METHODS OF TASK CLINCHES PREVENTION IN SOFTWARE APPLICATIONS FOR REAL-TIME SYSTEMS

Viktor Nikiforov, Vladimir Pavlov

This Publication has to be referred as: Nikiforov, V[iktor] & Pavlov, V[ladimir] (2017). Methods of Task Clinches Prevention in Software Applications for Real-Time Systems, Proceedings of the 28th DAAAM International Symposium, pp.0826-0834, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-11-2, ISSN 1726-9679, Vienna, Austria

DOI: 10.2507/28th.daaam.proceedings.116

Abstract

Real-time software applications are built as a set of tasks sharing hardware and information resources. A consistency of shared information resources is achieved by the usage of mutex-type synchronizing means. The usage of mutexes may lead into task clinches. A traditional mean to prevent task clinches is the Priority Ceiling Protocol (PCP). But PCP utilization excludes the using of efficient scheduling modes with task priorities reassigned dynamically. Therefore the implementation efficiency may be decreased significantly for a wide range of real-time software applications. Conducted experiments showed that such decrease can be especially significant when software applications are realized in a multi-core processor environment. The paper represents methods of the task clinch prevention which ensure that a wide range of applications can be scheduled with efficient scheduling modes permitting dynamic task priorities by using a specialized multi-partite oriented graph reflecting features of inter-task synchronizing links appearing when using mutexes. The absence of inter-partite contours means that task clinches are impossible. If such contours exist, it is possible to determine the code section that shall be modified to break these contours. If such modification is impossible then the new developed protocol can be utilized.

Keywords: real-time systems; multitasking software applications; task synchronization; scheduling modes; resource access protocols

1. Introduction

The study concerns a feasibility of software applications for real-time systems that relates to the problems of the efficient real-time operating systems development which provide an effective task scheduling without processor resource usage productivity decreasing. Real-time systems shall ensure a well-timed information exchange with external processes. The key property of real-time software applications is that they run in a “time structure” that is defined by the course of external processes [1]. Such correspondence between the course of the real-time system software application components execution and the course of external processes is caused by the presence of more or less hard time frames for information exchanges with external processes. One of the main consequences of this fact is a requirement to build a real-time software application as a set of cooperative tasks $\tau_1, \tau_2, \dots, \tau_n$.

During execution of the multitasking software application its cooperative tasks share common hardware resources: executive resources (first of all, processors and processor cores) and information resources (global data files, interface registers of peripheral devices, man-machine interface elements, etc.). To ensure an execution consistency the cooperative tasks exchange data and synchronizing (signal) information messages. A logical correctness of the task interaction structure consists in an assurance of the following: a) consistency of the shared information resources; b) absence of the risk of clinches when two or more tasks are in the state of mutual blocking. An additional condition of the dynamic correctness exists for the real-time systems: it is necessary to ensure a timeliness of the task executions.

A traditional approach to an assurance of the shared resources consistency is a usage of mutexes [2]. For each of the shared resources g_k a specific program object (mutex mut_k) is created that is a synchronizing element indicating occupancy of the resource g_k . The code section in which the task τ_i has an access to the resource g_k (critical interval on the access to g_k) is restricted by the synchronizing operators $lock(mut_k)$ and $unlock(mut_k)$ which restrict a capture and a release of the resource g_k , respectively.

The purpose of this study is to find new approaches to such computational process organization that ensure a task clinch prevention as well as a retention of the possibility to use highly efficient scheduling modes with dynamic task priorities.

2. Multitasking application profile

To fulfill a static analysis of the multitasking application with shared information resources or a simulation of the execution progress of such application the models are built in which a code of each task is represented as a chain of segments. Each segment contains a program code fragment that is terminated by the operator associated with the system event:

- segment terminated by the information resource request (by the *lock* operator);
- segment terminated by the *unlock* operator (resource release);
- final segment (task completion).

Tasks that don't contain resource access operators consist of the single (final) segment. A set of data about segment types sequence (with an indication of the specific information resource for which *lock* or *unlock* operator is executed) presents the task scheme. A set of schemes of all tasks represents the application scheme.

An example of the application scheme consisting of two tasks is presented in fig. 1a (using route net means [3]). The code of task τ_1 as well as the code of task τ_2 contains overlapping critical intervals on the access to resources g_1 and g_2 . The schemes of tasks τ_1 and τ_2 differ in the overlapping kind of critical intervals: the code of task τ_1 contains the intersection of critical intervals but the code of task τ_2 contains the nesting of critical intervals, i.e. the critical interval on the resource g_2 is nested into the critical interval on the resource g_1 . Application schemes contain all necessary and sufficient information to check a logical correctness of the application, in particular, to check an absence of the risk of clinches during the application execution.

To check a dynamic correctness of software applications used in real-time systems the scheme of each task shall be complemented with the following parameter values:

- weight w of each segment (i.e. an amount of computations performed in the frame of the segment);
- period T_i and deadline D_i of each task τ_i .

Fig. 1. Presentation of data about software application consisting of two tasks using route net means.

The weight w of the segment corresponds to the amount of computations executed within the segment (in a form of the equivalent amount of reference operations, e.g. arithmetic operations with floating point). The sum of weights of all segments contained in the code of task τ_i is the weight W_i of the complete task τ_i . If a processor throughput P is known (as a number of reference operations per second) then the task weight can be expressed in seconds of the processor time: $C_i = W_i/P$. The ratio $u_i = C_i/T_i$ is a processor load from the task τ_i (the load corresponds to the processor time quota that

shall be allocated for the execution of instances of the task τ_i). The total load $U = \sum u_i$ indicates a measure of the processor time usage by the application with the given profile and with the given processor throughput value P .

The period T_i represents a minimum acceptable duration of the time interval between adjacent activations of the task τ_i ; the deadline D_i represents a maximum acceptable duration of the time interval between a moment of the j -th activation of the task τ_i (i.e. a moment of the j -th task instance generation) and a moment of this instance execution termination.

The profile of the task τ_i is a set of data about a sequence of the segment kinds forming the task model with an indication of each segment weight as well as T_i and D_i values corresponding to this task. The application profile is a set of profiles of all its tasks.

The profile of the application with the scheme represented in fig. 1a is shown in fig. 1b. The first, second, third, fourth, and fifth segments of both tasks τ_1 and τ_2 have values of weights 3, 2, 5, 3, and 3 respectively. The weight C_1 of the task τ_1 (as well as the weight C_2 of the task τ_2) is equal to 16 units of the processor time measure (e.g. milliseconds). The value of the period T_i of the task τ_i is pointed out above a dotted arrow in front of the first segment of the task code. The deadline value is pointed out after the last task segment ($T_1 = 30$, $T_2 = 50$, $D_1 = 30$, $D_2 = 40$).

The ratio $H_i = T_i/D_i$ indicates a hardness of requirements for the task τ_i deadline. The hardness $H_1 = 1$ of the task τ_1 in fig. 1b means that the time interval of the τ_1 current execution shall not be overlapped with the time interval of its next execution. The hardness $H_2 = 1.25$ of the task τ_2 means that the time interval of the τ_2 current execution shall be terminated in advance so that it shall be not less than 20% of the period T_2 before the next τ_2 execution.

3. Processor throughput and execution configuration

An application execution is carried out in specific conditions defined by the known processor having known throughput P , known number m of cores, known scheduling mode and known access protocol to shared resources. If these conditions are defined and the application profile is known then the question about feasibility of this application with the given profile in the given conditions becomes actual one. Feasibility means a guaranteed deadline of all tasks in any acceptable scenarios of system events. If it is provided to be that in the given defined conditions one can find such task τ_i and such acceptable scenario of system events in the application with the given profile that some j -th instance of the task has no time to complete its execution in the frame of the time interval of the length D_i then this application is considered to be unfeasible in the given conditions.

Theoretically it is possible to get the application feasibility via increasing the value P of the processor throughput under invariability of the application profile and invariability of the rest application execution condition except P . Let's call these rest conditions the application execution configuration. The configuration specification includes the number m of processor cores, used scheduling mode, and used access protocol to the shared information resources.

When the value P increases, the values u_i of loads from each task τ_i and the value U of the total processor load from the whole application will decrease. Varying values P it is possible to find (with some prescribed accuracy) such maximum value $U_{\max}$ when the application still remains feasible. The value $Dens = U_{\max}/m$ (where m — number of processor cores) is called a density of the application in regard to the specific execution configuration [4].

The value of $Dens$ is no more than 1. This value reflects an efficiency of the specific execution configuration of the application with specific profile. Here efficiency is a maximum attainable quota of the processor time used effectively. For example, the value $Dens = 0.5$ means that an application with given profile cannot use effectively more than 50% of the processor time under guarantees of the each task deadline in the given execution configuration (the remaining time a processor will either stand idle or make calculations not relevant to an execution of tasks $\tau_1, \tau_2, \dots, \tau_n$). This circumstance allows to consider the value $Dens$ as a criterion of the comparative efficiency of various execution configurations usage in an application with the specific profile.

In particular, such comparison can be used to choose a scheduling mode in the real-time system functioning under control of the software application with the specific profile when it is necessary to resolve collisions between requirements to decrease a volume of allocated computational resources and requirements to keep a feasibility of the real-time software application.

Studies devoting to a development of efficient scheduling modes for real-time systems based on individual single-core processors have been started more than 40 years ago. In [5] the authors proved the optimality of the Rate Monotonic (RM) scheduling mode in the class of scheduling modes with static task priorities as well as the Earliest Deadline First (EDF) scheduling mode in the class of scheduling modes with task priorities assigned dynamically. They found a critical profile with the following features shown in fig. 2a:

- total load U is distributed evenly between application tasks (for each task $u_i = U/n$);
- values of periods T_i are increased exponentially with the increasing of index i .

A comparison of RM and EDF scheduling modes for the application consisting of 5 independent tasks executed on a single-core processor is presented in fig. 2b for hardness values $H_i = 1$ and $H_i = 1.25$ that shows an advantage of the EDF scheduling mode relative to the RM scheduling mode.

The application profile with parameter values such as in fig. 2a is the critical one to the effect that applications with such profile have the minimum density value $Dens$ for the RM scheduling mode. Below we will call application profiles with such correlations between periods and loads as Liu-Layland profiles.

Density values presented in fig. 2b have been derived as analytically as via simulation. To check a validity of the measurement results obtained during simulation these results have been compared with results obtained via analytical

methods. A coincidence of density values obtained via these two methods confirms a validity of the results obtained via simulation. The density values presented in fig. 2b indicate that EDF scheduling is more efficient than RM scheduling under standard and more hard requirements concerning the task deadline.

Fig. 2. Density values for the application consisting of five independent tasks with Liu-Layland profile

Thus the simulation confirmed that RM and EDF scheduling modes are the optimal ones in their classes when the applications are run on single-core processor. These scheduling modes win with respect to any other scheduling modes in their classes not only when applications with Liu-Layland profiles are executed but also when applications with any other profiles are executed.

Nowadays multitasking software applications are developed to be implemented on multiprocessor hardware or on multi-core processors. For example, scheduling modes for multiprocessor hardware are considered in [6]. But below we will discuss the scheduling modes for multitasking software applications implemented on multi-core processors.

In the case of multi-core processors RM and EDF scheduling modes are not optimal ones. They become very inefficient ones when the total load has essentially irregular distribution between tasks).

In the application with profile shown in fig. 3a 60 % of the total load carries the task τ_5 therefore it can be called “heavy”, then the tasks $\tau_1, \tau_2, \tau_3, \tau_4$ can be called “light”. It is reasonable to use the modified RM scheduling mode (ModifRM) for applications with the shifted load: a heavy task will have the highest priority while the light tasks scheduling is carried out with usual RM mode. The histograms presented in fig. 3b and 3c show that RM and EDF scheduling modes lose essentially with respect to ModifRM scheduling mode as for two-core processor as for four-core processor.

Fig. 3. Density values for the application consisting of five independent tasks with the shifted load

An efficiency of the scheduling mode with fixed task priorities is decreasing when the number of processor cores is increasing. When an application is run on the multi-core processor the best of the known scheduling modes with fixed task priorities ensure a dynamic application execution correctness if total load values do not exceed $0.4m$ (where m – number of processor cores) [7].

In the late 1990s it was proved the existence of Pfair scheduling modes that ensure the feasibility of applications consisting of independent tasks on multi-core processors when $Dens = 1$ [8]. Later the algorithms were developed that realize Pfair scheduling modes in systems with time slicing (e.g. [9]) as well as in systems without time slicing [10].

Thus, the scheduling modes with dynamic task priorities can ensure an essential gain in the processor resource usage efficiency in real-time systems especially when software applications are realized on multi-core processors.

The examples considered above relate to applications with independent tasks. In regard to multitasking software applications with interdependent tasks the specialized protocols of task requests processing on an access to information resources have been developed. These protocols differ in conditions and/or actions checking/performing in the order of *lock* and *unlock* operations execution [11].

When the simplest access protocol (SP) is applied the single condition of the requested resource allocation is that the resource is free at the current moment [12]. If the resource is occupied then the task requesting an access to it is converted into the state of this resource release waiting. No additional actions are performed in this case. A disadvantage of this protocol is that it allows:

- a clinch appearance and
- a priority inversion that can lead to the value $Dens$ significant decreasing.

The Priority Inheritance Protocol (PIP) requires a performance of the following additional (relative to SP) actions: if high priority task τ_i requests a resource g that is used by low priority task τ_j , then task τ_i (temporally, until τ_j will release g) gives its own priority to τ_j . In the case of PIP usage a priority inversion is prevented but a possibility of the clinch appearance is preserved [13].

The Priority Ceiling Protocol (PCP) prevents as priority inversion as clinch appearance possibility. When this protocol is used it is necessary to make a specific static processing of the scheme (on the stage of software application design) reflecting a structure of inter-task links in a multitasking software application of the real-time system [14]: each resource g is provided with the special scheduling parameter $pc(g)$ that is called a priority ceiling of g . The value $pc(g)$ is equal to the highest static priority $prio(\tau_i)$ among the tasks which have critical interval on g . The Priority Inheritance Protocol requires a check of the more strong condition (relative to SP and PIP) when task τ_i requests a resource g : if a resource g with $pc(g) \geq prio(\tau_i)$ is occupied by τ_j (j is not equal to i) then task τ_i is converted into the waiting state until all resources g with such high priority ceiling will be released.

New protocol of Inter-Partite Contours (IPC protocol) not only prevents a priority inversion and a clinch appearance but retains a possibility also to use efficient scheduling modes with dynamic priorities [15]. When IPC protocol is used as well as when PCP is used a specific static processing of the application scheme is required. This processing is based on construction and analysis of the bundle graph that is a specialized multi-partite oriented graph reflecting features of inter-task synchronizing links.

4. Bundle graph

A scheme of the multitasking software application containing interdependent tasks first of all should be checked on the clinch appearance possibility. If this check indicates that the clinch appearance is impossible then to ensure a consistency of the shared resources it is sufficiently to apply the SP protocol which allows to use any scheduling modes including (if necessary) the most efficient scheduling modes with dynamic task priorities.

The simplest method of this check is a search of the critical interval bundles. If the critical interval bundles are not discovered then it is sufficiently to apply the SP or PIP protocols which allow a usage of the scheduling modes with dynamic task priorities.

If the critical interval bundles exist then developers apply the PCP protocol. A difference between PCP and SP or PIP is that in addition to the check whether the requested resource is unoccupied it is necessary to make an additional check of the current state of other shared resources. To ensure a mutual blocking prevention by the PCP usage it is necessary to pay with a decreasing of the value $Dens$. The application density can decrease because of PCP does not allow the usage of the highly efficient scheduling modes with a dynamic reallocation of the task priorities.

An availability of the critical interval bundles is a necessary condition of the clinch appearance possibility. However this fact in itself does not mean that the mutual blocking is really possible. For example, the mutual blocking is impossible in the application with the scheme shown in fig. 1a despite the presence of the critical interval bundles but the application with the structure shown in fig. 2 can get into the state of the mutual blocking of the tasks τ_2 , τ_3 , τ_4 . Therefore when the critical interval bundles are available it is still unclear is it necessary to apply PCP which although protects an application from the clinch appearances but excludes a usage of the efficient scheduling modes. Real-time software application developers are obliged to apply PCP accompanying with the appropriate efficiency decreasing though it is not excluded that such precaution is unnecessary in fact.

To apply PCP in those cases only when it is really essential a developer shall have a necessary and sufficient condition of the clinch appearance possibility as well as an efficient tool to check a fulfillment of this condition. Such tool is a construction and analysis of the critical interval bundle graph (or simply the bundle graph) [16].

In the application consisting of two tasks τ_1 and τ_2 that is represented by the scheme in fig. 1a each task contains overlapping critical intervals on the access to resources g_1 and g_2 : critical intervals of the task τ_1 are concatenated ones and critical intervals of the task τ_2 are nested ones. Let's call two critical intervals of the task τ on resources g and g^* bundled ones (i.e. they form a bundle $L = \langle \tau, g, g^* \rangle$) if they overlap, i.e. they contain common segments of the task code.

In the starting (leading) section of the bundle $L = \langle \tau, g, g^* \rangle$ the task τ has an access to the head resource g of the bundle. In the central section the task τ has an access to the both resources of the bundle: head g and additional g^* . In the closing section one of the bundle resources is released already by the task τ .

The bundle graph is the multi-partite oriented graph that reflects dependence relations of the critical interval bundles. The bundle $L_x = \langle g_a, g_b \rangle$ depends on the bundle $L_y = \langle g_c, g_d \rangle$ if τ_i and τ_k are different tasks and $g_b = g_c$. In other words, the bundle L_x depends on L_y if L_x and L_y belong to different tasks and the head resource of the bundle L_y coincides with the additional resource of the bundle L_x . A node of the bundle graph has a one-to-one correspondence with a bundle existing in the application scheme.

If two graph nodes L_x и L_y are bundled by the arc directed from L_x to L_y then the bundle L_x depends on the bundle L_y . The profile and the bundle graph of the application consisting of five tasks which share four resources are represented in fig. 4a. Figure 4b represents the profile and the bundle graph of the application that differs from the application shown in fig. 4a in that the request operator of the resource g_2 in the task τ_4 is shifted so that the resource g_2 is captured before the resource g_4 .

Fig. 4. An example of the application profile modification eliminating a clinch appearance possibility

The bundle graph is a multi-partite graph (with one partition per each task) because dependent bundles cannot belong to the same task. The contour in the bundle graph shown in fig. 4a is an inter-partite contour in the sense that all its nodes belong to different partitions of the graph. Just the presence of such contour indicates a possibility of the appearance of the mutual blocking of tasks. It is necessary and sufficient for the mutual blocking appearance possibility during application execution that an inter-partite contour exists in the corresponding bundle graph.

The application with the structure shown in fig. 1 has no dependent bundles and the corresponding bundle graph does not contain arcs (and therefore inter-partite contours). The protocols SP or PIP allowing the efficient scheduling modes usage can be utilized during such application execution. In the case of the software application shown in fig. 4a the usage of these protocols could lead to the clinch appearance.

A single way exists to prevent a clinch without the bundle graph usage, namely, the PCP usage with the corresponding efficiency loss. When the bundle graph is used there are two ways of SP and PIP usage with the corresponding maintenance of the possibility to use highly efficient scheduling modes.

The first way is connected with a local modification of the application scheme that ensures a break of the contour in the bundle graph. So the contour corresponding to the application in fig. 4a will be broken if an order of the capture of resources g_2 and g_4 by the task τ_4 will be changed. In the scheme shown in fig. 4a the request of the resource g_4 precedes the request of the resource g_2 . If to relocate a request operator of the resource g_2 so that it precedes the request of the resource g_4 then the contour in the bundle graph will be broken (see fig. 4b), hence, the PCP usage becomes unnecessary and the possibility of the highly efficient scheduling modes usage is maintained.

The second way is connected with the possibility check of the ICP protocol usage. This protocol can be applied if the bundle graph does not contain overlapping inter-partite contours. If the required modification of the application scheme is unallowable on some reasons and overlapping inter-partite contours are absent (as in application represented in fig. 5) then the ICP protocol usage is allowed which, from the one hand, prevents clinches and, from the other hand, allows the usage of the scheduling modes with dynamic task priorities.

Fig. 5. Application with the bundle graph containing two non-overlapping inter-partite contours

When the IPC protocol is applied as well as the PCP is applied it is required a static processing of the application scheme. Values $pc(g)$ of the resource ceiling priorities are calculated during such processing for the PCP, but as for IPC protocol the list of inter-partite contours $\{C_1, C_2, \dots, C_k\}$ existing in the bundle graph is built. The IPC protocol usage is possible only when the bundle graph corresponding to the application does not contain overlapping inter-partite contours. In this case each critical interval bundle L existing in the application either is included in the specific inter-partite contour $C(L)$ or is not related to any inter-partite contour.

To execute an application with the IPC protocol usage each inter-partite contour C is provided with a counter $cnt(C)$ of the number of tasks which currently execute a code segment corresponding to any leading section of the bundle containing in the contour C . Besides the length $lng(C)$ of each contour is fixed.

When the task τ requests an access to the code segment being a leading section of the bundle L that belongs to C the additional check of the condition $cnt(C) < (lng(C) - 1)$ is performed besides conditions and actions corresponding to PIP. If this condition is satisfied then the task τ obtains an access to the requested resource and the value $cnt(C)$ is increased by one. If this additional condition is not satisfied then an execution of the task τ is suspended while it will be satisfied.

The bundle graph in fig. 5 contains two non-overlapping inter-partite contours. The length of each of these contours is equal to 3. If the application scheme can't be modified so that each contour will be broken then the IPC protocol may be used. In this case clinches will be impossible and efficient scheduling modes become available.

5. Density of applications with interdependent tasks

The means for a simulation of the application execution with interdependent tasks were included into the simulation software ensuring an acquisition of density values to compare a relative efficiency of the RM and EDF scheduling modes.

Histograms in fig. 6 represent a dependence of density on hardness for applications with the profile shown in fig. 4b when using EDF and RM scheduling modes. Note that this application profile contains critical interval bundles. Therefore a developer who has no a checking tool to detect a clinch appearance possibility during such application execution with SP or PIP protocols need to apply PCP with RM scheduling mode. In this case, when a real-time system is realized on a single-core processor and a hardness has a standard value $H = 1$, it will be received approximately 25 % loss of efficiency in comparison with the case when a developer knows about the absence of inter-partite contours and can apply EDF scheduling mode without fear of clinch appearances (fig. 6a). When a real-time system is realized on a two-core processor this loss will be 40 % (fig. 6b). When inter-partite contours exist in a bundle graph a developer can find such modification of the application scheme which provides inter-partite contour breaks (just as it occurs when the application scheme shown in fig. 4a is replaced by the application scheme shown in fig. 4b) and it becomes possible to increase the application implementation efficiency in the tens of percent.

Fig. 6. Execution of the application with interdependent tasks on a single-core processor

A modification of the application scheme by some reason can appear to be unwanted or impossible. In this case it is necessary to check a fitness for use of the new IPC protocol that prevents clinch appearances and at the same time allows applying scheduling modes with task priorities assigned dynamically. Fig. 7 represents density values obtained as a result of the simulation of the application with profile shown in fig. 4a realized on a single-core processor.

Fig. 7. Density values when IPC protocol is applied on the single-core processor

The data represented in fig. 7 indicate that IPC protocol usage together with EDF scheduling mode ensures the same gain in the tens of percent that is obtained due to application scheme modification eliminating inter-partite contours in the bundle graph.

6. Conclusion

The problem was that the usage of existing access protocols to shared information resources ensuring a task clinch prevention is connected with the necessity to reject the usage of highly efficient scheduling modes with dynamic task priorities that leads to processor resource usage productivity decreasing. The necessary condition for clinch appearances during multitasking applications execution is a presence of critical interval bundles in task codes. If an analysis of the application scheme detects an absence of bundles then a developer has no limitations in the configuration choice of such application execution (i.e. in the choice of appropriate scheduling modes and access protocols to shared resources). The traditional way to prevent clinches when critical interval bundles are present is a usage of the Priority Ceiling Protocol. But this protocol usage is inconsistent with the usage of highly efficient scheduling modes with task priorities assigned dynamically.

A suggested approach to solve this problem consists in the construction and analysis of the multi-partite graph of critical interval bundles. An application has no risk of clinch appearances if this graph does not contain inter-partite contours. In this case the most efficient execution configurations can be used for application implementation. The inter-partite contour existing in this graph indicates those sections of task codes a local modification of which leads to the contour break. Such modifications transfer the application into a class of applications which are free from the risk of clinch appearances.

The results obtained consist in a specific static processing of the application scheme based on the multi-partite graph analysis. This analysis can help a software application developer to find the task code sections, modifications of which will break inter-partite contours and prevent clinch appearance. If such task code modifications by some reason are unwanted or impossible then the new developed Inter-Partite Contours protocol can be used to prevent clinch appearance and to use highly efficient scheduling modes if there are no overlapping inter-partite contours in the multi-partite graph.

The future plans concern the Inter-Partite Contours protocol improvement that will give a possibility to use this protocol for software applications with overlapping inter-partite contours in the graph of critical interval bundles.

7. References

- [1] Davidenko, K. (1985). A Programming Technology for Automated Control Systems of Manufacturing Processes, Energoatomizdat, Moscow (in Russian)
- [2] Laplante, P. (2004). Real-Time Systems Design and Analysis, John Wiley & Sons, Inc., ISBN 0-471-22855-9, NJ, USA
- [3] Nikiforov, V. & Shkirtil, V. (2010). Route nets – graphical form for structural representation of real-time software applications. SPIIRAS Proceedings, No. 3(14), pp. 7–28, ISSN 2078-9599 (in Russian)
- [4] Baranov, S. & Nikiforov, V. (2015). Density of multi-task real-time applications, Proceedings of the 17th Conference of Open Innovations Association (FRUCT 2015), 20-24 April 2015, Yaroslavl, Russia, ISSN 2305-

- 7254, ISBN 978-5-7577-0493-7, Balandin, S. (Ed.), pp. 9-15, Publishing house of Research University of Information Technologies, Mechanics and Optics, Saint-Petersburg, Russia, DOI: 10.1109/FRUCT.2015.7117964
- [5] Liu, C. & Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, Vol. 20, No.1, January 1973, pp. 46-61, ISSN 0004-5411
- [6] Amol, A. S. (2016). Multicriteria dynamic scheduling by topsis and goal programming, *Proceedings of the 26th DAAAM International Symposium*, 21-24 October 2015, Zadar, Croatia, ISSN 1726-9679, ISBN 978-3-902734-07-5, Katalinic, B. (Ed.), pp. 0426-0434, Published by DAAAM International, Vienna, Austria, DOI: 10.2507/26th.daaam.proceedings.057
- [7] Anderson, B. (2008). Global static-priority preempted multiprocessor scheduling with utilization bound 38%, *Proceedings of the 12th International Conference on Principles of Distributed Systems*, 15-18 December 2008, Luxor, Egypt, ISSN 0302-9743, ISBN 3-540-92220-9, Baker, T. P., Bui, A. & Tixeuil, S. (Eds.), pp. 73-88, Springer-Verlag Berlin Heidelberg, DOI: 10.1007/978-3-540-92221-6_7
- [8] Baruah, S. (1995). Fairness in periodic real-time scheduling, *Proceedings of 16th IEEE Real-Time Systems Symposium*, 5-7 December 1995, ISSN 1052-8725, ISBN 0-7803-3199-0, Pisa, Italy, pp. 200-209, IEEE Computer Society Press Los Alamos, California, DOI: 10.1109/REAL.1995.495210
- [9] Anderson, J. & Srinivasan, A. (2000). Early-release fair scheduling, *Proceedings 12th Euromicro Conference on Real-Time Systems*, 19-21 June 2000, ISSN 1068-3070, ISBN 0-7695-0734-4, Stockholm, Sweden, pp. 35-43, IEEE Computer Society Washington, DC, USA, DOI: 10.1109/EMRTS.2000.853990
- [10] Cho, H.; Ravindran, B. & Jensen, D. (2006). An optimal real-time scheduling algorithm for multiprocessors, *Proceedings of 27th IEEE Real-Time Systems Symposium*, 5-8 December 2006, ISSN 1052-8725, ISBN 0-7695-2761-2, Rio de Janeiro, Brazil, pp.101-110, IEEE Computer Society Washington, DC, USA, DOI: 10.1109/RTSS.2006.10
- [11] Sha, L.; Rajkumar, R. & Lehoczky, J. P. (1990). Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, Vol. 39, No 9, September 1990, pp. 1175–1185, ISSN 0018-9340
- [12] Danilov, M. (2001). Methods of task scheduling in real-time systems. *Software Products and Systems*, No. 4, December 2001, pp. 28-35, ISSN 0236-235x (in Russian)
- [13] Baranov, S. & Nikiforov, V. (2015). Transitive task priority inheritance in real-time multi-task applications. *SPIIRAS Proceedings*, 2015, No. 6(43), pp. 114-134, ISSN 2078-9599 (in Russian)
- [14] Nikiforov, V. & Danilov, M. (2000). Static processing of real-time software system specifications. *Software Products and Systems*, No. 4, December 2000, pp. 13–19, ISSN 2036-235x (in Russian)
- [15] Nikiforov, V. (2014). A protocol for prevention of task mutual blocking in real-time systems. *Journal of Instrument Engineering*, Vol. 57, No. 12, December 2014, pp.21-27, ISSN 0021-3454 (in Russian)
- [16] Nikiforov, V. & Pavlov, V. (2011). Structured models for analysis of multitasking software systems. *Information-Measuring and Control Systems*, Vol. 9, No. 9, pp.19-29, ISSN 2070-0814 (in Russian)