HARD REAL-TIME SCHEDULING: THE DEADLINE-MONOTONIC APPROACH[1]

N. C. Audsley A. Burns M. F. Richardson A. J. Wellings

Department of Computer Science, University of York, York, YO1 5DD, England.

ABSTRACT. The scheduling of processes to meet deadlines is a difficult problem often simplified by placing severe restrictions upon the timing characteristics of individual processes. One restriction often introduced is that processes must have deadline equal to period. This paper investigates schedulability tests for sets of periodic processes whose deadlines are permitted to be less than their period. Such a relaxation enables sporadic processes to be directly incorporated without alteration to the process model. Following an introduction outlining the constraints associated with existing scheduling approaches and associated schedulability tests, the deadline-monotonic approach is introduced. New schedulability tests are derived which vary in computational complexity. The tests are shown to be directly applicable to the scheduling of sporadic processes.

KEYWORDS. computer control, deadlines, mathematical analysis, real-computer computer systems, scheduling theory

1. INTRODUCTION

A real-time system is one in which failure can occur in the time domain as well as in the more familiar value domain. If the consequence of such failure is catastrophic then the system is often referred to as a *hard real-time system.* Such systems are needed in a number of application domains including air-traffic control, process control, and numerous embedded systems.

In the development of application programs it is usual to map system timing requirements onto process deadlines. The issue of meeting deadlines therefore becomes one of process scheduling. The development of appropriate scheduling algorithms has been isolated as one of the crucial challenges for the next generation of real-time systems (Stankovic, 1988).

One scheduling method that is used in hard real-time systems is based upon rate-monotonic theory (Liu, 1973). At runtime a preemptive scheduling mechanism is used: the highest priority runnable process is executed. Priorities assigned to processes are inversely proportional to the length of period. That is, the process with the shortest period is assigned the highest priority. Rate-monotonic scheduling has several useful properties, including a simple "sufficient and not necessary" schedulability test based on process utilisations (Liu, 1973); and a complex sufficient and necessary schedulability test (Lehoczky, 1989). However, the constraints that it imposes on the process set 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 approach and have provided associated schedulability tests. Reported work includes a test to allow aperiodic processes to be included in the theory (Sha, 1989), and a test to incorporate processes that synchronise using semaphores (Sha, 1988). One constraint that has remained within rate-monotonic literature is that the deadline and period of a process must be equal.

Deadline-monotonic (Leung, 1982) priority assignment weakens this constraint within a static priority scheduling scheme. However, no schedulability tests were given in(Leung, 1982) for the scheme.

The weakening of the "period equals deadline" constraint would benefit the application designer by providing a more flexible process model. For example, precedence constraints in a distributed system can be modelled as a sequence of periodic processes (one per processor). The inevitable communication delay is modelled as an interval of "dead time" at the end of each processes period (apart from the last process in the sequence). These periodic processes must therefore complete their computations by a deadline that is before the end of the period. Such a model has been used to good effect in a process allocation scheme (Tindell, 1990) in which network communication overhead is traded against local schedulability. The greater the interprocessor traffic the greater the "dead time" and hence the lower the schedulability bound. Tindell's analysis (Tindell, 1990) uses the schedulability tests discussed in this paper.

Another important motivation for weakening the "deadline equal to period" constraint is to cater for sporadic (aperiodic) events in an efficient manner. Here the required response time is not, in general, related to the worst case arrival rate. Indeed the characteristics of such events often demand a short response time compared to minimum inter-arrival time. Hence polling in a periodic manner for sporadic events produces a non-optimal response time for sporadic processes.

This paper outlines the deadline-monotonic scheduling approach together with new simple and complex schedulability tests that are sufficient and in the latter case, necessary. The approach is then shown to encompass sporadic processes that have hard deadlines without any alteration to the theory and without resorting to the inefficiencies of a polling approach.

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

2. DEADLINE-MONOTONIC SCHEDULING THEORY

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

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

i.e. for each process i (where process 1 has the highest priority and process n the lowest in a system containing n processes):

$$C_i \le D_i \le T_i$$

where C gives computation time, D the deadline and T the period of process i.

Leung *et al* (Leung, 1982) have defined a priority assignment scheme that caters for processes with the above relationship. This is termed inverse-deadline or deadline-monotonic priority assignment. No schedulability tests were given however.

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 (Leung, 1982). Thus, the process with the shortest deadline is assigned the highest priority and the longest deadline process is assigned 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 (see theorem 2.4 in (Leung, 1982)). The implication of this is that if any static priority scheduling algorithm can schedule a process set where process deadlines are unequal to their periods, an algorithm using deadline-monotonic priority ordering for processes will also schedule that process set.

It is true, of course, that any process sets whose timing characteristics are suitable for rate-monotonic analysis would also be accepted by a static priority theory permitting deadlines and periods of a process to differ.

In general the deadline-monotonic scheme has not been employed because of the lack of adequate schedulability tests. Rate-monotonic scheduling schedulability tests could be used by reducing the period of individual processes until equal to the deadline. Obviously such tests would not be optimal as the workload on the processor would be over-estimated.

New schedulability tests have been developed by the authors for the deadline-monotonic approach (Audsley, 1990). These tests are founded upon the concept of *critical instants* (Liu, 1973). These represent the times that all processes are released simultaneously. When such an event occurs, we have the worst-case processor demand. Implicitly, if all processes can meet their deadlines for executions beginning at a critical instant, then they will always meet their deadlines. Thus, we have formed the basis for a schedulability test: check the executions of all processes for a single execution assuming that all processes are released simultaneously. One such schedulability test is given by[2]:

2,4. for derivation see (Audsley, 1990).

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

where I_i is a measure of higher priority processes interfering with the execution of τ_i:[3]

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

The test states that for a process τ_i to be schedulable, the sum of its computation time and the interference that is imposed upon it by higher priority processes executing must be no more than D_i. In the above, the interference is composed of the computation time of all higher priority processes that are released before the deadline of τ_i. This test is sufficient, but not necessary for the following reason. When the interference is being calculated, account is taken of executions of higher priority processes that start before D_i and could possibly complete execution after D_i. Therefore, I_i could be greater than the actual interference encountered by τ_i before D_i.

A more accurate test is given by[4]:

$$\forall i : 1 \le i \le n : \ \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1 \qquad (2)$$

where

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

The above test compensates within I_i for the parts of executions of higher priority processes that could not occur before D_i even though they were released before D_i. The test is not necessary as a pessimistic valuation is made of the time that will be utilised by higher priority processes before D_i.

The schedulability constraints given by equations (1) and (2) are sufficient but not necessary in the general case. To form a sufficient and necessary schedulability test the schedule has to be evaluated so that the exact interleaving of higher priority process executions is known. This is costly as this would require the solution of D_i equations per process τ_i.

The number of equations can however be reduced by observing that if τ_i meets its deadline at t'_i, where t'_i lies in $[0, D_i]$, we need not evaluate the equations in $(t'_i, D_i]$. Further reductions in the number of equations requiring solution can be made by limiting the points in $[0, D_i]$ that are considered as possible solutions for t'_i. Consider the times within $[0, D_i]$ that τ_i could possibly meet its deadline. We note that the interference due to high-priority processes is monotonically increasing within this interval. The points in time that the interference increases occur when there is a release of a higher priority process. This is illustrated by Fig. 1.

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

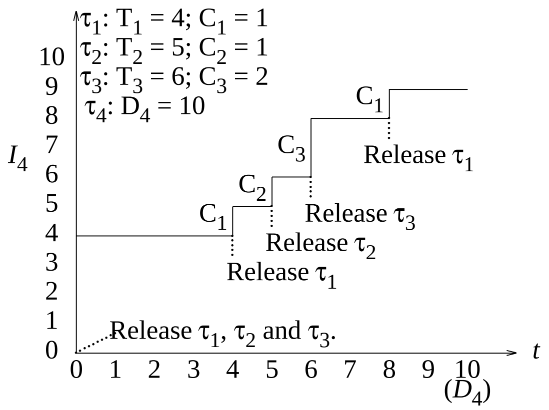

Fig. 1. Interference Increases"

In Fig. 1 there are three processes with higher priority than τ_4. We see that as the higher priority processes are released, I_4 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 rightmost 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 $[0, D_i]$ that correspond to a higher priority process release. Since as soon as one equation identifies the process set as schedulable we need test no further equations. Thus, the effect is to evaluate equations only in $[0, t'_i]$.

The number of equations has been reduced in most cases. We note that no reduction will occur if for each point in time in $[0, D_i)$ a higher priority process is released with τ_i meeting its deadline at D_i. The number of equations is reduced yet further by considering the computation times of the processes. Consider Fig. 2.

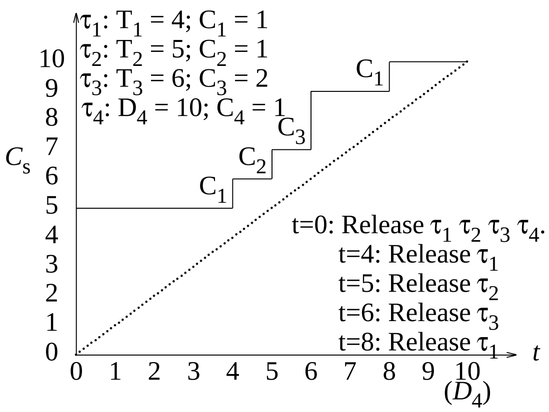

Fig. 2. Process Computation Time

In Fig. 2 the total computation requirement of the process set (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. In the above diagram this point in time coincides with the deadline of τ_4.

Considering Fig. 2, there is little merit in testing the schedulability of τ_i in the interval $[0, C_i)$. 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:

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

This gives a schedulability constraint of:

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

where

$$I_y^x = \sum_{z=1}^{y-1} \left\lceil \frac{x}{T_z} \right\rceil C_z$$

Since the value of t_1 assumes that only one release of each process occurs in $[0, t_0)$, the constraint will fail if there have been any releases of higher priority processes within the interval $(0, t_0)$. 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 $[t_0, t_1)$. Thus, we can build a series of equations to express the schedulability of τ_i.

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

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

$$(2) \qquad \frac{I_i^{t_1}}{t_1} + \frac{C_i}{t_1} \le 1$$

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

.

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

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

and where

$$I_y^x = \sum_{z=1}^{y-1} \left\lceil \frac{x}{T_z} \right\rceil C_z$$

If any of the equations hold, τ_i is schedulable. Obviously, the equations terminate if $t_k > D_i$ for process τ_i and equation k. At this point τ_i is unschedulable.

The series of equations above is encapsulated by the algorithm given in Fig. 3. The algorithm progresses since 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. We note that the algorithm can be used to evaluate the schedulability of any fixed priority process set where process deadlines are no greater than periods, whatever the assignment rule used for priorities.

Algorithm

foreach τ_i *do*

$\quad t = \sum_{j=1}^{i} C_j$

$\quad$ *continue* $= TRUE$

$\quad$ *while* $\left(continue \right)$ *do*

$\quad\quad$ if $\left(\frac{I_i^t}{t} + \frac{C_i}{t} \le 1 \right)$

$\quad\quad\quad$ *continue* $= FALSE$

$\quad\quad\quad$ */* NB* τ_i *is schedulable */*

$\quad\quad$ *else*

$\quad\quad\quad t = I_i^t + C_i$

$\quad\quad$ *endif*

$\quad\quad$ if $\left(t > D_i \right)$

$\quad\quad\quad$ *exit*

$\quad\quad\quad$ */* NB* τ_i *is unschedulable */*

$\quad\quad$ *endif*

$\quad$ *endwhile*

endfor

■

Fig. 3. Schedulability Algorithm

3. SCHEDULING SPORADIC PROCESSES

Non-periodic processes are those whose releases are not periodic in nature. Such processes can be subdivided into two categories (Burns, 1991): 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 3.1. Section 3.2 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.

3.1. 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 (Lehoczky, 1987). 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 (Lehoczky, 1987; Sha, 1989). These algorithms are founded upon a periodic server process being allotted a number of units of computation time per period. These units can be used by any sporadic process with outstanding computational requirements. 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 within periodic server processes due to the rigidly defined points in time at which the server computation time is replenished. The sporadic server (Sha, 1989) 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. However, this approach still requires additional processes with obvious extra overheads.

3.2. Sporadic Processes: the Deadline Monotonic Scheduling Approach

We now show how deadlines of sporadic processes can be guaranteed within the existing deadline-monotonic theory. Consider the timing characteristics of a sporadic process τ_s. The demand for computation time is illustrated in Fig. 4.

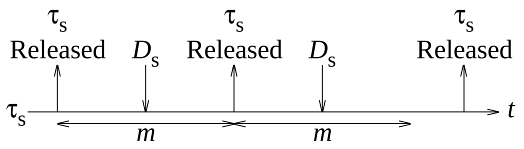

Fig. 4. Sporadic Execution

The minimum time difference between successive releases of τ_s is the minimum inter-arrival time m. This occurs between the first two releases of τ_s. At this point, τ_s 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 Fig. 4) τ_s 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 τ_s behaves exactly like a periodic process with period m and deadline D where $D \le m$. The characteristic of this behaviour is that a maximum of one release of the process can occur in any interval $[t, t + m]$ 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 $[t, t + D]$ 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 2.

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 will not be periodic, but successive releases will be separated by no less than m time units.

For the schedulability tests given in section 2 to be effective for this process system, we assume that at some instant all processes, both periodic and sporadic, are released simultaneously (i.e. a critical instant). We assume that this occurs at time 0. If the deadline of the sporadic can be guaranteed for the release at a critical instant then all subsequent deadlines are guaranteed. Examples of this approach are given in (Audsley, 1990). No limitations on the combination of periodic and sporadic processes are imposed by this scheme. Indeed, the approach is optimal for a fixed priority scheduling since sporadic processes are treated in exactly the same manner as periodic processes. All three schedulability tests outlined in section 2 are suitable for use with sporadic processes. To improve the responsiveness of sporadic processes, their deadlines can be reduced to the point at which the system becomes unschedulable.

4. CONCLUSIONS

The fundamental constraint of rate-monotonic scheduling theory has been weakened to permit processes that have deadlines less than period to be scheduled. The result is the deadline-monotonic scheduling theory. Schedulability tests have been presented for the theory.

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 sets than the simple test. Again, the test was pessimistic.

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 set. The complexity was reduced substantially when the number of equations required to determine the schedulability of a process were minimised. This test is able to determine the schedulability of any fixed priority process set where deadlines are no greater than periods, whatever the priority assignment criteria used.

Proposed methods for guaranteeing deadlines of sporadic processes using sporadic servers within the rate-monotonic scheduling framework were shown to have 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.

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. These issues remain for further investigation, although it is the authors' contention that the analysis that has been focussed upon the rate-monotonic approach shows that deadline-monotonic schedulability theory is easily extensible to address such issues.

REFERENCES

Audsley, N.C. (1990). Deadline Monotonic Scheduling. *YCS 146, Dept. of Comp. Sci., Univ. of York.*

Burns, A. (1991). Scheduling Hard Real-Time Systems: A Review. *Software Eng. Journal, 6, pp. 116-128.*

Lehoczky, J.P., L. Sha, and J.K. Strosnider. (1987). Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *Proc IEEE Real-Time Sys. Symp., pp. 261-270.*

Lehoczky, J.P., L. Sha, and Y. Ding. (1989). The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour. *Proc IEEE Real-Time Sys. Symp., pp. 166-171*.

Leung, J.Y.T., and J. Whitehead. (1982). On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Perf. Eval. (Netherlands), 2, pp. 237-250*.

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

Sha, L., and J.B. Goodenough. (1988). Real-Time Scheduling Theory and Ada. *CMU/SEI-88-TR-33, SEI, Carnegie-Mellon University*.

Sha, L. and J.P. Lehoczky. (1989). Aperiodic Task Scheduling for Hard Real-Time Systems. *J. of Real-Time Sys., 1, pp. 27-69*.

Stankovic, J.A. (1988). Real-Time Computing Systems: The Next Generation. *COINS Tech. Rep. 88-06, Dept. of Comp. and Inf. Sci., Univ. of Massachusetts.*

Tindell, K. (1990). Allocating Real-Time Tasks (An NP-Hard Problem made Easy). *YCS 147, Dept. of Comp. Sci., Univ. of York.*