

ANALYSIS OF TRANSIENT FAULT TOLERANCE IN HARD REAL-TIME SYSTEMS WITH TIME REDUNDANCY

UDC 007.52 621.397.132 629.073

Sandra Došić, Milun Jevtić

Department of Electronics, University of Niš, Faculty of Electronic Engineering, Serbia
E-mail: (sandra.djosic, milun.jevtic)@elfak.ni.ac.rs

Abstract. *In this paper we analyze timing constrains of one fault tolerant hard real-time system with time redundancy. Our goal is to analyze the possibility of overcoming transient faults detected during tasks executions. We created and presented in the paper a program which can help us estimate a probability of overcoming a transient fault in a real-time system. We also created a program for visual presentation of execution sequences for all tasks in a real-time system with a rate monotonic scheduling algorithm. With both created programs we can do different analysis with real-time systems and also see graphical model of tasks scheduling.*

Key words: *Real-time systems, Fault tolerance, Time redundancy*

1. INTRODUCTION

Systems for control and monitoring play an important role in many areas of daily life: robotics, cosmic research, automotive industry, process control, factory automation... Those systems have been designed in order to be safe and extremely reliable. They are usually realized as real time systems with the ability to tolerate some faults, [1].

A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed, [2]. The classical conception is that in a hard real-time system the completion of an operation after its deadline is considered useless which ultimately may cause a critical failure of the complete system. A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., omitting frames while displaying a video).

One of the goals during real-time system designing process is to create a predictable real-time system. The analysis of real-time system's timing constrains is fundamental for the design of such systems. Designing a predictable real-time system is easier with the assumption that there is no fault during system execution. However, this fault-free assumption is in fact not realistic because "non-faulty systems hardly exist, there are only

systems which may have not yet failed", [3]. So, if a fault occurs during real-time task execution, then it is necessary to overcome that fault and satisfy all timing constraints of real-time tasks.

The focus of our research is predictable fault tolerant hard real-time systems. For such systems it is very important to know: how big their probability to tolerate faults is? We created a program which can help us answer that question. The created program, on the basis of timing characteristics of real-time tasks, gives us the value for minimum time between two consecutive faults which real-time system can tolerate. Due to the result of the analysis we can conclude how fault tolerant one real-time system is.

In our research we usually use a basic algorithm for scheduling real-time tasks. One of them is rate monotonic (RM) algorithm [4], [5] for which we created a program for graphical illustration of tasks scheduling in time. With this program, execution sequences for all real-time tasks in one real-time system, according to rate monotonic scheduling algorithm, can be seen.

2. SOFTWARE REALIZATION OF RESPONSE TIME ANALYSIS

2.1 Response time analysis

One of the goals of our research is to design predictable hard real-time systems. Response time analysis is one approach that has successfully been used to achieve this goal. More about that analysis can be found in [6] and the basis of response time analysis is defined as:

$$R_i(T_E) = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(T_E)}{T_j} \right\rceil C_j + \left\lceil \frac{R_i(T_E)}{T_E} \right\rceil \max_{\tau_k \in hpe(i)} \overline{C}_k \quad (1)$$

We use response time analysis for set $\Gamma = \{\tau_1, \dots, \tau_n\}$ of n real-time tasks, called primary tasks, that must be scheduled by the system in the absence of faults. Any primary task τ_i , in a set Γ , has a period T_i , a deadline D_i ($D_i \leq T_i$), and a worst-case execution time, C_i . Each primary task τ_i can have some alternative tasks $\overline{\tau}_i$ associated with it, [7]. Each alternative task represents some extra processing that is necessary to recover a task from a given faulty state caused by a fault. Any alternative task has a worst-case execution time, called worst-case recovery time, \overline{C}_i .

We also consider n different priority levels (1, 2, ..., n), where 1 is the lowest priority level. We denote the priority of primary task τ_i and alternative tasks $\overline{\tau}_i$ as p_i and \overline{p}_i , respectively. We also assume in the analysis that there is a minimum time between two consecutive fault occurrences, T_E .

The input parameters of this analysis are:

- the task attributes (T_i, D_i, C_i and \overline{C}_i),
- the primary task priorities (p_i) and
- the assumed value of T_E .

The priorities of alternative tasks are assumed to be the same as their primary tasks $\overline{p}_i = p_i$.

If there are no faults in the system then the worst-case response time of task τ_i is the time necessary to execute τ_i and all tasks τ_j so that $p_j > p_i$. If there are faults in the system, in the calculation of the worst-case response time of τ_i the time necessary to recover the faulty task has to be included. We use time redundancy for systems recovering after faults, [8].

Since R_i appears on both sides of (1), the solution can be obtained iteratively by forming a recurrence relation with $R_i^0 = C_i$. This iterative procedure finishes either when $R_i^{m+1} = R_i^m$ (the worst-case response time of τ_i is found) or when $R_i^{m+1} > D_i$ (τ_i is considered unschedulable).

Fig. 1 illustrates response time analysis and possible scenarios of real-time tasks scheduling with different assumed values of T_E .

The first scenario, Fig. 1(a) presents scheduling of two periodic real-time tasks τ_1 and τ_2 when there is no fault in the system. A system of these two tasks are schedulable i.e. both tasks are executed before their deadlines, D_1 and D_2 .

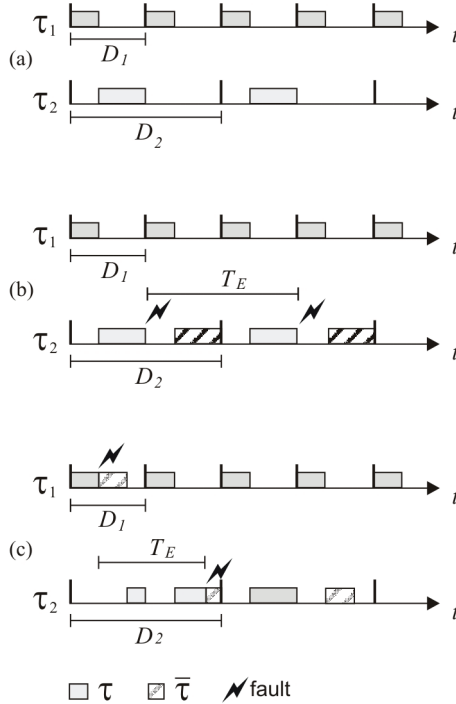


Fig. 1 Illustration of possible real-time tasks schedule when:
 (a) there is no fault; (b) value for T_E is long enough and real-time system is fault tolerant; (c) value for T_E is not long enough that real-time system stays fault tolerant

Fig. 1(b) presents scheduling of the same real-time tasks τ_1 and τ_2 when two faults occur in the system. The time between two consecutive faults T_E is long enough and real-time system can tolerate these faults. The first fault occurs just a little bit before the end of

tasks τ_2 execution. Real-time system overcomes this fault by executing task τ_2 again or executing alternative tasks with less or equal execution time as task τ_2 . The second fault occurs again just a little bit before the end of tasks τ_2 execution. Time redundancy is enough to tolerate this fault. As before, when the first fault occurs, the system overcomes the fault by executing task τ_2 again or executing some alternative tasks.

Fig. 1(c) presents scheduling of the same real-time tasks τ_1 and τ_2 when two faults occur in the system. Now, time between two consecutive faults T_E is not long enough and real-time system cannot tolerate these faults. The first fault occurs just a little before the end of tasks τ_1 execution. Real-time system can overcome this fault by executing task τ_1 again or executing alternative tasks with less or equal execution time as task τ_1 . In this case, the second fault occurs just a little before the end of tasks τ_2 execution. Now time redundancy is not enough to tolerate this fault. The second task τ_2 starts but its timing characteristics cannot be satisfied and τ_2 missing its deadline. This is not acceptable in one hard real-time system, so in this case real-time system is not fault tolerant.

2.2 The Algorithm

Based on (1) we realized an algorithm for the analysis of real-time systems timing constraints shown in Fig. 2.

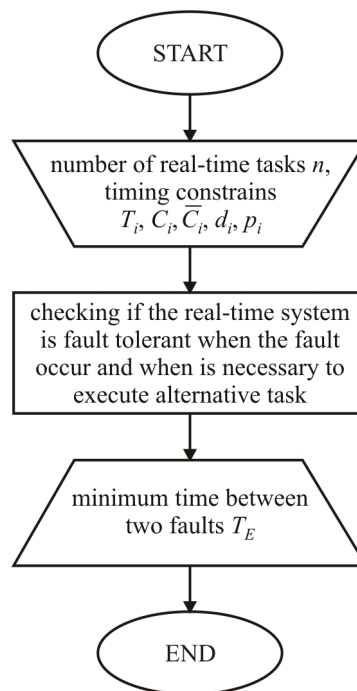


Fig. 2 Algorithm for analysis of RTS timing constrains

Input data are a number of real-time tasks n , task period T_i , worst-case execution time C_i , worst-case recovery time \bar{C}_i , task deadline d_i and task priority p_i . For these parameters an algorithm has to check if the real-time system is fault tolerant. We considered that a fault can occur during tasks execution and that it is necessary to execute some recovery tasks to overcome the faults. The goal of the algorithm is to find minimum time between two consecutive faults which a real-time system can tolerate.

Fig. 3 shows a more detailed algorithm for the analysis of real-time systems constrains.

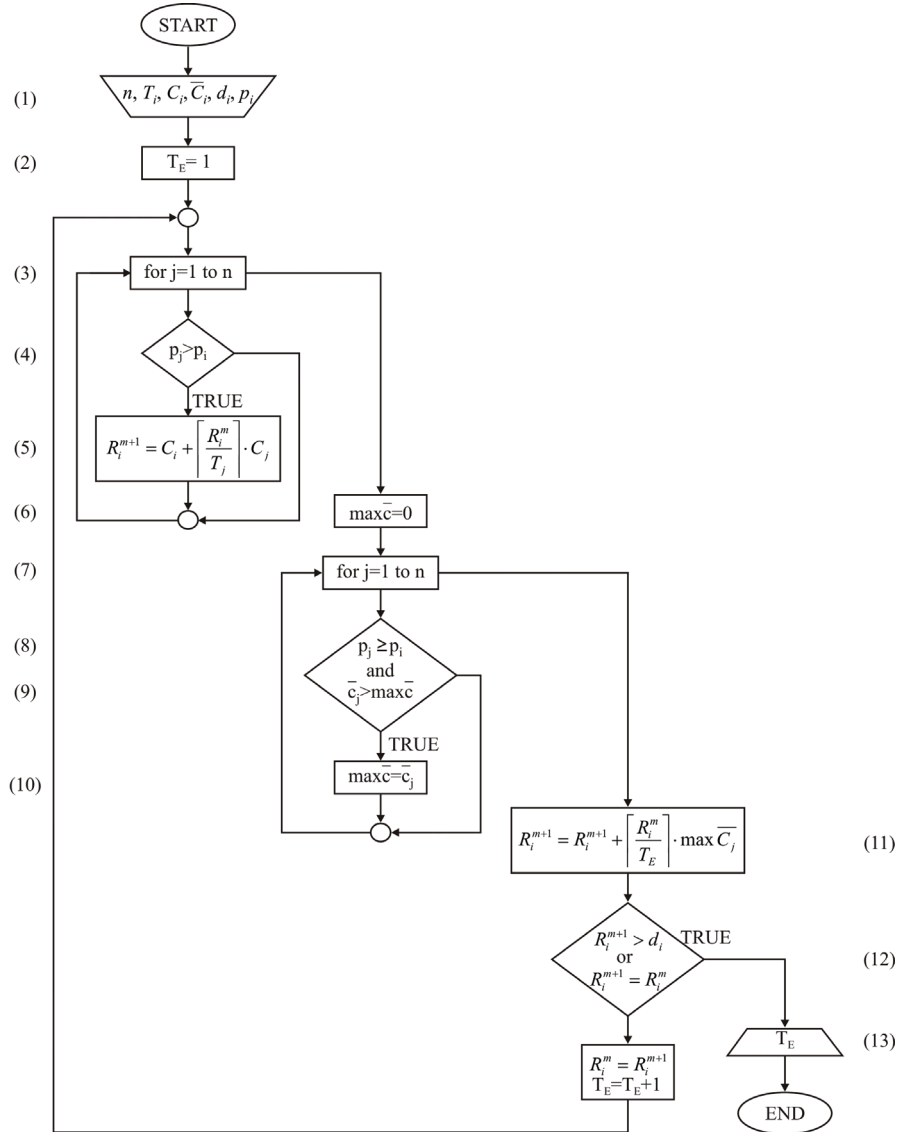


Fig. 3 More detailed algorithm for analysis of RTS timing constrains

Input data for the above algorithm are a number of real-time tasks n , task period T_i , worst-case execution time C_i , worst-case recovery time \overline{C}_i , task deadline d_i and task priority p_i , step (1) in Fig. 3. In the beginning of the analysis we assume that minimum time between two consecutive fault occurrence is $T_E = 1$ step (2) in Fig. 3.

In the first algorithm loop, step (3), step (4) and step (5) in Fig. 3, we calculate the first and the second addend of (1). In this loop only a task with higher priority then priority of task τ_i are important for us.

The second loop in algorithm, step (6) to (10) in Fig. 3, describes a process of finding maximum worst-case recovery time from the tasks with equal or higher priority then priority of task τ_i .

Step (11) in Fig. 3 calculates the worst-case response time R_i for task τ_i . According to (1) this process is iterative and it finishes either when $R_i^{m+1} > d_i$ (τ_i is considered unschedulable) or when $R_i^{m+1} = R_i^m$ (the worst-case response time of τ_i is found), step (12) in Fig. 3. If the condition step (12) is true then we have output result T_E step (13) in Fig. 3. If the condition step (12) is false then we must increase T_E and continue iterative process until it is necessary.

Using algorithm shown in Fig. 3 we wrote code and generated .exe file "AlgFix.exe" which could be started from the command line with the command:

```
ALGFIX [<input_file>] [<output_file>].
```

As it can be seen from the above command, the name of the input and output file could be written optionally. If the names for the input and output file are not written then their standard names "AlgFix Input.txt" and "AlgFix Output.txt" can be used.

Input file is .txt format with parameters separated by space. In the first line, the number of real-time tasks n should be written. After that in the next n line we have to specify timing characteristics of n real-time tasks: period T_i , worst-case execution time C_i , worst-case recovery time \overline{C}_i , deadline d_i and task priority p_i .

The output file is also .txt format with parameters separated by space. The first line is the required result T_E - minimum time between two consecutive faults which real-time system can tolerate. In the next n line are parameters R_i^m and R_i^{m+1} for each of n real-tasks.

2.3 Results of software realization

In order to prove the correctness of the realized algorithm and the whole program, we have done a number of tests and two of them are shown in Fig. 4.

Fig. 4(a) presents an input and output file for case I of three real-time tasks scheduling according to a rate monotonic algorithm.

Timing characteristics for these three tasks are shown in Table 1 and the inputs file "AlgFix Input" in Fig. 4(a). For these parameters, we started our program for analyses. The program considers that faults can occur in real-time system during the task execution and that the system recovers executing the task again. Therefore, for this case the worst-case recovery time is equal to worst-case execution time, $C_i = \overline{C}_i$.

The output file "AlgFix Output.txt" shows results of timing analyses. From that file, we can see that real-time systems can tolerate minimum time between two consecutive fault occurrences of 11 time units. For $T_E = 11$ parameters are $R_1(11) = 4$, $R_2(11) = 8$ and $R_3(11) = 22$ and this is also shown in the output file. For all three tasks we got that $R_i(11)$

$< d_i$, for $i = 1, 2$ and 3 , which means that all tasks finished before their deadlines. It can be concluded that the system is schedulable.



Fig. 4 Input and output file of realized software for
 (a) case I - system recovers executing task again
 (b) case II - system recovers executing alternative task

For $T_E = 10$ parameters are $R_1(10) = 4$, $R_2(10) = 8$ and $R_3(10) = 32$ and they are also shown in output file "AlgFix Output.txt". For task τ_3 we got that $R_3(10) > d_3$ which means that this task breaches its deadline, so the whole system is not schedulable.

Table 1 Real-time tasks timing characteristics - case I

Task	Task characteristics				
	T_i	C_i	\overline{C}_i	d_i	p_i
τ_1	13	2	2	13	3
τ_2	25	3	3	25	2
τ_3	30	5	5	30	1

Table 2 presents manually obtained results for the same input parameters. If we compare the output file "AlgFix Output.txt" and Table 2, it can be concluded that we got the same results, manually and software obtained.

Table 2 Results of timing analyses for case I

Task	$R_i(11)$	$R_i(10)$
τ_1	4	4
τ_2	8	8
τ_3	22	32

The second case presents a real-time system which tolerates fault executing some alternative tasks. Usually those tasks have less worst-case execution time than primary tasks, i.e. $\overline{C}_i < C_i$. This case is shown in Fig. 4(b).

Fig. 4(b) presents input and output file for case II of three real-time tasks scheduling also according to rate monotonic algorithm. Timing characteristics for these three tasks are shown in Table 3 and inputs file "AlgFix Input" in Fig. 4(b). For these parameters, we started our program for analyses. The program considers that faults can occur in real-time system during task execution and that the system recovers executing alternative tasks whose worst-case recovery time is less than worst-case execution time of primary task, $\overline{C}_i < C_i$.

Table 3 Real-time tasks timing characteristics – case II

Task	Task characteristics				
	T_i	C_i	\overline{C}_i	d_i	p_i
τ_1	13	2	1	13	3
τ_2	25	3	2	25	2
τ_3	30	5	3	30	1

The output file "AlgFix Output.txt" shows results of timing analyses. From that file, we can see that real-time systems can tolerate minimum time between two consecutive fault occurrences of 6 time units. For $T_E = 6$ parameters are $R_1(6) = 3$, $R_2(6) = 9$ and $R_3(6) = 24$ and this is also shown in the output file. For all three tasks we got that $R_i(6) < d_i$, for $i = 1, 2$ and 3 , which means that all tasks finished before their deadlines. It can be concluded that the system is schedulable.

For $T_E = 5$ parameters are $R_1(5) = 3$, $R_2(5) = 9$ and $R_3(5) = 35$ and they are also shown in output file "AlgFix Output.txt". For task τ_3 we got that $R_3(5) > d_3$ which means that this task breaches its deadline, so the whole system is not schedulable.

Table 4 Results of timing analyses for case II

Task	$R_i(6)$	$R_i(5)$
τ_1	3	3
τ_2	9	9
τ_3	24	35

Table 4 presents manually obtained results for the same input parameters. If we compare the output file "AlgFix Output.txt" and Table 4, it can be concluded that we got the same results, manually and software obtained.

3. GRAPHICAL ILLUSTRATION OF RATE MONOTONIC SCHEDULING ALGORITHM

3.1 The algorithm for graphical illustration

As stated in the previous section, rate monotonic (RM) algorithm has been used for real-time tasks scheduling. Our next step was to create software for the simulation of the rate monotonic algorithm. Fig. 5 shows an algorithm for graphical illustration of tasks scheduling in one real-time system if RM algorithm is active.

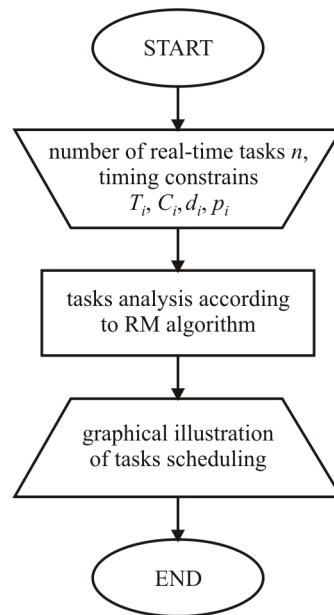


Fig. 5 Algorithm for graphical illustration of tasks scheduling according to rate monotonic algorithm

We realized the algorithm shown in Fig. 5 as a two part job, shown in Fig. 6.

The first part is the realization of the rate monotonic algorithm. The input data is a number of real-time tasks n and real-time tasks timing constrains: task period T_i , worst-case execution time C_i , task deadline d_i and task priority p_i . As can be seen in Fig. 6, the first part output data is the input data for the second part. They are parameters which describe how real-time tasks are scheduled according to RM algorithm. The second part has to give us graphical illustration of tasks scheduling i.e. execution sequences in time.

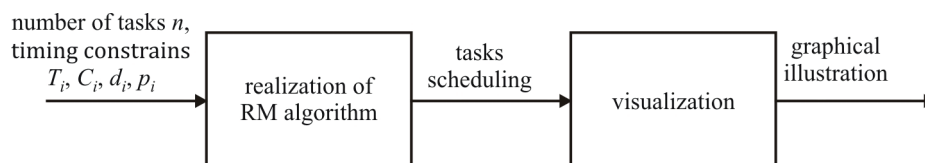


Fig. 6 Scheme of algorithm realization

3.2 Software Realization of Rate Monotonic Algorithm

A detailed algorithm for the realization of the rate monotonic algorithm is shown in Fig. 7.

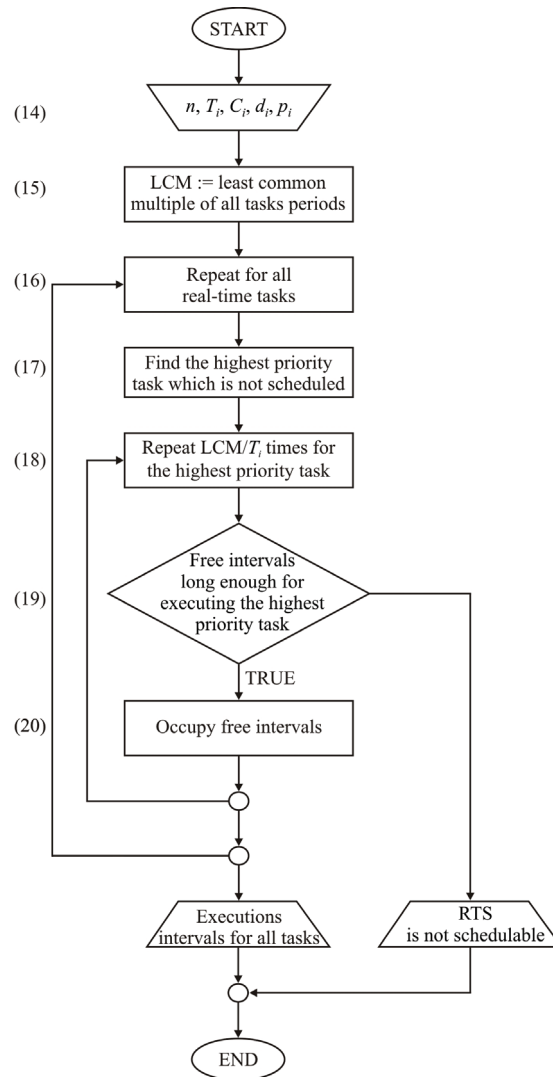


Fig. 7 Algorithm for realization of rate monotonic algorithm

This algorithm can be explained with one simple example of three real-time tasks τ_1 , τ_2 and τ_3 . Let's suppose that all tasks periods are equal to their deadlines $T_i = d_i$ and that all tasks are ready for execution at the same time. The rest real-time tasks timing characteristics are shown in Table 5.

Table 5 Real-time tasks timing characteristics - case III

Task	d_i	C_i
τ_1	10	2
τ_2	20	3
τ_3	30	5

According to RM algorithm task τ_1 has the highest priority and tasks τ_3 the lowest, i.e. $p_1 > p_2 > p_3$. Input data for the shown algorithm is a number of real-time tasks n , task period T_i , worst-case execution time C_i , task deadline d_i and task priority p_i , step (14) in Fig. 7. In the beginning the least common multiple (LCM) for all tasks period algorithm has to be calculated, step (15). In our case LCM is 60. Now, the focus is only on period of 60 time units because after that period tasks scheduling is repeated.

The first loop, step (16) is executed as many times as the number of real-time tasks n is. In our case the first loop will be executed three times. Then, the highest priority task has to be found, step (17). In our case this is task τ_1 .

The second loop, step (18) is executed as many times as the number of task release time of the highest priority task in LCM is. In our case the highest priority task is τ_1 with a period of 10 time units. In time interval of LCM=60 time units τ_1 will execute 6 times. Task release times are $t=10, t=20, t=30, t=40$ and $t=50$. The second loop will execute 6 times. In step (19) algorithm checks if in LCM there are free intervals long enough for executing the highest priority task. If there are, task takes that free intervals, step (20), and if there are not then the system is not schedulable. In our case this is possible and task τ_1 will execute as is shown in Fig. 8.

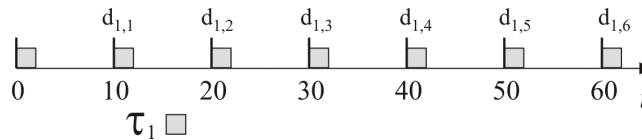


Fig. 8. Execution sequence for task τ_1

Steps from (16) to (20) are repeated for the next task by priority. In our case the next task is τ_2 . The first release time for task τ_2 is the same as for task τ_1 and it is $t=0$. As it can be seen from Fig. 8, time interval from 0 to 2 is not free and time interval from 2 to 10 is free. The execution time for task τ_2 is 3 time units, so τ_2 will execute from $t=2$ to $t=5$. The task period for τ_2 is 20 time units so in LCM task τ_2 will execute 3 times. The next task release times are $t=22$ and $t=42$ as shown in Fig. 9.

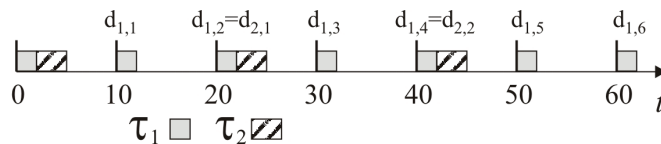


Fig. 9 Execution sequence for task τ_1 and τ_2

The last task in our simple example is τ_3 . Task period for τ_3 is 30 time units, so in LCM task τ_3 will execute 2 times. The first release time for task τ_3 is the same as for tasks τ_1 and τ_2 and it is $t=0$. As it can be seen from Fig. 9, time interval from 0 to 5 is not free. The first free appropriate interval is from 5 to 10. The execution time for task τ_3 is 5 time units, so the first execution for task τ_3 will be from $t=5$ to $t=10$. The second release time for τ_3 is in $t=30$. Time interval from $t=30$ to $t=32$ is not free, so the second execution of task τ_3 will be from $t=32$ to $t=37$. Execution sequence for all three tasks τ_1 , τ_2 and τ_3 is shown in Fig. 10.

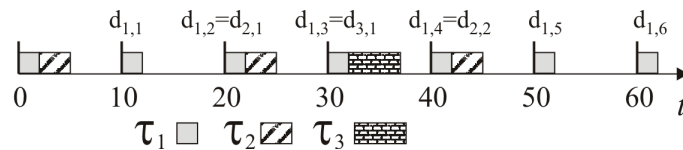


Fig. 10 Execution sequence for task τ_1 , τ_2 and τ_3

In our example all three tasks are schedulable according to RM algorithm.

Using the algorithm shown in Fig. 7 we wrote the code and generated the .exe file "AlgRM.exe" which can be started from the command line with the command:

ALGRM [<input_file>] [<output_file>].

From the above command, it can be seen that the name of the input and output file can be written optionally. If there are no names for the input and output files then their standard name "AlgRM Input.txt" and "ProcVis.txt" can be used.

The input file is .txt format with parameters separated with spaces. In the first line, the number of real-time tasks n should be written. After that, in the next n line we have to specify timing characteristics of n real-time tasks: deadline d_i , worst-case execution time C_i and release time r_i . Fig. 11 shows the input file for our simple example.

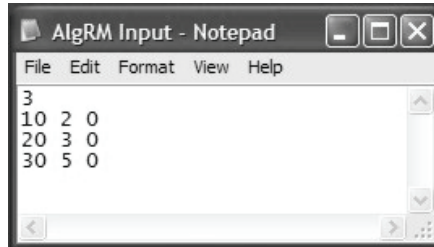
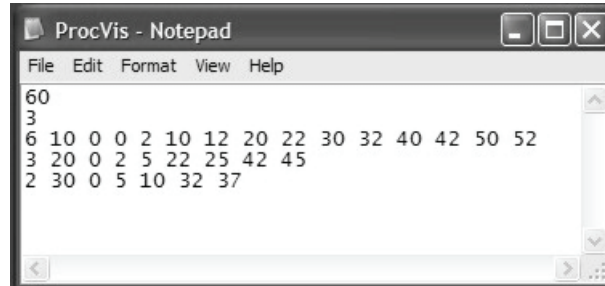


Fig. 11 The Input file for AlgRM software

The output file is also .txt format with parameters separated by spaces. If the system is schedulable then the value for LCM is in the first line of the output file and the number n of real-time tasks is in the second line. In the next n line are values for:

- number of task execution in LCM period of time,
- task period T_i ,
- the first release time for task τ_i - r_{i1} ,
- start time and end time of each task executions.

For our simple example the output file is shown in Fig. 12.



```

60
3
6 10 0 0 2 10 12 20 22 30 32 40 42 50 52
3 20 0 2 5 22 25 42 45
2 30 0 5 10 32 37

```

Fig. 12 The output file for AlgRM software

It can be seen from the output file that LCM is 60 time units and that number of real-time tasks is 3. The values in the third line are for task τ_1 . The number of task executions is 6 and execution intervals are [0, 2], [10, 12], [20, 22], [30, 32], [40, 42] and [50, 52]. Similar conclusions can be made from the second and the third line for tasks τ_2 and τ_3 .

If the tasks of one real-time system are not schedulable (according to RM algorithm), then the output file shows the message „Real-time system is not schedulable“.

3.3 Visualization of tasks scheduling

Program ProcVis was created for graphical illustration of tasks scheduling according to RM algorithm. The output AlgRM file was used as an input file for ProcVis program. ProcVis program has to use parameters from AlgRM output file and to draw execution sequences of real-time tasks.

Fig. 13 shows ProcVis output file for our simple example.

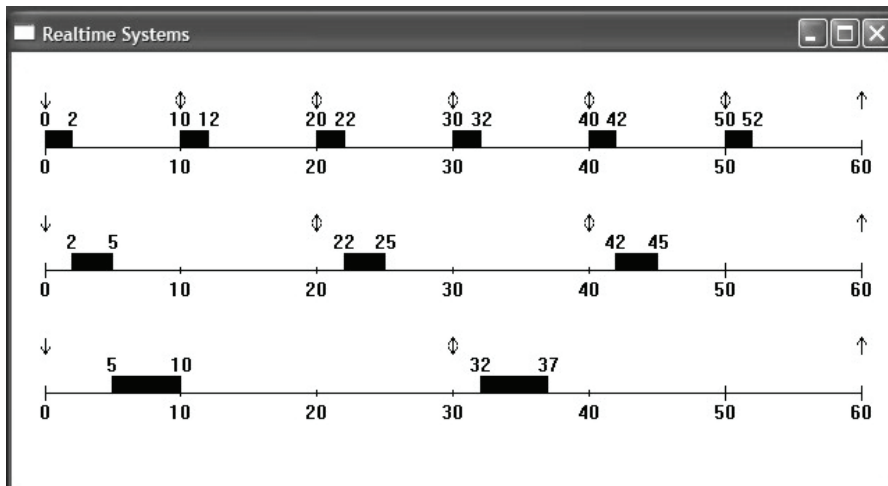


Fig. 13 The output file for ProcVis program

4. CONCLUSION

The paper presents realized software which can be used for estimating the possibility of overcoming transient faults in one process control hard real-time system using time redundancy. We presented two programs, both created in order to help us deal with real-time systems.

The first one can help us analyze timing constraints of real-time tasks in one real-time system. We considered that these tasks are scheduled according to rate monotonic algorithm and that faults can occur during tasks execution. In that situation real-time system recovers from faults executing task again (case I) or executing some alternative tasks (case II). In both cases, we use time redundancy for systems recovery after faults. For these two cases, we have done a number of tests and have proved the correctness of the realized algorithm and the whole program.

We specially presented two cases of tree real-time tasks whose input parameters are almost equal, the only difference is the value for the worst-case recovery time. Case I presents real-time system which recovers from faults executing task again, so $C_i = \overline{C}_i$. Case II presents real-time system which recovers from faults executing some alternative tasks whose worst-case recovery time is less than tasks worst-case recovery time, i.e. $\overline{C}_i < C_i$. It can be concluded that the shorter worst-case recovery time the shorter minimum time between two consecutive fault occurrences which systems can tolerate. This reduction of parameter T_E indicates increasing real-time fault tolerance, which is good.

The first realized program offers the possibility of analyzing timing constraints of multiple real-time tasks very fast, much faster than when manually obtained. From the output program result, we also got information about the minimum time between two consecutive fault occurrences that systems can tolerate. This is important information from which we can conclude how fault tolerant a real-time system is.

The second realized program gave us a possibility to simulate rate monotonic scheduling algorithm. This algorithm, as one of the basic algorithms, is usually used as scheduling algorithm in RTS. In our research we also use RM algorithm very often and we used it as scheduling algorithm in the presented cases in the paper. The program is realized as two part project. The first part is realization of rate monotonic algorithm and the second part is realization of graphical illustration for tasks scheduling.

With this realized program we can actually see on the screen the execution sequences for all tasks in one real-time system when RM scheduling algorithm is active. This visual way of presentation is definitely much easier for understanding. In our case the adage "A picture is worth a thousand words" is definitely true.

Both created programs help us analyze real-time systems (especially for fault tolerances) and see graphical model of tasks scheduling.

REFERENCES

1. M. Jevtić, M. Cvetković and S. Brankov, "Task Execution in Real-Time Systems for Industrial Control and Monitoring", Electronics, Faculty of Electrical Engineering University of Banjaluka, vol. 6, no. 2, pp. 56-61, december 2002.
2. N. Nisanke, Realtime Systems, Prentice Hall, 1997.
3. J. C. Laprie, Dependability: Basic Concepts and Terminology, Springer-Verlag, 1992.

4. F. Cottet, J. Delacroix and Z. Mammari, Scheduling in Real-Time Systems, John Wiley & Sons, 2002.
5. K. Juvva, Real-Time Systems, Carnegie Mellon University, 18-849b Dependable Embedded Systems, or http://www.ece.cmu.edu/~koopman/des_s99/real_time/index.html
6. G. Lima and A. Burns, "An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems", IEEE Transaction on Computers, Vol. 52, No. 10, pp. 1332-1346, October 2003.
7. B. Johnson, Design Analysis of Fault-Tolerant Digital Systems, Addison-Wesley Publishing Company, 1988.
8. S. Đošić and M. Jevtić, "Scheduling in RTS Using Time Redundancy for System Recovery After Faults", Proceedings of papers, Indel 2004, Banja Luka, pp. 146-149, November 2004.

ANALIZA TOLERANCIJE PROLAZNIH OTKAZA U HRTS-U SA VREMENSKOM REDUNDANSOM

Sandra Đošić, Milun Jevtić

Ovaj rad se bavi analizom vremenskih karakteristika sistema koji rade u realnom vremenu (RTS) i koji u određenoj meri imaju sposobnost tolerisanja prolaznih otkaza. U radu su prikazana dva realizovana alata za analizu rada RTS-a. Prvi alat omogućava procenu kolika je mogućnost prevazilaženja prolaznih otkaza u jednom upravljačko procesnom RTS-u sa rigidnim vremenskim ograničenjima (HRTS) korišćenjem vremenske redundanse. Drugi omogućava simulaciju rada rate monotonic algoritma i vizuelnu prezentaciju toka izvršavanja zadataka jednog RTS-a.

Ključne reči: *Sistemi za rad u realnom vremenu, Tolerisanja otkaza, Vremenska redundansa.*