Dual Priority Scheduling is Not Optimal

Pontus Ekberg
Uppsala University, Sweden
http://user.it.uu.se/~ponek616/
pontus.ekberg@it.uu.se

Abstract

In dual priority scheduling, periodic tasks are executed in a fixed-priority manner, but each job has two phases with different priorities. The second phase is entered after a fixed amount of time has passed since the release of the job, at which point the job changes its priority. Dual priority scheduling was introduced by Burns and Wellings in 1993 and was shown to successfully schedule many task sets that are not schedulable with ordinary (single) fixed-priority scheduling. Burns and Wellings conjectured that dual priority scheduling is an optimal scheduling algorithm for synchronous periodic tasks with implicit deadlines on preemptive uniprocessors. We demonstrate the falsity of this conjecture, as well as of some related conjectures that have since been stated. This is achieved by means of computer-verified counterexamples.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Scheduling

Keywords and phrases Scheduling, real time systems, dual priority

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.14

Supplement Material ECRTS 2019 Artifact Evaluation approved artifact available at https://dx.doi.org/10.4230/DARTS.5.1.1
The source code for a C program that verifies the correctness of the counterexamples in this paper can be found at https://github.com/pontusekberg/dualpriotest/.

Funding *Pontus Ekberg*: This work was supported in part by Vetenskapsrådet, grant 2018-04446.

Acknowledgements The author wants to thank Martina Maggio and Joël Goossens for independently verifying the counterexamples given in this paper, using their own computer implementations.

1 Introduction

Dual priority scheduling was introduced by Burns and Wellings [1] as a technique to schedule implicit-deadline periodic task sets with high utilization "whilst retaining an essentially static priority model". It has the capability to successfully schedule many high-utilization task sets, similar to what can be achieved by earliest deadline first (EDF) scheduling, but it also has low overhead and easy implementation, similar to that of ordinary fixed-priority scheduling.

Curiously, dual priority scheduling works very well despite its simplicity and mostly static behavior. In fact, it works so well that no feasible task set that is not dual priority schedulable has ever been found. Burns and Wellings conjectured in 1993 that dual priority scheduling is *optimal* for scheduling synchronous periodic tasks with implicit deadlines on a preemptive uniprocessor, just like EDF or Least Laxity First scheduling. This open problem has since been restated several times (e.g., in [4]) and stronger conjectures have also been given [7, 6] (these are detailed in Section 1.2). Renewed interest was likely sparked by a paper by Burns [2] dedicated to the problem in the Real-Time Scheduling Open Problems Seminar in 2010.

In this paper we give a counterexample to the conjecture by Burns and Wellings, as well as counterexamples to the related conjectures. These counterexamples can not feasibly be verified by hand because of the vast number of cases that have to be considered. Fortunately,

31st Euromicro Conference on Real-Time Systems (ECRTS 2019).
Editor: Sophie Quinton; Article No. 14; pp. 14:1–14:9



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

it is straightforward (though still time consuming) to do so by computer. A program that verifies all the counterexamples in this paper can be found by following the link under the heading *Supplement Material* above.

1.1 System model and definitions

We consider dual priority scheduling of synchronous periodic task sets on a preemptive uniprocessor. A task set $\mathcal{T}$ consists of n tasks, $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, where e_i and p_i are positive integers denoting the worst-case execution time and period of task τ_i, respectively.

In dual priority scheduling, each task τ_i is assigned two fixed priorities: a phase 1 priority π_i^1 and a phase 2 priority π_i^2. Each task is also assigned a phase change point δ_i, which is an integer such that $0 \leqslant \delta_i \leqslant p_i$.

Jobs from task τ_i are initially assigned the phase 1 priority π_i^1. If the job is still active δ_i time units after its release time, it changes its priority to π_i^2 for the remainder of its execution. Jobs are scheduled preemptively and strictly in priority order. Note that a task τ_i with $\delta_i = 0$ or $\delta_i = p_i$ is effectively a single-priority task.

We assume that priorities are distinct and that lower numbers denote higher priorities. We call a complete setting of priorities and phase change points to the tasks in a task set $\mathcal{T}$ a dual priority *configuration* of $\mathcal{T}$. If $\mathcal{T}$ meets all deadlines with a particular configuration, we call that a schedulable configuration.

There are $(2n)! \times \prod_{i=1}^{n} (p_i + 1)$ unique configurations for a task set $\mathcal{T}$ with n tasks, since there are $(2n)!$ different priority settings and $\prod_{i=1}^{n} (p_i + 1)$ different ways of assigning the phase change points. We will also consider some restrictions on the priority orderings:

▶ **Definition 1** (phase 1 RM). *A configuration of $\mathcal{T}$ is* phase 1 RM *if for each pair of tasks $\tau_i, \tau_j \in \mathcal{T}$, we have $\pi_i^1 < \pi_j^1$ if $p_i < p_j$.*

▶ **Definition 2** (phase 2 promoted). *A configuration of $\mathcal{T}$ is* phase 2 promoted *if for each task $\tau_i \in \mathcal{T}$, we have $\pi_i^2 < \pi_i^1$.*

▶ **Definition 3** (RM+RM). *A configuration of $\mathcal{T}$ is* RM+RM *if* (1) *for each pair of tasks $\tau_i, \tau_j \in \mathcal{T}$, we have $\pi_i^1 < \pi_j^1$ and $\pi_i^2 < \pi_j^2$ if $p_i < p_j$, and* (2) $\max_i(\pi_i^2) < \min_i(\pi_i^1)$.

Note that there are many priority settings that fulfill the requirements for phase 1 RM and phase 2 promoted (with distinct periods there are $\binom{2n}{n} \times n!$ and $(2n)!/2^n$ such priority settings, respectively). For RM+RM, however, there is exactly one valid priority setting if periods are distinct. For each priority setting there are $\prod_{i=1}^{n} (p_i + 1)$ unique configurations.

1.2 Related work and conjectures

In the original formulation by Burns and Wellings [1], all configurations were assumed to be both phase 1 RM and phase 2 promoted (see Definitions 1 and 2). Under these restrictions they conjectured that any feasible task set is dual priority schedulable.

▶ **Conjecture 4** (Burns and Wellings [1]). *For any feasible synchronous periodic task set $\mathcal{T}$ with implicit deadlines, there exists a schedulable dual priority configuration of $\mathcal{T}$ that is phase 1 RM and phase 2 promoted.*

However, in Burns' paper at the Real-Time Scheduling Open Problems Seminar [2], it was only assumed that configurations are phase 2 promoted, but conjectured[1] that phase 1 RM is an optimal choice for priorities.

[1] Burns did not use the word "conjecture" here, but wrote that "The phase 1 priorities are probably Rate Monotonic."

▶ **Conjecture 5** (Burns [2], but see the caveat in footnote 1)**.** *If there exists a schedulable dual priority configuration of $\mathcal{T}$, then there exists a schedulable configuration of $\mathcal{T}$ that is phase 1 RM.*

George et al. [7] and Fautrel et al. [6] made the stronger conjecture that RM+RM is an optimal choice for priorities.

▶ **Conjecture 6** (George et al. [7], Fautrel et al. [6])**.** *If there exists a schedulable dual priority configuration of $\mathcal{T}$, then there exists a schedulable configuration of $\mathcal{T}$ that is RM+RM.*

Fautrel et al. [6] also presented a heuristic called the First Deadline Missed Strategy (FDMS) for assigning the phase change points under RM+RM priorities. FDMS works by initially setting δ_i equal to p_i for every task τ_i. Then it simulates the hyper-period, and the first task τ_i to miss a deadline gets δ_i decremented by 1. This process is then repeated until no task misses a deadline (success) or a task that misses a deadline cannot further decrease its phase change point (failure). Fautrel et al. conjectured that FDMS is an optimal strategy for assigning phase change points with RM+RM priorities.

▶ **Conjecture 7** (Fautrel et al. [6])**.** *If there exists a schedulable dual priority configuration of $\mathcal{T}$ that is RM+RM, then the FDMS heuristic will find a schedulable configuration for $\mathcal{T}$.*

For task sets with only $n = 2$ tasks, Burns [2] proved that dual priority scheduling is indeed optimal. Burns also noted that given k priority levels per task (and $k - 1$ phase change points), k-priority scheduling can be made to simulate EDF and is therefore optimal. Pathan [11, 12] showed that $k = n$ priority levels is enough to simulate EDF, where n is the number of tasks. Pathan's result is also applicable to sporadic tasks and tasks without implicit deadlines. Nasri and Fohler [10] also considered scheduling with more than 2 priority levels per task and designed a heuristic for assigning the phase change points.

George et al. [7] demonstrated some properties of dual priority scheduling that differ from those of ordinary (single) fixed-priority scheduling: (1) The first job of each task does not always have the longest response time, (2) a first deadline miss can occur after the first busy period, and (3) the synchronous arrival sequence is not always the worst case.

In addition to the FDMS heuristic conjectured optimal above, Fautrel et al. [6] also designed another heuristic for finding a schedulable configuration. The latter was shown to be sub-optimal, but also faster than FDMS as it does not require repeated simulations of the hyper-period. Gu et al. [8] also designed heuristics for finding schedulable configurations, and presented a sufficient schedulability test for dual priority scheduling that is applicable to constrained-deadline periodic and sporadic tasks.

Last, there are other uses for dual priority scheduling as well. An early example is the work by Davis and Wellings [3]. Rather than using dual priority scheduling to maximize utilization of hard real-time tasks, they used it to improve the timeliness of soft real-time tasks that would otherwise run in the background. By giving the soft real-time tasks priorities in between the phase 1 and phase 2 priorities of the hard-real time tasks, they get the opportunity to run except when a hard-real time task is nearing its deadline.

2 Counterexamples

Here we list counterexamples that disprove the four conjectures in Section 1.2. Because Conjecture 6 implies Conjecture 5, we can disprove both by disproving Conjecture 5. We therefore have three counterexamples: one for disproving Conjecture 4, one for disproving Conjectures 5 and 6, and one for disproving Conjecture 7.

Verifying these counterexamples by hand is intractable, but they are readily verifiable by computer. A C program that verifies the counterexamples can be downloaded from the link given under *Supplement Material* on the first page. The run times reported below are for this program running on an AMD Ryzen 7 1700X CPU with Linux 4.20, compiled by GCC 8.2.1 and `-O3` optimization settings (memory usage is negligible).

The first counterexample actually disproves a weaker version of Conjecture 4 (thus obtaining a stronger result) by virtue of being a feasible task set that is not schedulable with *any* dual priority configuration.

▶ **Counterexample 8.** *The following task set is feasible, but is not dual priority schedulable with any configuration.*

	e_i	p_i
τ_1	8	19
τ_2	13	29
τ_3	9	151
τ_4	14	197

hyper-period:	16 390 597
utilization:	16 390 550 / 16 390 597 ≈ 0.9999971
simulated configurations:	728 082 432 000
verification run time:	∼ 61 *hours*

This counterexample is verified by simulating dual priority scheduling of the task set with all $8! \times \prod_{i=1}^{4}(p_i + 1) = 728\,082\,432\,000$ *possible configurations and noting that some deadline is missed in every simulation. The task set is feasible because its utilization is less than 1. [9]*

The above counterexample shows that dual priority scheduling is not optimal for synchronous periodic tasks with implicit deadlines. A corollary is that it is also not optimal for sporadic tasks with implicit deadlines because clearly the task set is feasible also if sporadic, but is still not dual priority schedulable. The sub-optimality also carries over to any task model that is a generalization of either of these two.[2]

The next counterexample is a task set that is dual priority schedulable, but not so with any configuration that is phase 1 RM. It thus disproves Conjectures 5 and 6.

▶ **Counterexample 9.** *The following task set is dual priority schedulable, but not with any configuration that is phase 1 RM.*

	e_i	p_i
τ_1	13	29
τ_2	17	47
τ_3	4	89
τ_4	28	193

hyper-period:	23 412 251
utilization:	23 412 240 / 23 412 251 ≈ 0.9999995
simulated configurations:	42 239 232 001
verification run time:	∼ 13 *hours*

This counterexample is verified by

- **(i)** *simulating dual priority scheduling with all* $\binom{8}{4} \times 4! \times \prod_{i=1}^{4}(p_i + 1) = 42\,239\,232\,000$ *possible configurations that are phase 1 RM and noting that some deadline is missed in every simulation, and*
- **(ii)** *simulating the full hyper-period with the (non-phase 1 RM) configuration given below and noting that no deadlines are missed with this configuration.*

A schedulable configuration that is not phase 1 RM:

	π_i^1	π_i^2	δ_i
τ_1	4	0	13
τ_2	5	1	17
τ_3	7	2	42
τ_4	6	3	139

[2] In fact, Gu et al. [8] restated Burns and Wellings' conjecture for constrained-deadline sporadic tasks. We can conclude that this variant also does not hold.

The last counterexample demonstrates that the FDMS heuristic is not guaranteed to find a valid setting of phase change points for RM+RM priorities, even though a schedulable configuration that is RM+RM exists. This disproves Conjecture 7.

▶ **Counterexample 10.** *The following task set is dual priority schedulable with a configuration that is RM+RM, but the FDMS heuristic fails to find a schedulable configuration.*

	e_i	p_i
τ_1	6	11
τ_2	6	20
τ_3	4	46
τ_4	5	74

hyper-period: 187 220
utilization: 46 804 / 46 805 ≈ 0.9999786
simulated configurations: 134
verification run time: ∼ 0.01 *seconds*

This counterexample is verified by

(i) *setting priorities to be RM+RM and noting that the FDMS heuristic fails to find phase change points that give a schedulable configuration (FDMS will try* 133 *configurations before failing), and*

(ii) *simulating the full hyper-period with the RM+RM configuration given below and noting that no deadlines are missed with this configuration.*

A schedulable configuration that is RM+RM:

	π_i^1	π_i^2	δ_i
τ_1	4	0	5
τ_2	5	1	3
τ_3	6	2	25
τ_4	7	3	35

In contrast to Counterexample 8, it can be noted that the results of Counterexamples 9 and 10 do not immediately carry over to sporadic tasks. The reason is that the synchronous arrival sequence is not necessarily the worst case [7], and so simulating it for the given custom configurations without any deadline misses does not guarantee schedulability for sporadic tasks with the same configurations.

3 On searching for counterexamples

For completeness we briefly outline the process of finding the reported counterexamples. Roughly speaking, to find these counterexamples one needs some intuition about which classes of task sets to search among (in addition to patience and a fast method for evaluating the schedulability of task sets).

The search space. After some trial and error, and after eventually finding Counterexample 10 using ad hoc methods, a more focused search was carried out in a search space delimited as follows.

- Exactly four tasks per task set.
- Task periods chosen from the first 100 prime numbers (i.e., the primes from 2 to 541).
- Product of periods at most 35 000 000.
- Utilization in the interval $[0.99999, 1]$.

In more detail, task periods were chosen uniformly at random (without replacement) from the 100 first primes, and if their product was at most 35 000 000, each possible combination of execution times for which total utilization ended up in the interval $[0.99999, 1]$ was used to generate a separate candidate task set.

"Quickly" determining schedulability. Each candidate task set was first evaluated using the FDMS heuristic [6] with RM+RM priorities. Despite being suboptimal (as shown in this paper), this heuristic could establish dual priority schedulability for the vast majority of task sets. FDMS runs in exponential time, but for these small task sets it was fast, and importantly much faster than exhaustive checking of configurations. Only task sets that were not FDMS-schedulable were finally evaluated using exhaustive methods as potential counterexamples to Conjectures 4 or 5.

Of course, checking is aborted as soon as a schedulable configuration has been found, and for each configuration, simulation is stopped at the first deadline miss. The latter point is very important as it keeps us from simulating the full hyper-period over and over. In fact, most configurations of the reported counterexamples lead to a deadline miss almost immediately. For example, in Counterexample 8 more than 99.7% of all configurations lead to a deadline miss within the largest period p_4 in that task set (i.e., within the first 197 time units). The average number of time units simulated over the 728 082 432 000 configurations is less than 31, which is smaller than 0.00019% of the hyper-period.

The bound of 35 000 000 on the product of the periods of the generated task sets is put in place to ensure reasonably fast checking of schedulability by limiting the number of combinations of phase change points. It also puts a limit on the hyper-period, but because of the above observation that we tend to not simulate even close to the full hyper-period, this is likely a less important effect.

Why four tasks in each task set? The counterexamples found all have four tasks, which seems to be a sweet spot of being complicated enough for dual priority scheduling to fail, and still small enough for exhaustive checking of all configurations to be feasible. We know from Burns [2] that there are no counterexamples with two tasks, and we have not been able to find counterexamples to any of the conjectures with three tasks either. On the other hand, five tasks seem to be beyond what can be checked with reasonable time and effort: As an example, consider what would happen if we added a fifth task with period 100 to the task set of Counterexample 8 (and adjusted execution times in some suitable way). The number of possible permutations of priorities would increase by a factor of $(2 \times 5)!/(2 \times 4)! = 90$, and the possible combinations of phase change points would increase by a factor of 101. The number of configurations we would have to check would therefore increase by a factor of 9090, and assuming everything else stays the same, the time required for checking all configurations would increase from around 2.5 days to over 60 years! We would have to be patient indeed, or at least have access to lots of processing capacity.

Why prime periods? We want the generated task sets to be as difficult as possible to schedule, and the intuition is that, in addition to high utilization, pairwise coprime periods[3] could be correlated to this. The reasoning is that the behavior of a given dual priority configuration is static with respect to the pattern of job releases (the phase change points are always at a certain offset from the job releases). Some configurations might therefore work well with some patterns, but not with others. By having all possible patterns of job releases occur in the hyper-period, the chance that no configuration will work might therefore increase. We know from the Chinese remainder theorem that all patterns will occur if the periods are pairwise coprime.

[3] Picking the periods from a list of primes was just a straightforward way of making sure they are always pairwise coprime.

The occurrence of counterexamples in the search space. The search space described above is somewhat ad hoc, and no claim is made that it is in any meaningful way the worst for dual priority scheduling. Here is, in any case, a characterization of the occurrence of "interesting" task sets in it. For this, candidate task sets were randomly generated according to the description above until a first unschedulable task set (like the one in Counterexample 8) was found. This happened after evaluating a total of 130 255 candidate task sets. These task sets are classified in the table below.

	# task sets	% of explored search space
Schedulable with RM+RM using FDMS	129 823	$\sim 99.67\%$
Schedulable with RM+RM, but not using FDMS	0	0%
Schedulable with phase 1 RM, but not RM+RM	284	$\sim 0.22\%$
Schedulable, but not with phase 1 RM	147	$\sim 0.11\%$
Unschedulable	1	$\sim 0.0008\%$

It can be noted that even in this search space, which was heavily narrowed down to what was believed to be "difficult" task sets, almost all task sets are still schedulable with the FDMS heuristic and RM+RM priorities. It is also interesting that among the evaluated task sets in this experiment, FDMS succeeded in finding a schedulable configuration to all task sets that were schedulable with RM+RM. (However, as is evidenced by Counterexample 10, there are cases where FDMS fails to find schedulable RM+RM configurations.)

4 Conclusions and open problems

Even though we show negative results about dual priority scheduling in this paper, most importantly that it is not an *optimal* algorithm, this does not directly imply that dual priority scheduling is not, in general, a *good* algorithm. It still successfully schedules the vast majority of task sets that have been tried – presumably the very reason it was conjectured optimal in the first place. One could argue, with some justification, that it is close enough to optimal for any *practical* uses. However, there are still major challenges ahead to make full use of the apparent near-optimality of dual priority scheduling. Here we list some of those challenges (see also Burns [2]).

1. **Can we efficiently test whether a schedulable configuration exists?** Had dual priority scheduling been optimal this would be trivial by simply checking whether the utilization is at most 100%, but now this cannot be a sufficient test. The current state-of-the art, to try all possible configurations and see if one works, is utterly unsatisfactory. This problem is clearly in PSPACE, but lower bounds are unknown.
2. **Can we efficiently find a schedulable configuration if one exists?** It is not unlikely that a good solution to the first point would somehow allow us to also find a schedulable configuration, but it is not necessarily so. For example, a non-constructive proof of optimality would have allowed us to use the 100% utilization bound to check for existence of a schedulable configuration, but it might not have helped us find one.
3. **Can we efficiently test whether a given configuration is schedulable?** The current state-of-the-art is to simulate a full hyper-period, which is straightforward but not very scalable. It also does not seem to be applicable to sporadic tasks as George et al. [7] have shown that the synchronous arrival sequence is not guaranteed to be the worst case. This problem is clearly in PSPACE and is also (weakly) NP-hard as it generalizes the ordinary fixed-priority schedulability problem [5].

4. **What is the utilization bound for dual priority scheduling?** Dual priority scheduling is suboptimal, but is more powerful than ordinary fixed-priority scheduling. Its utilization bound must therefore be in the half-open interval $[\ln(2), 1)$,[4] but where? It can be noted that a constant utilization bound can not be an exact schedulability test here.
5. **Does the apparent near-optimality of dual priority scheduling diminish with larger task sets?** Dual priority scheduling appears to work very well, but out of necessity it has only been systematically evaluated on small task sets so far (something that solutions to the issues above might change). At the same time, there is evidence that the number of priorities each task might need is related to the number of tasks in the task set: Burns [2] proved that dual priority scheduling is optimal for task sets with $n = 2$ tasks, while Pathan [11, 12] has shown that, in general, n-priority scheduling is optimal for task sets with n tasks. Does dual priority scheduling seem so good because our observations are biased?
6. **Is phase 2 promoted an optimal choice for priorities?** It is usually assumed in the literature that configurations should be phase 2 promoted (see Definition 2). While this seems very sensible, it has to the best of the author's knowledge never been proven optimal. After all, phase 1 RM and RM+RM seem like sensible choices as well, but they turn out to not be optimal.
7. **Would rational phase change points improve schedulability?** We have assumed that the phase change points, like task parameters, are integer. Integer phase change points are a reasonable assumption as we could always pick the clock cycle as our time unit, and it also seems to be in line with previous work on dual priority scheduling, even though it is not always spelled out. But what if phase change points can be rational? An interesting theoretical question is whether this adds any extra power. Naturally, it is impossible to exhaustively check all configurations in such a model.
8. **Is k-priority scheduling optimal for some constant k?** We know now that dual ($k = 2$) priority scheduling is not optimal, but also that given enough priority levels k per task, k-priority scheduling can be made to simulate EDF and is therefore optimal. The smallest bound on k known for optimal scheduling is $k = n$, where n is the number of tasks, as shown by Pathan [11, 12]. Is there some constant k, such that k-priority scheduling is optimal?
9. **Is there some deeper insight to be gained about why some task sets are unschedulable?** We have demonstrated that dual priority scheduling is not optimal, but the brute-force method with which this is done leaves something to be desired. It is not easy to see what actually makes a certain task set unschedulable, like the one in Counterexample 8. For a given configuration we could, in principle, analyze the concrete schedule to see where the "wrong" scheduling decision was made and explain why and how this happened. Explaining in a satisfiable way why none of billions of different configurations work seems to require some entirely different type of insight, however. This is insight that is currently lacking. Of course, it is entirely possible that there does not actually exist a more elegant and general characterization of dual priority unschedulability than "a task set is unschedulable if all configurations lead to a deadline miss." But if a more elegant characterization does exist, finding it would be key to deeper understanding. Such a characterization would likely be needed for a satisfactory solution to the first point in this list as well.

[4] As is evidenced by Counterexample 8, the utilization bound must in fact be in the slightly smaller interval $[\ln(2), 16\,390\,550 / 16\,390\,597)$.

References

1. A. Burns and A.J. Wellings. DUAL PRIORITY ASSIGNMENT: A Practical Method for Increasing Processor Utilisation. In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pages 48–53, 1993. doi:10.1109/EMWRT.1993.639052.
2. Alan Burns. Dual priority scheduling: Is the processor utilisation bound 100%? In *Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2010. URL: https://www.cs.york.ac.uk/ftpdir/reports/2010/YCS/455/YCS-2010-455.pdf#page=9.
3. Robert Davis and Andy Wellings. Dual Priority Scheduling. In *Proceedings of the 16th Real-Time Systems Symposium (RTSS)*, pages 100–109, December 1995. doi:10.1109/REAL.1995.495200.
4. Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64–82, 2016. doi:10.1016/j.sysarc.2016.04.002.
5. Pontus Ekberg and Wang Yi. Fixed-Priority Schedulability of Sporadic Tasks on Uniprocessors is NP-hard. In *Proceedings of the 38th Real-Time Systems Symposium (RTSS)*, pages 139–146, 2017. doi:10.1109/RTSS.2017.00020.
6. Tristan Fautrel, Laurent George, Joël Goossens, Damien Masson, and Paul Rodriguez. A Practical Sub-Optimal Solution for the Dual Priority Scheduling Problem. In *Proceedings of the 13th International Symposium on Industrial Embedded Systems (SIES)*, June 2018. doi:10.1109/SIES.2018.8442075.
7. Laurent George, Joël Goossens, and Damien Masson. Dual Priority and EDF: a closer look. In *Proceedings of the Work-in-Progress Session of 35th Real-Time Systems Symposium (RTSS-WiP)*, December 2014. URL: https://hal.archives-ouvertes.fr/hal-01217433.
8. Xiaozhe Gu, Arvind Easwaran, and Risat Pathan. Design and Analysis for Dual Priority Scheduling. In *Proceedings of the 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 164–173, 2018. doi:10.1109/ISORC.2018.00033.
9. C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.
10. Mitra Nasri and Gerhard Fohler. Some Results in Rate Monotonic Scheduling with Priority Promotion. In *Proceedings of the Work-in-Progress Session of the 27th Euromicro Conference on Real-Time Systems (ECRTS-WiP)*, pages 5–8, 2015. URL: https://people.mpi-sws.org/~mitra/papers/Nasri_WIPECRTS15.pdf.
11. Risat Mahmud Pathan. Unifying Fixed- and Dynamic-Priority Scheduling based on Priority Promotion and an Improved Ready Queue Management Technique. In *Proceedings of the 21st Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 209–220, April 2015. doi:10.1109/RTAS.2015.7108444.
12. Risat Mahmud Pathan. Real-Time Scheduling on Uni- and Multiprocessors based on Priority Promotions. *Leibniz Transactions on Embedded Systems*, 3(1):02–1–02:29, 2016. doi:10.4230/LITES-v003-i001-a002.