The yielding-first rate-monotonic scheduling approach and its efficiency assessment

Mahmoud Naghibzadeh* and K H (Kane) Kim†

*Department of Computer Engineering, Ferdowsi University of Mashad, Iran. Email: naghib@ferdowsi.um.ac.ir
†Department of Electrical Engineering & Computer Science, University of California, Irvine, USA. Email: khkim@uci.edu

The rate-monotonic (RM) fixed-priority scheduling approach is one of the oldest scheduling approaches devised for use in RT computing systems. Three decades ago researchers found that if the workload presented by the set of processes is below a certain bound, the system using the RM approach can execute safely, i.e., without introducing the possibility of a deadline violation. In this paper, we identify some cases where a bound for the workload which is higher than the previously known bound can be used without compromising the execution safety. We then present an extension of the RM approach named the yielding-first rate-monotonic (YFRM) approach under which each ready process goes through a fixed period of yielding to lower-priority ready processes before exercising its priority fully. For two special cases we show that full processor utilization is possible with the YFRM approach. The YFRM and RM approaches are then compared and it is shown that there are many systems that execute safely under the YFRM approach but are prone to deadline misses under the RM approach. Also, for a set of randomly generated systems, the number of process preemptions under each approach is examined.

Keywords: Yielding-first, rate-monotonic, scheduling, real-time, deadline, fixed priority, periodic, processes, execution safety

1. INTRODUCTION

In real-time (RT) computing systems, a service function execution including an output action is typically subject to a deadline. Scheduling algorithms play a significant role in the design of such systems. They are the major factor in determining the execution safety, i.e., the possibility of always executing control service functions without missing deadlines in a given execution engine possessing a limited set of execution resources.

The rate-monotonic (RM) fixed-priority scheduling approach is one of the oldest scheduling approaches devised for use in RT computing systems [1–3]. It can be used effectively in a narrow class of RT application situations. In this paper, we attempt to add some to the knowledge-base related to the RM approach. The essence of the RM approach is:

(E1) To structure the RT application software in the form of a set of cyclic fixed-priority processes, each iterating the sequence of receiving a service request signal coming from a controlled device or timer and executing a control service function;

(E2) To make the priority number associated with each process to be either inversely proportional to the minimum request interval (MRI), i.e., the minimum interval between two successive arrivals of service request signals, or proportional to the maximum request rate which is the maximum rate of arrivals of service request signals, which is also called the maximum service rate; and

(E3) To select the highest-priority process at every selection point.

A key step in using any approach to scheduling of RT processes is to determine at design time whether a given appli-
cation software, e.g. set of processes, can be executed safely, i.e. without any deadline violation, under the scheduling approach. This execution safety analysis, also called the schedulability analysis, is complicated in many RT computing systems. In the case of using the RM approach, the following class of situations is somewhat easier to handle than other classes are:

(S4) The process execution engine has just one CPU;
(S5) Preemption of the running process by a newly ready higher-priority process is permitted [1, 2, 4];
(S6) The maximum execution time allowed (META), also called the deadline parameter, is the same as the MRI [1, 2, 5–7].
(S7) Processes are mutually independent, i.e., there is no blocking message communication or blocked access to shared data structures among processes.
(S8) The process switch overhead, i.e., the time overhead incurred in switching the processor from the execution of one process to that of another, is negligible.

The above class of situations represents a relatively small subset of the situations encountered in practice. The RT computing systems belonging to the above class of situations are called the RM-1 systems in this paper.

RM-1 systems have been dealt with in a great majority of research publications related to the RM approach because of the relatively simple and yet effective analyses that the systems yield. This paper also deals with RM-1 systems. Analysis results for RM-1 systems can sometimes serve as rough approximations of the performance expected in other situations involving the RM approach.

For other types of RT computing systems using the RM approach, the execution safety analysis becomes much more complicated [8–13]. Some researchers have considered the RM approach even for network environments (message scheduling), fault-tolerant environments, etc. [12–15].

The main results reported in this paper are two-fold. First, we identify some cases where increased processor utilization can be allowed without compromising the execution safety, i.e., without introducing the possibility of a deadline violation. This result is presented in Section 2.

Secondly, a modified version of the RM approach named the yielding-first rate-monotonic (YFRM) approach, is introduced. Under YFRM each ready process goes through a fixed period of yielding to lower-priority ready processes before exercising its priority fully. The YFRM approach uses two ready states, foreground and background, instead of the single ready state used in the RM approach. For two special cases we show that full processor utilization is possible with the YFRM approach. The two approaches, namely YFRM and RM, are then compared and it is shown that there are many systems that run safely with the YFRM approach but are unsafe with the RM approach. Also, for a set of randomly generated system cases, the number of process preemptions incurred under each approach is examined.

The YFRM approach is presented in Section 3 and comparisons of the YFRM and RM approaches are made in Section 4. Additional consideration on the implementation of the YFRM approach is discussed in Section 5 and the paper concludes in Section 6.

2. THE RATE-MONOTONIC APPROACH

This section deals with RM-1 systems.

(N1) Let \( \tau_i \) denote a process and \( v_i \) the device which generates a request signal for a service by \( \tau_i \). The two successive request signals generated by device \( v_i \) are separated in time by at least the minimum request interval (MRI), \( r_i \).

(N2) Each service execution by process \( \tau_i \) takes at most \( e_i \), the maximum execution time (MET).

(D1) The processor load factor is defined to be the sum of load factors of all processes in the system, i.e.,
\[
U = \sum_{i=1}^{n} \frac{e_i}{r_i}, \text{ where } r_i, i = 1, 2, ..., n,
\]

is the MRI seen by process \( \tau_i \) and \( e_i \), \( i = 1, 2, ..., n \), is the MET of every service task by process \( \tau_i \).

It was proved [2, 3] that an RM-1 system can always execute safely if its processor load factor \( U \) does not exceed \( n \left( 2^{1/n} - 1 \right) \). This bound is the smallest upper bound known for the processor load factor of any \( n \)-process RM-1 system that ensures execution safety.

An RM-1 system for which the processor load factor is greater than \( n \left( 2^{1/n} - 1 \right) \), may still be able to execute safely, depending upon the exact values of MRIs and METs of processes. Therefore, more elaborate and accurate procedures for determining the execution safety of RM-1 systems are desirable. Some procedures of such a kind have been studied [5, 11, 16]. It was also proved in [2, 3] that for checking the execution safety of an RM-1 system, it is sufficient to check only the most demanding test-case where for every process \( \tau_i \), \( 1 \leq i < n \), the first service request signal for arises at time 0 and subsequent service request signals arise precisely at the interval of \( r_i \).

Let us now consider a subset of RM-1 systems that have the following characteristics:

(S9) Controlled devices are strictly periodic in that each of them, \( v_i \), generates a service request signal at the fixed interval of \( r_i \). In addition, each device \( v_i \) may be activated and deactivated at any time by the controlling process \( \tau_i \) running on the CPU. Therefore, multiple devices may be activated and operated synchronously.

The special class of strictly periodic RM-1 systems that have the above characteristics (S9) are called the RM-IS systems in this section.

Consider an RM-IS system that runs \( n \) processes satisfying the relationship: for some process \( \tau_i \) and process \( \tau_{i+1}, \ r_i + 1 = k \times r_i \), where process indices are ordered in the ascending order of their request intervals (RIs) and \( k \) is a non-zero positive integer. Then we can think about the case of operating this special RM-IS system in such a way that the two devices \( v_i \) and \( v_{i+1} \) are operated synchronously by activating them simultaneously. Therefore, whenever \( v_{i+1} \) generates a service request signal, \( v_i \) also does it but not the other way around.
Now imagine that we remove process \( \tau_{i+1} \) and instead make process \( \tau_i \) spend extra execution time of \( e_i + 1/k \) for a \((1/k)\) portion of a) control service task for \( \tau_{i+1} \) following its completion of a control service task in response to a request from \( v_i \) that comes at the interval of \( r_i \). This means a modified version of the RM approach under which process \( \tau_{i+1} \) is "merged" into process \( \tau_i \) and the newly expanded process \( \tau_i \) is executed as follows.

1. Whenever a service request signal from device \( v_i \) arises, the newly expanded process \( \tau_i \) is scheduled normally according to the RM priority rule.
2. A service task for \( v_i+1 \) may be executed through up to \( k \) rounds, each taking no more than \( e_i + 1/k \) amount of time except the last round which is generally shorter.
3. The MET for \( \tau_{i+1} \) is \( e_i + e_i + 1/k \). Therefore, when \( \tau_{i+1} \) is running on the CPU, it tries to complete the service requested by \( v_i \) first and then checks if there is a service task for device \( v_i+1 \) that has not been completed. If so, \( \tau_{i+1} \) tries to execute the latter service until one of the following events occurs:
   a. A new service request signal from \( v_i \) has arisen, i.e. the amount of time equal to \( r_i \) has passed since the arrival of the last service request signal from \( v_i \);
   b. The amount of execution time spent in the service task for \( v_i+1 \) since the arrival of the last service request signal from \( v_i \) has reached the limit \( e_i + 1/k \).

The above modification of the RM approach is called the \textit{RM-ISM approach} and a system using the RM-ISM approach is called an \textit{RM-ISM system}.

In the RM-ISM system described above, process \( \tau_{i+1} \) does not exist and instead the MET for process \( \tau_{i+1} \) is greater than the MET for process \( \tau_{i+1} \) in the original RM-ISM system by \( e_i + 1/k \). Therefore, the number of processes in the RM-ISM system is \( n-1 \). This means that if we apply the upper bound formula developed in [2, 3] to the RM-ISM system, the bound becomes \((n-1)\left(2^{(1/(n-1))-1}\right)\), which is larger than \( n \left(2^{1/(n-1)}\right)\).

**Proposition 1**

Given an RM-1S system with \( n \) processes in which a certain pair of processes \( \tau_i \) and \( \tau_{i+1} \) satisfy the relationship \( r_{i+1} = k\tau_{i} \), where process indices are ordered in the ascending order of their MRIs, \( 1 \leq i < n \), and \( k \) is a non-zero positive integer, it can be converted into a functionally equivalent RM-1SM system with \( n-1 \) processes and the smallest upper bound for the processor load factor that ensures the execution safety is \((n-1)\left(2^{1/(n-1)}-1\right)\) rather than \( n \left(2^{1/(n-1)}\right)\).

This rule can be used over and over again for all pairs of consecutive processes that satisfy \( r_{i+1} = k\tau_{i} \). The proposition covers many special cases of systems, too. For example, an RM-1S system with \( n \) processes with equal MRIs can be converted into a safe RM-1SM system if and only if \( U \leq 1 \).

**Example 1**

Consider an RM-1S system of four processes handling four periodic devices, respectively, which generate service request signals with \( r_1 = 5 \), \( r_2 = 15 \), \( r_3 = 30 \), and, \( r_4 = 40 \). In this system, \( r_3 = 2r_2 \) and thus it can be converted into an equivalent RM-1SM system with three processes by merging process \( \tau_2 \) into an expansion of \( \tau_2, \tau_3 \). Also, since \( r_2 = 3r_1 \), the RM-1SM system can be converted into yet another RM-1SM system by merging the new \( \tau_2 \) into an expansion of \( \tau_1, \tau_2 \). The upper bound for the processor load factor in the original four-process RM-1S system is \( 4*\left(2^{1/4}-1\right) = 0.756 \) while for the equivalent two-process RM-1SM system, the bound is \( 2*\left(2^{1/2}-1\right) = 0.828 \). Suppose in the original RM-1S system, \( e_1 = 1.0, e_2 = 4.5, e_3 = 6.0 \), and \( e_4 = 5.12 \). The new RM-1SM system then has \( r_1 = 5, r_4 = 40, \) \( e_1 = 3.5 \), and \( e_4 = 5.12 \) with the processor load factor equal to 0.828, and thus it is safe.

**3. THE YIELDING-FIRST RATE-MONOTONIC SCHEDULING APPROACH**

Under the RM approach a ready process with a smaller MRI (or a higher maximum service rate) has unconditional priority over a process with a larger MRI (or a lower maximum service rate). This fixed priority arrangement may cause unnecessary deadline violations, especially in execution of lower-priority processes, and unnecessarily weak processor utilization.

In this section a new extension of the RM approach, called the yielding-first rate-monotonic (YFRM) approach, is presented. The essence of this approach is to let each process assigned a service task go through the "yielding" state and spend up to a fixed amount time there in order to yield the CPU to processes with larger MRIs before it enters the ready state and overshadows lower-priority processes. Figure 1 depicts how a process transmits among four different states under the YFRM approach.

(N3) Let \( U_{i,j} \), where \( 1 \leq i \leq j \leq n \), be called the load factor of a process-group \{ \( \tau_i, \ldots, \tau_j \) \} and denote:

\[
\sum_{k=i}^{j} e_k/\tau_k
\]

The \textit{yielding period} of a process \( \tau_i \), i.e. the amount of time that \( \tau_i \) spends continuously in the yielding state or the "Running while ready to yield" state, is limited by \( \Delta_i = \alpha * r_i * U_{i+1,n} \), where \( 0 \leq \alpha \leq 1 \) and \( \alpha \) called the yield coefficient, is a constant. In this section, the constant \( \alpha \) is assumed to be 1.

Therefore, for process \( \tau_1 \), the yielding period \( \Delta_1 \) is \( r_1 * U_{2,n} \) and it can be as large as \( r_1 - e_1 \) when \( U = 1 \). On the other hand, for process \( \tau_n \), the yielding period \( \Delta_n \) is zero.

A process \( \tau_i \) stays in the yielding state or the "Running while ready to yield" state for a period of \( \Delta_i \).
ready to yield” state until one of the following events occurs:

a. The yielding period $\tau_i$ has expired;
b. The service task by $\tau_i$ has been completed.

A process $\tau_i$, which has been in the yielding state moves to the “Running while ready to yield” state when all of the following conditions are met:

a. No process is in the ready state, the running state, and the “Running while ready to yield” state;
b. Of the processes in the yielding state, $\tau_i$ is the process with the largest MRI.

Consequently, a process $\tau_i$ which has been in the “Running while ready to yield” state moves to the yielding state when any of the following events occurs:

a. A process has entered the ready state;
b. A process with the MRI larger than that of $\tau_i$ has entered the yielding state.

In a sense, the ready queue (i.e. the queue of processes in the ready state) can be considered a foreground ready queue while the queue of processes in the yield state is considered a background ready queue. A modification of the RM-1 system which uses the YFRM approach instead of the RM approach is called a YFRM-1 system.

**Lemma 1**
Suppose a YFRM-1 system that runs two periodic processes is given and its processor load factor $U \leq 1$. Let $t_{2i}$, $i = 1, 2, \ldots$, denote the end of the ith request interval of process $\tau_2$, and $t_{li}$ denote the time when the last request from device $v_1$ is generated prior to time $t_{2i}$. The last request from $\tau_2$ will receive at least $(e_1 / r_1) * (t_{2i} - t_{li})$ execution time within the interval $[t_{li}, t_{2i}]$.

**Proof**
Induction on the request interval number of process $\tau_2$ is used to prove this lemma. Suppose both processes start simultaneously at time zero. In the context of this lemma the worst case occurs when no request is missed during the first request interval of process $\tau_2$, $[0, r_2]$, and all requests with deadlines up to time 0 + $r_2$ are executed in time. If there is no request from process $\tau_1$ that is generated prior to time $t_{2i}$ with deadline beyond $t_{2i}$, i.e. $t_{2i} = k * r_1$ where $k$ is a positive integer, the last request from process $\tau_1$ has received $e_1$ execution time and thus in this case, the lemma holds true.

Otherwise, since $U \leq 1$,

$$U * r_2 = ((e_1 / r_1) + (e_2 / r_2)) * r_2 = (r_2 / r_1) * e_1 + e_2$$

$$= (t_2 / t_1) * e_1 + (e_1 / r_1) * length([t_1, t_2]) + e_2$$

$$= (t_2 / t_1) * e_1 + (e_1 / r_1) * (t_2 - t_1) + e_2$$

$$\leq t_2$$

Therefore, even if the service task by $\tau_2$ is fully executed by using $e_2$ amount of CPU time and the CPU is fully busy during the first request interval of $\tau_1$, $[0, r_1]$, the CPU will have $(e_1 / r_1) * (t_2 - t_1)$ time to spend for $\tau_1$. This means that for the first request interval of process $\tau_2$, the lemma is true.

We now assume that the statement is true for the $i^{th}$ request interval of $\tau_2$. We must prove that the statement is also true for the $(i + 1)^{th}$ request interval. If a request from $v_1$ coincides with the $(i + 1)^{th}$ request of device $v_2$, the situation during the $(i + 1)^{th}$ request interval is the same as that which existed during the first request interval of $\tau_2$. Hence the statement is true. Otherwise, there exists a request from $v_1$ that is generated prior to time $t_{2i}$ with the deadline falling after time $t_{2i}$, i.e. $t_{2i} + r_1 > t_{2i}$. Therefore, this request from $v_1$ is during the $i^{th}$ request interval of $\tau_2$. From the hypothesis associated with the $i^{th}$ request interval, the service task requested by $\tau_1$ must have received at least its proportionate share of CPU prior to time $t_{2i}$. This means that during the interval from $t_{2i}$ to the deadline for the service task requested by $v_1$, i.e. $[t_{2i}, t_{2i} + r_1]$, the service task requested by $\tau_1$ receives $e_1s$ amount of CPU time, which is no more than its proportionate share of CPU. During the interval $[t_{2i}, t_{2i} + r_1]$, the amount of CPU time that process $\tau_i$ receives, does not exceed $(length([t_{2i}, t_{2i} + r_1])) / r_1) * e_1$. So, again from the expression for $U * r_2$, one can deduce that even if the service task requested by $v_2$ at $t_{2i}$ is fully executed during the $(i + 1)^{th}$ request interval by using $e_2$ amount of CPU time, there still remains at least $(e_1 / r_1) * (t_{2i} - t_{2i+1})$ amount of CPU time available to process $\tau_2$ during $[t_{2i}, t_{2i+1}]$. This means that the lemma holds true for the $(i + 1)^{th}$ request interval of $\tau_2$ and the lemma has been proved.

**Theorem 1**
An YFRM-1 system running two periodic processes is safe if and only if $U \leq 1$.

**Proof**
The necessity of $U \leq 1$ is obvious.

We need to prove that if $U \leq 1$, the system is safe. A service task requested by device $v_1$ cannot miss the deadline because process $\tau_1$ will stay in the yielding state for at most $r_1 - e_1$ amount of time.

Suppose the $j^{th}$ request from device $v_2$ is generated at time $t_{2j-1}$ and the requested service task missed the deadline at time $t_{2j}$, $j \in \{1, 2, \ldots \}$. Let $t_{2j-1}$ be the time at which the last request from device $v_1$ prior to time $t_{2j-1}$ arises and $t_{2j}$ be the time at which the last request from device $v_1$ prior to time $t_{2j}$ arises. See Figure 2. The execution of the service task requested by device $v_1$ at time $t_{2j}$ must start before time $t_{2j}$ since otherwise, it would mean a violation of Lemma 1.

On the other hand, if we assume that the execution of the aforementioned service task started before time $t_{2j}$ but after the yielding period of $\tau_1$ was over, there could not have been enough CPU time left for $\tau_1$ to satisfy Lemma 1 as shown below:

$$[\{(j * r_2)/r_1\} * e_1 + length([t_{2j-1}, t_{2j}]) - (r_1 - e_1) + j * e_2 > j * r_2]$$

$$=> [\{(j * r_2)/r_1\} + 1] * e_1 + j * e_2 > \{(j * r_2)/r_1\} + 1] * r_2$$
\( > (e_1/r_1) + j \cdot e_2/((j \cdot r_2/r_1) + 1) \cdot r_1 > 1 \)

Since \( (j \cdot r_2/r_1) + 1 \cdot r_1 > j \cdot r_2 \)

\( (e_1/r_1) + j \cdot e_2 / (j \cdot r_2) > (e_1/r_1) + j \cdot e_2 / ((j \cdot r_2/r_1) + 1) \cdot r_1 > 1 \)

\( \Rightarrow U > 1 \).

This is a contradiction. Therefore, the execution of the service task requested by device \( v_j \) at time \( t_1 \) must start before time \( t_1 \) and before the yielding period of \( \tau_i \) is finished. This is possible only if the request generated by \( v_j \) at \( t_{j+1} \) is completed prior to its deadline at time \( t_j \).

Let us now consider a subset of YFRM-1 systems which deal with strictly periodic devices possessing the characteristics specified in (S9) earlier. Such systems are called the YFRM-1S systems in this section.

**Theorem 3**

An YFRM-1S system running \( n \) periodic processes for which \( r_{i+1} = k_i \cdot r_i \), where \( k_i \) is a positive integer, \( i = 1, 2, ..., n - 1 \), is safe if and only if \( U \leq 1 \).

**Proof**

For a safe system we must have \( U \leq 1 \).

Within each period of process \( \tau_i \), \( i = 1, 2, ..., n \), exactly \( k_{i+1} \cdot k_{i+2} \cdot ... \cdot k_i \) requests from device \( v_j \), \( 1 \leq j < i \) are generated and their deadlines are also within the period. Moreover, if we activated the devices simultaneously, there cannot be any request that is generated within the period with the deadline falling after the period. Similarly, there cannot be any request that is generated before the start of this period and has a deadline falling within the period. The total execution time needed by all these requested service tasks including the service task provided by process \( \tau_i \) is equal to:

\[ \sum_{m=1}^{i} (r_i/r_m) \cdot e_m \]

The yielding period of process \( \tau_i \) that starts at the beginning of the mentioned period is:

\[ r_i \cdot \sum_{m=i+1}^{n} (e_m/r_m) \]

The worst case that we can imagine in terms of maximizing the chance for \( \tau_i \) to miss the deadline for its service task is where the CPU will not execute any request from processes \( \tau_1, \tau_2, ..., \tau_i \) during the yielding period of \( \tau_i \). If the sum of the above two terms does not exceed \( r_i \), \( \tau_i \) cannot miss its deadline. The condition,

\[ \sum_{m=1}^{i} (r_i/r_m) \cdot e_m + r_i \cdot \sum_{m=i+1}^{n} (e_m/r_m) \leq r_i \]

can be shown to be equivalent to \( U \leq 1 \) by dividing both sides by \( r_i \) since it will result in:

\[ \sum_{m=1}^{i} (r_i/r_m) \cdot e_m \leq 1 \]

Hence, the theorem is proved.

4. **COMPARISON OF RM AND YFRM APPROACHES**

To compare the RM and the YFRM approaches, many process sets were randomly generated. To generate a set of cardinality \( n \), request intervals were drawn randomly from the interval \([0, n]\) and execution times were then chosen such that the total processor load factor is exactly one and the load factors of all processes are equal, i.e.

\[ e_j/r_1 = e_2/r_2 = ... = e_n/r_n \]

Minor changes had to be done on request intervals so that \( e_j/r_i, i = 1, 2, ..., n \), may become a terminating real number. In Table 1 properties of some of these process sets are represented. Simulation programs for both approaches were implemented. Overhead time was considered zero for both scheduling approaches.

Table 2 shows the number of deadline misses for 1000 service tasks executed (either to successful completions or to deadline misses) by every process set in Table 1, that occurred under the RM approach as well as under the YFRM approach. It also shows the exact time at which the request signal for the first service task that missed the deadline had occurred and the process which was handling the task. RM-1S systems and YFRM-1S systems were dealt with in this simulation study. It has been assumed that all processes start simultaneously at time zero.

There is quite a difference in the number of deadline misses between RM and YFRM approaches. In Figure 3, this difference is clearly noticeable. It also shows that in most cases the first deadline miss is from the process with the largest request interval \( \tau_i \), and this was the same under both RM and YFRM.

It was our intuition that YFRM might cause more process preemptions than RM does. To check this, the number of process preemptions per 1000 service tasks executed (either to successful completions or to deadline misses) by the process sets in Table 1, was counted and is summarized in Table 3. It shows that contrary to our anticipation, the number of process preemptions under YFRM is less than that under RM in all cases, except one. However, the differences did not appear significant.

An efficient and broadly applicable procedure for verification of the execution safety of any given YFRM-1 system is yet to be developed. For most practical YFRM-1S systems for which request intervals of processes can be expressed in terms of integer numbers, a reasonably efficient safety verification procedure can be devised. Given such a system, let \( M \) denote LCM \( (r_1, r_2, ..., r_n) \), where LCM stands for the least common multiple of the given integer values of request intervals. The system can be simulated for a period of \( M \) and this was the same under both RM and YFRM.

If request intervals are non-integers in terms of a given time unit, changing the time unit to make them take integer representations is a possible approach. For example, a request interval may be 5.75 seconds. This request interval can be represented as 5750 milliseconds, an integer form but
then the simulation time may be lengthened significantly. It is also reasonable in some circumstances to apply approximation on request intervals for the purpose of checking the execution safety, e.g., adopting the smallest integer above the given real number.

5. EFFECT OF THE YIELD COEFFICIENT $\alpha$

In this section the effect of the yield coefficient is considered. In Section 3 the yielding period for process $\tau_i$, $i = 1, 2, ..., n$, was taken to be

$$\Delta_i = \alpha * r_i * U_{i+1..n}$$

but $\alpha$ was assumed to be equal to one. Here, different values of $\alpha$ (0.25, 0.5, 0.75, 1.0, 1.25, 1.5, and 1.75) are considered and their impacts on the number of deadline misses are examined. Table 5 shows the number of deadline misses per 1000 service tasks executed (either to successful completions or to deadline misses) by the process sets in Table 1 under each different yield coefficient.

The table shows that, for all process sets, the number of deadline misses is the smallest when $\alpha$ is equal to one. The number of deadline misses clearly decreases as $\alpha$ increases from 0.25 to 1.0, but it suddenly explodes as $\alpha$ increases further from 1.0 to 1.25.

6. SUMMARY AND FUTURE WORK

A special class of systems running under the RM approach was recognized and for this class an improvement over the known upper bound to the processor load factor that ensures execution safety was shown. We then presented a new flexible way to schedule fixed-priority processes, the YFRM approach. Its performance was compared to that of the RM approach and shown to be superior. There are many process sets that run safe with YFRM and are unsafe with RM. Development of an efficient and broadly applicable procedure for verification of the execution safety of any given YFRM-1 system is an important open research topic in this area.
Table 3 Number of process preemptions under RM and YFRM approaches

<table>
<thead>
<tr>
<th>Process set</th>
<th>No. of Preemptions under RM</th>
<th>No. of Preemptions under YRM</th>
<th>Difference (RM – YFRM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>722</td>
<td>564</td>
<td>158</td>
</tr>
<tr>
<td>2</td>
<td>776</td>
<td>713</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>676</td>
<td>672</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>725</td>
<td>723</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>795</td>
<td>759</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>852</td>
<td>844</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>638</td>
<td>634</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>683</td>
<td>708</td>
<td>-25</td>
</tr>
<tr>
<td>9</td>
<td>707</td>
<td>682</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 4 Sample systems that are safe with YFRM and unsafe with RM

<table>
<thead>
<tr>
<th>No. of processes</th>
<th>Request intervals</th>
<th>Execution times</th>
<th>Process or load</th>
<th>Time of 1st miss under RM</th>
<th>Process missing a deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3.0, 5.0,</td>
<td>0.4, 1.4,</td>
<td>0.98</td>
<td>9.0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4.6, 7.2,</td>
<td>0.92, 1.8,</td>
<td>0.97</td>
<td>30.0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>3.0, 8.0,</td>
<td>0.3, 2.4,</td>
<td>0.975</td>
<td>32.0</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>3.0, 5.0,</td>
<td>0.45, 0.65,</td>
<td>0.985</td>
<td>20.0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>3.0, 5.0,</td>
<td>0.32, 0.77,</td>
<td>1.22, 1.26,</td>
<td>24.0</td>
<td>4.0, 7.0,</td>
</tr>
<tr>
<td>8</td>
<td>4.0, 7.0,</td>
<td>0.32, 0.77,</td>
<td>1.22, 1.26,</td>
<td>24.0</td>
<td>4, 0, 7.0,</td>
</tr>
<tr>
<td>9</td>
<td>4.0, 7.0,</td>
<td>0.32, 0.77,</td>
<td>1.22, 1.26,</td>
<td>24.0</td>
<td>4.0, 7.0,</td>
</tr>
</tbody>
</table>

Table 5 Number of deadline misses under YFRM and different yield coefficients

<table>
<thead>
<tr>
<th>Process</th>
<th>$\alpha = 0.25$</th>
<th>$\alpha = 0.5$</th>
<th>$\alpha = 0.75$</th>
<th>$\alpha = 1.0$</th>
<th>$\alpha = 1.25$</th>
<th>$\alpha = 1.5$</th>
<th>$\alpha = 1.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73</td>
<td>55</td>
<td>18</td>
<td>0</td>
<td>218</td>
<td>218</td>
<td>218</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>16</td>
<td>12</td>
<td>3</td>
<td>172</td>
<td>172</td>
<td>190</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
<td>27</td>
<td>12</td>
<td>2</td>
<td>101</td>
<td>106</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
<td>17</td>
<td>6</td>
<td>4</td>
<td>99</td>
<td>97</td>
<td>84</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>178</td>
<td>157</td>
<td>136</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>232</td>
<td>219</td>
<td>194</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>11</td>
<td>6</td>
<td>1</td>
<td>34</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>79</td>
<td>70</td>
<td>68</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>13</td>
<td>6</td>
<td>3</td>
<td>38</td>
<td>28</td>
<td>26</td>
</tr>
</tbody>
</table>

In research on extensions of the RM approach, the fact that a different approach, the earliest-deadline-first approach [2], allows up to 100% utilization of the processor time in safe execution of a set of processes satisfying the conditions, (E2) and (S4)–(S9) stated in Section 1, should not be forgotten. The system designer may be interested in using an RM extension over EDF only in a special class of systems, e.g., systems in which there is strong desire to avoid preemption of the top-priority process or a top few processes beyond a certain threshold. Therefore, only a simple extension of the RM approach will be worth exploring.

Also, some conditions such (S6) and (S8) stated in Section 1 are not met in most practical application situations. Therefore, both RM and YFRM are just starting points in searching for scheduling approaches that can be used with high degree of effectiveness in many practical situations.

ACKNOWLEDGMENT

The research reported here was supported in part by Mashad University, Iran, and in part by the NSF under Grant Numbers 02-04050 (NGS) and 00-86147 (ITR).

REFERENCES
