Buletinul Ştiinţific al Universităţii Politehnica Timişoara

TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Volume 60(74), Issue 1, 2015

INVERTA – Specification of Real-Time Scheduling Algorithms

V. Stangaciu[1] , O. Datcu[2], M. Micea[3], V. Cretu[4]

Abstract – This paper describes how the scheduling algorithms for real time applications can be specified formally and the development of a simulator that verifies if a set of tasks for a real time application can be scheduled with an existing scheduling algorithm or with an algorithm defined by the user. This simulator is part of the integrated visual environment for designing and analysing real-time applications called INVERTA.
Keywords: scheduling, real-time, simulator

I. INTRODUCTION

Embedded systems and digital signal processing (DSP) systems are used in a variety of application today. Such applications include: automotive control, nuclear plant surveillance, flight control systems, and industrial mechatronics. These systems usually run hard real-time tasks, for which the violation of their time requirements (deadlines), may have catastrophic impacts, thus special task scheduling policies must be used. This class of hard real time scheduling policies must provide schedulability tests which state if a certain set of tasks is feasible or not. If a set of task is feasible with a certain algorithm there is a guarantee that no deadline is missed. Thus, these algorithms have been and still are, heavily analyzed [1, 2].

OPEN-HARTS (Operating Environment for Hard Real-Time Systems) is a methodology that was introduced recently for development and implementation of hard real-time systems and applications and is based on signals and tasks. This system is represented by the interconnection of two sub-systems: one for analysis of the task set called INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development) and one for running the task set called HARETICK (Hard Real-Time Compact Kernel).

The paper has the following structure which will be further described: problem statement, theoretical foundations, related work, proposed solution and research methodology, implementation, experimental results, contribution and conclusions.

II. PROBLEM STATEMENT

INVERTA allows the building, specification and visual display of real-time applications, designed as a set of tasks of different types, each task having a characteristic set of parameters (including parameters of time) and a set of control links with other tasks of the application.

The INVERTA sub-system which is presented in this paper, along with HARETICK (Hard Real-Time Compact Kernel) sub-system is part of OPEN-HARTS (Operating Environment for Hard Real-Time Systems) system. The role of the INVERTA sub-system is to take the running context of the current application from the HARETICK module, to analyse the application, to modify its parameters and to send the modified application back to it.

Most scheduling simulators do not offer the possibility to simulate a customized real time scheduling algorithm. This is a drawback because users that propose new algorithms cannot test them to see if they are feasible or not. Another disadvantage of some of the existing scheduling simulators is that they are not optimized to work for high number of tasks.

III. THEORETICAL FOUNDATIONS

A real time system is defined by J.S. Ostroff as: "*A real-time system (RTS) is any system in which the time at which the output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.*" [3]

Real time system can be divided in the following: critical RTS (not meeting the deadline can result in a catastrophe), strict RTS (not meeting the deadline results in a wrong behaviour of the system), and soft

[1] Faculty of Automation and Computers, Dept. of Computer and Software Engineering Bd. V. Parvan 2, 300223 Timisoara, Romania, e-mail valys@dsplabs.cs.upt.ro
[2] Faculty of Automation and Computers, Dept. of Computer and Software Engineering Bd. V. Parvan 2, 300223 Timisoara, Romania, e-mail olivia.datcu@gmail.com
[3] Faculty of Automation and Computers, Dept. of Computer and Software Engineering Bd. V. Parvan 2, 300223 Timisoara, Romania, e-mail mihai.micea@cs.upt.ro
[4] Faculty of Automation and Computers, Dept. of Computer and Software Engineering Bd. V. Parvan 2, 300223 Timisoara, Romania, e-mail vladimir.cretu@cs.upt.ro

RTS (not meeting the deadline results in the loss of the system's value and of the quality provided by the system).

Tasks scheduling refers to finding reliable solutions for the processor's assignment, for each tasks, in a way in which there is no overlapping in their execution while the system operates.[4]

Taking into consideration if they admit or not interruptions, the scheduling algorithms can be classified as follows: preemptive (the execution of a task can be interrupted by a task with a higher priority) and non-preemptive (the execution of a task cannot be interrupted).

Off-line non-preemptive scheduling techniques provide solutions to hard real-time constraints and predictability, which are important demands in critical applications. On the other hand, these scheduling techniques do not provide flexibility, as online scheduling techniques like the ones that rely on task prioritization (RM, EDF, LLF and others).

A scheduler is the part of a system that deals with the operation of scheduling a task set. In order to find a valid schedule for a task set the scheduler executes a schedulability test. The scheduler can be preemetive if the execution of a task can be interrupted by another task and non-preemtive if no interruption is allowed.

Fig. 1 presents a real-time scheduler [5]. As it can been seen in Fig. 1, the scheduling algorithm needs the task set and the resource management protocol to apply the schedulability test, for a given system architecture, and give an answer if the task set can be scheduled or not.

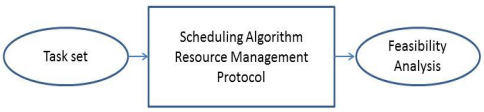

Fig. 1. Real-time scheduler

IV. RELATED WORK

Liu and Layland [6] showed that RM is the best fixed priority algorithm to be used in a uniprocessor system. They proved that a task set that is not schedulable by RM it cannot be scheduled by any other fixed priority scheme. They were the first authors who provided a necessity condition for a set of n periodic tasks under RM, based on the processor utilization factor U (1) and an upper bound bn (2), both defined below:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}, \tag{1}$$

where, Ci represents the computation time of task i and Ti represents the period of the same task i.

$$b_n = n(2^{1/n} - 1) \tag{2}$$

The condition is that if the processor utilization factor is greater than bn, then the set of tasks is not schedulable by RM. This condition was improved by Bini in [7] where the Hyperbolic Bound (HB) improves the acceptance ratio by a factor of $\sqrt{2}$ for large n, compared with the Liu and Layland test. According to HB method, a set of periodic tasks is schedulable by RM if condition (3) is satisfied:

$$\prod_{i=1}^{n} (U_i + 1) \le 2 \tag{3}$$

In [8] a sufficiency test is provided for the same RM algorithm. The task set is proven to be schedulable if the utilization factor is smaller or equal to:

$$U \le n(\sqrt[n]{2} - 1) \tag{4}$$

The first formulation of the Rate Monotonic Analysis was done by Lehoczky in [9]. The goal of the article was to present an exact characterization of the ability of the rate monotonic algorithm to meet the deadlines for a set of period tasks. The article also includes a stochastic analysis of the performance of the algorithm when the task sets are generated randomly. Manabe and Aoyagi improved this article in [10] by reducing the number of points where the time demand has to be checked. Another improvement was done by Bini and Buttazzo [11], who proposed a way to trade complexity versus accuracy of the RM feasibility tests.

In [5] Chen presents an overview of the existing real-time scheduling tool-kits. These tools are useful for real-time system designers and programmers to verify if a task set is schedulable with a scheduling algorithm. Chen divides these scheduling tool-kits based on their functionality in the following categories: simulators, simulation languages and frameworks.

A drawback of the simulators is that they have all the functionality predefined and the user cannot add new code. Among the developed simulators there are: GAST [12], DET/SAT/SIM, PERTS SAT, DTRESS/PERTSSim, AFTER, Brux, CAISARTS, and Scheduler 1-2-3.

A simulation language called STRESS was proposed in [13]. Although STRESS is a good tool to evaluate scheduling algorithms and can be used to design new algorithms, the cost of a context switch is considered to be zero, a task can only start on a tick of the system clock and resources are limited to semaphores. Asserts (A Software Simulation Environment for Real-Time Systems) [14] is another simulation language which is focused on distributed and heterogeneous systems. The user can define nonstandard systems by specifying the task body in pseudo-code.

Frameworks take into consideration the user requirements and the possibility of extension. A framework is able to generate, compile and the run code based on the user specification of a simulation environment, scheduler, resource management

protocols, and task set. A framework of the Oregon State University, which is implemented in C++ was presented in Chen's study from [12]. Another framework that targets failure analysis and hierarchical scheduling was described by Matthew Francis Storch in [15].

Cheddar [16] is another framework, which was implemented in Ada language, and allows the user to check if a real time application meets its temporal constraints. The purpose for creating this framework was mainly educational. This framework can connect to other tools such as editors, design tool and simulations, easily because the data sent to the framework and received by the framework is in XML format.

V. PROBLEM STATEMENT

This paper defines a meta-language for the INVERTA environment, which has the ability to model numerous schedulers (executives). The simulation will be based on scripts that will be translated into simulation parameters and interpreted by the simulation engine.

The general architecture of the simulator described in this paper is presented in Fig 2. The simulator was developed as a plugin for INVERTA application. As it can been seen in the figure, the simulator plugin receives as an input a configuration for a task set and an XML file in which the scheduling algorithm is specified. INVERTA environment is used to describe the configuration of the task set. The XML specification file is generated by the Formal Specification plugin from INVERTA. This plugin offers a User Interface where the scheduling algorithm can be defined in an XML format.

Fig. 3 illustrates the structure of the XML file used for describing the scheduling algorithm. The XML file is composed of five tags. The first one is the *ScheduleName*, in which the name for the scheduling algorithm is entered. The second tag, *Acronym*, identifies the acronym used for the algorithm. The value from this tag is optional. The next tag, *DeclarePriority*, describes the type of the scheduling algorithm: static, dynamic or special. The forth tag, *DeclarePreemtiveBehavior*, specifies if the algorithm is preemptive or non-preemptive. The condition for priority assignment is defined in the last tag, called *PriorityAssignement*.

In order to evaluate the expression that defines the priority assignment for a scheduling algorithm, the expression is first split into atoms, which are stored in a list of atoms. An atom can be an operator, a numeric constant or a task parameter. Based on the literature review, a set of task parameters were identified:

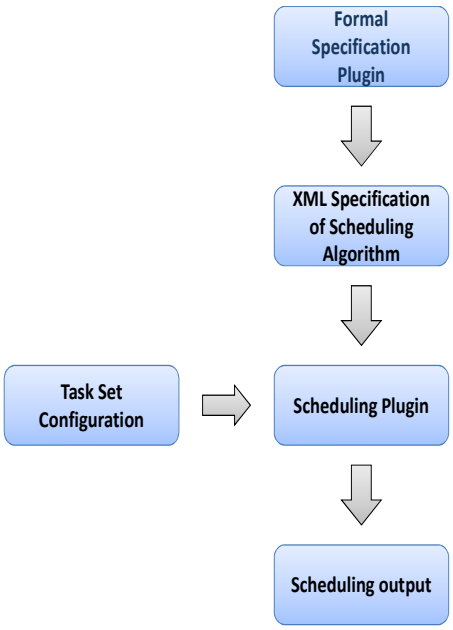

Fig. 2. *General architecture of Task Simulator*

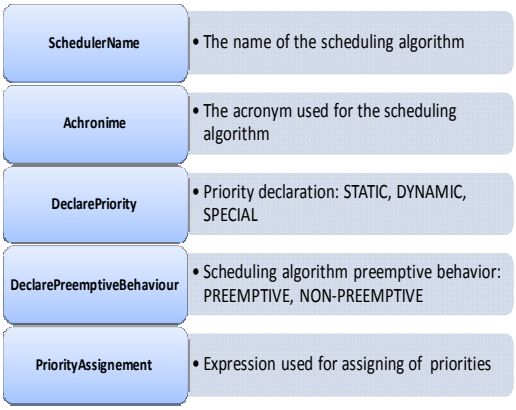

Fig. 3. *XML Specification file structure*

- *T[i]* - The task relative period
- *D[i]* - The task relative deadline
- *C[i]* - The task computation time
- *P[i]* - The task priority
- *S[i]* - The task start time inside current period
- *d[i]* - The task absolute deadline
- *s[i]* - The task absolute start time

In the next step, the expression is transformed in Reversed Polish Notation. From this notation the binary evaluation tree was constructed. The result of the expression is obtained from the in-order traversal of the tree. The above steps are presented in Fig. 4.

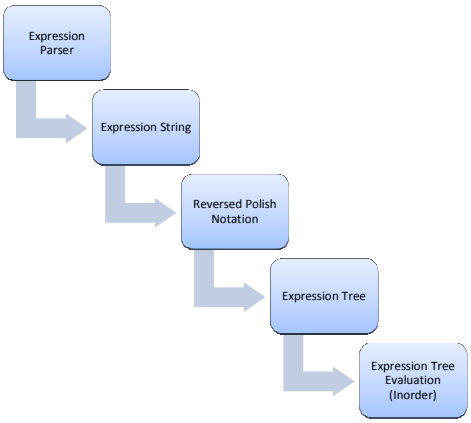

Fig. 4. *Expression evaluation steps*

The list of atoms is iterated in order to verify each atom. If an atom is a number, it is added to the Reversed Polish Notation list. If the atom is an operator and the stack is empty the atom is pushed on the stack. If the stack is not empty, the precedence of the current atom is compared with the precedence of the atom from the top of the stack, and a specific action is performed based on the precedence. If the atom is a start of parenthesis character the atom is pushed on the stack. On the other hand, if the end of parenthesis is encountered the content of the stack until the start of parenthesis is stored in the output RPN list. The pseudo-code used to specify the RPN list construction algorithm is very similar with C programing language. The reserved words are written in bold and the main operations are listed in italic style:

- isNumber – returns true if an atom is a number and false otherwise
- isOperator – returns true if an atom is an operator and false otherwise
- isStartParan – returns true if the atom is a start of parenthesis character and false otherwise
- isStopParan – returns true if the atom is a stop of parenthesis character and false otherwise
- isStackEmpty – returns true if the sack is empty and false otherwise
- Push – adds an element to the stack
- Pop – removes the element from the top of the stack
- Peek – returns the element from the top of the stack
- Precedence – returns the precedence of the operator given as a parameter
- AddRPNList – adds an element to the Reversed Polish Notation list

Reversed Polish Notation construction algorithm

```
1:  foreach (Atom in AtomList)          do
2:      if isNumber(Atom) do
3:        AddRPNList(Atom)
4:      else if isOperator(Atom) do
5:              if isStackEmpty() do
6:                 Push(Atom)
7:              else if isStartParan(Peek()) do
8:                    Push(Atom)
9:              else if Precedence(Atom) > Precedence(Peek()) do
10:                   Push(Atom)
11:             else
12:                   while (!isStackEmpty() && !isStartParan(Peek())
13:                         && Precedence(Atom) < Precedence(Peek())) do
14:                     TempAtom = Pop()
15:                     AddRPNList(TempAtom)
16:                   end do
17:                   Push(Atom)
18:             end if
19:     else if isStartParan(Atom) do
20:             Push(Atom)
21:     else if isStopParan(Atom) do
22:             while (!isStackEmpty() && !isStartParan(Peek())) do
23:                    TempAtom = Pop()
24:                    AddRPNList(TempAtom)
25:             end do
26:             Pop(Atom)
27:     end if
28:     while (!isStackEmpty()) do
29:              TempAtom = Pop()
30:              AddRPNList(TempAtom)
31:     end do
32: end foreach
```

Fig. 5 *Reversed Polish Notation Construction Algorithm*

VI. EXPERIMENTAL RESULTS

The output of the Scheduling PlugIn from INVERTA for the task set defined in Fig. 6 and scheduled with Rate Monotonic Non-Preemptive, a static algorithm, is presented in Fig. 8. Fig. 7 presents the XML file that specifies the Rate Monotonic Non-Preemptive algorithm.

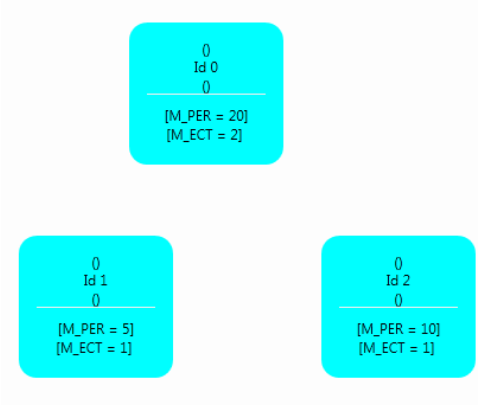

Fig. 6 Task set scheduled with RM algorithm

```
<?xml version="1.0"?>
- <Algorithm>
    <SchedulerName>Rate Monotonic</SchedulerName>
    <Achronime>RM</Achronime>
    <DeclarePriority> Static </DeclarePriority>
    <DeclarePreemptiveBehavior> Non-preemptive </DeclarePreemptiveBehavior>
    <PriorityAssignment> T[i] < T[j] </PriorityAssignment>
  </Algorithm>
```

Fig. 7 *XML specification for RM algorithm*

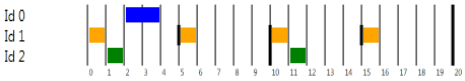

Fig. 8 *RM scheduling example*

The output of the Scheduling PlugIn from INVERTA for the task set defined in Fig. 9 and planned with MLFNP - Minimum Laxity First Non-Preemptive, a dynamic algorithm, is presented in Fig. 11. The task set from Fig. 9 was taken from the example that was treated in [1] for MLFNP algorithm. Fig. 10 presents the XML file that specifies the MLNFNP algorithm.

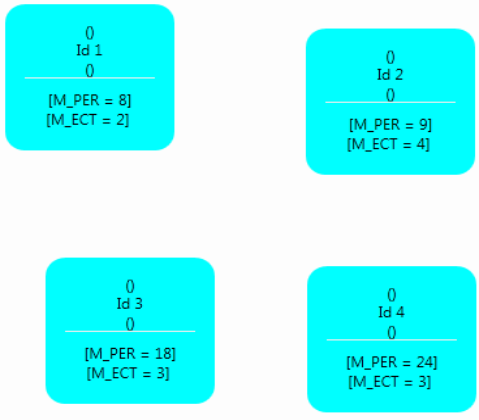

Fig. 9 *Task set scheduled with MLFNP algorithm*

```
<?xml version="1.0"?>
- <Algorithm>
    <SchedulerName>Minimum Laxity First Non-Preemtive</SchedulerName>
    <Achronime>MLFNP</Achronime>
    <DeclarePriority> Dynamic </DeclarePriority>
    <DeclarePreemptiveBehavior> Non-preemptive </DeclarePreemptiveBehavior>
    <PriorityAssignment> T[i] - D[i] < T[j] - D[j] </PriorityAssignment>
  </Algorithm>
```

Fig. 10 *XML specification for MLFNP algorithm*

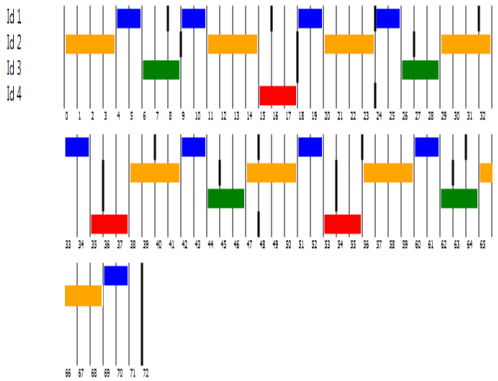

Fig. 11 *MLFNP scheduling example*

VII. CONCLUSION

The development of real-time systems remains a very important research domain because of the complexity of the problems which characterize these systems. Task scheduling is one of the most important problems from real-time systems and without which the function of the system would be unfeasible. This fact is supported by the tremendous number of research papers from this domain which treat different types of scheduling algorithms. INVERTA environment is intended to help users define real-time applications in a visual user friendly environment, analyse these applications from the feasibility point of view and simulate existing and custom defined scheduling algorithms.

ACKNOWLEDGMENT

This work was partially supported by the strategic grant POSDRU/159/1.5/S/137070 (2014) of the Ministry of National Education, Romania, co-financed by the European Social Fund – Investing in People, within the Sectoral Operational Programme Human Resources Development 2007-2013.

REFERENCES

[1] S. Baruah, M. Bertogna, and G. Buttazzo, "A Review of Selected Results on Uniprocessors," in *Multiprocessor Scheduling for Real-Time Systems*, ed: Springer International Publishing, 2015, ISBN: 978-3-319-08695-8, pp. 29-33.

[2] Yan Feng Zhai and Feng Xiang Zhang, "A Review of Sufficient Schedulability Analysis for Fixed Priority Scheduling Systems," *Applied Mechanics and Materials,* vol. 741, no. 1, pp. 856-859 2015.

[3] J. S. Ostroff, "Formal methods for the specification and design of real-time safety critical systems," *J. Syst. Softw.,* vol. 18, no. 1, pp. 33-60, 1992.

[4] M. V. Micea, "Proiectarea si implementarea sistemelor timp-real pentru aplicatii critice de achizitie si prelucrare numerica de semnal," PhD, Politehnica Timisoara, 2004.

[5] J. Chen, "Extensions to Fixed Priority with Preemption Threshold and Reservation-Based Scheduling," PhD, University of Waterloo, 2005.

[6] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, 1973.

[7] E. Bini, G. C. Buttazzo, and G. M. Buttazzo, "Rate monotonic analysis: the hyperbolic bound," *Computers, IEEE Transactions on,* vol. 52, no. 7, pp. 933-942, 2003.

[8] R. Devillers, Jo, #235, and l. Goossens, "Liu and Layland's schedulability test revisited," *Inf. Process. Lett.,* vol. 73, no. 5-6, pp. 157-161, 2000.

[9] J. Lehoczky, S. Lui, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Real Time Systems Symposium, 1989., Proceedings.*, 1989, pp. 166-171.

[10] Y. Manabe and S. Aoyagi, "A Feasibility Decision Algorithm for Rate Monotonic andDeadline Monotonic Scheduling," *Real-Time Syst.,* vol. 14, no. 2, pp. 171-181, 1998.

[11] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *Computers, IEEE Transactions on,* vol. 53, no. 11, pp. 1462-1473, 2004.

[12] J. Johnson, "The Impact of Application and Architecture Properties of Real-Time Multiprocessor Scheduling," PhD, CTH Department of Computer Engineering, Computer Architecture Laboratory (CAL), MicroMultiProcessor Group, 1997.

[13] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "STRESS: a simulator for hard real-time systems," *Softw. Pract. Exper.,* vol. 24, no. 6, pp. 543-564, 1994.

[14] K. Ghose, S. Aggarwal, P. Vasek, S. Chandra, A. Raghav, A. Ghosh, and D. R. Vogel, "ASSERTS: a toolkit for real-time software design, development and evaluation," in *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, 1997, pp. 224-232.

[15] M. F. Storch, "A framework for the simulation of complex real-time systems," University of Illinois at Urbana-Champaign, 1997.

[16] F. Singhoff, J. Legrand, L. Nana, L. Marc, and #233, "Cheddar: a flexible real time scheduling framework," presented at the Proceedings of the 2004 annual ACM SIGAda international conference on Ada: The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies, Atlanta, Georgia, USA, 2004.