IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%

Björn Andersson

HURRAY-TR-081002

Version: 0

Date: 10-01-2008

Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: bandersson@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling real-time tasks on a multiprocessor with the goal of meeting deadlines. Tasks arrive sporadically and have implicit deadlines, that is, the deadline of a task is equal to its minimum inter-arrival time. Consider this problem to be solved with global static-priority scheduling. We present a priority-assignment scheme with the property that if at most 38% of the processing capacity is requested then all deadlines are met.

Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%

Björn Andersson

IPP Hurray Research Group,
Polytechnic Institute of Porto, Portugal

Abstract. Consider the problem of scheduling real-time tasks on a multiprocessor with the goal of meeting deadlines. Tasks arrive sporadically and have implicit deadlines, that is, the deadline of a task is equal to its minimum inter-arrival time. Consider this problem to be solved with global static-priority scheduling. We present a priority-assignment scheme with the property that if at most 38% of the processing capacity is requested then all deadlines are met.

1 Introduction

Consider the problem of preemptively scheduling n sporadically arriving tasks on $m \geq 2$ identical processors. A task τ_i is uniquely indexed in the range $1..n$ and a processor likewise in the range $1..m$. A task τ_i generates a (potentially infinite) sequence of jobs. The arrival times of these jobs cannot be controlled by the scheduling algorithm and are a priori unknown. We assume that the arrival time between two successive jobs by the same task τ_i is at least T_i. Every job by τ_i requires at most C_i time units of execution over the next T_i time units after its arrival. We assume that T_i and C_i are real numbers and $0 \leq C_i \leq T_i$. A processor executes at most one job at a time and a job is not permitted to execute on multiple processors simultaneously. The utilization is defined as $U_s = (1/m) \cdot \sum_{i=1}^{n} \frac{C_i}{T_i}$. The utilization bound UB_A of an algorithm A is the maximum number such that all tasks meet their deadlines when scheduled by A, if $U_s \leq UB_A$.

Static-priority scheduling is a specific class of algorithms where each task is assigned a priority, a number which remains unchanged during the operation of the system. At every moment, the highest-priority task is selected for execution among tasks that are ready to execute and has remaining execution. Static-priority scheduling is simple to implement in operating systems and it can be implemented efficiently. Therefore, it is implemented in virtually all real-time operating systems and many desktop operating systems support it as well, accessible through system calls specified according to the POSIX-standard [1]. Because of these reasons, a comprehensive toolbox (see [2, 3]) of results (priority-assignment schemes, schedulability analysis algorithms, etc) has been developed for static-priority scheduling on a single processor. The success story of static-priority scheduling on a single processor started with the development of the rate-monotonic (RM) priority-assignment scheme [4]. It assigns task τ_j a higher priority than task τ_i if $T_j < T_i$. RM is an optimal priority-assignment scheme, meaning that for every task set, it holds that if there is an assignment of priorities that causes deadlines to be met then deadlines are met as well when RM is used. It is also known [4] that $UB_{RM} = 0.69$ for the case that $m = 1$. This result is important because it gives designers an intuitive idea of how much a processor can be utilized without missing a deadline.

Multiprocessor scheduling algorithms are often categorized as partitioned or global. Global scheduling stores tasks which have arrived but not finished execution in one queue, shared by all processors. At any moment, the m highest-priority tasks among those are selected for execution on the m processors. In contrast, partitioned scheduling algorithms partition the task set such that all tasks in a partition are assigned to the same processor. Tasks may not migrate from one processor to another. The multiprocessor scheduling problem is thus transformed to many uniprocessor scheduling problems.

Real-time scheduling on a multiprocessor is much less developed than real-time scheduling on a single processor. And this applies to static-priority scheduling as well. In particular, it is known that it is impossible to design a partitioned algorithm with $UB > 0.5$ [5]. It is also known that for global static-priority scheduling, RM is not optimal. In fact, global RM can miss a deadline although U_s approaches zero [6]. For a long time, the research community dismissed global static-priority scheduling for this reason. But later, it was realized that other priority-assignment schemes (not necessarily RM) can be used for global static-priority scheduling and the research community developed such schemes. Many priority-assignment schemes and analysis techniques for global static-priority scheduling are available (see for example [7–10]) but so far, only two priority-assignment schemes, RM-US($m/(3m-2)$) [11] and RM-US(x) [12] have known (and non-zero) utilization bounds. These two algorithms categorize a task as heavy or light. A task is said to be heavy if $\frac{C_i}{T_i}$ exceeds a certain threshold number and a task is said to be light otherwise. Heavy tasks are assigned the highest priority and the light tasks are assigned a lower priority; the relative priority order among light tasks is given by RM. It was shown that among the algorithms that separate heavy and light tasks and use RM for light tasks, no algorithm can achieve a utilization bound greater than 0.374 [12]. And in fact, the current state-of-art offers no algorithm with utilization bound greater than 0.374.

In this paper, we present a new priority-assignment scheme SM-US($2/(3+\sqrt{5})$). It categorizes tasks as heavy and light and assigns the highest priority to heavy tasks. The relative priority order of light tasks is given by slack-monotonic (SM) though, meaning that task τ_j is assigned higher priority than task τ_i if T_j - C_j < T_i - C_i. We prove that the utilization bound of SM-US($2/(3+\sqrt{5})$) is $2/(3+\sqrt{5})$, which is approximately 0.382.

We consider this result to be significant because (i) the new algorithm SM-US($2/(3+\sqrt{5})$) breaks free from the performance limitations of the RM-US framework, (ii) the utilization bound of SM-US($2/(3+\sqrt{5})$) is higher than the utilization bound of the previously-known best algorithm in global static-priority scheduling and (iii) the utilization bound of SM-US($2/(3+\sqrt{5})$) is reasonably close to the limit $\sqrt{2}-1 \approx 0.41$ which is known (from Theorem 8 in [13]) to be an upper bound on the utilization bound of every global static-priority scheduling algorithm which assigns a priority to a task τ_i as a function only of T_i and C_i.

Section 2 gives a background on the subject, presenting the main ideas behind algorithms that achieve a utilization bound greater than zero. It also presents results that we will use, in particular (i) lemmas expressing inequalities, (ii) a lemma from previous research on the amount of execution performed and (iii) a new schedulability test. Section 3 presents the new algorithm SM-US($2/(3+\sqrt{5})$) and proves its utilization bound using the schedulability test in Section 2. Conclusions are given in Section 4.

2 Background

2.1 Understanding global static-priority scheduling

The inventor of RM observed [14] that

> Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.

Example 1 gives a good illustration of this.

Example 1. [From [6]]. Consider a task set with n=m+1 tasks to be scheduled on m processors. The tasks are characterized as $\forall i \in \{1,2,\ldots,m\} : T_i = 1, C_i = 2\epsilon$ and $T_{m+1} = 1+\epsilon, C_{m+1} = 1$. If we assign priorities according to RM then τ_{m+1} is given the lowest priority and when all tasks arrive simultaneously then τ_{m+1} misses a deadline. Letting $\epsilon \to 0$ and $m \to \infty$ gives us a task set with $U_s \to 0$ and it misses a deadline.

Based on Example 1, one can see that better performance can be achieved by giving high priority to tasks with high $\frac{C_i}{T_i}$. And in fact this is what the algorithms, RM-US($m/(3m-2)$) [11] and RM-US(x) [12] do. The algorithm RM-US(x) [12] computes the value of x and its utilization bound is x. The value of x depends on the number of processors; it is given as (1-y)/($m \cdot$ (1+y))+ln(1+y)=(1-y)/(1+y)=x. Solving it for $m \to \infty$ gives us that y=0.454 and x=0.375. One can see that $m \to \infty$ gives us the least value of x. Hence the utilization bound of RM-US(0.375) is 0.375. And there is no other choice of x which gives a higher utilization bound. Example 2 illustrates this.

Example 2. [Partially taken from [12]]. Figure 1 illustrates the example. Consider $n = m \cdot q + 1$ tasks to be scheduled on m processors, where q is a positive integer. The task τ_n is characterized by $T_n = 1+y$ and $C_n = 1-y$. The tasks with index

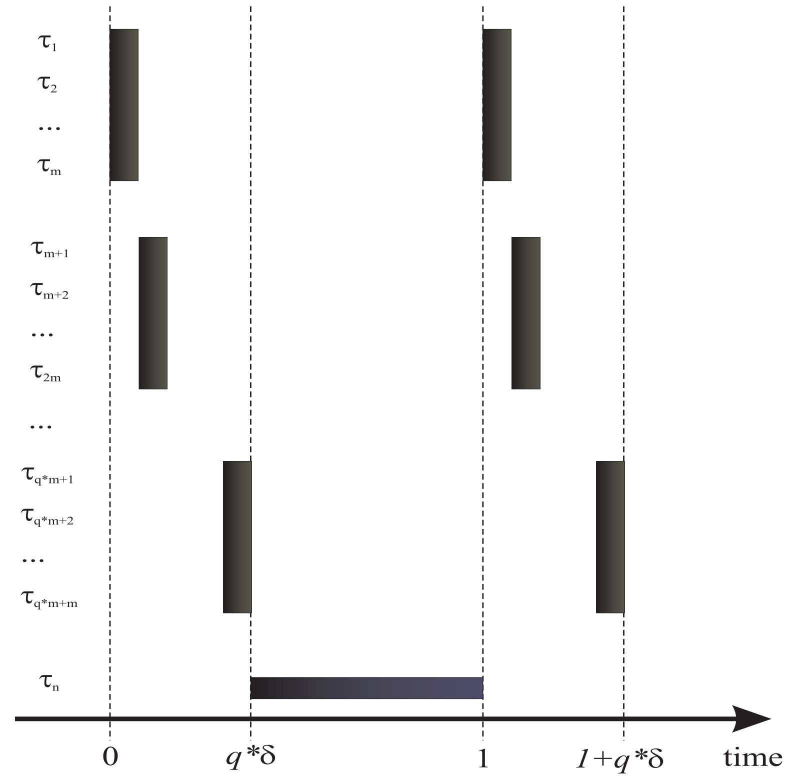

Fig. 1. An example of a task set where RM-US(0.375) performs poorly. All tasks arrive at time 0. Tasks τ_1, τ_2,..., τ_m are assigned the highest priority and execute on the m processors during $[0,\delta)$. Then the tasks τ_{m+1}, τ_{m+2},..., τ_{2m} execute on the m processors during $[\delta,2\delta)$. The other groups of tasks execute in analogous manner. Task τ_n executes then until time 1. Then the groups of tasks arrive again. The task set meets its deadlines but an arbitrarily small increase in execution times causes a deadline miss.

$i \in \{1, 2, \ldots, n-1\}$ are organized into groups, where each group comprises m tasks. One group is the tasks with index $i \in \{1, 2, \ldots, m\}$. Another group is the tasks with index $i \in \{m+1, m+2, \ldots, 2 \cdots m\}$ and so on. The r:th group comprises the tasks with index $i \in \{r \cdot m + 1, r \cdot m + 2, \ldots, r \cdot m + m\}$. All tasks belonging to the same group have the same T_i and C_i. Clearly there are q groups. The tasks in the r:th group have the parameters $T_i = 1 + r \cdot \delta$ and $C_i = \delta$, where δ is selected as $y = q \cdot \delta$. Hence, specifying m and y gives us the task set. By letting $y = 0.454$ and $m \to \infty$ we have a task set that where all tasks are light. The resulting task set is depicted in Figure 1. Also, all tasks meet their deadlines but an arbitrarily small increase in execution time of τ_n causes it to miss a deadline. That is, RM-US(0.375) misses a deadline at a utilization just slightly higher than 0.375.

One can see that if the light tasks in Example 2 would have been assigned priorities such that $T_j - C_j < T_i - C_i$ implies that τ_j has higher priority than τ_i then deadlines would have been met. In fact, we will use this idea when we design the new algorithm in Section 3.

2.2 Results we will use

Lemma 1-4 state four simple inequalities that we will find useful; their proofs are available in the Appendix.

Lemma 1. *Let m denote a positive integer. Consider u_i to be a real number such that $0 \le u_i < \frac{2}{3+\sqrt{5}}$ and consider S to denote a set of non-negative real numbers u_j such that*

$$(\sum_{j \in S} u_j) + u_i \le \frac{2}{3+\sqrt{5}} \cdot m \tag{1}$$

then it follows that

$$\frac{1}{m} \cdot (\sum_{j \in S} (2 - u_i) \cdot u_j) + u_i \le 1 \tag{2}$$

Lemma 2. *Consider two non-negative real numbers u_j and u_i such that $0 \le u_j < 1$ and $0 \le u_i < 1$. For those numbers, it holds that:*

$$u_j \cdot \frac{1-u_i}{1-u_j} + (1 - u_j \cdot \frac{1-u_i}{1-u_j}) \cdot u_j \le (2 - u_i) \cdot u_j \tag{3}$$

Lemma 3. *Consider two non-negative real numbers u_j and u_i such that $0 \le u_j < 1$ and $0 \le u_i < 1$. And two non-negative real numbers T_j and T_i such that*

$$T_j \cdot (1 - u_j) \le T_i \cdot (1 - u_i) \tag{4}$$

For those numbers, it holds that:

$$u_j \cdot \frac{T_j}{T_i} + (1 - u_j \cdot \frac{T_j}{T_i}) \cdot u_j \le u_j \cdot \frac{1-u_i}{1-u_j} + (1 - u_j \cdot \frac{1-u_i}{1-u_j}) \tag{5}$$

Lemma 4. *Consider two integers T_j and C_j such that $0 \le C_j \le T_j$. For every $t > 0$ it holds that:*

$$\lfloor \frac{t}{T_j} \rfloor \cdot C_j + \min(t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j, C_j) \le C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{6}$$

Predictable scheduling. Ha and Liu [15] have studied real-time scheduling of jobs on a multiprocessor; a job is characterized by its arrival time, its deadline, its minimum execution time and its maximum execution time. The execution time of a job is unknown but it is no less than its minimum execution time and no greater than its maximum execution time. A scheduling algorithm A is said to be predictable if for every set J of jobs it holds that:

Scheduling all jobs by A with execution times equal to their maximum execution times causes deadlines to be met. $\Rightarrow$ Scheduling all jobs by A with execution times being at least their minimum execution times and at most their maximum execution times causes deadlines to be met.

Intuitively, the notion of predictability means that we only need to analyze the case when all jobs execute according to their maximum execution time. Ha and Liu also found that global static priority scheduling of jobs on a multiprocessor is predictable. Our paper deals with tasks that generate jobs with a certain constraint (given by the minimum inter-arrival time, T_i). But since our model is a special case of the model used by Ha and Liu, it also follows that global static-priority scheduling with our model is predictable as well.

The notion of active. We let $active(t, \tau_i)$ be true if at time t, there is a job of τ_i which has arrived no later than t and has a deadline no earlier than t; otherwise $active(t, \tau_i)$ is false. Observe that a task τ_i may release a job and at time t this job has no remaining execution but its deadline is greater than t. Because of our notion active, this task τ_i is active at time t. Note that with our notion of active, a periodically arriving task is active all the time after its first arrival. Because we study sporadically arriving tasks, there may be moments when a task is not active though. The notion of gap measures that.

The notion of gap. We let gap($[t_0,t_1)$, τ_i) denote the amount of time during $[t_0,t_1)$ where $active(t, \tau_i)$ is false.

Optimal algorithm. Consider a task τ_i and a time interval of duration ϵ such that the task τ_i is active during the entire time interval. Let OPT denote an algorithm which executes task τ_i for $(C_i/T_i) \cdot \epsilon$ time unit during the time interval of duration ϵ, where ϵ is arbitrarily small.

Work-conserving. We say that a scheduling algorithm is work-conserving if it holds for every t that: if there are at least k tasks with unfinished execution at time t then at least k processors are busy at time t. In particular, we note that global static-priority scheduling is work-conversing.

Execution. Let t_0 denote a time such that no tasks have arrived before t_0. Let W(A, τ, $[t_0,t_1)$) denote the amount of execution performed by tasks in τ during $[t_0,t_1)$ when scheduled by algorithm A. Philips et al. [16] studied the amount of execution performed by a work-conserving algorithm. They found that the amount of execution in a time interval performed by work-conserving algorithm is at least as much as the amount of execution performed by any other algorithm assuming that the work-conserving algorithm is given processors that are $(2m-1)/m$ times faster. Previous research [11] in real-time computing has used this result by comparing the amount of execution performed by global static-priority scheduling against the algorithm OPT but that work considered only the model of periodically arriving tasks. That result can be extended in a straightforward manner to the model we use in this paper (the sporadic model) though, as expressed by Lemma 5.

Lemma 5. *Let G denote an algorithm with global static-priority scheduling. If*

$$\forall j : \frac{C_j}{T_j} \leq \frac{m}{2m-1} \tag{7}$$

and

$$\sum_{\tau_j \in \tau} \frac{C_j}{T_j} \leq \frac{m}{2m-1} \cdot m \tag{8}$$

then

$$W(G, \tau, [t_0, t_1)) \geq \sum_{\tau_j \in \tau} (t_1 - t_0 - gap([t_0, t_1], \tau_j)) \cdot \frac{C_j}{T_j} \tag{9}$$

Proof. From Equation 7 and Equation 8 it follows that the task set τ can be scheduled to meet deadlines by OPT on a multiprocessor with m processors of speed $m/(2m-1)$. The amount of execution during $[t_0,t_1)$ is then given by the right-hand side of Equation 9. And the result by Philips et al gives us that also algorithm G performs as much execution during $[t_0,t_1)$. Hence Equation 9 is true and it gives us that the lemma is true.

Schedulability analysis. Let t_0 denote a time such that no tasks arrive before t_0. Let us consider a time interval that begins at time t_0; let $[t_0,\ t_2)$ denote this time interval. We obtain that the amount of execution performed by the task set τ during $[t_0,\ t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(\lfloor \frac{t_2 - t_0 - gap([t_0, t_2), \tau_j)}{T_j} \rfloor \cdot C_j + \\ \min(t_2 - t_0 - gap([t_0, t_2), \tau_j) - \lfloor \frac{t_2 - t_0 - gap([t_0, t_2), \tau_j)}{T_j} \rfloor \cdot T_j, C_j) \Big) \tag{10}$$

From Lemma 5 we obtain that the amount of execution performed by the task set τ during $[t_0,\ t_1)$ is at least:

$$\sum_{\tau_j \in hp(i)} (t_1 - t_0 - gap([t_0, t_1], \tau_j)) \cdot \frac{C_j}{T_j} \tag{11}$$

Let us consider the case that a deadline was missed. Let us consider the earliest time when a deadline was missed. Let t_1 denote the arrival time of the job that missed this deadline and let τ_i denote the task that generated this job. Let $hp(i)$ denote the set of tasks with higher priority than τ_i. Let t_2 denote the deadline that was missed; that is, t_2=t_1+T_i. Applying Equation 8 and Equation 9 on $hp(i)$ gives us that the amount of execution by $hp(i)$ during $[t_1,t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(\lfloor \frac{t_2 - t_0 - gap([t_0, t_2), \tau_j)}{T_j} \rfloor \cdot C_j +$$

$$\min(t_2 - t_0 - gap([t_0,t_2),\tau_j) - \lfloor\frac{t_2 - t_0 - gap([t_0,t_2),\tau_j)}{T_j}\rfloor \cdot T_j, C_j)\Big) - \sum_{\tau_j \in hp(i)} (t_1 - t_0 - gap([t_0,t_1],\tau_j)) \cdot \frac{C_j}{T_j} \quad (12)$$

Using $t_2 = t_1 + T_i$ and rewriting gives us that the amount of execution by $hp(i)$ during $[t_1,t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(\lfloor\frac{T_i + t_1 - t_0 - gap([t_0,t_1),\tau_j) - gap([t_1,t_2),\tau_j)}{T_j}\rfloor \cdot C_j + \min(T_i + t_1 - t_0 - gap([t_0,t_1),\tau_j) - gap([t_1,t_2),\tau_j) - \lfloor\frac{T_i + t_1 - t_0 - gap([t_0,t_1),\tau_j) - gap([t_1,t_2),\tau_j)}{T_j}\rfloor \cdot T_j, C_j)\Big) - \sum_{\tau_j \in hp(i)} (t_1 - t_0 - gap([t_0,t_1],\tau_j)) \cdot \frac{C_j}{T_j} \quad (13)$$

Applying Lemma 4 on Equation 13 gives us that the amount of execution by $hp(i)$ during $[t_1,t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(C_j + (T_i + t_1 - t_0 - gap([t_0,t_1),\tau_j) - gap([t_1,t_2),\tau_j) - C_j) \cdot \frac{C_j}{T_j}\Big) - \sum_{\tau_j \in hp(i)} (t_1 - t_0 - gap([t_0,t_1],\tau_j)) \cdot \frac{C_j}{T_j} \quad (14)$$

Simplifying Equation 14 gives us that the amount of execution by $hp(i)$ during $[t_1,t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(C_j + (T_i - gap([t_1,t_2),\tau_j) - C_j) \cdot \frac{C_j}{T_j}\Big) \quad (15)$$

Relaxing gives that the amount of execution by tasks in $hp(i)$ during $[t_1,t_2)$ is at most:

$$\sum_{\tau_j \in hp(i)} \Big(C_j + (T_i - C_j) \cdot \frac{C_j}{T_j}\Big) \quad (16)$$

From Equation 16 it follows that the amount of time during during $[t_1,t_2)$ where all processors are busy executing tasks in $hp(i)$ is at most:

$$\frac{1}{m} \cdot \sum_{\tau_j \in hp(i)} \Big(C_j + (T_i - C_j) \cdot \frac{C_j}{T_j} \Big) \tag{17}$$

Lemma 6. *Consider global static-priority scheduling. Consider a task τ_i. If all tasks in $hp(i)$ meet their deadlines and*

$$\forall j \in hp(i) : \frac{C_j}{T_j} \leq \frac{m}{2m-1} \tag{18}$$

and

$$\frac{C_i}{T_i} \leq \frac{m}{2m-1} \cdot m \tag{19}$$

and

$$\Big(\sum_{\tau_j \in hp(i)} \frac{C_j}{T_j} \Big) + \frac{C_i}{T_i} \leq \frac{m}{2m-1} \cdot m \tag{20}$$

and

$$\frac{1}{m} \cdot \Big(\sum_{\tau_j \in hp(i)} \Big(C_j + (T_i - C_j) \cdot \frac{C_j}{T_j} \Big) \Big) + C_i \leq T_i \tag{21}$$

then all deadline of τ_i are met.

Proof. Follows from the discussion above.

3 The new algorithm

Section 3.1 presents Slack-monotonic (SM) scheduling and analyzes its performance for restricted task sets (called light tasks). This restriction is then removed in Section 3.2; the new algorithm is presented and its utilization bound is proven.

3.1 Light tasks

We say that a task τ_i is light if $\frac{C_i}{T_i} \leq \frac{2}{3+\sqrt{5}}$. We let Slack-Monotonic (SM) denote a priority assignment scheme which assigns priorities such that task τ_j is assigned higher priority than task τ_i if $T_j - C_j < T_i - C_i$.

Lemma 7. *Consider global static-priority scheduling with SM. Consider a task i. If all tasks in hp(i) meet their deadlines and*

$$\forall j \in hp(i) : \frac{C_j}{T_j} \leq \frac{2}{3+\sqrt{5}} \tag{22}$$

and

$$\frac{C_i}{T_i} \leq \frac{2}{3+\sqrt{5}} \tag{23}$$

and

$$(\sum_{\tau_j \in hp(i)} \frac{C_j}{T_j}) + \frac{C_i}{T_i} \leq \frac{2}{3+\sqrt{5}} \cdot m \tag{24}$$

then all deadline of τ_i are met.

Proof. The Inequalities 22,23 and 24 imply that Inequalities 18,19 and 20 are true. Applying Lemma 1 on Inequalities 24 gives us:

$$\frac{1}{m} \cdot (\sum_{j \in hp(i)} (2 - \frac{C_i}{T_i}) \cdot \frac{C_j}{T_j}) + \frac{C_i}{T_i} \leq 1 \tag{25}$$

Applying Lemma 2 on Inequalities 25 gives us:

$$\frac{1}{m} \cdot \left(\sum_{j \in hp(i)} (\frac{C_j}{T_j} \cdot \frac{1 - \frac{C_i}{T_i}}{1 - \frac{C_j}{T_j}} + (1 - \frac{C_j}{T_j} \cdot \frac{1 - \frac{C_i}{T_i}}{1 - \frac{C_j}{T_j}}) \cdot \frac{C_j}{T_j}) \right) + \frac{C_i}{T_i} \leq 1 \tag{26}$$

From the fact that SM is used we obtain that

$$\forall j \in hp(i) : T_j - C_j < T_i - C_i \tag{27}$$

Considering Inequality 26 and Inequality 27 and Lemma 3 gives us:

$$\frac{1}{m} \cdot \left(\sum_{j \in hp(i)} (\frac{C_j}{T_j} \cdot \frac{T_j}{T_i} + (1 - \frac{C_j}{T_j} \cdot \frac{T_j}{T_i}) \cdot \frac{C_j}{T_j}) \right) + \frac{C_i}{T_i} \leq 1 \tag{28}$$

Multiplying both the left-hand side and the right-hand side of Inequality 28 by T_i and rewriting yields:

$$\frac{1}{m} \cdot \left(\sum_{j \in hp(i)} (C_j + (T_i - C_j) \cdot \frac{C_j}{T_j}) \right) + C_i \leq T_i \tag{29}$$

Using Inequality 29 and Lemma 6 gives us that all deadline of τ_i are met. This states the lemma.

Lemma 8. *Consider global static-priority scheduling with SM. If it holds for the task set that*

$$\forall \tau_j \in \tau : \frac{C_j}{T_j} \leq \frac{2}{3 + \sqrt{5}} \tag{30}$$

and

$$\sum_{\tau_j \in \tau} \frac{C_j}{T_j} \leq \frac{2}{3 + \sqrt{5}} \cdot m \tag{31}$$

then all deadline of τ_i are met.

Proof. Follows from Lemma 7.

3.2 Light and heavy tasks

We say that a task is heavy if it is not light. We let the algorithm SM-US($2/(3+\sqrt{5})$) denote a priority assignment scheme which assigns the highest priority to heavy tasks and assigns a lower priority to light tasks; the priority order between light tasks is given by SM.

Theorem 1. *Consider global static-priority scheduling with SM-US(*$2/(3+\sqrt{5})$*). If it holds for the task set that*

$$\forall \tau_j \in \tau : \frac{C_j}{T_j} \leq 1 \tag{32}$$

and

$$\sum_{\tau_j \in \tau} \frac{C_j}{T_j} \leq \frac{2}{3+\sqrt{5}} \cdot m \tag{33}$$

then all deadlines are met.

Proof. The proof is by contradiction. If the lemma was false then it follows that there is a task set such that Inequality 32 and Inequality 33 are true and when this task set was scheduled by SM-US($2/(3+\sqrt{5})$) a deadline was missed. Let τ^{failed} failed denote this task set and let m denote the number of processors. Let k denote the number of heavy tasks. Because of Inequality 33 it follows that $k \leq m$. Also, because of Lemma 8 is follows that $k \geq 1$.

Let $\tau^{failed2}$ denote a set which is constructed from τ^{failed} as follows. For every light task in τ^{failed} there is a light task in $\tau^{failed2}$ and their T_i and C_i are the same. For every heavy task in τ^{failed} there is a heavy task in $\tau^{failed2}$ and its T_i is the same. For the heavy tasks in $\tau^{failed2}$ it holds that $C_i = T_i$. From Inequality 33 it follows that

$$\sum_{\tau_j \in light(\tau^{failed})} \frac{C_j}{T_j} \leq \frac{2}{3+\sqrt{5}} \cdot (m-k) \tag{34}$$

where $light(\tau^{failed})$ denotes the set of light tasks in τ^{failed}. Since the light tasks are the same in τ^{failed} and $\tau^{failed2}$ it clearly follows that

$$\sum_{\tau_j \in light(\tau^{failed2})} \frac{C_j}{T_j} \leq \frac{2}{3+\sqrt{5}} \cdot (m-k) \tag{35}$$

If the task set $\tau^{failed2}$ would meet all deadlines when scheduled by SM-US($2/(3+\sqrt{5})$) then it would follow (from the fact that global static-priority scheduling is predictable) that all deadlines would have been met when τ^{failed} was scheduled by SM-US($2/(3+\sqrt{5})$). Hence it follows that at least one deadline was missed by $\tau^{failed2}$. And since there are at most $k \leq m-1$ heavy tasks it follows that no deadline miss occurs for the heavy tasks. Hence it must have been that a deadline miss occurred from a light task in $\tau^{failed2}$. But the scheduling of the light tasks in $\tau^{failed2}$ is identical to what is would have been if we deleted the heavy tasks in $\tau^{failed2}$ and deleted the k processors. That is, we have that scheduling the light tasks on $m-k$ processor causes a deadline miss. But Inequality 35 and Lemma 8 gives that no deadline miss occurs. This is a contradiction. Hence the theorem is correct.

4 Conclusions

We have presented a new priority-assignment scheme, SM-US($2/(3+\sqrt{5})$), for global static-priority multiprocessor scheduling and proven that its utilization bound is $2/(3+\sqrt{5}$, which is approximately, 0.382. We left open the question whether it is possible to achieve a utilization bound of $\sqrt{2}-1$ with global static-priority scheduling.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

References

[1] Gallmeister, B.: POSIX.4 Programmers Guide: Programming for the Real World. O'Reilly Media, 1995.
[2] Sha, L., Rajkumar, R., Sathaye, S.: Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems. Proceedings of the IEEE, vol. 82, pp. 68-82, 1994.
[3] Tindell, K. W.: An Extensible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Technical Report, Department of Computer Science, University of York, UK YCS 189., 1992.
[4] Liu, C. L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Technical Report, Department of Computer Science, University of York, UK YCS 189., 1992. Journal of the ACM, vol. 20, pp. 46 - 61, 1973.
[5] Oh, D., Baker, T.P.: Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. Real-Time Systems, vol. 5, pp. 183-192, 1998.
[6] Dhall, S., Liu, C.: On a real-time scheduling problem. Operations Research, vol. 6, pp. 127-140, 1978.
[7] Baker, T.P.: An Analysis of Fixed-Priority Schedulability on a Multiprocessor. Real-Time Systems, vol. 32, pp. 49-71, 2006.
[8] Bertogna, M., Cirinei, M.,: Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms. IEEE Real-Time Systems Symposium, Tucson, Arizona, 2007.
[9] Bertogna, M., Cirinei, M., Lipari, G.: New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors. 9th International Conference on Principles of Distributed Systems, Pisa, Italy, 2005.
[10] Cucu, L.: Optimal priority assignment for periodic tasks on unrelated processors. Euromicro Conference on Real-Time Systems (ECRTS'08), WIP session, Prague, Czech Republic, 2008.

[11] Andersson, B., Baruah, S., Jonsson, J.: Static-Priority Scheduling on Multiprocessors. IEEE Real-Time Systems Symposium, London, UK, 2001.
[12] Lundberg, L.: Analyzing Fixed-Priority Global Multiprocessor Scheduling. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), 2002.
[13] Andersson, B., and Jonsson, J.: The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. Euromicro Conference on Real-Time Systems, Porto, Portugal, 2003.
[14] Liu, C.L.: Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary, vol. 37-60, pp. 28-31, 1969.
[15] Ha, R., Liu, J.W.S: Validating timing constraints in multiprocessor and distributed real-time systems. Proceedings of the 14th International Conference on Distributed Computing Systems, Pozman, Poland, 1994.
[16] Phillips, C.A, Stein, C., Torng, E. Wein, J.: Optimal time-critical scheduling via resource augmentation. ACM Symposium on Theory of Computing, El Paso, Texas, United States, 1997.

Appendix

Lemma 1. *Let m denote a positive integer. Consider u_i to be a real number such that $0 \leq u_i < \frac{2}{3+\sqrt{5}}$ and consider S to denote a set of non-negative real numbers u_j such that*

$$(\sum_{j \in S} u_j) + u_i \leq \frac{2}{3+\sqrt{5}} \cdot m \tag{36}$$

then it follows that

$$\frac{1}{m} \cdot (\sum_{j \in S} (2 - u_i) \cdot u_j) + u_i \leq 1 \tag{37}$$

Proof. Let us define f as:

$$f = (2 - u_i) \cdot \frac{2}{3+\sqrt{5}} \cdot m + m \cdot u_i - m - u_i + \frac{2}{3+\sqrt{5}} \tag{38}$$

We have:

$$\frac{\partial f}{\partial u_i} = -\frac{2}{3+\sqrt{5}} \cdot m + m - 1 > 0 \tag{39}$$

From Inequality 39 and the constraint $u_i \leq \frac{2}{3+\sqrt{5}}$ we obtain that f is no greater than f for the value $u_i = \frac{2}{3+\sqrt{5}}$. And we have $f(u_i = \frac{2}{3+\sqrt{5}}) = 0$. This gives us:

$$f \leq (2 - u_i) \cdot \frac{2}{3+\sqrt{5}} \cdot m + m \cdot u_i - m - u_i + \frac{2}{3+\sqrt{5}} \leq 0 \tag{40}$$

Applying Inequality 40 to Inequality 36 and rewriting yields:

$$(2 - u_i) \cdot \left((\sum_{j \in S} u_j) + u_i\right) + m \cdot u_i - u_i + \frac{2}{3+\sqrt{5}} \leq m \tag{41}$$

Rearranging terms in Inequality 41 gives us:

$$\frac{1}{m} \cdot \Big(\sum_{j \in S} (2 - u_i) \cdot u_j\Big) + u_i + \frac{(2 - u_i) \cdot u_i - u_i + \frac{2}{3+\sqrt{5}}}{m} \leq 1 \tag{42}$$

Recall that $u_i \leq \frac{2}{3+\sqrt{5}}$. Clearly this gives us $2 - u_i \geq 1$. And hence the last term in the left-hand side of Inequality 42 is non-negative. This gives us:

$$\frac{1}{m} \cdot \Big(\sum_{j \in S} (2 - u_i) \cdot u_j\Big) + u_i \leq 1 \tag{43}$$

And this states the lemma. Hence the lemma is correct.

Lemma 2. *Consider two non-negative real numbers u_j and u_i such that $0 \leq u_j < 1$ and $0 \leq u_i < 1$. For those numbers, it holds that:*

$$u_j \cdot \frac{1 - u_i}{1 - u_j} + (1 - u_j \cdot \frac{1 - u_i}{1 - u_j}) \cdot u_j \leq (2 - u_i) \cdot u_j \tag{44}$$

Proof. The proof is by contradiction. Suppose that the lemma is false. Then we have:

$$u_j \cdot \frac{1 - u_i}{1 - u_j} + (1 - u_j \cdot \frac{1 - u_i}{1 - u_j}) \cdot u_j > (2 - u_i) \cdot u_j \tag{45}$$

Let us explore the following cases.

1. $u_i = 0$ and $u_j = 0$
   Applying this case on Inequality 45 gives us:

$$0 > 0 \tag{46}$$

   which is a contradiction. (end of Case 1)
2. $u_i = 0$ and $u_j > 0$
   Applying this case on Inequality 45 gives us:

$$u_j \cdot \frac{1}{1 - u_j} + (1 - u_j \cdot \frac{1}{1 - u_j}) \cdot u_j > 2 \cdot u_j \tag{47}$$

   Since $u_j > 0$ we can divide Inequality 47 by u_j and this gives us:

$$\frac{1}{1 - u_j} + 1 - u_j \cdot \frac{1}{1 - u_j} > 2 \tag{48}$$

   Rewriting Inequality 48 yields:

$$\frac{1}{1 - u_j} \cdot (1 - u_j) > 1 \tag{49}$$

   which is a contradiction. (end of Case 2)

3. $u_i > 0$ and $u_j = 0$
   Applying this case on Inequality 45 gives us:
   $$0 > 0 \tag{50}$$
   which is a contradiction. (end of Case 3)
4. $u_i > 0$ and $u_j > 0$
   Since $u_j > 0$ we can divide Inequality 45 by u_j and this gives us:
   $$\frac{1-u_i}{1-u_j} + (1 - u_j \cdot \frac{1-u_i}{1-u_j}) > 2 - u_i \tag{51}$$
   Rewriting Inequality 51 yields:
   $$\frac{1-u_i}{1-u_j} - u_j \cdot \frac{1-u_i}{1-u_j} > 1 - u_i \tag{52}$$
   Further rewriting yields:
   $$\frac{1}{1-u_j} - u_j \cdot \frac{1}{1-u_j} > 1 \tag{53}$$
   Further rewriting yields:
   $$1 > 1 \tag{54}$$
   which is a contradiction. (end of Case 4)

Since a contradiction occurs for every case we obtain that the lemma is false.

Lemma 3. *Consider two non-negative real numbers u_j and u_i such that $0 \leq u_j < 1$ and $0 \leq u_i < 1$. And two non-negative real numbers T_j and T_i such that*
$$T_j \cdot (1-u_j) \leq T_i \cdot (1-u_i) \tag{55}$$
For those numbers, it holds that:
$$u_j \cdot \frac{T_j}{T_i} + (1 - u_j \cdot \frac{T_j}{T_i}) \cdot u_j \leq u_j \cdot \frac{1-u_i}{1-u_j} + (1 - u_j \cdot \frac{1-u_i}{1-u_j}) \tag{56}$$

Proof. Rewriting Inequality 55 yields:
$$\forall j \in hp(i) : \frac{T_j}{T_i} \leq \frac{1-u_i}{1-u_j} \tag{57}$$
Let $q_{i,j}$ denote the left-hand side of Inequality 57. There are two occurrences $q_{i,j}$ in the left-hand side of Inequality 56. Also observe that the left-hand side of Inequality 56 is increasing with increasing $q_{i,j}$. For this reason, combining Inequality 57 and the left-hand side of inequality 56 gives us that the lemma is true.

Lemma 4. *Consider two integers T_j and C_j such that $0 \leq C_j \leq T_j$. For every $t > 0$ it holds that:*

$$\lfloor \frac{t}{T_j} \rfloor \cdot C_j + \min(t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j, C_j) \leq C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{58}$$

Proof. The proof is by contradiction. Suppose that the lemma is false. Then there is a $t > 0$ such that:

$$\lfloor \frac{t}{T_j} \rfloor \cdot C_j + \min(t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j, C_j) > C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{59}$$

Let us consider two cases:

1. $t - \lfloor t/T_j \rfloor \cdot T_j \leq C_j$
   Let Δ be defined as: $\Delta = C_i - (t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j)$. Let us increase t by Δ. Then the left-hand side of Inequality 59 increases by Δ and the right-hand side increases by $(C_j/T_j) \cdot \Delta$. Since $C_j/T_j \leq 1$ it follows that Inequality 59 still true. That is:

$$\lfloor \frac{t}{T_j} \rfloor \cdot C_j + \min(t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j, C_j) > C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{60}$$

   Repeating this argument gives us that $t - \lfloor t/T_j \rfloor \cdot T_j = C_j$. Applying it on Inequality 60 yields:

$$\frac{t - C_j}{T_j} \cdot C_j + C_j > C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{61}$$

   Rewriting Inequality 61 gives us:

$$(t - C_j) > (t - C_j) \tag{62}$$

   which is impossible. (end of Case 1)
2. $t - \lfloor t/T_j \rfloor \cdot T_j \geq C_j$
   Let Δ be defined as: $\Delta = (t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j) - C_j$. Let us decrease t by Δ. Then the left-hand side of Inequality 59 is unchanged and the right-hand side decreases by $(C_j/T_j) \cdot \Delta$. Since $0 \leq C_j/T_j$ it follows that Inequality 59 still true. That is:

$$\lfloor \frac{t}{T_j} \rfloor \cdot C_j + \min(t - \lfloor \frac{t}{T_j} \rfloor \cdot T_j, C_j) > C_j + (t - C_j) \cdot \frac{C_j}{T_j} \tag{63}$$

   Repeating this argument gives us that $t - \lfloor t/T_j \rfloor \cdot T_j = C_j$. Applying it on Inequality 63 and applying similar rewriting as in Inequality 61 and Inequality 62 yields:

$$(t - C_j) > (t - C_j) \tag{64}$$

   which is impossible. (end of Case 2)

We can see that regardless of which case occurs a contradiction occurs and hence the lemma is correct.