbitrarily large number of them active at any time. Sporadic processes deadlines can be guaranteed since it is possible, by means of the maximum release frequency, to define the maximum workload they place upon the system.

One approach is to use static periodic polling processes to provide sporadics with executions time. This approach is reviewed in section 4.1. Section 4.2 illustrates how to utilise the properties of the deadline monotonic scheduling algorithm to guarantee the deadlines of sporadic processes without resorting to the introduction of polling processes.

4.1. Sporadic Processes: the Polling Approach

To allow sporadic processes to execute within the confines of a static schedule (such as that generated by the rate-monotonic algorithm) computation time must be reserved within that schedule. An intuitive solution is to set up a periodic process which polls for sporadic processes [Leh87a]. Strict polling reduces the bandwidth of processing as

- processing time that is embodied in an execution of the polling process is wasted if no sporadic process is active when the polling process becomes runnable;
- sporadic processes occurring after the polling process's computation time in one period has been exhausted or just passed have to wait until the next period for service.

A number of bandwidth preserving algorithms have been proposed for use with the rate-monotonic scheduling algorithm. One such algorithm is the deferrable server [Leh87a, Sha89b, Sha89a]. The server is a periodic process that is allotted a number of units of computation time per period. These units can be used by any sporadic process with outstanding computational requirements. When the server is run with no outstanding sporadic process requests, the server does not execute but defers its assigned computation time. The server's time is preserved at its initial priority. When a sporadic request does occur, the server has maintained its priority and can thus run and serve the sporadic processes until its allotted computation time within the server period has been exhausted. The computation time for the server is replenished at the start of its period.

Problems arise when sporadic processes require deadlines to be guaranteed. It is difficult to accommodate these with a deferrable server due to the rigidly defined points in time at which the server computation time is replenished. The sporadic server [Sha89a] provides a solution to this problem. The replenishment times are related to when the sporadic uses computation time rather than merely at the period of the server process.

The sporadic server is used by Sha *et al* [Sha89a] in conjunction with the rate-monotonic scheduling algorithm to guarantee sporadic process deadlines. Since the rate-monotonic algorithm is used, a method is required to map sporadic processes with timing characteristics given by

$$\textit{computation time} \le \textit{deadline} \le \textit{period}$$

onto periodic server processes that have timing characteristics given by

$$\textit{computation time} \leq \textit{deadline} = \textit{period}$$

The method adopted in [Sha89a] lets the computation time, period and deadline of the server be equal to the computation time, minimum inter-arrival time and deadline of the sporadic process. The rate-monotonic scheduling algorithm is then used to test the schedulability of the process system, with runtime priorities being assigned in a deadline monotonic manner.

The next section details a simpler approach to guaranteeing sporadic deadlines based upon the deadline monotonic scheduling algorithm.

4.2. Sporadic Processes: the Deadline Monotonic Scheduling Approach

Consider the timing characteristics of a sporadic process. The demand for computation time is illustrated in Figure 15.

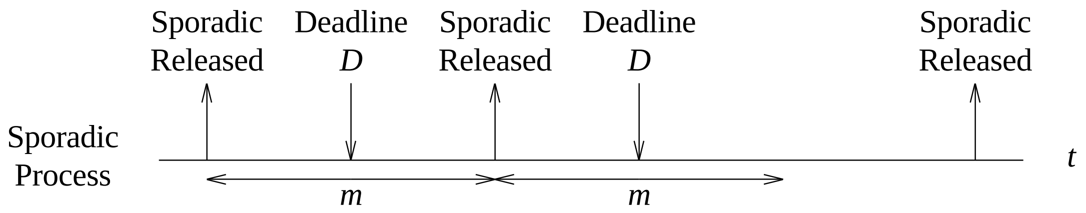


Figure 15.

The minimum time difference between successive releases of the sporadic process is the minimum inter-arrival time m. This occurs between the first two releases of the sporadic. At this point, the sporadic is behaving exactly like a periodic process with period m: the sporadic is being released at its maximum frequency and so is imposing its maximum workload.

When the releases do not occur at the maximum rate (between the second and third releases in Figure 15) the sporadic behaves like a periodic process that is intermittently activated and then laid dormant. The workload imposed by the sporadic is at a maximum when the process is released, but falls when the next release occurs after greater than m time units have elapsed.

In the worst-case the sporadic process behaves exactly like a periodic process with period m and deadline D $(D \leq m)$. The characteristic of this behaviour is that a maximum of one release of the process can occur in any interval $\left[t, t+m \right]$ where release time t is at least m time units after the previous release of the process. This implies that to guarantee the deadline of the sporadic process the computation time must be available within the interval $\left[t, t+D \right]$ noting that the deadline will be at least m after the previous deadline of the sporadic. This is exactly the guarantee given by the deadline-monotonic schedulability tests in section 3.

For schedulability purposes only, we can describe the sporadic process as a periodic process whose period is equal to m. However, we note that since the process is sporadic, the actual release times of the process wil not be periodic, merely separated by at least m time units.

For the schedulability tests given in section 3 to be effective for this process system, all processes, both periodic and sporadic, have to be released simultaneously. We can assume that all the processes are released simultaneously at time 0: a critical instant. This forms the worst-case workload on the processor. If the deadline of the sporadic can be guaranteed for the release at a critical instant then all subsequent deadlines are guaranteed.

An example is now given.

Example

Consider the following process system.

$$\tau_1 : \ C_1 = 1, \ D_1 = 5, \ T_1 = 6$$

$$\tau_2 : \ C_2 = 2, \ D_2 = 6, \ T_2 = 8$$

$$\tau_3 : \ C_3 = 2, \ D_3 = 7, \ T_3 = 9$$

$$\tau_4 : \ C_3 = 2, \ D_3 = 8, \ T_3 = 10$$

Processes τ_1 and τ_3 are periodic, whilst τ_2 and τ_4 are sporadic with minimum inter-arrival times given by T_2 and T_4 respectively.

We check the schedulability of the system using the equations given in section 3. The simplest test (equation (8)) is used.

(a) check process τ_1

$$\frac{C_1}{D_1} + \frac{I_1}{D_1} \leq 1$$

$$\frac{1}{5} < 1$$

Hence τ_1 is schedulable.

(b) check process τ_2

$$\frac{C_2}{D_2} + \frac{I_2}{D_2} \leq 1$$

$$\frac{2}{6} + \frac{I_2}{6} \leq 1$$

where

$$I_2 = \left\lceil \frac{6}{6} \right\rceil 1 = 1$$

substituting

$$\frac{2}{6} + \frac{1}{6} < 1$$

Hence τ_2 is schedulable.

(c) check process τ_3

$$\frac{C_3}{D_3} + \frac{I_3}{D_3} \leq 1$$

$$\frac{2}{7} + \frac{I_3}{7} \leq 1$$

where

$$I_3 = \left\lceil \frac{D_3}{T_1} \right\rceil C_1 + \left\lceil \frac{D_3}{T_2} \right\rceil C_2$$

$$I_3 = \left\lceil \frac{7}{6} \right\rceil 1 + \left\lceil \frac{7}{8} \right\rceil 2 = 4$$

substituting

$$\frac{2}{7} + \frac{4}{7} < 1$$

Hence τ_3 is schedulable.

(d) check process τ_4

$$\frac{C_4}{D_4} + \frac{I_4}{D_4} \leq 1$$

$$\frac{2}{8} + \frac{I_3}{8} \leq 1$$

where

$$I_4 = \left\lceil \frac{D_4}{T_1} \right\rceil C_1 + \left\lceil \frac{D_4}{T_2} \right\rceil C_2 + \left\lceil \frac{D_4}{T_3} \right\rceil C_3$$

$$I_4 = \left\lceil \frac{8}{6} \right\rceil 1 + \left\lceil \frac{8}{8} \right\rceil 2 + \left\lceil \frac{8}{9} \right\rceil 2 = 6$$

substituting

$$\frac{2}{8} + \frac{6}{8} = 1$$

Hence τ_4 is schedulable.

The process system is schedulable. An example run is given in Figure 16.

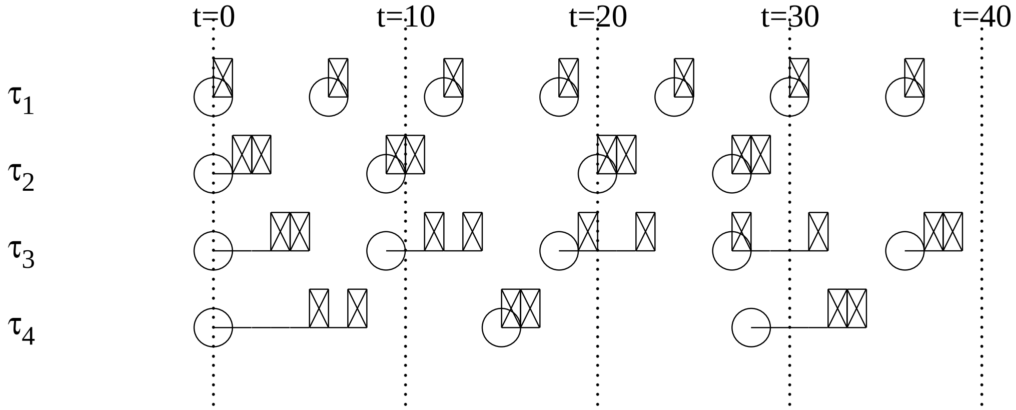


Figure 16.

■

In the example run (Figure 16) all deadlines are met. Each of the sporadic processes are released at time 0. This forms a critical instant and thus the worst-possible scenario for scheduling the process system. A combination of many periodic and many sporadic processes was shown to be schedulable under this scheme without the need for server processes which are required for scheduling sporadic processes with the rate-monotonic scheduling algorithm (see section 4.1).

4.3. Summary

The proposed method for guaranteeing the deadlines of sporadic processes using sporadic servers within the rate-monotonic scheduling framework has two main drawbacks. Firstly, one extra periodic server process is required for each sporadic process. Secondly, an extra run-time overhead is created as the kernel is required to keep track of the exact amount of time the server has left within any period.

The deadline-monotonic approach circumvents these problems since no extra processes are required: the sporadic processes can be dealt with adequately within the existing periodic framework.

5. CONCLUSIONS

The fundamental constraints of the rate-monotonic scheduling algorithm have been weakened to permit processes that have deadlines less than period to be scheduled. The resulting scheduling mechanism is the deadline-monotonic algorithm. Schedulability tests have been presented for the deadline-monotonic algorithm.

Initially a simple sufficient and not necessary schedulability test was introduced. This required a single equation per process to determine schedulability. However, to achieve such simplicity meant the test was overly pessimistic.

The simplifications made to produce a single equation test were then partially removed. This produced a sufficient and not necessary schedulability test which passed more process systems than the simple test. The complexity of the second test was $O(n^2)$ compared with $O(n)$ for the simple test. Again, the test was pessimistic.

To complement the second schedulability test, a similar unschedulability test was developed. The combination of sufficient and not necessary schedulability and unschedulability tests was shown to be useful for identifying some unschedulable systems. However, it was still possible for a process system to fail both the schedulability and unschedulability tests.

This problem was resolved with the development of a sufficient and necessary schedulability test. This was the most complex of all the tests having a complexity related to the periods and computation times of the processes in the system. The complexity was reduced substantially when the number of equations required to determine the schedulability of a process were minimised.

The problem of guaranteeing the deadlines of sporadic processes was then discussed. Noting that schedulability tests proposed for sporadic processes and the rate-monotonic algorithm require the introduction of special server processes, we then proposed a simple method to guarantee the deadlines of sporadic processes within the confines of the deadline-monotonic algorithm. The simplicity of the method is due to sporadic processes being treated exactly as periodic processes for the purpose of determining the schedulability. Using this scheme, any mixture of periodic and sporadic deadlines can be scheduled subject to the process system passing the deadline-monotonic schedulability constraint.

A number of issues raised by the work outlined in this paper require further consideration. These include the effect of allowing processes to synchronise and vary their timing characteristics. Another related issue is the effect of deadline-monotonic scheduling upon system utilisation. These issues remain for further investigation.

ACKNOWLEDGEMENTS

The author thanks Mike Richardson, Alan Burns and Andy Wellings for their valuable comments and diatribes.

REFERENCES

Bur89a. A. Burns and A. J. Wellings, *Real-Time Systems and Their Programming Languages*, Addison Wesley (1989).

Sta88a. J.A. Stankovic, ‘‘Misconceptions About Real-Time Computing: A Serious Problem for Next Generati on Systems’’, *IEEE Computer* **21**(10), pp. 10-19 (October 1988).

Aud90a. N. C. Audsley and A. Burns, ‘‘Scheduling Real-Time Systems’’, YCS 134, Department of Computer Science, University of York (1990).

Liu73a. C. L. Liu and J. W. Layland, ‘‘Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment’’, *Journal of the ACM* **20**(1), pp. 40-61 (1973).

Leh89a. J. Lehoczky, L. Sha and Y. Ding, ‘‘The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour’’, *Proceedings IEEE Real-Time Systems Symposium*, Santa Monica, California, pp. 166-171, IEEE Computer Society Press (5-7 December 1989).

Sha89a. L. Sha, B. Sprunt and J. P. Lehoczky, ‘‘Aperiodic Task Scheduling for Hard Real-Time Systems’’, *The Journal of Real-Time Systems* **1**, pp. 27-69 (1989).

Sha88a. L. Sha and J. B. Goodenough, ‘‘Real-Time Scheduling Theory and Ada’’, CMU/SEI-88-TR-33, Software Engineering Institute, Carnegie-Mellon University (November 1988).

Sha90a. L. Sha and J. B. Goodenough, ‘‘Real-Time Scheduling Theory and Ada’’, *IEEE Computer* (April 1990).

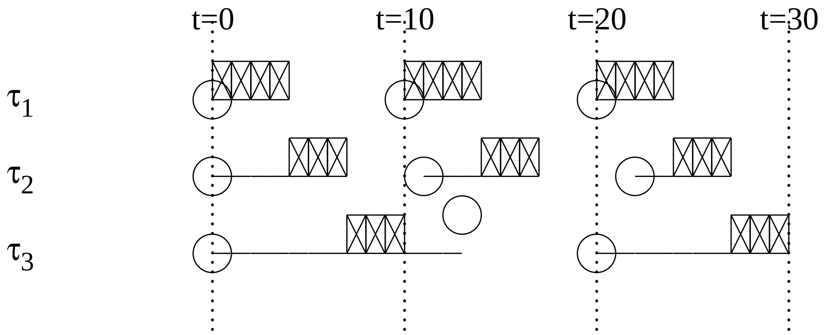
Sha87a. L. Sha, R. Rajkumar and J. P. Lehoczky, ‘‘Priority Inheritance Protocols: An Approach to Real-Time Synchronisation’’, CMU-CS-87-181, Computer Science Department, Carnegie-Mellon University (December 1987).

Leu82a. J. Y. T. Leung and J. Whitehead, ‘‘On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks’’, *Performance Evaluation (Netherlands)* **2**(4), pp. 237-250 (December 1982).

Leh87a. J. P. Lehoczky, L. Sha and J. K. Strosnider, ‘‘Enhanced Aperiodic Responsiveness in Hard Real-Time Environments’’, *Proceedings IEEE Real-Time System Symposium*, San Jose, California, pp. 261-270 (1987).

Sha89b. L. Sha, J. B. Goodenough and T. Ralya, ‘‘An Analytical Approach to Real-Time Software Engineering’’, DRAFT (1989).

APPENDIX 1 : Run-Time Simulation Diagrams



The runtime diagrams are produced by a real-time system simulator and show the following aspects of a process simulation:

- Process Release - circle on the time line of a process (see τ_1 at time 0).
- Process Execute - a hatched box on the time line of a process (see τ_1 at time 0 to 3).
- Preempted Process - when a higher priority process runs and a lower priority process has remaining computation time, a solid line extend along the time line of the process (see τ_2 at time 0 to 3).
- Missed Deadline - raised circle above the time line of a process (see τ_3 at time 13).