Analysis of Hard Real-Time Systems

Abha Moitra

TR 85-693
July 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

ANALYSIS OF HARD REAL-TIME SYSTEMS

Abha Moitra
Department of Computer Science
Cornell University

ABSTRACT

In this paper we study *hard real-time systems* : systems where strict time deadlines have to be met. We analyze a special case as well as a general model for hard real-time systems and study pre-emptive, static, scheduling policies for a single processor. The analysis is exact and can handle any arbitrary choice of strict deadlines. For any specification of a hard real-time system, a feasible priority assignment is one where all deadlines are met. An optimal scheduling algorithm is an algorithm that always produces a feasible priority assignment if one exists. For both the special and general model we present an optimal scheduling algorithm.

Index Terms - Deadline, pre-emptive, rate-monotonic, real-time systems, scheduling policy.

1. Introduction

A real-time processing system can be modelled as a set of processors handling requests that arrive from a number of sources, with the constraint that each request must be processed within some fixed time limits. For any given scheduling policy, it is necessary to be able to evaluate the response times for the requests. Queuing theory [8] has traditionally been used to estimate these quantities. However, in systems where strict deadlines must be met, termed as *hard real-time systems* [10], probabilistic analysis is not adequate.

A hard real time-time system P consists of n sources, $S_1,..,S_n$ with $n \geq 2$ and analysis of hard real-time systems can be undertaken when the following information is available for each source S_i.

1. The *inter-arrival* or *cycle time*, T_i, which is the minimum time between successive requests from source S_i.
2. The *processing* or *run time*, C_i, which is the maximum amount of processing required for any request from source S_i.
3. The *deadline*, D_i associated with source S_i.

Typically, the deadline for a source is specified by requiring that a request be processed within some fixed time interval.

The scheduling policy for one processor assumes that a different process is dedicated to processing requests from each source and that the arrival of an input from source S_i causes the associated process to be automatically executed. However, the system has different priority levels assigned to sources; and a request from a source of higher priority causes execution of a lower priority process to be temporarily suspended (priority value 1 is the highest priority). Execution of a suspended process is resumed only when there is no higher priority process to be executed. Such a model implies that a request may not be processed immediately, therefore requests have to be buffered.

In this paper we will only consider real-time systems to be scheduled on a uniprocessor and where a buffer of size 1 is associated with each source. We will also assume that each source is assigned to a unique priority level and that there is no cost associated with switching the processor. We call this *Model*1.

In general, the inter-arrival, processing and deadline times may be real numbers but to take account of finite machine precision we can always scale the original problem P by multiplying all these times by a suitable constant to yield an equivalent problem with only integer parameters. We will assume that such a scaling is always done. We will also use $LCM(X_1,..,X_m)$, $\lceil X_1/X_2 \rceil$ and $\lfloor X_1/X_2 \rfloor$ to denote the least common multiple, the ceiling and the floor functions for integer arguments. We also use $[t_1, t_2)$ to denote the closed-open interval from t_1 up to, but not including, t_2.

In this paper we will first analyze a special case of *Model*1 obtained by setting $D_i = \infty$ for all sources. This is an important class of problems and we term this as *Model*2. We will also show how the analysis for this model can be extended to handle arbitrary deadlines and hence cover the entire class of problems in *Model*1.

DEFINITION 1. For a given priority assignment, a request X is said to be *lost* or *overwritten* if a new request from the same source arrives so that the processing for request X cannot be started.

In *Model*2, a priority assignment in which no requests are ever lost is acceptable. In our analysis we will make a simplification that a request is available for reading (and hence processing) until another request arrives from the same source. This assumption is realistic, provided a process that handles a request reads the entire request as its first action in near-zero time.

Although we investigate scheduling algorithms under *Model*1, a large number of other models have been studied in the literature. Scheduling of unit time requests has been studied by Garey et. al [4] and Frederickson [3]. Scheduling on a multiprocessor systems has been studied by Gonzalez and Soh [6], Dhall and Liu [2], Gonzalez and Sahni [5], Simons [13], Lawler and Martel [9] and Bertossi and Bonuccelli [1]. Scheduling of three source systems on a single processor with limited pre-emption is studied in Mok[12]; the other details of their model are also different.

The remainder of the paper is organised as follows. Section 2 introduces the terminology used. Section 3 investigates priority assignments for *Model*2 while section 4 analyzes scheduling algorithms for *Model*2. Section 5 extends the analysis for *Model*2 to handle *Model*1. Section 6 gives some possible extensions of our work, section 7 compares the work presented here with the existing work and section 8 presents some conclusions.

2. Terminology and Assumptions

We make the following assumption about the environment.

A1. A request is available for processing until it is overwritten by the arrival of the next request from the same source.

Since we are interested in dealing with strict deadlines; we will always perform a *worst case analysis*, where for each source S_i requests arrive at times $m * T_i$, $m = 0,1,2,..$. This will also allow us to infer that if all the requests meet their deadlines in the time interval $[0, LCM(T_1,..,T_n)]$ for the worst case analysis then all requests will always meet their deadlines. In the worst case the n sources can impose a *processor utilization factor* or *processor load* $U_n = \sum_{i=1}^{n} \frac{C_i}{T_i}$ and we will assume that $U_n \leq 1$.

A *scheduling algorithm* is a set of rules determining which request must be processed at any particular moment. We shall be studying ***pre-emptive***, ***priority-driven*** scheduling algorithms; i.e., whenever a request arrives from a higher priority source than the one being currently served, the running process is suspended and the requested process is started. We shall only consider ***static*** scheduling algorithms : i.e. those where the priorities assigned to sources are never changed. A scheduling algorithm generates one ***priority assignment*** for the sources of a given problem. A priority assignment for a problem is said to be ***feasible*** for a source if all requests from that source meet their deadlines. A priority assignment for a problem is said to be a ***feasible priority assignment***, ***fpa***, if under that priority assignment requests always meet their deadlines. An ***optimal scheduling algorithm*** is a scheduling algorithm that generates a feasible priority assignment for a problem whenever one exists for that problem.

DEFINITION 2. For problem P let $A = \{j_1,..,j_m\}$ be a non-empty subset of the source indices in P and let $Y = LCM(T_{j_1},..,T_{j_m})$. Define *Maximum Cover* for A, $MC_{j_1,..,j_m}$, to be the maximum time less than or equal to Y for which the processor can be utilized continuously by the sources named in A.

LEMMA 1. Consider a problem P where source S_j is assigned priority j. If $S_1,..,S_i$, is an fpa then T is equal to $MC_{1,..,i}$ iff the following hold

$$\sum_{j=1}^{i} \lceil T / T_j \rceil * C_j = T \tag{1}$$

$$\forall T' : T' < T : \sum_{j=1}^{i} \lceil T' / T_j \rceil * C_j \geq T' \tag{2}$$

$$\sum_{j=1}^{i} \lceil T + 1 / T_j \rceil * C_j < T + 1 \tag{3}$$

PROOF. (1) and (2) imply that the processor can be continuously utilized up to time T by the sources $S_1,..,S_i$. In addition if (3) is satisfied then T represents the maximum time for which the processor can be continuously utilized by requests arriving from sources $S_1,..,S_i$. □

3. Feasible Priority Assignment for *Model2*

We now consider how a priority assignment can be checked for feasibility and under what general conditions a feasible priority assignment always exists for a problem under *Model2*.

3.1 Checking Priority Assignment

For any particular priority assignment it is a straightforward task to check whether the priority assignment is feasible or not. This can be done using a graph paper and allocating the processor to the processor according to the priority. This allocation has to be done for the interval $[0,\mathrm{LCM}(T_1,..,T_n)]$ and if each request meets its deadline then the priority assignment is feasible otherwise not. This construction can be mimicked by the following algorithm.

```
var A : array [0:Y] of boolean;          (* Y = LCM(T1,..,Tn) *)
    C,T : array [1:n] of integers;
    FPA : array [1:n] of booleans;
    J,K,AMT,ARRIVAL : integer;
for J:=1 to Y do A[J]:= true;
for J:= 1 to n do
  begin
    ARRIVAL:= 0; AMT:= C[J]; FPA[J]:= true;
    while ARRIVAL < Y do
```

```
        begin
          K:= ARRIVAL;
          while AMT > 0 do
            begin
              if K ≥ ARRIVAL + T[J] & AMT = C[J]
                then FPA[J]:= false;
              if A[K] then begin A[K]:= false; AMT:= AMT-1 end;
              K:=K+ 1
            end;
          ARRIVAL:= ARRIVAL+ T[J]
        end;
    end;
```

Fig. 1. Feasibility Checking Algorithm.

In the above algorithm we could have halted as soon as $FPA[J]$ is set to false; however, we will later make use of the entire FPA array. The cost of the above algorithm is $O(n*\ \mathrm{LCM}(T_1,..,T_n))$ (however, its average cost can be reduced considerably by rearranging the computation so that the minimal time interval required for demonstrating feasibility/infeasibility is used as opposed to the time interval $[0,\mathrm{LCM}(T_1,..,T_n)]$).

3.2 Critical Processor Utilization

Consider Ex. 1 for which we will show that no fpa exists.

Ex. 1

	S_1	S_2	S_3	S_4
C	1	1	1	1
T	3	4	5	6

Here $U = 57/60$.

We say that a source S_i should not be assigned the lowest priority if such an assignment would lead to some request being lost. We can show that no fpa exists by showing that none of the sources should be assigned the lowest priority. For Ex. 1 we have

$MC_{1,2,3} = 7$ which is greater than T_4 and hence T_4 cannot have lowest priority
$MC_{1,2,4} = 5$ which is equal to T_3 and hence T_3 cannot have lowest priority
$MC_{1,3,4} = 4$ which is equal to T_2 and hence T_2 cannot have lowest priority
$MC_{2,3,4} = 3$ which is equal to T_1 and hence T_1 cannot have lowest priority

The processor utilization factor for a problem P is said to be *critical*, if no fpa exists for P but a unit decrease in the run-time of any request, or a unit increase in the cycle-time of any request, makes an fpa possible. Altering the cycle-time or the run-time of a problem changes the problem being considered; but the number of sources being considered remains constant. We can therefore consider the set of all problems with the same number of sources and determine for that set the minimum load for which no fpa exists. Hence we will now define the critical processor utilization factor, CU_m, for m sources such that for all m source problems that have process utilization factor below the value derived for CU_m, there exists an fpa. On the other hand, if the processor utilization is above the derived value of CU_m, an fpa may exist if the run-times and the cycle-times for the problem are suitably related.

THEOREM 1. Without loss of generality let the sources of P be numbered so that $T_i \leq T_{i+1},\ 1 \leq i < n$. None of the sources should be assigned the lowest priority and the processor utilization factor U_n is minimized if

$$T_i = 2*\sum_{j=1}^{i-1} C_j + \sum_{j=i+1}^{n} C_j \qquad 1 \leq i \leq n-1 \tag{4}$$

$$T_n = C_1 + 2*\sum_{j=1}^{n-1} C_j \tag{5}$$

$$C_i = C_1 \qquad\qquad 1 \le i \le n \tag{6}$$

PROOF. The proof is by induction on i in (4), (5) and (6); i.e., on the structure of T_i and C_i.

Base Case. $i = 1$. If S_1 is assigned the lowest priority and all the requests arrive at time=0 then at least time $T = \sum_{i=2}^{n} C_i$ is spent by the processor in processing all the requests arriving at time $= 0$. So $MC_{2,3,..,n} \ge \sum_{i=2}^{n} C_i$. (Note $MC_{2,3,..,n}$ will be greater than $\sum_{i=2}^{n} C_i$ if $T_2 \le \sum_{i=2}^{n} C_i$.) Since we are interested in minimizing U over all possible cases, we are interested in determining, irrespective of the run times and cycle times for requests from sources S_i, $2 \le i \le n$, the largest value of T_1 that will ensure that S_1 should not be assigned the lowest priority. We get $T_1 = \sum_{i=2}^{n} C_i$ and hence (4) and (6) are satisfied.

Induction Step. Assume that for all $k < i$, $i \le n-1$,

$$T_k = 2*\sum_{j=1}^{k-1} C_j + \sum_{j=k+1}^{n} C_j \quad \text{and} \quad C_k = C_1$$

If S_i, $i > 1$, is assigned the lowest priority and requests from all the sources arrive at time $= 0$, then time $T = \sum_{j=1}^{i-1} C_j + \sum_{j=i+1}^{n} C_j$ is spent by the processor in processing the first requests that arrive from all the sources except S_i. So $MC_{1,2,..,i-1,i+1,..,n} \ge T$ (note if $i = n$ then $MC_{1,2,..,i-1,i+1,..,n} = MC_{1,2,..,n-1}$). We now consider two cases.

I. $C_1 < C_i$. In that case $T < \sum_{j=2}^{n} C_j$ and hence $T < T_1$. Since $T_j \geq T_1$, $1 \leq j \leq n$ it follows that a second request from any source in $\{S_1,..,S_{i-1}, S_{i+1},..,S_n\}$ cannot arrive in time to give $MC_{1,..,i-1,i+1,..,n} > T$. Hence $MC_{1,..,i-1,i+1,..,n} = T$ and so $T_i \leq T$ but this yields $T_i < T_1$ which is a contradiction.

II. $C_1 \geq C_i$. In that case $T = \sum_{j=2}^{n} C_j$ and hence $T = T_1$. Therefore a second request from source S_1 can arrive in time to extend $MC_{1,..,i-1,i+1,..,n}$ to $T + C_1$. Since for $2 \leq j \leq i$-1, $T_j = T_{j-1} + C_{j-1}$ it follows that a second request can arrive from sources $S_2,..,S_{i-1}$ so as to give $MC_{1,..,i-1,i+1,..,n} \geq 2*\sum_{j=1}^{i-1} C_j + \sum_{j=i+1}^{n} C_j$. If $i = n$ then $2*\sum_{j=1}^{n-1} C_j \geq 2*T_1$ and hence a third request from source S_1 can arrive in time to give $MC_{1,..,n-1} \geq C_1 + 2*\sum_{j=1}^{n-1} C_j$. As before, we are interested in minimizing U over all possible cases, and hence (4), (5) and (6) hold. □

THEOREM 2. For problem P of n sources, if $n \leq 3$, then $CU_n = 1$ otherwise $CU_n = \sum_{i=-1}^{n-3} \frac{1}{n+i} + \frac{1}{2n-1}$.

PROOF. From Theorem 1 we obtain the result that none of the sources in problem P should be assigned the lowest priority and the process utilization factor is minimized if (4), (5) and (6) are satisfied. For $n \leq 3$ these imply $U_n > 1$.

Hence for $n \leq 3$ there is no way in which the cycle and run times can be arranged so that no fpa exists while $U_n \leq 1$. So for $n \leq 3$ $CU_n = 1$. For $n \geq 4$, (4), (5) and (6) give $CU_n = \sum_{i=-1}^{n-3} \frac{1}{n+i} + \frac{1}{2n-1}$. □

If $n = 4$, then from Theorems 1 and 2 we obtain that problems $P1$ and $P2$ both do not have any fpa.

$P1$	S_1	S_2	S_3	S_4
C	1	1	1	1
T	3	4	5	7

$P2$	S_1	S_2	S_3	S_4
C	2	2	2	2
T	6	8	10	14

Further, problem $P3$ also does not have any fpa

$P3$	S_1	S_2	$\cdots$	S_{n-1}	S_n
C	1	1	$\cdots$	1	1
T	$n-1$	n	$\cdots$	$2*n-3$	$2*n-1$

In $P3$ if T_i is increased by any amount, d, then the process utilization factor for the altered problem falls below CU_n and an fpa should exist for it. Fig. 2 shows that the assignment of priority j to S_j when $d < 1$ yields an fpa (here for simplicity we have not scaled the problem). Similarly, if C_i is decreased by any amount, then assigning priority j to S_j also yields an fpa.

processing | I S_n S_i II S_{n-1} S_1 S_{n-1} S_n S_2 S_n S_3 free

d 1-d d 1-d

time 0 n+ i-2 n+ i-1+ d 2n-3+ d 2n-2 2n-1 2n 2n+ 1 2n+ 2 2n+ 3

S_n arrival ^ ^

"I" accounts for the complete processing for requests from S_1, S_2,..,S_{n-1}, S_1, S_2,..,S_{i-1} in that order

"II" accounts for the complete processing for requests from S_{i+1},..,S_{n-2}

Fig. 2. Processor Allocation.

4. Scheduling Algorithms for *Model*2

Having investigated conditions under which an fpa always exists, we now derive a scheduling algorithm to generate an fpa whenever possible. A very simple and extensively used scheduling algorithm is based on the idea that sources with lower cycle times should be assigned higher priorities; this is also called *rate-monotonic* priority assignment. Unfortunately, for *Model*2 this scheduling algorithm does not always give an fpa when one exists. We illustrate such this situation in Ex. 2.

Ex. 2

	S_1	S_2	S_3	
C	4	2	1	
T	5	12	30	
PA	1	2	3	rate-monotonic priority assignment which is not fpa
PA'	1	3	2	a priority assignment which is an fpa

We will derive an optimal scheduling algorithm for *Model*2 which will be a variant of the rate-monotonic scheduling algorithm and hence we will first investigate the rate-monotonic scheduling algorithm in more detail.

4.1 Rate-Monotonic Scheduling Algorithm

We will now investigate the conditions under which the rate-monotonic scheduling algorithm always yields an fpa for *Model*2. That is, for a problem of n sources we derive the bound TU_n, on the processor load, such that for any problem P with load less than TU_n the rate-monotonic scheduling algorithm gives an fpa while for any problem P with processor load greater than or equal to TU_n the rate-monotonic scheduling algorithm may or may not give an fpa.

THEOREM 3. For problem P, if $MC_{1,..n} = y$, then another problem specification P' obtained from P in which all the cycle times and the run times are multiplied by a factor m has $MC_{1,..,n} = m*y$.

PROOF. Obvious, since the multiplication of all the cycle times and the run times by a constant can be thought of as changing the unit of measurement. □

THEOREM 4. For problem P with $U_n \leq 1$ and sources numbered so that $T_i \leq T_{i+1}$, $1 \leq i < n$; the rate-monotonic algorithm does not give an fpa if

$$MC_{1,..,n-1} \geq T_{n-1} \tag{7}$$

PROOF. Let

$$1 - \sum_{i=1}^{n-1} \frac{C_i}{T_i} = \frac{x}{y} \qquad \text{where } y \geq T_{n-1} \text{ and let } z = MC_{1,..,n-1}$$

So, x/y represents the maximum amount of load that can be imposed by the source S_n so as to keep $U_n \leq 1$. If $z \geq T_{n-1}$ then construct another problem P' from P by scaling up the cycle and run times of the first $n-1$ sources by a factor m such that $m*z \geq y$. Then P and P' have the same load and $MC_{1,..,n-1}$ for P' is $m*z$ (by Theorem 3). Now we can choose $C_n = x$ and $T_n = y$ so that $U_n \leq 1$ and the rate-monotonic scheduling algorithm will not result in an fpa for P'. So, a rate-monotonic scheduling algorithm does not result in an fpa if $MC_{1,..,n-1} \geq T_{n-1}$ □

THEOREM 5. Consider problem P of n sources and processor load $U_n \leq 1$. For $n = 2$, $TU_n = 1$ otherwise $TU_n = (n-1)*(2^{1/(n-1)} - 1)$.

PROOF. Without loss of generality assume that the sources for the problem P are numbered so that $T_i \leq T_{i+1}$, $1 \leq i < n$. From Theorem 4 we know that the rate-monotonic scheduling algorithm does not give an fpa if $MC_{1,..,n-1} \geq T_{n-1}$. If the problem P has only 2 sources then $MC_1 = C_1 \leq T_1$, which contradicts the assertion $MC_{1,..,n-1} \geq T_{n-1}$ and hence the rate-monotonic scheduling algorithm gives an fpa if $U_n \leq 1$. So, for $n = 2$, $TU_n = 1$.

We now consider $n > 2$. $MC_{1,..,n-1} \geq T_{n-1}$ implies that at least a second request from all the highest $n-1$ priority sources can arrive in time to extend $MC_{1,..,n-1}$. That is,

$$2*\sum_{j=1}^{i-1} C_j + \sum_{j=i}^{n-1} C_j \geq T_i \quad 1 \leq i \leq n-1 \tag{8}$$

Since, we want to minimize U we get

$$2*\sum_{j=1}^{i-1} C_j + \sum_{j=i}^{n-1} C_j = T_i \quad 1 \leq i \leq n-1$$

Thus,

$$C_i = T_{i+1} - T_i \qquad 1 \leq i \leq n-2$$

and a third request from S_1 can arrive in time to extend $MC_{1,..,n-1}$ so

$$C_{n-1} = 2*T_1 - T_{n-1}$$

Since in the construction of P' the processor load imposed by the source S_n can be made arbitrarily small we have

$$U = \sum_{i=1}^{n-1} \frac{C_i}{T_i} = \sum_{i=1}^{n-2} \frac{T_{i+1} - T_i}{T_i} + \frac{2*T_1 - T_{n-1}}{T_{n-1}}$$

To minimize U, we must set the first derivative of U with respect to each T_i equal to zero, i.e.

$$\frac{\partial U}{\partial T_1} = 0 \Rightarrow 2*T_1^2 = T_2*T_{n-1}$$

$$\frac{\partial U}{\partial T_i} = 0 \Rightarrow T_i^2 = T_{i-1}*T_{i+1} \qquad 2 \leq i \leq n-2$$

$$\frac{\partial U}{\partial T_{n-1}} = 0 \Rightarrow T_{n-1}^2 = 2*T_1*T_{n-2} \qquad n > 2$$

These equations can be solved to give

$$T_i = 2^{1/n-1} * T_{i-1} \qquad \text{for } 2 \leq i \leq n-1$$

Hence, $U = (n - 1)*(2^{1/(n-1)} - 1)$ □

4.2 Optimal Scheduling Algorithm

In this section we develop an optimal scheduling algorithm for *Model*2. Consider the rate-monotonic priority assignment for the problem P. If $U_n \leq (n - 1)*(2^{1/(n-1)} - 1)$ then the rate-monotonic priority assignment is a fpa and we are done with a total cost $= O(n \log n)$. Otherwise, check whether this priority assignment is a fpa using the feasibility checking algorithm of section 3.1. The cost of applying this algorithm is $O(n * \mathrm{LCM}(T_1,..,T_n))$. If the priority assignment is a fpa then the total cost is $O(n \log n + n * \mathrm{LCM}(T_1,..,T_n)) = O(n * \mathrm{LCM}(T_1,..,T_n))$.

If the rate-monotonic priority assignment is not a fpa then we alter the priority assignment as follows. Let Fig. 3 describe the *FPA* array after the initial application of the algorithm.

S_1			S_i	
t	...	t	f	...
1			i	

Fig. 3. Array FPA

where i is the leftmost index with $FPA(i)=false$. Now consider the subproblem of finding a fpa for the i sources, $S_1,..,S_i$ of Fig. 3. Now if a fpa exists then there exist some source to which we can assign the lowest priority (i.e. ith priority). To check whether jth source can be assigned the lowest priority we consider a new priority assignment obtained by shifting S_j behind S_i, i.e.

$$S_k \text{ has } \begin{array}{l} \text{priority } k \text{ if } k < j \\ \text{priority } i \text{ if } k = j \\ \text{priority } k-1 \text{ if } k > j \end{array}$$

and then apply the feasibility checking algorithm. We work leftwards from the $(i-1)$th source until we succeed in finding a source that can be shifted behind S_i. If we do not succeed then no fpa exists for this subproblem as well as for the original problem. Suppose S_j can be shifted behind S_i then the sources ordered on priority are $S_1,..,S_{j-1},S_{j+1},..,S_i,S_j$. We will now argue that this priority assignment is a fpa (for this subproblem).

THEOREM 6. Let the sources ordered on rate-monotonic priority assignment be $S_1,..,S_{j-1},S_j,S_{j+1},..,S_i$; and let this priority assignment be feasible for $S_1,..,S_{i-1}$ but not for S_i. Consider the priority assignment $S_1,..,S_{j-1},S_{j+1},..,S_i,S_j$; this is a fpa if the priority for S_j is feasible.

PROOF. For all sources other than S_i the priority assignment is feasible follows immediately. We will now prove that this priority assignment is feasible for S_i. We do this by proving that for all $k \geq 1$, at least $1 + (k-1)C_i$ units of processing is allocated to source S_i in time interval $[0,kT_i]$. The proof is by induction.

Base Case. $k = 1$. Since S_j has a feasible priority it implies that at least 1 unit of processing is allocated to S_j in $[0,T_j]$. However, since S_i is at a higher priority than S_j it implies that C_i units of processing is allocated in $[0,T_j]$. Hence at least 1 unit of processing is allocated to S_i in $[0,T_i]$.

Induction Step. Assume that $\forall\, h \leq m$; at least $1 + (h-1)C_i$ units of processing is allocated to S_i in $[0,hT_i]$. Since $T_i \geq T_j$ it follows that $\exists\, p : pT_j \leq mT_i$ and $mT_i < (p+1)T_j \leq (m+1)T_i$.

Case 1. The amount of processing allocated to S_j in $[pT_j,mT_i]$ is zero. Since S_j

has a feasible priority it must be allocated at least 1 unit of processing in $[mT_i,(p+1)T_j]$. However since S_i has a higher priority than S_j all processing for its requests must be completed before processing can be done for S_j. Hence, we obtain $(m+1)C_i$ units of processing for S_i in $[0,(m+1)T_i]$; i.e. at least $1 + mC_i$ units of processing for S_i in $[0,(m+1)T_i]$.

Case 2. The amount of processing allocated to S_j in $[pT_j,mT_i]$ is greater than zero. This implies that mC_i units of processing is allocated to S_i in $[0,mT_i]$. We now have to show that we get at least one more unit of processing for S_i in $[mT_i,(m+1)T_i]$. For this note that $MC_{1,..,j-1,j+1,..,i-1} \leq T_j - C_i - 1 < T_i$ (because we obtain at least 1 unit of processing for S_j in $[0,T_j]$). By definition of MC it follows that from time mT_i onwards the maximum amount of time for which the processor can be continously utilized by the sources $S_1,..,S_{j-1},S_{j+1},..S_{i-1}$ is less than T_i and hence at least 1 more unit of processing is obtained for S_i in $[mT_i,(m+1)T_i]$. So we obtain at least $1 + mC_i$ units of processing for S_i in $[0,(m+1)T_i]$. $\square$

So the total cost of removing the infeasibility of 1 source out of n sources is $O(n \log n + n^2 * \mathrm{LCM}(T_1,..,T_n)) = O(n^2 * \mathrm{LCM}(T_1,..,T_n))$.

If the rate-monotonic priority assignment yields the following situation

S_1			S_{i_1}				S_{i_2}	
t	...	t	f	t	...	t	f	...
1			i_1				i_2	

Fig. 4. Array FPA.

then we can attempt to remove the infeasibility of both S_{i_1} and S_{i_2} at the same time. That is we will first attempt to move (simultaneously) S_{i_1-1} and S_{i_2-1} over

S_{i_1} and S_{i_2} respectively. Notice that if we succeed in finding S_{j_m} ($m = 1,2$) that can be moved over S_{i_m} then S_{i_m} will be feasible since $T_{i_m} \geq T_{j_m}$. This can be extended so that we attempt to remove all the infeasibilities at the same time. If at some point we have the situation

S_1			S_i	S_{i+1}	
t	...	t	f	f	...
1			i	$i+1$	

Fig. 5. Array FPA.

then we would first attempt to move S_{i-1} over both S_i and S_{i+1}.

Clearly, the worst case complexity of obtaining a fpa is when only S_n is infeasible in the rate-monotonic priority assignment and a fpa can only be obtained by shifting S_1 behind S_n. In that case the total cost is $O(n \log n + n^2 * \mathrm{LCM}(T_1,..,T_n)) = O(n^2 * \mathrm{LCM}(T_1,..,T_n))$. We now present the optimal scheduling algorithm developed for *Model*2.

Optimal Scheduling Algorithm for *Model*2 :

1. Assign rate-monotonic priorities to the sources.
2. If $U_n \leq (n-1)*(2^{1/(n-1)} - 1)$ then go to step 6.
3. Apply algorithm of section 3.1 to obtain array *FPA*.
4. If $\forall i$, $1 \leq i \leq n$: $FPA[i] =$ true then go to step 6.
5. For each j such that $FPA[j] =$ false find the leftmost index $n_j : n_j < j$ and the n_jth source can be shifted behind the jth source. If n_j cannot be found then no fpa exists for the problem.

6. Stop.

5. Handling Specific Deadlines in *Model* 1

So far we have dealt with problems where the deadline was specified implicitly since a buffer of size 1 was associated with a source. This implicit deadline is the weakest possible that can be allowed with buffers of size 1, but at times the problem may specify stronger deadlines that have to be met.

First of all, the algorithm of section 3.1 has to be altered slightly to check that that the processing of each request is completed within its deadline. The cost of this algorithm remains $O(n * \mathrm{LCM}(T_1,..,T_n))$. Secondly when we have an implicit deadline, D_i', imposed for source S_i by a buffer of size 1 then $D_i' \geq T_i$. Since $D_i = \infty$, in *Model* 2 we start the optimal scheduling algorithm by first assigning priorities according to the rate-monotonic priority assignment. In *Model* 1, the D_i and T_i can be related in any arbitrary way so we first define T_i' $= \min(T_i, D_i)$ and start the optimal scheduling algorithm by assigning priorities according to T_i' (i.e. higher T_i' have lower priority). The critical thing is that if any point S_i is infeasible and S_j can be shifted behind S_i then the new priority assignment for S_i is feasible. This follows from the fact that $T_i' \geq T_j'$ and there are two cases that we have to consider.

1. If $T_i > T_j$ then the argument used in the optimal scheduling algorithm for *Model* 2 can be applied here.
2. If $D_i > D_j$ then if S_j at a lower priority than S_i can meet a stricter deadline then S_i must also meet its deadline.

So, the optimal scheduling algorithm for *Model* 1 is the same as the optimal scheduling algorithm for *Model* 2 except that Step 2 is not used and the feasibility checking algorithm of section 3.1 is altered suitably. The cost of this algorithm is $O(n^2 * \mathrm{LCM}(T_1,..,T_n))$.

6. Various Extensions

In this section we indicate some of ways in which our analysis can be extended easily. Quite often in an analysis of real-time systems we are interested in obtaining bounds for response time for each source. This can be done by a slight modification to the algorithm of section 3.1, to determine response time for source S_i we would obtain

$$\max \{\text{completion of processing of } j\text{th request of } S_i$$
$$- \text{ arrival of } j\text{th request of } S_i \mid j = 1,..,\frac{\mathrm{LCM}(T_1,..,T_n)}{T_j}\}.$$

While it is necessary to go up to $\mathrm{LCM}(T_1,..,T_n)$ in checking the feasibility of an arbitrary priority assignment, we believe that for the rate-monotonic priority assignment and the priority assignments encountered in the application of the optimal scheduling algorithms for *Model* 1 and *Model* 2 it should be possible to improve the performance of the feasibility checking algorithm. We propose to investigate this idea in future work.

Throughout this paper we have assumed that a buffer of size 1 is associated with each source. But the analysis that we have presented can be extended to handle arbitrary buffer size. This work should also be extended to handle the situation where more than one source is assigned the same priority.

7. Comparison With Other Work

In this section we compare the work presented here with that existing in the literature. We confine our attention to the relevant work done on scheduling on a single processor. Liu and Layland [10] have studied a special case of *Model*1 in which the deadline for some source is specified by requiring that the processing of a request be completed before the arrival of the next request from the same source, i.e. $D_i = T_i$ for all sources S_i. We shall call this the LL model. They impose an arbitrary but very specific deadline and hence are able to deal with only a subset of problems that can be dealt with in model *Model*1. Even between *Model*2 and the LL model, we feel *Model*2 is more realistic since quite often the requirement for a real-time problem is that all requests be handled, in which case the buffer size imposes an implicit deadline. We present below some comparison between *Model*2 and the LL model.

Since the deadline imposed by the LL model is more strict than that in *Model*2, it is reasonable to expect that a larger class of problems possess a fpa under *Model*2. For LL model, it has been shown [10] that the critical processor utilization factor for a problem of n sources is $n * (2^{1/n} - 1)$ and Table 1 summarizes the corresponding maximum values of process utilization factor which ensure the existence of fpa for LL model and *Model*2.

n	LL model $CU_n = n * (2^{1/n}-1)$	*Model2* $CU_n = \sum_{i=-1}^{n-3} \frac{1}{n+i} + \frac{1}{2*n-1}$
2	.828	1.0
3	.779	1.0
4	.756	.926
5	.743	.870
6	.734	.836
7	.728	.813
8	.724	.796
9	.720	.784
10	.717	.774
100	.695	.700
1000	.693	.693
∞	ln 2	ln 2

Table 1. Existence of fpa.

The major difference is that the rate-monotonic scheduling policy is an optimal scheduling policy for the LL model. While this policy is very efficient to implement (cost $= O(n \log n)$); a priority assignment obtained by it may not be a fpa for the LL model. So to compare the costs between the LL model and *Model*1 and *Model*2; we must also include the cost of checking for feasibility. Hence the cost of the LL model is $O(n \log n + n * \mathrm{LCM}(T_1,..,T_n))$ while the cost for *Model*1 and *Model*2 is $O(n \log n + n^2 * \mathrm{LCM}(T_1,..,T_n))$. The cost of *Model*2 does not compare favorably with that of the LL model. However, *Model*1 can handle arbitrary deadlines and its cost is not prohibitive compared to that of LL model. The interesting thing to note is that in the LL model since $D_i = T_i$, the deadline D_i is more strict than the implicit deadline D_i' and no adjustment to the rate-monotonic priority assignment is required to obtain a fpa. However, in *Model*1 and *Model*2 it is not possible to obtain exactly the implicit deadline imposed by the buffer size and hence adjustment to the rate-monotonic priority assignment is required to obtain a fpa. In fact, whenever $D_i \leq T_i$ for all sources then a priority assignment based on lower priority to higher D_i will be an optimal scheduling policy.

In Liu, Liu and Liestman [11] it is assumed that the deadline for completion of a requested computation coincides with the earliest next arrival from that source. In contrast we impose no constraint on how the cycle time, run time and deadline are to be related with each other.

The model that we have used in this paper is based on that used in Joseph [7]. The difference between [7] and our work is that we are able to do an exact analysis whereas only an approximate analysis is done in [7].

8. Conclusions

In this paper we have investigated pre-emptive, static scheduling policies for hard real-time systems for a single processor. An exact analysis for this class of problems can be undertaken when the cycle time, run time and the deadline associated with each source is known. When these values are known, we derive an optimal scheduling algorithm for this class of problems which whenever possible assigns priorities so that all the deadlines are met. The priority assignment given by the optimal scheduling algorithm is an adjustment to the well known rate-monotonic priority assignment. This work must be contrasted with the previous work for this class of problems where the optimal scheduling algorithm was known only when the cycle time and the deadline for each source was the same [10].

Acknowledgement

I would like to thank M. Joseph for introducing me to this problem and many helpful comments while this work was being done. I would also like to thank R. Cleaveland and F.B. Schneider for their helpful comments on previous versions of this paper.

REFERENCES

[1] A.A. Bertossi and M.A. Bonuccelli, "Preemptive scheduling of periodic jobs in uniform multiprocessor systems," *IPL,* vol. 16, pp. 3-6, Jan. 1983.

[2] S.K. Dhall, and C.L. Liu, "On a real time scheduling problem," *Oper. Res 2,* vol. 6, pp. 127-140, Jan. 1978.

[3] G.N. Frederickson, "Scheduling unit-time tasks with integer release times and deadlines," *IPL,* vol. 16, pp. 171-173, May 1983.

[4] M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan, "Scheduling unit-time tasks with arbitrary release times and deadlines," *SIAM J. Comput.,* vol. 10, pp. 256-269, May 1981.

[5] T. Gonzalez and S. Sahni, "Preemptive scheduling of uniform processor systems," *J. ACM,* vol. 25, pp. 92-101, 1978.

[6] M.J. Gonzalez and J.W. Soh, "Periodic job scheduling in a distributed processor system," *IEEE Trans. Aerospace Electron. Systems,* vol. 12, pp. 530-535, 1976.

[7] M. Joseph, "On a problem in real-time computing," to appear in *IPL.*

[8] L. Kleinrock, *Queueing Systems, Vol. II : Computer Applications.* Wiley Interscience, New York, 1976.

[9] E.L. Lawler and C.U. Martel, "Scheduling periodically occurring tasks on multiple processors," *IPL,* vol. 12, pp. 9-12, 1981.

[10] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM.,* vol. 20, pp. 46-61, Jan. 1973.

[11] C.L. Liu, J.W.S. Liu and L. Liestman, "Scheduling with slack time," *Acta Informatica,* vol. 17, pp. 31-41, 1982.

[12] A.K. Mok, "The design of real-time programming systems based on process models," *Proc. of the IEEE Real-Time Systems Symposium,* Austin, Texas, Dec. 1984, pp. 5-17.

[13] B. Simons, "A fast algorithm for multiprocessor scheduling," *Proc. 21st Symp. on Foundations of Computer Science,* Syracuse, NY, 1980, pp. 50-53.