Guaranteeing Timing Constraints Under Shortest Remaining Processing Time Scheduling

R.I. Davis, A. Burns and W. Walker
Real-Time Systems Research Group
Department of Computer Science
University of York

Abstract

The scheduling scheme "shortest remaining processing time" (SRPT) has the advantage that it minimises mean response times. In this paper we present feasibility tests for SRPT that will enable this scheduling approach to be used for real-time systems. Examples are given of task sets that are schedulable under SRPT but not by fixed priority based scheduling.

1. Introduction

In scheduling theory, the notion of priority is used to describe the attribute of a task or process which is used to determine which of a set of competing tasks will utilize the processor at any given time. Scheduling algorithms themselves can be classified according to the way in which this notional "priority" is assigned and how it varies with time. The main distinction is between *fixed* and *dynamic* priority schemes.

In fixed priority preemptive scheduling, task priorities assume constant values, allocated off-line by some priority assignment policy such as Deadline Monotonic priority assignment. At run time, each invocation of a given task has the same fixed priority. This priority does not vary as the task executes (other than to implement some concurrency control protocol for resource sharing).

In 1972, Liu and Layland [4] (and others) showed that a simple sufficient feasibility test could be used to determine if a set of independent periodic tasks assigned priorities according to the Rate Monotonic priority assignment policy would always meet their deadlines when dispatched to the processor on a fixed priority preemptive basis. Subsequently, exact feasibility tests have been developed for task sets scheduled according to general fixed priority preemptive dispatching [3], [1].

Dynamic priority algorithms may be divided into two types:

1. *EDF like algorithms*: The priority of each invocation of a given task is determined dynamically and then remains fixed for the duration of the invocation. For example with EDF scheduling (earliest deadline first), the priority of a task at invocation depends on its deadline, which is a fixed value determined at the release of the task, similarly for FCFS (first come first served) scheduling, the priority of a task is equivalent to its release time. Liu and Layland [4] showed that a set of independent periodic task will always meet their deadlines when scheduled according to EDF provided that the total utilisation of the task set is no more then 100%.

2. *RPT like algorithms*: The priority of every invocation of a given task is the same at the release of that invocation. However the priority of each invocation then varies in proportion to the remaining execution time of the invocation. We refer to this type of scheduling algorithm as a 'RPT' algorithm (*Remaining Processing Time*). The shortest remaining processing time (SRPT) and maximum value density first (MVDF) algorithms are examples of RPT algorithms.

In 1956, Smith [7] showed that scheduling a set of tasks in shortest remaining processing time order results in the minimum mean response time for the set of tasks. Building upon this result, Locke [5] showed that MVDF results in a schedule which maximised the total value accrued by completing tasks at any given time.

Since 1956, there has been much interest in RPT type algorithms in the field of job-shop scheduling, since these algorithms minimise/maximise some metric of interest, such as the mean flow time (response time). However, since the early 1970's the majority of research into real-time scheduling has focused on fixed priority (FP) or EDF algorithms. The development of feasibility tests for these algorithms meant that *a priori* analysis could be performed to determine if tasks scheduled by FP or EDF would always meet their deadlines at run-time.

Although the primary aim of (hard) real-time system scheduling is to ensure that time constraints (deadlines) are always met, once this criteria is fulfilled, other metrics become important. For example, a system which meets all deadlines and provides the minimum mean response time for jobs may be considered to provide a higher quality of service than a comparable system which meets deadlines but results in longer average delays.

In this paper, we show that there is a critical instant for tasks scheduled according to the SRPT algorithm analogous to that given by Liu and Layland for FP scheduling. We provide a simple sufficient but not necessary feasibility test for SRPT based upon similar tests derived for fixed priority scheduling. This approach is then extended to provide an exact feasibility test for tasks scheduled by SRPT. Examples are given of task sets which are feasible under SRPT but not under FP preemptive scheduling and vice versa.

2. Computational Model

In this paper, we consider a uniprocessor system executing a set of n tasks. Each task, τ_i, is assumed to have a minimum inter-arrival time (between invocations) of T_i, a worst case execution time (wcet) C_i and a deadline D_i. Task arrival may therefore be periodic or sporadic. We assume that $\forall \tau_i : D_i \leq T_i$. The set of tasks $\tau_1 \ldots \tau_n$ are ordered according to their execution times. Thus τ_1 is the task with the shortest wcet (i.e. smallest C). We use $sp(i)$ to denote the set of tasks with shorter worst case execution times than τ_i. The alternative set, $lp(i)$, have longer processing time.

Throughout this paper, we assume preemptive shortest remaining processing time scheduling. At any given time, the task which is allocated the processor is the runnable task with the shortest remaining processing time. It is assumed that nothing is known about the remaining processing time of a task save the worst case execution time C_i and the time for which the current invocation of the task has executed. Thus if C_i is the wcet of task τ_i and it has executed for time t' then the remaining execution time is assumed to be $C_i - t'$. In general, we use $C_i(t)$ to denote the remaining processing time of the current invocation of task τ_i at some arbitrary time t. The value $C_i(t) = 0$ implies that the task has completed its current invocation.

3. Critical Instant

In this section, we derive a critical instant for SRPT scheduling. By a critical instant, for task τ_i, we refer to the arrangement of task releases and executions such that task τ_i exhibits the largest possible delay between release and completion. We refer to this largest possible delay as the worst case response time, R_i, of task τ_i.

We now give a general formula for the worst case response time of task τ_i:

$$R_i = C_i + B_i + I_i$$

Where I_i is the maximum interference which τ_i is subject to between its release and completion due to tasks in the set $sp(i)$. Similarly B_i is the maximum time for which τ_i is prevented from executing due to the execution of tasks in the set $lp(i)$. In fixed priority scheduling, B_i is referred to as blocking.

First we introduce a simple theorem about the execution order of tasks:

Theorem 1 *At any arbitrary time t there can be at most one task τ_j which has a wcet C_j greater than some arbitrary constant C and yet at time t has a remaining execution which is less than C (i.e. $C_j > C$ and $C_j(t) \leq C$).*

Proof

At time 0 (system start up) no tasks have the desired property: $C_j > C$ and $C_j(t) \leq C$. Without loss of generality, we assume that at time t there is one task τ_k with $C_k > C$ and $C_k(t) \leq C$. Whilst $C_k(t) \leq C$, no other task τ_j with $C_j > C$ and $C_j(t) > C$ can execute until τ_k completes. Hence at any given time, there can only be one task with a wcet greater than C which has a remaining execution time which is less than C.

□

It follows that at time, t, only one task, τ_j, from the set $lp(i)$ can have $C_j(t) \leq C_i$.

Theorem 2 *The maximum interference which task τ_i may be subject to, due to the execution of tasks in the set $sp(i)$, occurs when τ_i and all the tasks in the set $sp(i)$ are released simultaneously and all subsequent instances are released periodically.*

Proof

We prove this theorem in two steps.

Step 1: *we assume that there exists some arbitrary pattern of releases of tasks (τ_j) in the set $sp(i)$, characterised by offsets O_j ($0 \leq O_j < T_j$) which leads to the worst case response time for task τ_i.*

In this worst case arrangement let q_j be the number of invocations of τ_j which interfere with τ_i. Thus the interference suffered by τ_i is given by:

$$\sum_{\forall j : \tau_j \in sp(i)} q_j C_j$$

Let $I_j(t)$ be the cumulative task τ_j processing released in the period [0,t), and $C_i(kT_j + O_j)$ be the remaining execution time of task τ_i at the kth release of task τ_j. (Note,

$C_i(kT_j + O_j) > C_j \ \forall k : 0 \leq k \leq q_j)$ as all q invocations interfere with τ_i).

Step 2: *we now show that changing the pattern of task releases assumed in Step 1 such that any arbitrary task τ_j ($\tau_j \in sp(i)$) is released at times $t = 0, T_j, 2T_j, \ldots$ instead of $t = O_j, T_j + O_j, 2T_j + O_j, \ldots$ results in a worse case response time for task τ_j which is at least as large as it is with the pattern of task releases assumed in Step 1.*

Given that τ_j is released at time $t = 0, T_j, 2T_j, \ldots$ let $C_i'(kT_j)$ be the remaining execution time of task τ_i at the kth release of task τ_j and $I_j'(t)$ be the cumulative task τ_j processing released in the interval [0, t). The interference due to τ_j increases by $C_j$at each release of τ_j thus:

$$I_j'(kT_j) = I_j(kT_j + O_i)$$

and therefore:

$$I_j'(t) \geq I_j(t) \ \ \forall t : 0 < t < q_j T_j + O_j$$

As all other task releases are at the times assumed in Step 1, all other task invocations which interfered with τ_i in the interval [0, kT_j) still interfere and therefore: $C_i'(t) \geq C_i(t)$ and $C_i'(kT_j) \geq C_i(kT_j + O_j)$.

Thus all q_j invocations of τ_j still interfere with τ_i giving τ_i a response time at least as large as in Step 1.

Repeatedly applying Steps 1 and 2 proves that $O_j = 0$ ($\forall \tau_j \in sp(i)$) gives a response time for τ_i which is at least as long as that for any arbitrary set of task offsets.
□

4. A Sufficient Feasibility Test

From Theorem 1, the maximum time for which task τ_i may be prevented from executing due to the execution of tasks with longer worst case execution times is:

$$B_i = C_i$$

unless the task has the largest computation time (i.e τ_n), in which case the blocking time (B_i) is zero.

From Theorem 2, an upper bound on the interference which task τ_i is subject to, due to the execution of tasks with shorter worst case execution times, is given by:

$$I_i = \sum_{j \in sp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

An upper bound on the worst case response time of task τ_i is thus given by:

$$R_i \ = \ C_i \ + \ B_i \ + \sum_{j \in sp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

where $B_i = C_i$ for all τ_i except τ_n; for τ_n, $B_n = 0$

As R_i appears on both sides of this equation (1) and the summation term is a monotonically increasing function of R_i, it may be solved via a recurrence relation (this is a standard technique in FP analysis [1]).

$$r_i^{n+1} \ = \ C_i \ + \ B_i \ + \sum_{j \in sp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j$$

Iteration starts with $r_i^0 = C_i$ and terminates when $r_i^{n+1} = r_i^n$ or when $r_i^n > D_i$ in which case the response time of task τ_i is greater than its deadline and the task is unschedulable.

We note that this test is pessimistic, it assumes that releases of a task τ_j with a shorter wcet than τ_i will always interfere with task τ_i. However this is only in fact the case if the remaining computation time of τ_i is greater than C_j. Consider the task set given in the following table:

task	C	T	R	R'
τ_1	2	4	4	4
τ_2	3	7	5	7

The values in column R give the actual worst case response time of the task, whilst the values in column R' give the pessimistic values calculated using the above sufficient feasibility test.

The timing diagram given below illustrates the actual execution of the tasks under SRPT. The following points should be noted:

- At time 4, τ_1 is released for a second time but it does not preempt τ_2 as $C_2(4) = 1$ and $C_1 = 2$. The pessimistic analysis assumes τ_1 does preempt and hence R_2' takes a value of 7 rather than 5.
- The third release of τ_1 shows the effect of 'blocking'. At time 8, $C_2(8) = 2$; C_1 is not less than this value so τ_1 is blocked until τ_2 completes.

This simple example also illustrates a further interesting point. If the deadlines of the two tasks were (3,7) then they can be scheduled by FP but not by SRPT. Alternatively deadlines of (4,5) are amenable to SRPT but not FP. For FP the tasks have worst case response times of (2,7), for SRPT they are (4,5).

5. An Exact Feasibility Test

We now derive an exact feasibility test for tasks scheduled according to the shortest remaining processing time algorithm. This test follows the same form as the sufficient test given above. The pessimism of the above approach comes from the assumption that later arrivals of 'higher priority' tasks will always interfere. In reality they will only

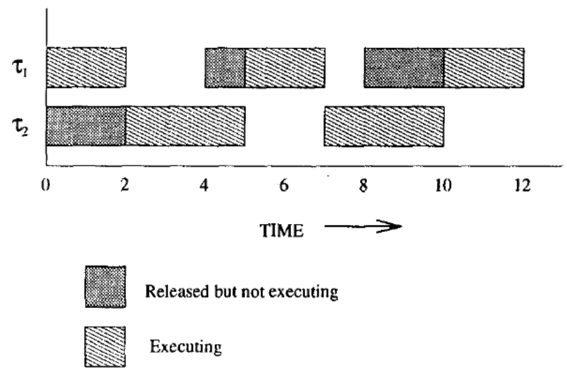

Figure 1. Example task set execution

interfere if they arrive with a computation time shorter than the remaining processing time of the task under consideration (τ_i).

The method for finding the worst case response time of task τ_i follows that used by Davis et al [2] to calculate slack time, it relies upon two equations. Equation (2) determines the length of the busy period $w_i^{n+1}(t)$ starting at time t, during which tasks with a remaining execution time of less than $C_i(t)$ execute in preference to τ_i.

$$w_i^{n+1}(t) = B_i(t) + \sum_{j \in sct(C_i(t))} \left(\left\lfloor \frac{w_i^n(t) + t}{T_j} \right\rfloor + 1 - \left\lceil \frac{t}{T_j} \right\rceil \right) C_j \quad (2)$$

where $sct(X)$ is the set of tasks with shorter computation time than X.

Iteration starts with $w_i^0 = 0$ and ends when $w_i^{n+1} = w_i^n$, w_i^{n+1} then gives the length of the busy period.

Given that time t is the end of a 'busy period' during which tasks with remaining computation times less than $C_i(t)$ execute, equation (3) determines the length of time for which τ_i executes before being pre-empted by a task with a shorter remaining execution time.

$$V_i(t, C_i(t)) = min \left[\begin{array}{l} C_i(t), \\ min_{\forall j \in sct(C_i(t))} M_i(t, C_i(t)) \end{array} \right] \quad (3)$$

where

$$M_i(t, C_i(t)) = \left\lceil \frac{t}{T_j} \right\rceil T_j - t \quad if C_j < C_i(t) - \left(\left\lceil \frac{t}{T_j} \right\rceil T_j - t \right); \quad \infty \quad otherwise$$

Combining equations (2) and (3) our method for determining the worst case response time (R_i) proceeds as follows:

1. The remaining execution time of τ_i, $C_i(t)$ is initially set to C_i and its response time R_i is set to zero.
2. Equation (2) is used to compute the length of the busy period. This is added to R_i.
3. The end of the busy period is used as the start of a period of task τ_i execution, the length of which is calculated using equation (3).
4. The remaining execution time of τ_i is decremented by the length of the 'idle period' found in Step 3. The response time is incremented by the length of the 'idle period'.
5. If the remaining execution time of τ_i is zero then R_i gives its response time. Otherwise if R_i is less than the deadline of task τ_i we repeat Steps 2 - 5. If R_i is greater than or equal to the deadline and the remaining execution time is non-zero, then task τ_i is unschedulable.

This method may be implemented as detailed in the algorithm below:

```
for each task i do
  t := 0
  C := C_i
  w(n+1) := 0
  B := B_i
  while t <= D_i and C > 0 do
    w(n) := w(n+1)
    w(n+1) := -- via equation (2)
    if w(n) = w(n+1) then
      t := t + w(n)
      V := -- via equation (3)
      t := t + V
      C := C - V
      w(n+1) := 0
      B := 0
    end if
  end do

  if C = 0 then
    Task is schedulable,
    response time is R = t
  else
    Task is not schedulable, exit

end do
```

6. An Example Task Set

In this section we present a more extensive example that was analysed by a prototype tool that implements the exact algorithm described above. The example task set is based upon the GAP case study described by Locke et al [6]. In Table 1 the task set is given in the order defined by the rate monotonic algorithm. The response times are calculated using standard analysis for FP scheduling.

task	T	D	C	R
ω_1	250	250	9	9
ω_2	250	250	25	34
ω_3	400	400	10	44
ω_4	500	500	35	79
ω_5	500	500	60	139
ω_6	590	590	62	201
ω_7	700	700	28	229
ω_8	700	700	37	300
ω_9	1000	1000	61	361
ω_{10}	2000	2000	11	372
ω_{11}	2000	2000	12	384
ω_{12}	2000	2000	18	412
ω_{13}	2000	2000	39	451
ω_{14}	2000	2000	40	491
ω_{15}	10000	10000	19	800
ω_{16}	10000	10000	20	830

Table 1. FP Priority order

Table 2 gives the ordering dictated by SRPT and the response times found by the exact analysis described earlier. The example task set was also simulated under the standard earliest deadline first (EDF) scheduling algorithm.

task	T	D	C	R
ω_1	250	250	9	18
ω_3	400	400	10	29
ω_{10}	2000	2000	11	41
ω_{11}	2000	2000	12	54
ω_{12}	2000	2000	18	78
ω_{15}	10000	10000	19	98
ω_{16}	10000	10000	20	119
ω_2	250	250	25	149
ω_7	700	700	28	180
ω_4	500	500	35	222
ω_8	700	700	37	270
ω_{13}	2000	2000	39	336
ω_{14}	2000	2000	40	377
ω_5	500	500	60	467
ω_9	1000	1000	61	563
ω_6	590	590	62	564

Table 2. SRPT Priority order

Two ways of comparing the FP, SRPT and EDF approaches (other than noting that they all schedule this task set) is to examine the observed mean response times for all tasks; either worse case or mean actual response time up to the LCM (Least Common Multiple) of their periods. These results are contained in Table 3, Table 4 and Table 5. It is clear from these tables that while all approaches schedule the task set, the mean response times of the tasks are significantly lower under the SRPT scheduling algorithm. Figure 7 illustrates this point. Here the cumulative density of the tasks' response times is show.

task	Mean R FP	Mean R EDF	Mean R SPTF
ω_1	9.0	21.5	9.1
ω_2	34.0	29.5	45.0
ω_3	16.8	22.6	11.9
ω_4	71.5	101.5	90.2
ω_5	134.0	114.0	193.6
ω_6	104.4	106.3	150.7
ω_7	76.5	103.7	41.8
ω_8	133.9	109.4	93.6
ω_9	265.8	265.8	313.4
ω_{10}	295.1	366.1	30.0
ω_{11}	309.6	367.2	42.2
ω_{12}	334.8	379.9	60.3
ω_{13}	395.6	414.5	187.51
ω_{14}	450.6	400.5	234.21
ω_{15}	545.6	565.8	79.2
ω_{16}	579.2	566.8	99.3

Table 3. Mean response times

	Mean	Standard Deviation
FP	321.0	240.1
EDF	352.6	246.7
SRPT	216.8	176.8

Table 4. Comparison of mean worst-case response times

	Mean	Standard Deviation
FP	104.3	124.1
EDF	110.7	128.8
SRPT	82.2	93.2

Table 5. Comparison of mean actual response times

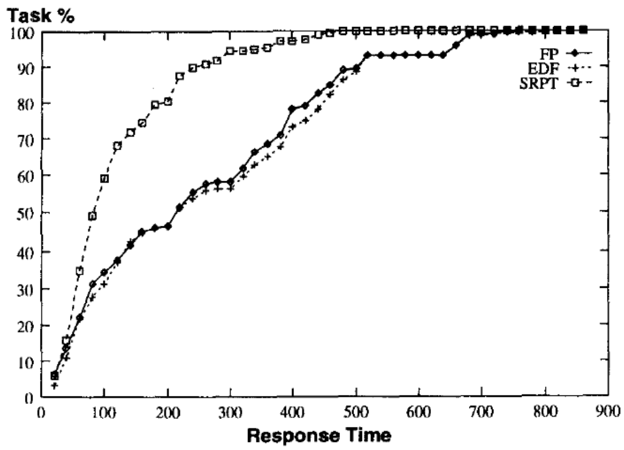

Figure 2. Cumulative response times

7. Summary

In this paper, we provided sufficient and exact feasibility tests for task sets scheduled under the SRPT algorithm. These tests enable *a priori* analysis to be used to determine if tasks will always meet their deadlines. This analysis therefore permits for the first time, the use of SRPT and other similar algorithms as a fundamental scheduling approach in real-time systems.

The SRPT algorithm is of particular interest as its use along with the analysis described in this paper allows task deadlines to be guaranteed whilst also minimising the mean response times of tasks.

References

[1] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] R. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings Real-Time Systems Symposium*, 1993.

[3] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.

[4] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

[5] C. Locke. Best-effort decision making for real-time scheduling. CMU-CS-86-134 (PhD Thesis), Computer Science Department, CMU, 1986.

[6] C. Locke, D. Vogel, and T. Mesler. *Building a Predictable Avionics Platform in Ada: A Case Study*. Proceedings of the IEEE 12th Real Time Systems Symposium, 1991.

[7] W. E. Smith. Various optimisers for single-stage production. *Naval research and Logistics Quarterly*, 3(1), 1956.