SAFETY VERIFICATION OF REAL-TIME COMPUTING SYSTEMS SERVING PERIODIC DEVICES

M. Naghibzadeh

Department of Computer Engineering
Faculty of Engineering
Ferdowsi University of Mashad
Mashad, Iran

Abstract In real-time systems response to a request from a controlled object must be correct and timely. Any late response to a request from such a device might lead to a catastrophy. The possibility of a task overrun, i.e., missing the deadline for completing a requested task, must be checked and removed during the design of such systems. Safe design of real-time systems running periodic tasks under the rate monotonic preemptive (RM) scheduling strategy is considered in this paper. A safety verification procedure that is an improvement over previously developed procedures is presented.

Key Words Real-Time Systems, Scheduling, Rate-Monotonnic, Safety Verification

چکیده دراین مقاله سیستمهای بلادرنگ کامپیوتری را مطالعه کرده ایم که درآنها اجرای درخواستها باید صحیح و بموقع باشد. اگر نتوان به درخواستی بموقع جواب گفت، ممکن است حادثه جبران ناپذیری ببار آید. چنین امکانی را باید هنگام طراحی سیستم بررسی و رفع کرد. دراین مقاله می خواهیم سیستمهای بلادرنگی طراحی کنیم که در آنها هر درخواست قبل از ضرب الاجل آن اجرا شود (طراحی مطمئن). درخواستها ادواری بوده و زمانبندی اجرای آنها مبتنی بر الگوریتم «اولویت متناسب با فرکانس درخواست» است. روشی که دراین مقاله ارائه کرده ایم، روشهای پیشین را بهبود قابل توجهی بخشیده است.

INTRODUCTION

In real-time applications needed for air-traffic control, avionics, process control, patient monitoring and other mission critical computations, execution of time-critical tasks must be correct and timely. Failure of the computer system to execute service tasks within deadlines might lead to catastrophic situation and must be prevented. Any possibility of task overrun must be detected and eliminated during the design of the system. [1,3,8,10].

In this paper we consider uniprocessor systems that control periodic time-critical devices. Each device, U_i, i= 1,2, ..., n, makes service requests repeatedly at a fixed rate and each request must be accepted by the processor and the corresponding service task must be executed. Each device U_i, i= 1,2, ..., n, is characterized by three parameters R_i, E_i and D_i. R_i represents the request interval of U_i, i.e., the amount of time that the device waits between generating successive service requests, E_i represents the maximum execution time required for each service task requested by U_i, and D_i represents the deadline parameter, i.e., the amount of time that U_i allows for the system to complete the execution of its requested service task. In this paper, the deadline parameter for each device is considered to be equal to its request interval ($D_i = R_i$), i.e., the deadline for a service request generated at time t by device U_i is at time t+R_i. The inverse of R_i, i= 1,2, ..., n is a fixed value and it is called the request rate of device U_i. The devices are thus cyclic and the request rates are fixed [7,8,9].

The rate monotonic preemptive (RM) scheduling strategy [4,5,8] is used to schedule tasks for execution. Under the RM strategy the tasks requested by

the devices are assigned priorities on the basis of R_i's of the devices; a task requested by a device with a shorter R_i is assigned a higher priority than a task requested by a device with longer R_i. Moreover, a lower priority task in execution is preempted upon arrival of a request for a higher priority task.

The RM scheduling strategy is perhaps one of the most widely practiced strategies in time-critical computer control applications due to its simplicity in implementation. One question of practical importance about the RM or any other time-critical task scheduling is whether or not a given set G of devices can be served safely, i.e., without any failure of the processor to execute every requested task within the associated deadline. This question is dealt with in the following sections.

SOME BASIC PROPERTIES

In this section some basic properties of the rate monotonic scheduling strategy that are used in real-time systems for schedulig tasks, requested by periodic time-critical devices, are discussed. These properties form the basis for developing an efficient safety determination procedure.

A uniprocessor system that controls a set G_n of n periodic time-critical devices using the RM strategy will be represented by a three-tuple (1, G_n, RMP)-system. Without loss of generality, it is assumed that devices in a given system are ordered such that $R_1 \leq R_2 \leq \ldots \leq R_n$.

It was shown by Liu and Layland that the RM strategy is an optimal fixed priority strategy [8]. They also demonstrated that a (1, G_n, RMP)-system is safe if $F \leq n(2^{1/n}-1)$, where F is the processor utilization defined as

$$F = E_1/R_1 + E_2/R_2 + \ldots + E_n/R_n.$$

This condition is only a sufficient condition, not a necessary one. There are still many systems with processor utilization greater than $n(2^{1/n}-1)$ that are safe under the RM strategy. One such case which consists of four devices will be given later in this paper (see example 2).

Definition. The critical instant for a device is a time point in the system operation at which a request for a service task by the device will have the longest response time.

Liu and Layland [8] found that the critical instant for a device U_i, i= 1,2, ... , n, occurs whenever a request by the device is made simultaneously with requests from all higher priority devices. Therefore, the following corollary could be stated.

***Corollary* 1.** A (1, G_n, RMP)-system is safe if and only if whenever all devices start simultaneously at time 0, there is no task overrun either before or at time R_n.

This corollary provides the basis for an overrun possibility detection procedure. Again the detection should be done off-line during the system design. It also suggests that in order to make sure that a (1, G_n, RMP)-system is safe, i.e., without the possibility of a task overrun, the system could be simulated for the duration of R_n. This simulation method will perform well whenever the ratio R_n/R_1 is reasonably small.

A different approach is taken by Lehoczky, et. al, [6]. In this approach, the devices are checked for their safety one at a time starting from device U_1 to device U_n. In order to make sure that device U_i, i= 1,2, ..., n can be served safely, they checked to see if a certain linear inequality condition is met at any of the finite number of time instances whitin the interval $[0, R_i]$. These time instances are request-points of devices U_j, j= 1,2, ..., i. The check is made for all request-points of devices U_1, U_2, ... and U_i within the interval $[0, R_i]$.

In this paper, given a (1, G_n, RMP)-system, the devices are also checked one at a time starting with device U_1 and continuing towards device U_n. If a device is safe, it will be included in set S as a new

member. Device set S represents the set of all devices for which we have already made sure that there wili be no overrun; it is originally a null set.

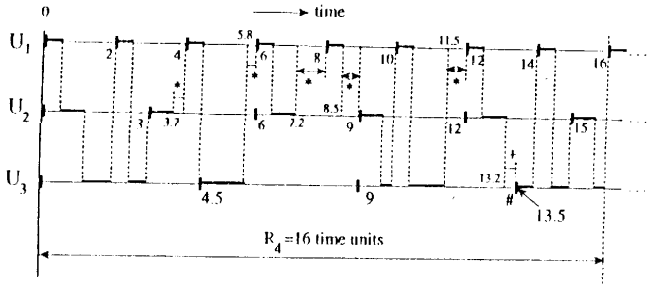
Definitions.(a) Given a (1, G_n, RMP)-system, a processor idle period (PIP) for device set S within a given interval $[t_1, t_2]$ is defined to be a period [a, b], if any, in which the processor is not needed by any of the devices in the set S.

(b) A time point T at which there are no incomplete tasks requested by the devices in set $\{U_1, U_2, \ldots, U_k\}$, except those that might have been requested at time T, is called a fully served point (FSP) of the device set.

(c) The last fully served point (LFSP) of the device set $\{U_1, U_2, \ldots, U_i\}$ within the interval $[t_1, t_2]$ is denoted by LFSP ($\{U_1, U_2, \ldots, U_i\}$, $[t_1, t_2]$).

If interval $[T_1, T_2]$ is a PIP of device set $\{U_1, U_2, \ldots, U_i\}$ and all requests generated by the devices before time T_1 are completed prior to their deadlines, any point within this interval is an FSP of the devices.

***Example* 1.** Consider a (1, G_4, RMP)-system for which $R_1=2, E_1=0.5, R_2=3, E_2=0.7, R_3=4.5, E_3=1.3$, $R_4=16$ and $E_4=2.0$. Figure 1 shows the behavior of the (1, G_3, RMP)-subsystem when it starts at time 0 and runs for R_4(= 16) time units. There are 6 PIPs of device set $\{U_1, U_2, U_3\}$ within the test interval [0, 16]. All requests generated by the devices in the device set



* PIPs of device set $\{U_1, U_2, U_3\}$ within the interval [0, 16]
\+ Last PIP of device set $\{U_1, U_2, U_3\}$ within the interval [0, 16]
# The last fully served point (LFSP) within the interval [0, 16]

Figure 1. Test execution of (1, G_3, RMP)-subsystem showing PIPs and the LFSP.

$\{U_1, U_2, U_3\}$ within the test interval are executed prior to their deadlines. Therefore, any time point within any PIP of this device set is also a fully served point (FSP) of the device set. Besides, time 0 is also a fully served point of the devices in the set. The last fully served point (LFSP) of the device set $\{U_1, U_2, U_3\}$ within the test interval is at time 13.5 as it is shown in Figure 1.

***Lemma* 1.** Given a (1, G_n, RMP)-system started at time 0, the device U_i, i= 1,2, ..., n, is safe if and only if at time T, the LFSP of devices $U_1, U_2, \ldots, U_{i-1}$ within the interval $[0, R_i]$,

$$\lceil T/R_1 \rceil * E_1 + \lceil T/R_2 \rceil * E_2 + \ldots + \lceil T/R_i \rceil * E_i \le T \qquad (1)$$

Proof. The sufficiency is obvious. As a matter of fact, if Condition 1 is satisfied at any time instant t, within the interval $[0, R_i]$, i= 1,2, ..., n, the device U_i is safe. Since devices are checked one at a time, starting from device U_1 and continuing towards device U_n, when device U_i is being checked all lower indexed devices are already checked and they are safe. Within the interval $[T, R_i]$ the processor will not be available for executing the task which is requested at time 0 by the device U_i. Therefore, it is concluded that Condition 1 is necessary, too.

Notations. (a) l_i ($\le R_{i+1}$) denotes the last request point of device U_i which is not later than R_{i+1}.

(b) l'_i ($\le R_{i+1}$) denotes the next-to-last request point of device U_i which is not later than R_{i+1}.

Definition. The maximum processor demand (MPD) of a device U_i, i= 1,2, ..., n, within an interval [0, t] is denoted by $MPD_i([0, t])$ and defined as:

$$MPD_i([0, t]) = \lfloor t/R_i \rfloor * E_i + min\{E_i, t - \lfloor t/R_i \rfloor * R_i\}. \qquad (2)$$

where $\lfloor x \rfloor$ represents the largest integer that is not greater than x and $min\{y,z\}$ represents the minimum of y and z values.

***Lemma* 2.** The execution time consumed by de-

vice U_i, i= 1,2, ..., n, within the interval [0,t] cannot exceed $MPD_i([0, t])$.

Proof. Suppose $l_i(\le t)$ is the last request point of the device U_i which is not later than t, i.e., $l_i(\le t) = \lfloor t/R_i \rfloor * R_i$. The maximum execution time that can be consumed by the device within the interval $[0, l_i(\le t)]$ is $\lfloor t/R_i \rfloor * E_i$. Since there is only one request that is made by the device within the interval $[l_i(\le t), t]$ the maximum execution time that can be consumed by the device within the interval cannot exceed E_i. At the same time, the execution time consumed by the device within the interval $[l_i(\le t), t]$ cannot exceed the length of the interval, i.e., $t - \lfloor t/R_i \rfloor * R_i$. Therefore, the execution time consumed by the device within the interval $[l_i(\le t), t]$ cannot exceed $min\{E_i, t - \lfloor t/R_i \rfloor * R_i\}$. And for the whole interval [0, t] the maximum execution time consumed by the device cannot exceed $\lfloor t/R_i \rfloor * E_i + min\{E_i, t - \lfloor t/R_i \rfloor * R_i\} = MPD_i([0, t])$.

SAFTY DETERMINATION OF REAL-TIME SYSTEMS

The following corollary shows that the amount of execution time that is consumed by the first device, i.e., the device with the highest priority, within any given interval is easily computable.

Corollary 2. Let $ET_1([0, t])$ denote the exact amount of execution time that will be consumed by the device U_1 within any given interval [0, t]. $ET_1([0, t])$ is equal to $MPD_1([0, t])$, the maximum processor demand of the device within the same interval, i.e.,

$$ET_1([0, t]) = MPD_1([0, t]) = \lfloor t/R_1 \rfloor * E_1 + min\{E_1, t - \lfloor t/R_1 \rfloor * R_1\} \quad (3)$$

Proof. Any request from the device U_1 will immediately be picked up and executed without any preemption; therefore,

$$ET_1([0, t]) = MPD_1([0, t]) = \lfloor t/R_1 \rfloor * E_1 + min\{E_1, t - \lfloor t/R_1 \rfloor * R_1\}$$

Consequently the exact amount of execution time needed by the device U_1 within an interval $[t_1, t_2]$, i.e., $ET_1([t_1, t_2])$, is equal to

$$ET_1([t_1, t_2]) = ET_1([0, t_2]) - ET_1([0, t]).$$

Two Device Systems

Since it is easy to compute $ET_1(0, R_2)$ as stated in Corollary 2, the safety determination of a 2-device system is easy to accomplish.

Theorem 1. A $(1, G_2, RMP)$-system is safe if and only if

$$ET_1(0, R_2) + E_2 = \lfloor R_2/R_1 \rfloor * E_1 + min\{E_1, R_2 - \lfloor R_2/R_1 \rfloor * R_1\} + E_2 \le R_2 \quad (4)$$

Proof. Within the interval $[0, R_2]$ whenever the processor is not needed by the device U_1, it can execute the task that is requested by the device U_2 at time 0. The execution time used by the device U_1 within the interval $[0, R_2]$ is $ET_1(0, R_2)$. Therefore the system is safe if

$$ET_1(0, R_2) + E_2 = \lfloor R_2/R_1 \rfloor * E_1 + min\{E_1, R_2 - \lfloor R_2/R_1 \rfloor * R_1\} + E_2 \le R_2.$$

If the Inequality 4 is not satisfied, it means that the execution time needed by devices U_1 and U_2 within the interval $[0, R_2]$, in order for the device U_2 to be safe, is greater than the length of the interval. But, this is not possible.

N- Device Systems

Suppose that devices $U_1, U_2, \ldots, U_{i-1}$ are all safe and the safety of the device U_i is being considered. The following theorem shows that it is not necessary to check Condition 1 for all request-points within the interval $[0, R_i]$. In Reference 6 it was suggested that

the Condition 1 be checked for all request-points within the interval.

***Theorem* 2.** If devices in the set $\{U_1, U_2, \ldots, U_{i-1}\}$ are safe, the LFSP $(\{U_1, U_2, \ldots, U_{i-1}\}, [0, R_i])$ is always later than $l'_{i-1}(\leq R_i)$, i.e., the next-to-last request point of U_{i-1} that is made within the interval $[0, R_i]$.

Proof. It is sufficient to show that there exists at least one FSP of device set $\{U_1, U_2, \ldots, U_{i-1}\}$ which is later than $l'_{i-1}(\leq R_i)$. The request generated by device U_{i-1} at time $l'_{i-1}(\leq R_i)$ must be completed by time $l'_{i-1}(\leq R_i) + R_{i-1}$. Suppose T is the time at which the task requested by device U_{i-1} is completed. At time T there does not exist any incomplete task for a device in $\{U_1, U_2, \ldots, U_{i-1}\}$ and thus T is an FSP of $\{U_1, U_2, \ldots, U_{i-1}\}$ which is later than $l'_{i-1}(\leq R_i)$.

It can be concluded from the above theorem that only the request points of all devices within the interval $[l'_{i-1}(\leq R_i), R_i]$ may be used as test-points for Condition 1. This is the main source of improvement over the methods in [6,8]. Yet further improvement is discussed below . In order to find the request-point that is the LFSP $(\{U_1, U_2, \ldots, U_{i-1}\}, [0, R_i])$, two search intervals, located within the interval $[l'_{i-1}(\leq R_i), R_i]$, are first determined. The first search interval is within the interval $[l_{i-1}(\leq R_i), R_i]$, and the second one is within the interval $[l'_{i-1}(\leq R_i), l_{i-1}(\leq R_i)]$. Both, the complete intervals $[l_{i-1}(\leq R_i), R_i]$ and $[l'_{i-1}(\leq R_i), l_{i-1}(\leq R_i)]$ can be used as search intervals. In such a case, all request-points falling within the intervals may have to be checked to see if Condition 1 is satisfied. But, it is useful to narrow down the search intervals by using an efficient procedure. The test-points are request of device set $\{U_1, U_2, \ldots, U_i\}$ that fall within the search intervals. The test-points within the first search interval are considered first and only if Condition 1 is not satisfied for any of the test-points within this search interval, the test-points within the second search interval are considered.

***Theorem* 3.** Suppose that the safety of device U_i, $i = 3,4, \ldots, n$, in a $(1, G_n, RMP)$-system is to be checked. Consider one of the two initial search intervals and let T_0 represent the lower bound of the interval, i.e., $l_{i-1}(\leq R_i)$ or $l'_{i-1}(\leq R_i)$, and T_x represent the upper bound, i.e., R_i or $l_{i-1}(\leq R_i)$.

(1) The search interval can be shortened from $[T_0, T_x]$ to $[T_1, T_x]$, where $T_1 = T_0 + E_{i-1}$. That is, if the LFSP $(\{U_1, U_2, \ldots U_{i-1}\}, [0, R_i])$ falls within the interval $[T_0, T_x]$, then it is never earlier than T_1.

(2) In addition, the search interval can be shortened by the iterative application of the following derivation formula until either $T_j > T_{i-1}$ or $j = i-2$.

$$\begin{aligned} T_{j+1} &= T_j + \lceil T_j/R_j \rceil * E_j - MPD_j[0, T_0]) \\ &= T_j + \lceil T_j/R_j \rceil * E_j - (\lfloor T_0/R_j \rfloor * E_j + min\{E_j, T_0 \\ &\quad - \lfloor T_0/R_j \rfloor * R_j\}) \end{aligned} \tag{5}$$

Proof. The task that is requested by the device U_{i-1} at time T_0 is represented by t_{i-1}. The execution time needed for this task is E_{i-1}. If $[T_0, T_x]$ is the original search interval, the execution time that is needed for the task t_{i-1} will shorten the search interval to $[T_1, T_x]$, where $T_1 = T_0 + E_{i-1}$, i.e., a PIP $\{S, [T_0, T_x]\}$ cannot begin until task t_{i-1} is completed.

To prove the second assertion, it is clear that since devices $U_1, U_2, \ldots, U_{i-2}$ have higher priorities than the device U_{i-1}, the task t_{i-1} might not be completed by time T_1. The completion of the task t_{i-1} will be delayed by all higher priority tasks that are requested by these devices while the task t_{i-1} is not completed. Specially, the completion time of the task t_{i-1} will be delayed by all tasks from the device U_1 that are either incomplete at time T_0 or requested after time T_0 and prior to time T_1. That, is all tasks requested by the device U_1 prior to time T_1 will be completed before the task t_{i-1} is completed. The total execution time needed by these tasks is $\lceil T_1/R_1 \rceil * E_1$, of which exactly $ET_1([0, T_0]) = MPD_1([0, T_0])$ is consumed by time T_0. The difference, i.e.,

$\lceil T_1/R_1 \rceil * E_1$ - $MPD_1([0, T_0])$

will be consumed after time T_0 and prior to completion of the task t_{i-1}. Therefore, the task t_{i-1} will not be completed prior to time T_2, where

$$T_2 = T_1 + \lceil T_1/R_1 \rceil * E_1 - MPD_1([0, T_0])$$

Substituting $MPD_1([0, T_0])$ from Equation 3,

$$T_2 = T_1 + \lceil T_1/R_1 \rceil * E_1 - (\lfloor T_0/R_1 \rfloor * E_1 + min\ \{E_1, T_0 - \lfloor T_0/R_1 \rfloor * R_1\}),$$

the search interval is narrowed down to $[T_2, T_x]$.

To prove the second assertion, All tasks requested by the device U_2 prior to time T_2 will be completed before the task t_{i-1} is completed. The total execution time needed by these tasks is $\lceil T_2/R_2 \rceil * E_2$, of which at the most $MPD_2([0, T_0])$ is consumed by the time T_0. Hence, the minimum amount of time that the device U_2 will delay the completion of the task t_{i-1} is

$$\lceil T_2/R_2 \rceil * E_2 - MPD_2([0, T_0]).$$

Therefore, the completion time of the task t_{i-1} must be later than time T_3, where

$$T_3 = T_2 + (\lceil T_2/R_2 \rceil * E_2 - MPD_2[0, T_0]).$$

Substituting $MPD_2([0, T_0])$ from Equation 2,

$$T_3 = T_2 + \lceil T_2/R_2 \rceil * E_2 - (\lfloor T_0/R_2 \rfloor * E_2 + min\ \{E_2, T_0 - \lfloor T_0/R_2 \rfloor * R_2\})$$

Using similar argument for devices $U_3, U_4, \ldots, U_{i-2}$, the search interval can be further shortened by the following iterative formula.

$$T_{j+1} = T_j + (\lceil T_j/R_j \rceil * E_j - MPD_j([0, T_0])$$

Or,

$$T_{j+1} = T_j + \lceil T_j/R_j \rceil * E_j - (\lfloor T_0/R_j \rfloor * E_j + min\ \{E_j, T_0 - \lfloor T_0/R_j \rfloor * R_j\}).$$

Using the assertions of Theorem 3 the procedure for locating test-points that are candidates for the LFSP(S, $[0, R_i]$), i= 1,2, ..., n, and checking condition 1 is developed as follows.

The LFSP Locating Procedure

(i) Take $T_0 = l_{i-1}(\leq R_i) = \lfloor R_i/R_{i-1} \rfloor * R_{i-1}$ and $T_1 = R_i$. The interval $[T_0, T_1]$ can be considered to be the first search interval.

(ii) The length of this inteval is further shortened by advancing the lower bound of the interval from T_0 to $T_1 = T_0 + E_{i-1}$ on the basis of Theorem 3(1).

(iii) Following Theorem 3(2) the search interval is repeatedly shortened by advancing the lower bound of the interval from T_1 through $T_2, T_3, \ldots$ to T_{i-1}. Since the upper bound of the search interval is T_i, which is a fixed value, the final search interval is $\{T_{i-1}, T_i]$. At any time during the process of computing T_j, j= 1,2, ..., i-1, if T_j turns out to be greater than T_i, the interval $[T_{i-1}, T_i]$ becomes an invalid interval and we should then consider the second search interval.

(iv) If $T_{i-1} \leq T_i$, we then check to see if Condition 1 is satisfied for any of the request points of devices U_1, U_2, ..., U_i within the interval $[T_{i-1}, T_i]$, i.e., a point in time at which a request is made by any of the devices. If the condition holds true for at least one such point, the device U_i is safe. Otherwise, we should then consider the second search interval. The request points of each device within a search interval are checked backwards, i.e., starting with the request point nearest to time T_i and continuing towards the request point nearest to time T_{i-1}.

(v) To consider the second search interval take $T_0 = l'_{i-1}(\leq R_i) = \lfloor R_i/R_{i-1} \rfloor * R_{i-1} - R_{i-1}$ and $T_i = l_{i-1}(\leq R_i)$, and steps (ii) through (iv) should be considered out one more time.

(vi) If there does not exist a request-point within

either of the two search intervals for which Condition 1 is satisfied, device U_i is unsafe.

The following algorithm, when called, produces a reduced length search interval from a given original one. The search interval that is produced is an actual search interval. The algorithm is called twice, once for $T_0 = l_{i-1}(\leq R_i) = \lfloor R_i/R_{i-1} \rfloor * R_{i-1}$ and $T_x = R_i$ and once for $T_0 = l'_{i-1}(\leq R_i) = \lfloor R_i/R_{i-1} \rfloor * R_{i-1} - R_{i-1}$ and $T_x = l_{i-1}(\leq R_i)$. In order to make this algorithm a real pascal program minor changes are necessary.

```
Procedure search-int (i: integer; R, E: vector; T0, Tx: real;
var T: real);
        {i: index of the device being checked}
        {R: array of device request intervals}
        {E: array of device execution times}
        {this procedure uses the results of Theorem 3 to shorten
the original search interval [T0, Tx] and produce the actual
search interval [T, Tx]}
begin
    T: T0 + Ei;
    J:= 1;
    while (j≤i-2 & T≤Tx) do
    being
      T:= T+⌈T/Rj⌉*Ej(⌊T0/Rj⌋*Ej min{Ej,T0-⌊T0/Rj⌋*Rj});
      j:= j+1
    end
end.
```

***Algorithm* 1.** This algorithm furhter shortens a given search interval.

The overall method for verification of the safety of a general $(1,G_n,RMP)$-system is summarized as follows.

A New Safety Determination Procedure

For a general n-device system, devices are checked one at a time, starting from the first device, i.e., the device with the shortest request interval, and continuing towards the last device. When we made sure that a device is safe it will be added to device set S which is originally a null set.

Considering a new device U_i, i= 2,3, ..., n, if the total processor utilization of devices in set S plus the device being considered is not greater than $i(2^{1/i}-1)$, the new device is also safe. It is clear that the device is unsafe if the utilization is greater than one. Otherwise, the safety of the device is checked as follows:

(a) For the first device the checking is simple and it is safe if and only if $E_1 \leq R_1$.

(b) The second device, U_2, is safe if and only if condition 4 in Theorem 1 holds true.

(c) Device U_i, i= 3,4, ..., n, is safe if and only if when checking the Condition 1 for any of the request-points which is a candidate for the LFSP $(S, [0, R_i])$, where $S= \{U_1, U_2, \ldots U_{i-1}\}$, the condition is satisfied. The candidates are obtained by using the LFSP locating procedure.

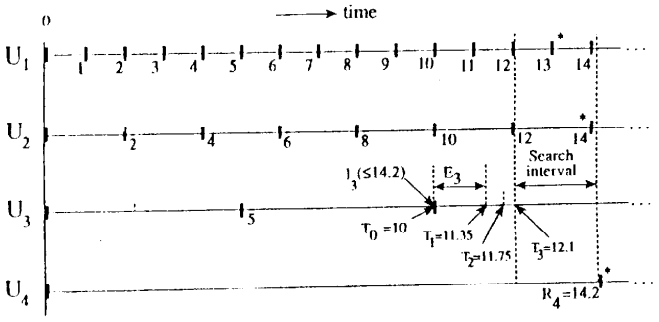
The following algorithm clarifies how the safety determination method presented in this paper could be implemented. It is clear that it is not a real Pascal program and some details are left for the implementor. The algorithm is called once for every device until either all devices are checked or an unsafe device is detected.

```
Procedure safe_i(i: integer; R, E: vector): boolean;
{this procedure, when called, checks for safety of device
Ui, i= 1,2, ..., n}
begin
    safe_i:= "false";
    if i= 1 then if E1≤R1 then safe_i:= "true"
                          else;
    else if i=2 then if ⌊R2/R1⌋*E1 + min{E1, R2-⌊R2/ R1⌋*R1}
          + E2≤R2
                    then safe_i:= "true" else;
    else if processor utilization≤i(2^(1/i)-1) then safe_i:= "true"
else begin
        k:= 1;
      while (k≤ number of request points within actual
```

```
    search intervals & safe_i= "false") do
      begin
        T:= time of k^th request point within actual search
        intervals;
        if ⌈T/R_1⌉*E_1 + ⌈T/R_2⌉*E_2 + ... + ⌈T/R_i⌉*E_i≤T
        then safe_i:= "true";
        k:= k+1
      end
    end
end.
```

***Algorithm* 2.** This algorithm, when called, checks for safety of a given device.

***Example* 2.** Consider a (1, G_4, RMP)-system with R_1= 1, E_1= 0.2, R_2= 2, E_2= 0.6, R_3= 5, E_3= 1.35, R_4= 14.2 and E_4= 2.95. For devices U_1, U_2 and U_3, no checking is requied. The processor utilization for these three devices is 0.77 which is less than $3(2^{1/3}-1)$= 0.779; therefore, S= {U_1, U_2, U_3}. Processor utilization of all four devices is 0.977 which is much higher than $4(2^{1/4}-1)$= 0.756. To check for safety of device U_4, T_0 and T_4 are set to $l_3(\leq R_4)$= 10 and R_4= 14.2, respectively (Figure 2). T_1 can be computed as $T_1 = T_0 + E_3$= 11.35. T_2 and T_3 are then computed as 11.75 and 12.1 by using Theorem 3. Within the interval [12.1, 14.2] there are three time points at which one or more requests are generated by devices U_1, U_2, U_3 and U_4. These time points are 13, 14, and 14.2. Condition 1 is satisfied for T= 14. Therefore, the last device is also safe and the system can run without any task overrun under the RM scheduling strategy. The safety determination procedure presented here checked Condition 1 for only two request-points and concluded that the system is safe, whereas previously known procedures check for 14 request points.



*time point for which Condition 1 is checked

Figure 2. A four-device system with load factor equal to 9.77 which is safe.

The safety determination procedure presented here is a substantial improvement over the procedures developed in [6,8]. It performs specially well when $r_i > 2r_{i-1}$, for some or all i, i=1,2,..., n.

CONCLUSION

In this paper real-time systems that run periodic time-critical devices under the RMP scheduling strategy were studied. The important problem considered here was to make sure that the system is safe and every request generated by the devices are executed prior to their respective deadline. The safety determination time was reduced in this paper. In earlier algorithms to make sure that the device U_i, i= 1,2,..., n, is safe, Condition 1 had to be checked for all request-points within the interval [0, R_i], until either this condition is satisfied or all request-points are exhusted. In the method presented here, to check for safety of device U_i two search intervals are produced. Only request-points within these intervals are considered. In almost all cases, total length of these search intervals is by far less than the length of interval [0, R_i]. Therefore, the number of request-points for which Condition 1 has to be checked is much smaller in comparison to earlier methods. Only in few cases, e.g., when $R_1 = R_2 = ... = R_n$, the present method performs similar to earlier methods. The improvement is even greater when $r_i > 2r_{i-1}$, for some or all devices.

REFERENCES

1. S. K. Baruah, A. K. Mok and L. E. Rosier, "Preemptive Scheduling Hard Real-time Sporadic Tasks on one Proces-

sor," Proc. Real-time Systems Symposium, (1990), pp. 182-190.

2. E. J. Jr. Coffman, ed, "Computer Job-shop Scheduling Theory," John Wiley and sons, (1976).

3. J. R. Lala, R. E. Harper and L. S. Alger, "A Design Approach for Ultrareliable Real-time Systems," *IEEE computer*, May (1991), pp. 12-22.

4. B. Sprut, L. Sha and J. Lehoczky, "Apriodic Task Scheduling for Hard Real-time Systems," *Journal of real-time systems*. Vol 1, No 1, (1989).

5. M. S. Finberg and O. Sirline, "Multiprogramming for Hybrid Computation," Proc. AFIPS Fall Joint Comp. Conf., (1967), pp. 1-13.

6. J. Lehoczky, L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," Proc. Real-time Systems Symposium, (1989), pp. 166-171.

7. J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines,", Proc. Real-time Systems Symposium, (1990), pp. 201-209.

8. C. L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-time Environments," *Journal of the ACM*, Vol. 20, No. 1, Jan (1973), pp. 46-61.

9. L. Sha and J. B. Goodenough, "Real-time Scheduling Theory and the Ada," *IEEE Computer*, April (1990), pp. 53-62.

10. J. A. Stankovic and K. Ramamrithan, "Hard Real-time Systems," Computer society press of *IEEE*, (1988).