

Producer-consumer paradigm in real-time applications

Doina Zmaranda^{*}, Gianina Gabor^{**} and Antoniu Nicula^{***}

^{*} Department Computer Science,
University of Oradea, Faculty of Electrical Engineering and Information Technology,
410087 Oradea, Romania, E-Mail: zdoina@uoradea.ro

^{**} Department Computer Science,
University of Oradea, Faculty of Electrical Engineering and Information Technology,
410087 Oradea, Romania, E-Mail: gianina@uoradea.ro

^{***} Department Computer Science,
University of Oradea, Faculty of Electrical Engineering and Information Technology,
410087 Oradea, Romania, E-Mail: niculaantoniu@uoradea.ro

Abstract - One of the most important characteristic of real-time systems is that they must react to external stimuli (inputs) and, in a timely manner, affect the environment in which they operate. Often real-time systems are used in control systems where the input data must be analyzed, computed and afterwards the real-time system must send responses to control the overall system in a pre-defined timely manner.

In this paper a model for constructing real-time application that interacts with external environment is described. The model is based on the producer/consumer paradigm for inter-process communication and provides a framework for defining initial task specifications. Based on this model, the resulting specification of system's tasks could be tested against schedulability using a graphical framework.

Keywords: *producer-consumer real-time system, schedulability, Earliest Deadline First*

I. INTRODUCTION

The important issue regarding real-time computer systems defines them as the class of computer systems that have to perform computations in a pre-defined time frame. Computation includes also possible I/O operations and therefore the time frame generally depends on the computer external environment [1].

Consequently, one of the most important features of a real-time systems is that it interacts with it's external environment: it is not a stand-alone systems, it must respond to external stimuli that may come from a large variety of external devices, such as sensors, and afterwards, it must process the data and responses

must be sent in order to control the overall system based on the computed results [2].

For real-time systems, in addition to functional requirements other issues are equally important such as: timing, reliability, security and fault tolerance requirements. Therefore the development of complex real-time control systems implies extensive analysis of design tradeoffs and efficient and accurate prediction of system's behaviour. Thus, means for specifying real-time systems timeliness requirements are important as well as developing methods for predicting real-time behaviour of a program [3].

This paper tries to develop a practical and flexible method for development and analyzing of hard real-time systems prior to their implementation based on the producer-consumer paradigm.

The method formally models a set of tasks that define the system and verifies if there are enough resources for all tasks to meet their deadlines. This will allow the programmer to easily test if the desired real-time behaviour will be realized at run-time.

II. THE PRODUCER-CONSUMER PARADIGM

A. General issues and related work

This is a standard paradigm in traditional multitasking systems for modelling interaction between two processes: one called as producer and other as consumer. The main issue considered is to synchronize both tasks so that no data is lost, that means that all data produced by the producer are consumed by the consumer. Generally, communication between the producer and the consumer is done using a buffer implemented in different ways (for example - a linked circular list could be used) and the access to this buffer is synchronized using a monitor based on some condition variables.

```

process Producer
{
while (1)
{
produce data;
Buffer.Insert(data);
}
}
process Consumer
{
while (1)
{
data = Buffer.Remove();
consume data;
}
}

```

Because of the implementation of the monitor, the two processes have to wait one after each other when the buffer is full (wait for consumer to consume data) and when the buffer is empty (wait for producer to produce some data). A possible implementation of the monitor could be the following:

```

monitor Buffer
{
bool full, empty;
signal notEmpty, notFull;
void Insert(int data)
{
if (full) wait(notFull);
insert in the buffer;
signal(notEmpty);
}
int Remove()
{
if (empty) wait(notEmpty);
remove from buffer;
signal(notFull);
}
}

```

The above implementation is well known and works correctly in multitasking systems where no real-time requirements exist. We can use also the model for real-time systems starting from the idea that, as it was stated before, in real-time systems software processes interact with processes from external world.

In this approach, interaction between internal/external processes could be modelled as in the producer-consumer scenario by thinking that the external process is the producer (because it provides data collected from external devices) and the consumer is the internal process that make computations based on the data provided from the external process [4]. The relationship between the two processes generally corresponds to producer/consumer paradigm, but, in practice the things are slightly different. There are mainly two reasons for which the above producer-

consumer paradigm is not viable for real-time systems:

- if the producer is an external process, it is not under control of the computer; thus, it may not be possible to force this process to wait for the consumer. In this case, it is important to assure that all data produced by the producer will be consumed by the consumer in the absence of the wait statement from Insert function;

- also, in real-time systems using wait statements as the ones from producer-consumer model is not acceptable. The main problem regarding these wait statements is that they induce non-determinism: practically, it is impossible to assess an upper bound for the waiting time and therefore, it will be further impossible to verify if the process meet its deadline.

Moreover, the above reasons don't cover transient workloads that could overload a system and damage its timeliness or usefulness. For these situations there are some proposed models such as window-constraint scheduling and the firm task model, presented in [5]; If we want the producer-consumer model to be applicable for real-time processes, the wait statement from Insert and Remove functions should be removed. But, in this situation another problem arises: if the consumer is not fast enough when consuming data, some data produced by the producer will be lost. In this case, knowledge of the semantics of the synchronization and the buffering algorithm is not enough to correct the functioning of the system and a temporal component should be included. This temporal component relies at least on knowing the rate at which the producer produces data and on knowing also how much time needs the consumer for computation on received data. If both producer and consumer tasks are periodic, in order to deliver reasonably stable service, an arbitrary rate (period) adjustment for tasks could be useful when possible, because sometimes, as stated in [6], this approach could conflict with hardware characteristics. The proposed model investigates possibilities of period adjustment for a given set of periodic tasks, timing analysis being carried on along an efficient implemented schedulability test.

B. Producer-consumer paradigm for real-time

Based on the issues presented above, we might conceive a model for producer-consumer tasks in real-time systems. The model is based on two processes one being the producer (that correspond to the process from the external environment and produces data, for example from a sensor) and the second one being the consumer (that corresponds to one internal process that computes received data).

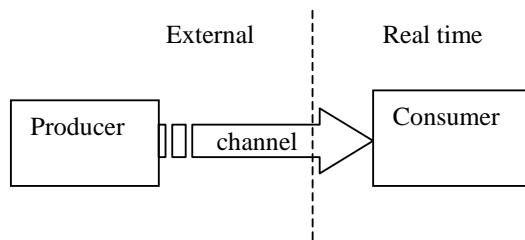


Figure 1: Producer-consumer model for real-time systems

These processes are connected via a unidirectional channel, from producer to consumer, since we assumed that the consumer cannot communicate with the producer or control it. This is a relatively fair assumption, because in most cases the data collected from the sensors are produced independently and asynchronously from the task that uses them (Figure 1).

But, in order to avoid the loss of data, the rate at which the producer produce data is important to be considered. Consequently, if we consider and denote with r the maximum rate at which data is produced (based on values provided from the corresponding sensor) and if this rate is realizable over an arbitrarily long interval of time and if the consumer does not consume the data at rate r , then there is no amount of buffering that should be implemented between producer and consumer. If data is consumed at the rate it is produced, then no buffering and consequently no wait statements are necessary, and no data loss will occur.

The real-time systems producer-consumer model is based on the idea that all consumer tasks must process data in a time frame imposed by its producer. Thus, producer defines a specific discrete time domain and produce data at a fixed “tick” of the discrete clock. Based on this, constraints could be defined over the real-time processing (internal) tasks based on the external world that they are connected with.

C. Applying the paradigm in analysing real-time systems

The conceptual model

Real-time producer-consumer paradigm provides a framework for both expressing processor-dependent computations and for reasoning about real-time behaviour of the programs. At low level, the producer-consumer paradigm for real time specifies an upper bound on the time to process each received

data. At a high level, the paradigm allows general analysing of the behaviour of the system.

In order to apply the producer-consumer paradigm in the analysing process of a real-time system, we consider that all processes from our system are periodic and therefore, the following characteristics for each computing task are considered:

- task identifier ID_i that defines a task uniquely
- task i execution or computing time C_i – the execution time of a task can vary within an interval $[C_{Bi}, C_{wi}]$ where C_{Bi} and C_{wi} are the best respectively the worst case execution times for the task i (in most of the cases Worst Case Execution Time is denoted by $WCET_i$); generally, only the best and the worst execution time of each task are known. When modelling the timing behaviour of tasks, this is a natural approach, because exact computation time of a task cannot be establish. Consequently, both times have to be considered when creating a task behaviour model, and results must be compared; another approach could analyze the timing behaviour starting from the idea of variable execution times. In our model we will rely on WCET.

- task deadline D_i – deadline is a typical task constraint in real-time systems: is the time before which the task must complete its execution. Usually, the deadline of the task is relative, meaning that, from the moment when a task T_i arrives (from its arrival time), it should finish within D_i time units.

- task period T_{iper} – for periodic tasks, task period T_{iper} is considered to be equal to task deadline D_i . Because in most real-time control and monitoring systems the tasks typically arise from sensor data or control loops are periodic, we will consider in our model that all tasks are periodic [7].

- task ready (arrival) time R_i – represents the moment of time when the task T_i is ready for execution.

A periodic job consists of an infinite sequence of requests with periodic ready times, where deadline of the request is the ready time of the succeeded request. Earliest Deadline First algorithm is the optimal algorithm for periodic tasks; Earliest Deadline First is a priority driven algorithm, based on dynamic priorities, in which higher priorities are assigned to the requests that have the earliest deadlines, and a higher priority request always preempts a lower priority one [8].

In our model, the deadline D_i is used to guarantee that communication between producer and consumer adheres to the real-time producer-consumer paradigm. Therefore, if data are sent from the producer (external) task on the communication channel with the worst-case (maximum) rate r , then, according to the model, the consumer task will be

required to complete execution within $1/r$ time units of each invocation:

$$D = \frac{1}{r} \quad (1)$$

where:

- r is the rate at which the data is produced
- D = deadline of task – the time limit in which the data should be consumed.

An execution of the task corresponds to the execution of the code required to consume the data received through the channel. For a given process, estimating the execution time consists of measurement of all operations called by the process as well as the sequential code of the process itself. For the time being, estimations are made manually, although we intend to use some specific tools in the future.

Depending on the scheduling policy but not only on that, the task set could be schedulable under those conditions or not. From the known scheduling algorithms, for real-time systems Earliest Deadline First is preferred; the algorithm is optimal in the sense that it can schedule a set of tasks in such a manner that all imposed deadlines are met whenever it is possible to do so. But, although the used scheduling policy is optimal, it is still possible to have a non-schedulable set of tasks (for these tasks cannot be guaranteed that all deadlines are met).

Generally, the necessary condition for achieve schedulability using Earliest Deadline First algorithm was given by Liu & Layland [9] for a medium performance:

$$\sum_{i=1}^n \frac{C_i}{T_{iper}} \leq 1 \quad (2)$$

where:

- C_i represents the Worst Case Execution Time for task i , C_i - WCET _{i}
- T_{iper} represents the period of task i

But, in our model, which take into consideration periodic tasks, the period of task is assumed to be equal with the task deadline, so $T_{iper} = D_i$ and, by considering (1), we can say that a set of tasks can be scheduled on a processor using Earliest Deadline First scheduling algorithm, if:

$$\sum_{i=1}^n C_i r_i \leq 1 \quad (3)$$

The product $C_i r_i$ is the fraction of the processor that must be allocated to process data by task T_i , the relation from (3) stipulating the condition that the processor not to be overloaded. By evaluating this

condition for the given set, a general schedulability assessment could be met.

Analyzing the model by simulation

The rates and implied deadlines and execution times are further tested against schedulability using a specific developed framework that permits modelling, simulation and schedulability analysis for real-time systems tasks. The framework is divided into two parts: a specification part and an analysis part (Figure 2) and it is implemented in C++.

In the specification part, user models the analyzed real-time system, by given all initial data about the set of tasks. The framework provides a graphical user interface for specifying system's (tasks) initial characteristics and a simulator for simulation of tasks execution. In the analysis part, the given real-time system is analyzed from the schedulability point of view. The framework implements the most used scheduling algorithms, grouped into two categories: periodic and aperiodic.

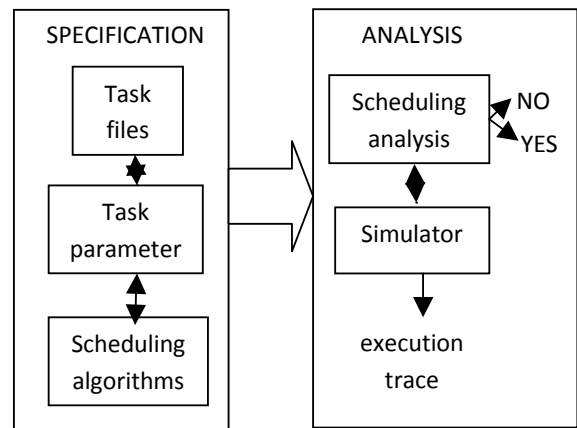


Figure 2: The framework structure

The main features of the analysis tool are the following:

- a graphical user interface for introducing tasks parameters, such as: deadline, execution time, priority; these parameters are saved in a (text) file for each given task, in a specific format
- several scheduling algorithms are implemented, and the user could choose from a list of implemented algorithms. These algorithms are grouped into two categories: for periodic and non-periodic tasks
- a simulator, that shows a graphical representation of the generated trace of execution for the set of tasks, according to the chosen scheduling algorithm. Using this simulation, the points when

tasks arrive, when they are suspended or completed could be easily observed.

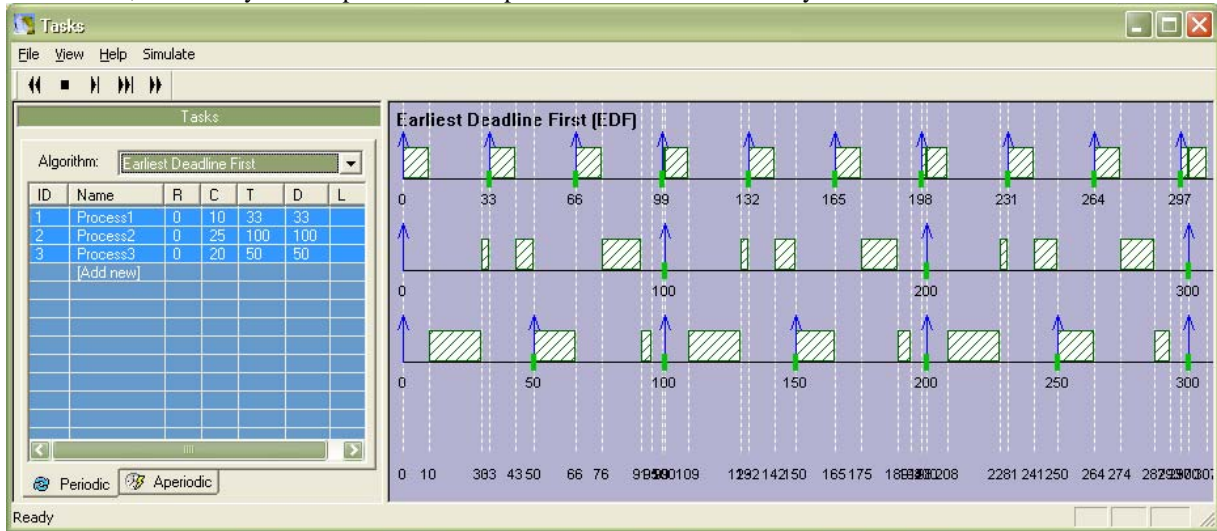


Figure 3: Tasks scheduled using Earliest Deadline First algorithm

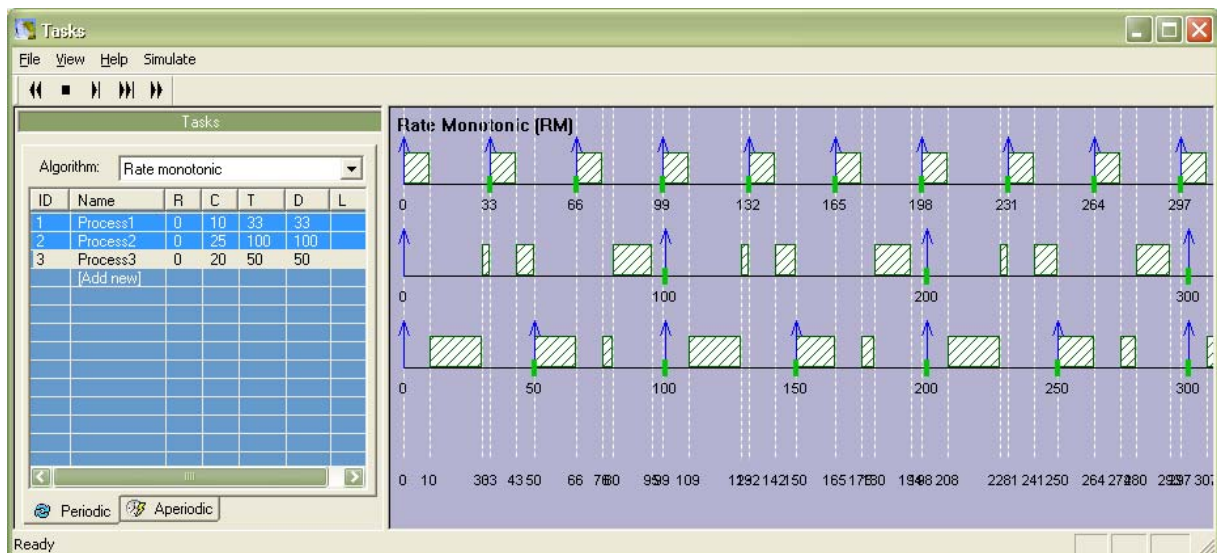


Figure 4: Tasks scheduled using Rate Monotonic algorithm

- also, when a tasks miss its deadline (for a non-schedulable set of tasks), this situation is marked in the simulation view using a red cross. By simulation user can validate the dynamic behaviour of the system and see how tasks are executed according to their given parameters and scheduling policy

- schedulability analysis implies checking if all tasks meet their deadlines; a message to the user is displayed, that indicates is system is schedulable or not.

Given the framework described above, framework that has been presented and used in other papers [10][11], we consider as an example the following set

of tasks and their characteristics (see Table 1):

- task T_1 - responsible for update the display with real-time information that are read from specific sensors; for this, an update rate of 30 updates per second is acceptable, that means a rate r of $1000\text{ms}/30 \Rightarrow r = 1/33\text{ms} \Rightarrow D_1 = 33\text{ms}$;

- task T_2 - responsible for alarm condition verification; for this, an update rate of 10 updates per second is acceptable, that means a rate r of $1000\text{ms}/10 \Rightarrow r = 1/100\text{ms} \Rightarrow D_2 = 100\text{ms}$;

- task T_3 - responsible for statistic computation; for this, an update rate of 20 updates per second is

acceptable, that means a rate r of $1000\text{ms}/20 \Rightarrow r = 1/50\text{ms} \Rightarrow D_3 = 50\text{ms}$;

TABLE 1: Task characteristics

ID	Name	R	C	D
1	Proces1 (T_1)	0	10	33
2	Proces2 (T_2)	0	25	100
3	Proces3 (T_3)	0	20	50

Verifying the relation from (3) we obtain:

$$\sum_{i=1}^3 C_i r_i = \sum_{i=1}^3 \frac{C_i}{T} = \frac{10}{33} + \frac{25}{100} + \frac{20}{50} = \frac{3145}{3300} = 0.9530 \leq 1$$

After modelling the set of tasks using Earliest Deadline First algorithm, the result from Figure 3 can be obtained.

If Earliest Deadline First algorithm is used, the scheduling policy is optimal, but is it still possible, that a particular set of tasks not to be schedulable. Rate Monotonic algorithm could be also used when possible; as it is shown in Figure 4, our example of set of tasks is also schedulable using Rate Monotonic algorithm even if this solution is not optimal [7].

This graphical framework present not only that the task set is schedulable, but also gives a pretty good idea about how the tasks will be executed. Simulation could be carried step by step or fully simulation could be released.

Analysis based on producer-consumer paradigm enables assessment of schedulability of the task set that was constructed based on initial conditions derived from real-time producer-consumer model. Also, because the rates are given as constants in the model, then the schedulability conditions could be used to derive maximum input rates that can be sustained by a set of processes.

III. CONCLUSIONS

In this paper, a conceptual model based on producer-consumer paradigm is developed for real-time tasks. In this approach the process interaction is modelled as a producer-consumer system with a time constraint on the rate at which the consumer “consumes” the data received from the producer.

The model provides issues for expressing processor dependent computation and for reasoning about real-time behavioural of the program. The conceptual model is then simulated using specific developed

framework that permits modelling, simulation and schedulability analysis for real-time systems tasks. Analysis by simulation is carried on based mainly on Rate monotonic and Earliest Deadline First scheduling algorithms ant permits assessment of schedulability of the task set that was constructed based on initial conditions derived from real-time producer-consumer model.

REFERENCES

- [1] J. Liu, “*Real-Time Systems*”, Prentice Hall, 2000
- [2] W. Svrcek, D. Mahoney, B. Young, “A Real-Time Approach to Process Control”, John Wiley & Sons, 2000
- [3] Xu, J.: “*On Inspection and Verification of Software with Timing Requirements*”, IEEE Transactions on Software Engineering, vol. 29, no. 8, 2003
- [4] K. Jeffay, “*The Real-Time Producer-Consumer Paradigm: A Paradigm for the construction of Efficient, Predictable real-Time Systems*”, Proc. Of the ACM/SIGAPP Symposium of Applied Computing, pp. 796-804, 1993
- [5] A.K. Mok, W. Wang, „*Window-Constrained Real-Time Periodic Task Scheduling*”, Proceedings of the 22nd IEEE Real-Time Systems Symposium 7 Audsey, 2001
- [6] L., Chang, „*Event-Driven Scheduling for Dynamic Workload Scaling in Uniprocessor Embedded Systems*”, Proceedings of the 2006 ACM symposium on Applied computing, pp. 1462- 1466 , ISBN:1-59593-108-2, 2006
- [7] A. Burns, R. Davis, K. Tindell and A. Wellings, “*Fixed Priority Preemptive Scheduling: An Historical Perspective*”, Real Time Systems, vol. 8, pp. 173-198, 1995.
- [8] Stankovic, J. and Ramamritham, K.: “*Deadline Scheduling for Real-time systems: EDF and Related Algorithms*”, Kluwer Academic Publishers (1998)
- [9] C. L. Liu and J. W. Layland, “*Scheduling Algorithms for Multiprogramming in a Hard real Time Environment*”, Journal of the ACM, vol. 20(1), pp. 46-61, 1973
- [10] D. Zmaranda, C. Rusu and M. Gligor, „*A Framework for Modeling and Evaluating Timing Behaviour for Real-Time Systems*”, proc. of SINTES, An International Symposium on Systems Theory – Software Engineering, vol III, pp. 514-520, ISBN 973-742-148-5, 2005
- [11] D. Zmaranda, G. Gabor, „*Tool for Modeling and Simulation of Real-Time Systems Behavior*”, Proc. of SOFA 2007, 2nd IEEE International Workshop on Soft Computing Applications, Gyula, Hungary - Oradea, Romania, ISBN: 978-1-4244-1608-0, pp. 211-215, 2007