Redundant Free Efficient Task Execution System

Sridhar N.S.
Computer science Engg.
Trivandrum

Akarsh Venugopal
Computer science Engg.
Chennai

ABSTRACT
In this paper, we will discuss about the key features to consider while designing for a fault tolerant system. Different models are being used in different applications such as space, air traffic control, nuclear power plant etc.

A real time system must be reliable if a failure to meet its timing requirements may endanger human life, damage equipment etc. Fault tolerant system improves reliability by incorporating rating redundancy into the system design. This system is called the 'existing system'. Current space missions deal with such system. We have introduced another system called as the proposed model which deals with efficiency in terms of performance by removing the duplicity and dividing the total number of tasks among the different processors with a term called as 'Load Balancing' .So the actual mission can be completed before the time taken by the existing system to complete a mission.

General Terms
A Fault Tolerant System basically deals with safe and efficient execution of the different tasks used in On Board Computers used in Space application.

Keywords

- FTS- Fault Tolerant Systems
- RFETES- Redundant Free Efficient Task Execution system
- Processor Switching and Load Sharing
- Dual Redundant Processors.

1. INTRODUCTION
FAULT TOLERANCE

- Error Detection

It can be Ideal check, which is determined solely from the specification, but check should be independent from system and fails if system crashes

It can be acceptable check where rate of change of system is reasonably monitored and powered up diagnostics done.

- Damage confinement

Any error occurring many propagate and spread. We should thus identify boundaries for the same dynamically or by static means like firewall.

- Error recovery

Can be backward recovery where the state is restored to a earlier state by making checkpoints. This is most frequently used but suffers from recovery overhead.

Can be forward recovery where we need accurate assessment of damage. We try to make the state error-free. It is highly application dependent.

- Fault treatment

If any transient fault occurs, the system is restarted to go to error-free state else the system is repaired online without any manual intervention and by dynamic system reconfiguration..

.

2. DESIGNING ISSUES
Before deciding what type of tasking should be done or what scheduling to be approached , first thing to be done for any system is:-

1. Define a computational model

a) Now our main focus is how to divide the tasking among the two-processor systems say A and B among which the tasks are divided.

b) One executive which handles the two processors and monitors them as they perform their actions and decides what measures to take if any fault occurs at any point during the execution of a task in a processor and lead to normal functioning of the system. The term used here is GRACEFUL DEGRADATION.

2. Identify the tasks and their reliability level

a) Set out which tasks to be assigned to which processor and setting up deadline for their execution which is the most critical part when it comes to fault tolerant computation.

b) Checking how reliable the system is with respect to efficiency of executing tasks, weight (number of processors it uses) and other aspects as well.

3. Task allocation and prioritization

a) Number of tasks needed to be allocated to each processor i.e. suppose we have 10 tasks in total, we can have 5->A and 5->B

b) Prioritizing the tasks involves segregating the tasks into critical and non-critical tasks where all critical tasks are given greater priorities than the non-critical tasks and are grouped together.

4. Carrying out the execution and finishing them before the deadline

a) Carrying out the execution of tasks especially the critical ones on which the system depends .If any fault occurs then the executive must recover procedure without affecting the execution.

b) Most importantly the execution of the tasks must finish before the deadline so that the system can have some safety margin.

Greater the safety margin, greater is program efficiency so more complexities can be added to the tasks.

Based on these, two algorithms are produced for two different systems. The primary algorithm provides security and backup management facilities but it reduces system performance by duplicating the same algorithm for different processors, which is been mastered, by the secondary algorithm but the only problem that exists seems to be about reliability in case of failure since none of the tasks are duplicated.

Now, numerous algorithms for scheduling real-time tasks exist. Broadly speaking, however, they can be classified as follows:-

a) STATIC scheduling algorithms require the programmer to define the entire schedule prior to execution. At run time , this pre-determined schedule is then used to guide a simple task dispatcher. Cyclic executives are one way to program static task scheduling.

b) DYNAMIC scheduling algorithms make decisions about which task to execute at run time, based on the priorities of the task invocations in the ready queue.

They require a more complex run-time dispatcher or scheduler.

Also, Scheduling theory usually assumes tasks are of three types, characterized by the arrival pattern of their individual invocations.

a) PERIODIC tasks consist of an infinite sequence of identical invocations which arrive at fixed intervals .Their arrival pattern is thus time driven.

b) APERIODIC tasks consist of a sequence of invocations, which arrive randomly, usually in response to some external triggering event .Their arrival pattern is thus event driven.

c) SPORADIC tasks are a special case of aperiodic ones where there is a known worst-case arrival rate for the task, i.e., they have a fixed minimum inter arrival.

Here, we are applying the 'STATIC – PERIODIC' scheduling for both the following models.

3. THE TWO MODELS

3.1 Existing Model

The system currently in use is the "Dual Redundant Fault Tolerant System." This system deals with having different processors among which redundant tasks are distributed i.e. If suppose we have 10 tasks to be executed in processor A at one stretch, we introduce the method of redundancy by getting the same 10 tasks into the processor B. So we have the same set of tasks getting executed in both processors.

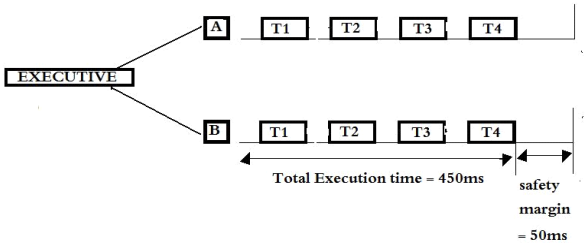


Fig 1

In Fig 1 we have two processors A and B.Tasks T1,T2,T3,T4 are the tasks which are distributed in both the processors but both the processors have the same set of tasks which are also called redundant tasks .

The 'EXECUTIVE' or The Scheduler(scheduler algorithm) is another part of this real time system which orders assignment of CPU and the resources to the tasks. The function of scheduling algorithm is to determine, for a given task set, a sequence of task step executions (a schedule).In particular, we are interested by scheduling algorithms which give(if any) a schedule for executing the tasks such that their timing, precedence and resource constraints are satisfied.

In Fig1 we see that both the processors are given a deadline of 450ms and thetotal time is set as 500ms. So we get a safety margin of 50ms.

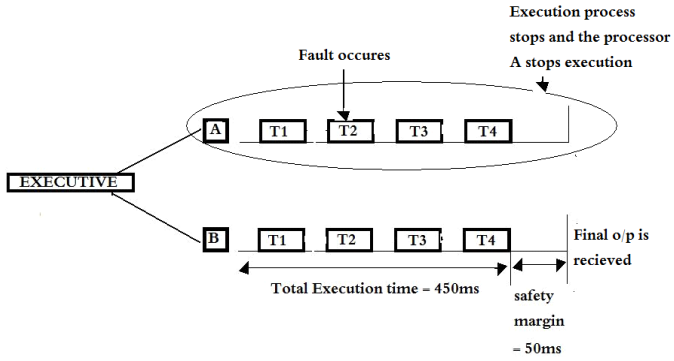


Fig 2

In Fig 2 we do the analysis of the performance of present system. Suppose we find a fault at execution of task T2 in processor A and T2 is a critical task, No problem , the present system has an advantage of task redundancy often known as software redundancy which causes the same task T2 to get executed in processor B. So what happens is the processor A which had detected the fault is temporarily shut down and execution continues in processor B which ultimately gives the final output. This process of creating hardware and software redundancy causes the safe execution of tasks through graceful degradation.

In this model we use Primary/backup copies (PB) and triple modular redundancy (TMR), which are two basic methods that allow multiple copies of a task to be scheduled on different processors.

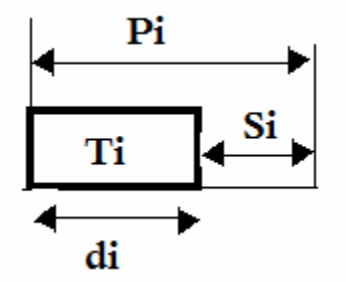
The scheduling algorithm that we use to implement this system is (RM) Rate Monotonic Algorithm. because we are considering static elements here where efficiency is directly proportional to the frequency rate at which tasks get executed. So in such cases RM is better than any other algorithms like FCFS, Round Robin etc.

The basic idea here is to assign different and fixed priorities to tasks with different execution rates, highest priority being assigned to task with highest frequency (lowest execution time) and lowest priority to lowest frequency task. (highest execution time). RM algorithms can schedule a set of tasks to meet their

deadlines if total resource utilisation is lower than the maximum CPU utilisation.

Algorithm:-

```
function docalc()
{
1: arrival: Time;
……
2: arrival: = Clock; -- first arrival time
3: loop -- loop forever
4: delay until arrival; -- suspend task until arrival time
5: 'action'; -- code for task invocation
6: -- deadline: arrival + D                    (\\D is the deadline)
7: arrival: = arrival + T; -- determine next arrival time  (\\T is the inter arrival time (or period))
8: end loop;
}
Task body SYSTEM is
Begin
        Loop
                Accept CLOCK_ENTRY;
                GETIN;
Docalc();
                PUTOUT;
        End loop;
End SYSTEM;
```



Safety margin = Si = Pi – di

In this model we give a specific safety margin for each task and cumulative of this Si has to be less than the deadline i.e.:- for a set of 4 tasks:-

$$\sum_{i=1}^{4} Si < 450$$

Limitation:-

The present model poses a Disadvantage of multiple redundancies, which adds to high power consumption, incomplete use of resources, loss of ½ of computing power & performance degradation mainly due to low safety margin and increased bus bandwidth. Due to fact that we are using multiple processors to perform a particular set of tasks mainly in space programs where a failure in machine could lead to catastrophic consequences with huge monitory losses, there is a need to have about 5 to 6 processors, each having same set of tasks .This adds to weight of the space craft. That is a big disadvantage.

So to remove that disadvantage, we have introduced a new model which could be used in near future. As if for now there is no such fixed structure for this system but it could vary according to need of the mission.

3.2 Proposed Model

We call this RFETES (redundant free efficient task execution system) . Salient features:-

- Processor Switching
- Load sharing
- Efficient performance

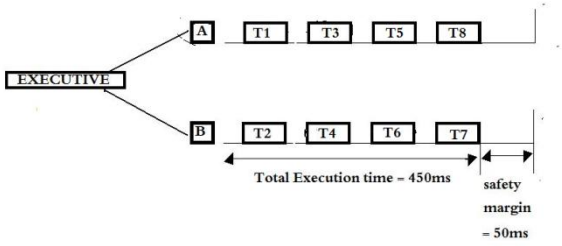


Fig 3

In Fig 3, we have processor A which is assigned 4 tasks T1,T3,T5,T8 and processor B with 4 tasks T2,T4,T6,T7 and both are given the deadline of 450ms.

We task T1, T3, T6 as critical tasks and rest as non-critical tasks .So our priority becomes the execution of the critical tasks at any stage. This model deals with switching between the processors A, B. After execution of T1 in A, B. After execution of T1 in A , the execution goes to T4 in B as T4 is dependent on T1 and so on. The execution continues in the same fashion. This switching process increases the system performance and leads to maximum resource utilization.

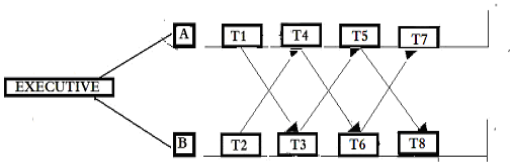


Fig 4

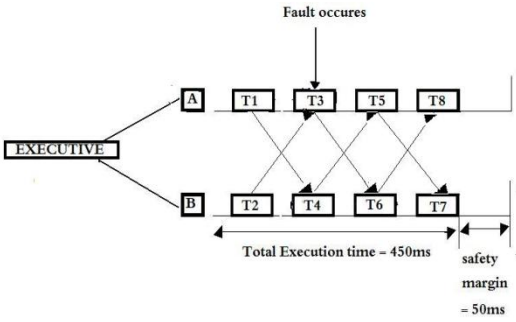


Fig 5

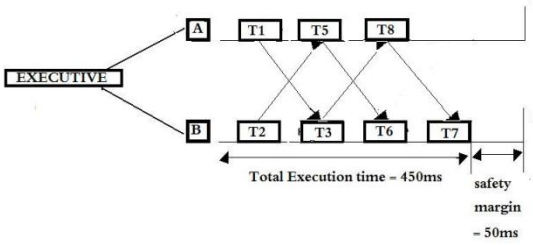


Fig 6

If any fault occurs at any time, say before the execution of T3 then we can shift or replace T4 with T3 in B since T4 is a non critical task and T3 is critical and its execution is a must. This model makes sure that all critical tasks get executed without any interruption.

So at the end , processor A is having a large safety margin shown in Fig 6 ; so it is quite obvious that we can add some more complexity to the system or add another task T9 which is done in the Fig 7.

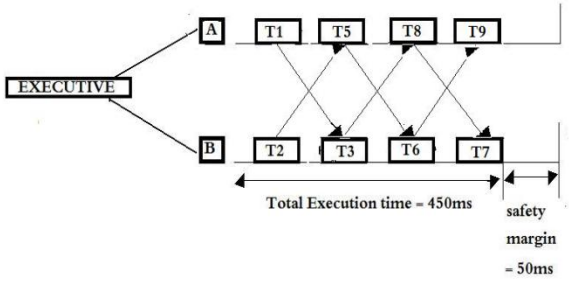


Fig 7

3.3 The control flow

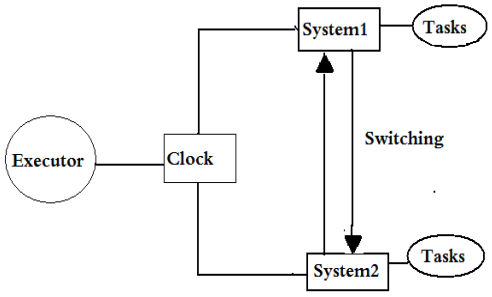


Fig 8

ALGORITHM-

1: arrival : Time;

...

2: arrival := Clock; -- first arrival time

3: loop -- loop forever

4: delay until arrival; -- suspend task until arrival time

5: 'action'; -- code for task invocation

6: -- deadline: arrival + D;

7: switch processors(); --switches between processors

8. if(error)

8.1: Check_processor(); --checks the fault occurred at which processor

8.2: perform_operation(); --performs push and pop operation

9: Else

Do_calc(); --does all the calculation

10: compute_time(); --shows time of computation

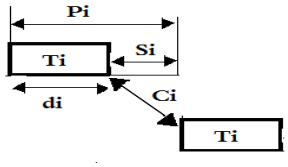
11:end

Now consider,

Di= Execution time of a particular task

Si=Relaxation time between two tasks

Pi=Total time for execution of one task



$$di + ci < Pi$$

Under worst case

$$di + ci = Pi$$

Safety margin = Si = Pi – di

For 4 tasks, condition is

$$\sum_{i=1}^{4} Ti + Ci < 450$$

Problems faced with proposed model:-

Our proposed model deals with communication overhead and existing model deals with relaxation time, so there is no guarantee that it becomes lesser than relaxation time. Hence it is completely dependent upon the hardware Recovery procedure to be laid out for intermediate output.

But inspite of this, in addition to the redundancy removed, we are going to prove with the following case study the efficiency of 'RFETES' is better than the existing model.

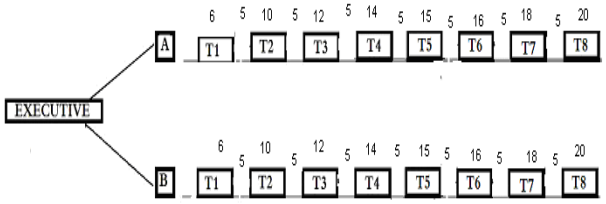


Fig 9

$$\sum_{i=1}^{8} Di + Si$$

Total execution time = =(6+5)+ (10+5)+ (12+5)+ (14+5)+ (15+5)+ (16+5)+ (18+5)+ (20+5) =146=150(Approx)

Here, Di= Execution time of a particular task

Si=Relaxation time between two tasks

$$\sum_{1<=i<=n} (Si/Di) <= 1$$

The Condition here:-

Under worst case scenario we mentioned earlier the above summation becomes equal to 1.

Proposed Model:-

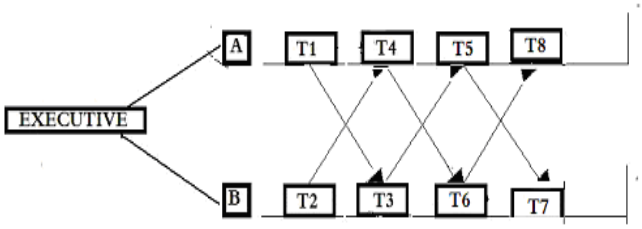


Fig 10

To = (6+4)+(12+4)+(15+4)+(18) = 63

Te = (10+4)+(14+4)+(16+4)+(20) = 72

Without any error it gives O/P at times t=63 and t=72 which is much lesser than the one we got for the existing model which proves its efficiency without any error happening in the background.

$$\sum_{0<=i<=n/2} (C\,i/Di) <= n/2(2^{\wedge}(2/n)-1)$$

Ci = Communication overhead

Di= Execution time of task i

This is the basic formula that any case under the proposed model has to follow to get the desired result to be obtained

3.4 Dependent tasks

Consider system where all the tasks are dependent upon the tasks that are preceding them so their O/P has bearing over the result of the execution of other task.

a) Consider the situation where all the tasks are critical & have equal priority. So under this condition , omission of a task could prove to be catastrophic so it is given that all the tasks need to be executed. So here is how the situation handled

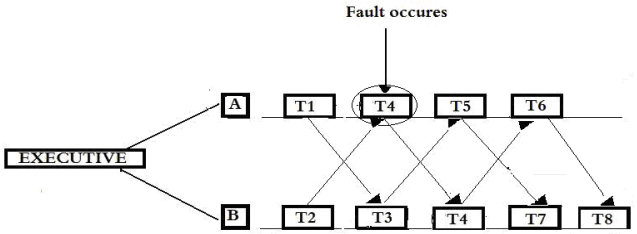


Fig 11

To = (6+4)+(12+4)+(15+4)+(18+4) = 67

Te = (10+4)+(13+4)+(16+4)+(20+4)+14 = 89

Here if a particular task say T4 catches an error then it performs a push & pop operation where T4 is pushed onto B in place of T6 which is popped & pushed in place of T8 & finally pushed onto B extending the time limit & consuming some safety margin which is quite large in this case since we are taking the same time limit as we did for existing system.

So we get final result at time t = 89ms.

b) Consider an error occurring at both the processors considering the same constraints as we had before since all tasks are critical so , all needs to be executed at any cost. Here fault occurs at both T4 and T3 so we are doing the same push and pop operation

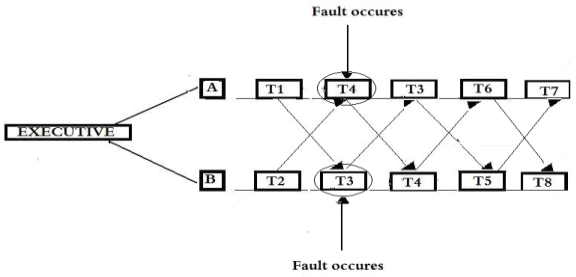


Fig 12

To = (6+4)+(11+4)+(15+4)+(18+4)+(18) = 78

Te = (10+4)+(13+4)+(14+4)+(16+4)+20 = 89

So, final O/P is received at time t = 89ms.

3.5 Independent tasks

Consider a situation where all the tasks are independent of each other so there may be a situation where a particular task may or may not be critical. So let us see what happens then.

a) Here we consider that all the tasks are arranged according to increasing order of their priority. Here is a situation that shows this case:-

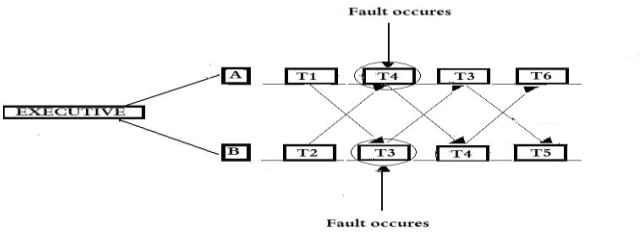


Fig 13

To = (6+4)+(11+4)+(12+4)+15 = 66

Te = (10+4)+(13+4)+(14+4)+16 = 65

Here we have fault at both T4 and T3 in A and B respectively. So we perform the same push and pop operation .So finally we reach that T8 and T7 are lowest in their priority list which are rejected since their removal won't affect the outcome of the system.

So we get the final O/P at time t=66ms.

b) Here we consider that all the tasks are scattered which is shown below

Here we store the list of tasks as critical or non critical where we consider the execution of all the critical tasks to be mandatory. We have a fault at task T4 so it looks into the list whether T4 is critical or not. If it is critical then it performs the same push & pop operation until a task from non critical list comes which is rejected. If it is not then T4 is rejected. If all tasks are critical then we get condition 3.5 (a).

So any way we get the result early.

4. CONCLUSION

Finally if we apply the above formula for the above cases then we will come to know that these cases do follow the formula. So this system is highly efficient i.e. :- If we put n = 8, then the proposed formula would give us the result

RHS = 8/2(2^ (2/8)-1) = 4(2^(1/4)-1) = 4(1.414^(1/2)-1) = 4(1.2 – 1) = 0.8 approx

As compared to worst case scenario for the existing system which takes RHS =1, it takes only 0.8 .

So if total time taken for the worst case by the existing system = 150ms, the proposed system takes 150 *0.8 =120ms.

Thus, with 'RFETES' , feel the difference!!!

5. ACKNOWLEDGMENTS

Our sincere thanks to Mrs. Radhamani Pillai, Professor of Amrita University, Coimbatore, India who had inspired us to work for this idea based on fault tolerant computation. We would like to thank our college, Amrita school of Engineering, Amritapuri campus for giving us such an opportunity to have a deeper insight into the field of embedded systems related to computer applications.

6. REFERENCES

[1] D.G. Feitelson and L. Rudolph. Parallel job scheduling Issues and approaches. In D.G. Feitelson and L. Rudolph, editors, IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing, pages 1{18. Springer{{Verlag, Lecture Notes in computer science LNCS 949,1995

[2] Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, Ltd. A.J. 1992, Deadline monotonic scheduling theory. In proceedings 18th IFAC/IFIP Workshop on real-time programming (WRTP'92) (June 1992).

[3] Bate, I. and Burns, A. 1999. A framework for scheduling and schedulability analysis for safety-critical embedded control systems. Draft, Department of Computer Science, The University of York.