**Fault-Tolerant Deadline-Monotonic and Round-Robin Scheduling Algorithms for Uniprocessors in Hard Real-Time Systems**

Rogatiya Mohmad Aspak Arif (A0179741U)

Department of ECE, National University of Singapore

[e0269760@u.nus.edu](mailto:e0269760@u.nus.edu)

+65 84073081

**ABSTRACT**

*The reliability of a Real-Time System (RTS) is measured in terms of probability of meeting the hard deadlines even in the presence of hardware or software failures. The general approach to fault tolerance in uniprocessor system is to have time-redundancy in scheduling algorithms so that task instances can be re-executed in the presence of transient failures. In this paper, Fault-Tolerant Deadline-Monotonic (FTDM) and Fault-Tolerant Round-Robin (FTRR) scheduling algorithms are presented. An algorithm is proposed to exclusively calculate an Intelligent Time Slice (ITS) for individual tasks in RR to avoid frequent context switches and make RR scheduling suitable for RTS. Schedulability (S) analysis are done for FTDM and FTRR algorithms for various pairs of randomly generated tasksets with various task utilization (U) and mean time to failure (MTTF), i.e. S = f(U, MTTF). It was concluded that FTRR misses much more deadlines compared to FTDM.*

*Keywords: Fault Tolerance, Round-Robin, Deadline Monotonic, CPU Scheduling, RTS*

**INTRODUCTION**

In this paper, time redundancy has been employed to provide a predictable performance in the presence of failures. Deadline-Monotonic (DM) and Round-Robin (RR) scheduling policies are used in various critical applications. However, the classic DM and RR policies do not provide any mechanism for managing task in the presence of transient failures. The main goal of this paper is to provide an appropriate and efficient time redundancy to DM and RR scheduling policies to make them suitable for fault-tolerant real-time applications.

Transient faults in uniprocessor RTS are generally tolerated using time redundancy, which involves the retry or re-execution of any task running during the occurrence of transient faults. Several studies have been done for using time redundancy in embedded RTS for tolerating faults. In the event of a failed primary task, primary task is discarded and a backup copy is executed before the deadline. Two algorithms FTRR and FTDM are proposed to reserve time for the recovery of periodic real-time tasks on a uniprocessor. We will discuss the schedulability tests for fault-tolerant task sets. In DM and RR scheduling policy with task utilization less than 100%, there is a natural amount of slack available. For example, for a task utilization of 80%, CPU is idle for 20% of the time. If somehow we can distribute this 20% time throughout the schedule, then it can be used to re-execute a task in presence of transient failure. To have an efficient fault-tolerant mechanism in the schedule it is necessary that enough slack time is available during the schedule.

**INPUT DATA DESCRIPTION**

**Task Model**

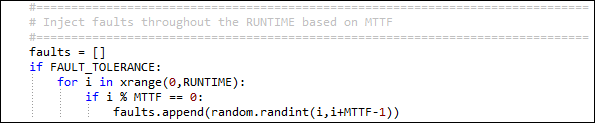
A tasksets generator program TaskGenerator [4] based on UUnitFast-Discard algorithm was used to generate tasksets for the simulation. TaskGenerator can generate specified number of tasksets for given range of worst case execution time (WCET), period and deadlines. Period *Pi* is generated following a uniform distribution with 10 ≤ *Pi* ≤ 100 and *Ci* is generated following a uniform distribution with 1 <= *Ci <= 2αPi*. Where *α* is the average task utilization.

Following **assumptions** are made for the task model.

* *Ci ≤ Di ≤ Pi*
* *Ai = 0.* (Arrival time is same for all task to simulate critical instant.)
* 10 ≤ *Pi ≤* 100 (Uniform distribution of task periods.)
* WCET of tasks is uniformly distributed. 1 <= *Ci <= 2αPi.* Where *α* is average task utilization*.*
* *0.6 ≤ U≤ 0.9 i.e.* Tasksets are generated with CPU utilization varying from 60% to 90%.Upper bound of 0.9 is applied to make sure there is enough room available for fault tolerance.
* Task pre-emption is allowed.
* Tasks are independent, i.e. No precedence relation exists among tasks and no inter-task communications or task synchronization is permitted among tasks.
* RUNTIME = 10000. Each simulation runs for 10000 time units.

**Fault Model**

The code shown below generates random faults during the run time. Various fault sets with MTTF of 25, 50, 75 and 100 were generated to simulate faults during FTDM and FTRR simulation.



An example of fault set generated over a runtime of 1000 with MTTF=50 is given below.



Following **assumptions** are made for the fault model.

* *0 < MTTF ≤ Runtime, i.e.* at least one fault will occur during run time.
* Faults are transient, i.e. faults occur temporarily and can be recovered within very short time.
* A fault gets detected at the very moment of occurrence. The task on CPU during fault will be scheduled for re-execution.

**ALGORITHMS**

1. **Fault-Tolerant Deadline-Monotonic (FTDM)**

This paper presents FTDM algorithm which extends the conventional DM algorithm by using the available CPU slack for fault tolerance. In a fixed priority scheduling algorithm with CPU utilization less than 100%, a natural free time slack is available. Higher the CPU utilization, lesser the available time slack. This time slack is used to implement fault tolerance using time redundancy by allowing re-execution of tasks with transient failure. At any given moment, even during re-execution of failed tasks, the rule of thumb for the DM algorithm still holds, i.e. the task with the shortest deadline will have the highest priority.

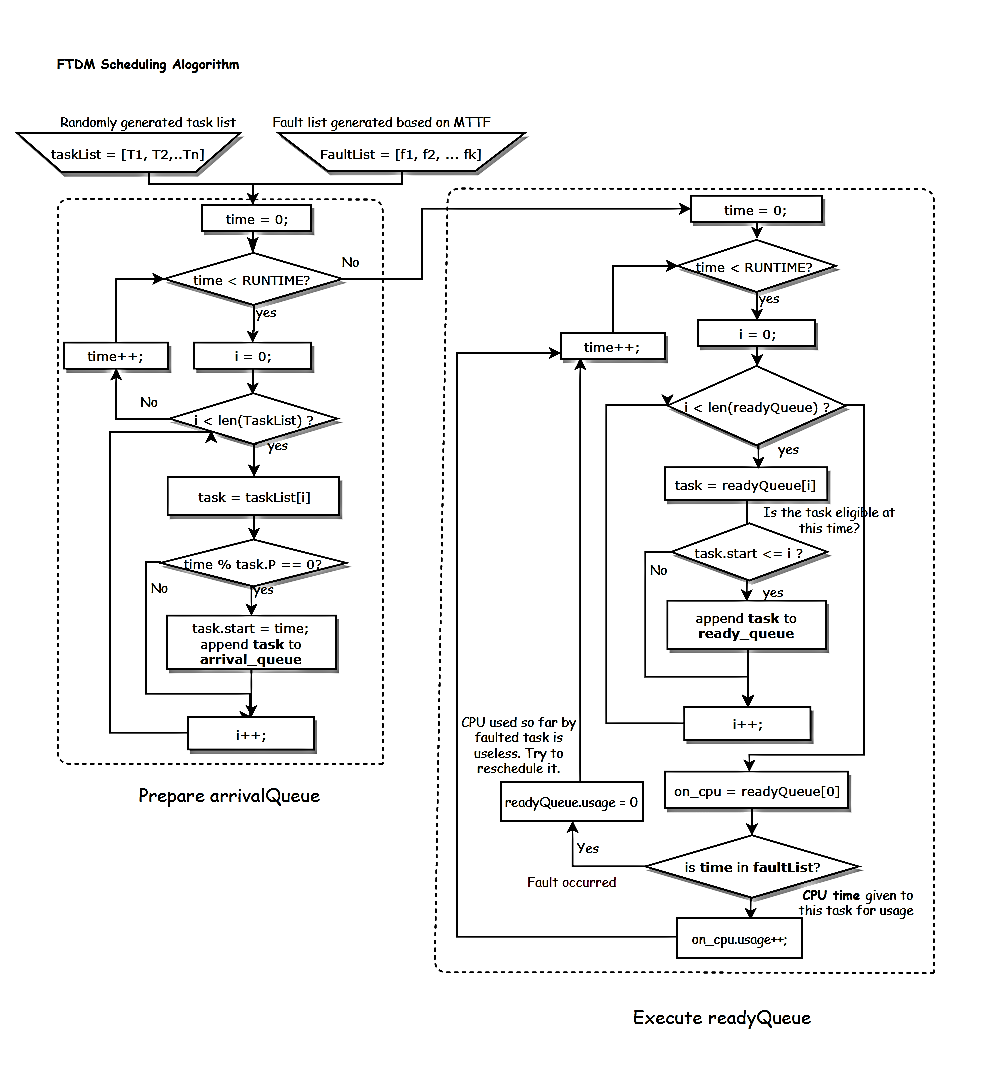
****

Figure 1: Flow chart for FTDM algorithm

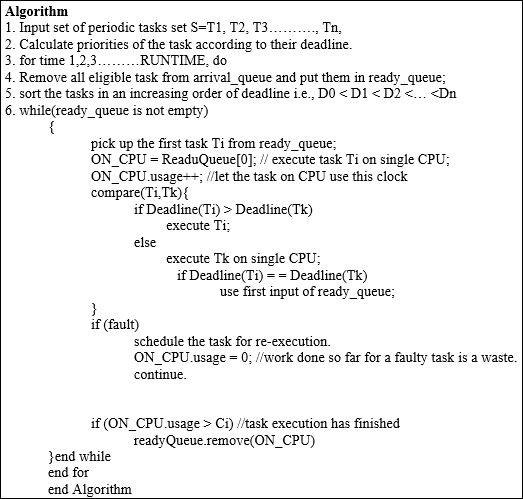


Figure 2: Pseudo code for FTDM algorithm

1. **Fault-Tolerant Round-Robin (FTRR)**

Round robin architecture is a pre-emptive scheduling policy which arranges tasks in the ready queue in first come first serve manner and the processor executes the task from the ready queue based on time slice. If the time slice ends and the tasks are still executing on the processor the scheduler will forcibly pre-empt the executing task and keeps it at the end of ready queue then the scheduler will allocate the processor to the next task in the ready queue. Each task is given the same amount of CPU time regardless of task’s priority, deadline or shortness. Yaashuwanth & Ramesh [3] has proposed an algorithm to dynamic quantum for each task depending on task’s shortness and deadlines to make RR suitable for RTS. The time slice must not be too small which results in frequent context switches and it should be slightly greater than average execution time. Algorithm to calculate an ITS is as shown below.

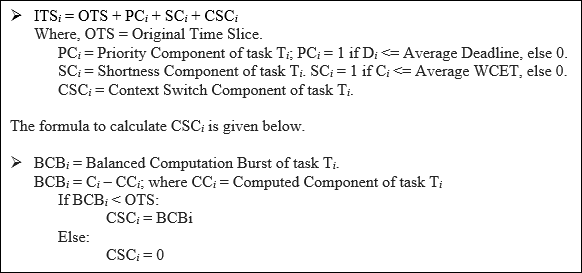


Figure 3: Formulas for Intelligent Time Slice calculations

An example of ITS calculation for 3 tasks T1 (8, 20, 40), T2 (2, 12, 20) and T3 (6, 9, 10) is shown below.



Implementing fault tolerance in RR is easier compared to DM. Each task is assigned an ITS based on its shortness and priority component. If the slice assigned to a task is CPU is enough, the task will be finished and removed from ready queue.

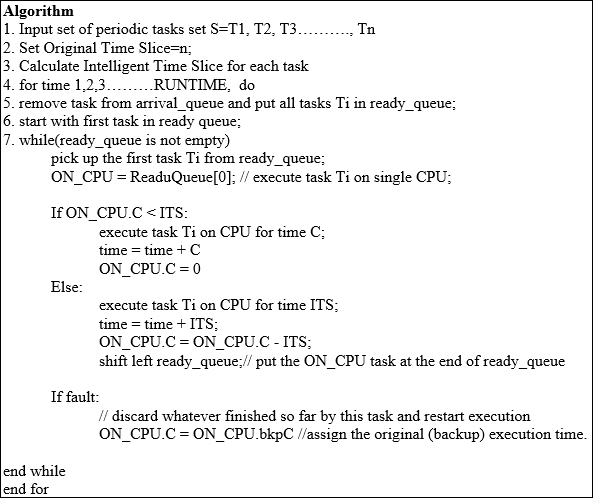


Figure 4: Pseudo code for FTRR algorithm

If the slice is less than execution time of a task, the task will be forcibly pre-empted at the end of time slice and it will be put at the back of the ready queue. If a task is failed due to transient failures, the work done so far for that particular task has to be discarded and the task has to be scheduled for re-execution. The task will restart within the same time slice and put at the back of ready queue if time slice was not sufficient for re-execution of the task.

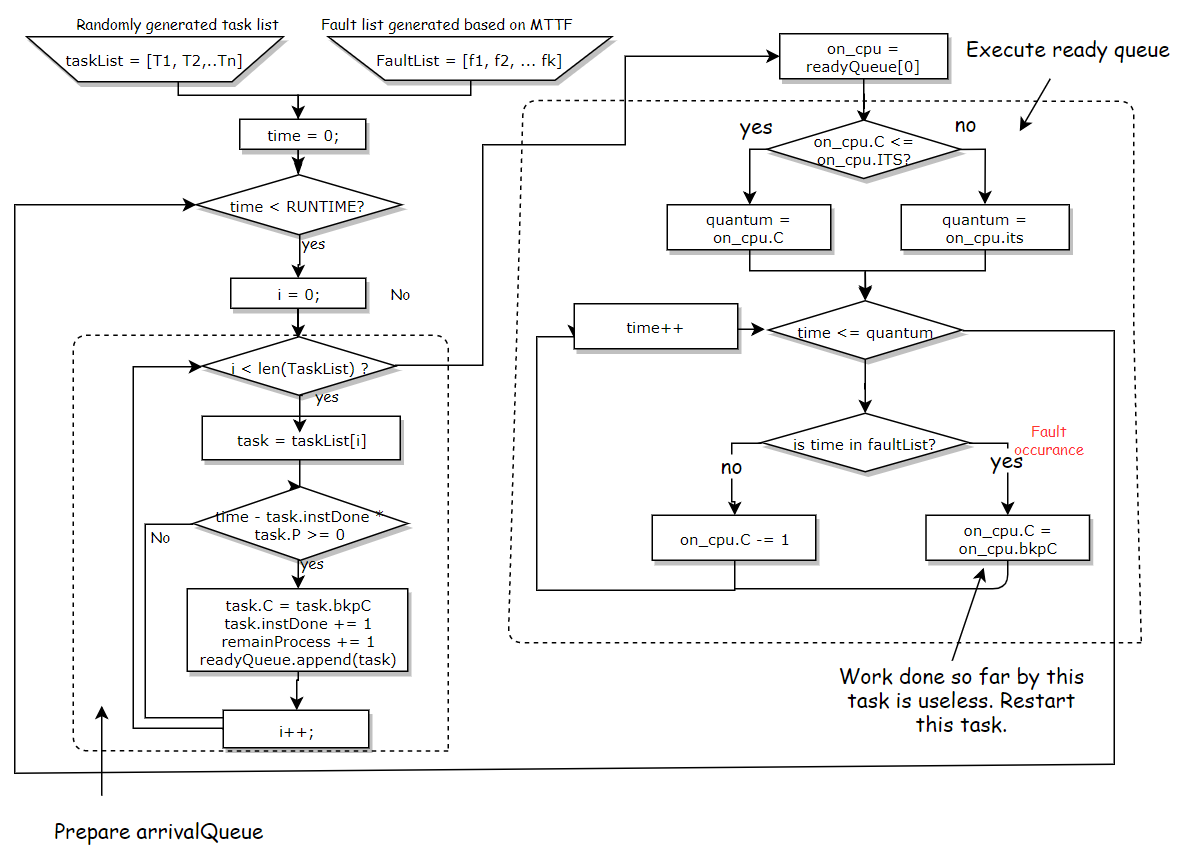


Figure 5: Flow chart for FTRR algorithm

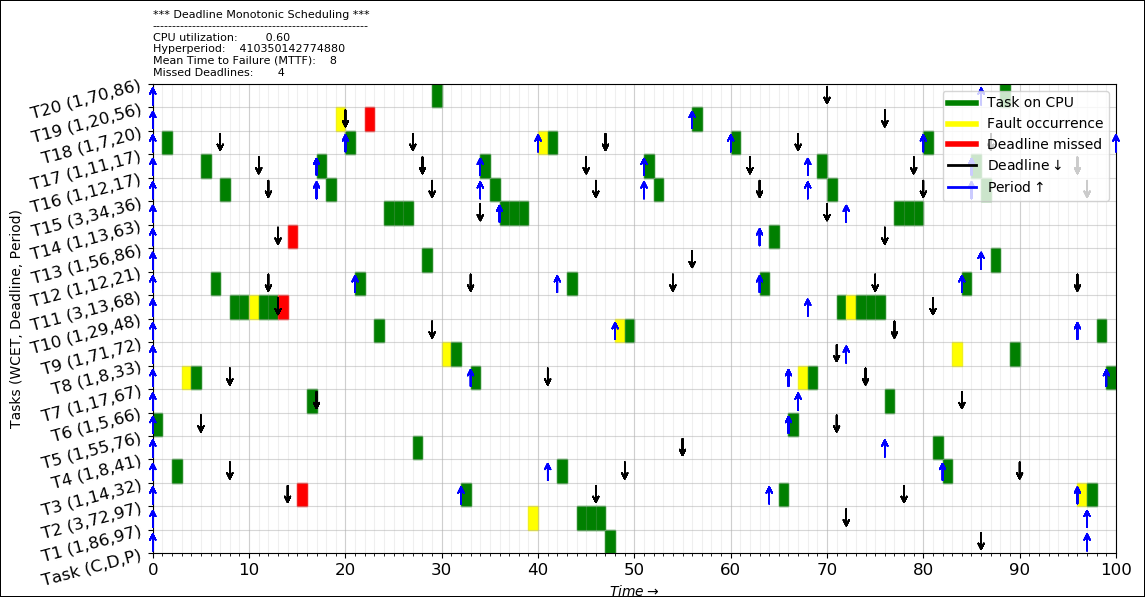
**Simulation Results**

Figure 6: Gantt chart for FTDM with 20 tasks over 100 time units

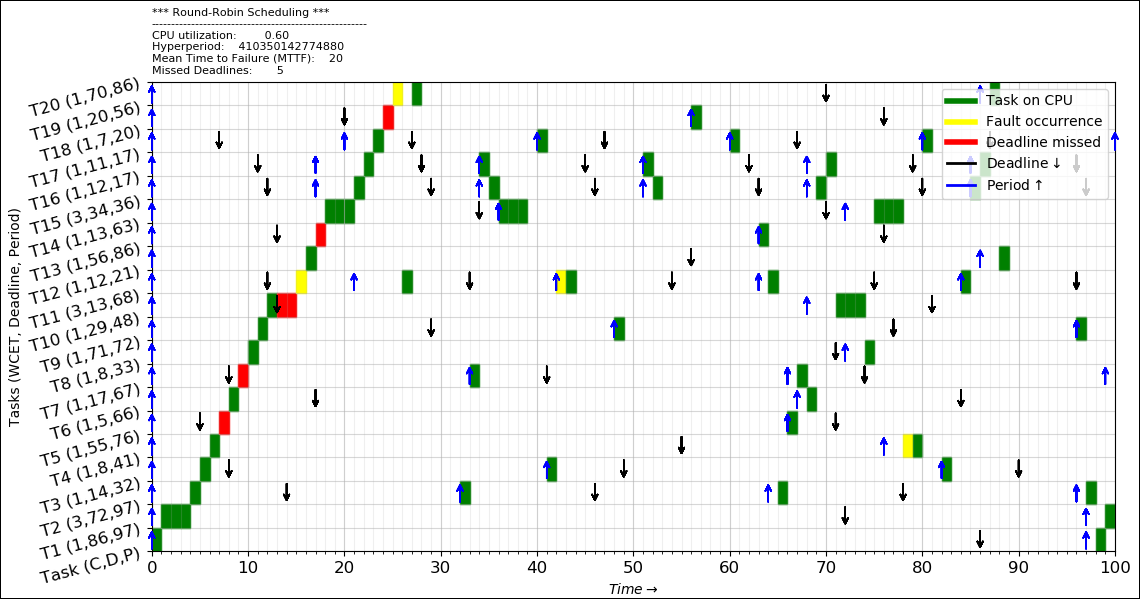


Figure 7: Gantt chart for FTRR with 20 tasks over 100 time units

*Figure 8:* ***FTDM*** *scheduling results showing missed deadlines for various (MTTF, U) pairs*

*Figure 9:* ***FTRR*** *scheduling results showing missed deadlines for various (MTTF, U) pairs*

**Conclusions**

It can be easily seen from figure 8 and figure 9 that FTDM outperforms FTRR algorithm. Number of missed deadlines are much lesser for FTDM algorithm compared to FTRR for the same task and fault model. This main **disadvantage** **of FTRR** **is its inherent nature of frequent context switching**. FTRR with ITS of course gives much better results compared to conventional fix time quantum policy. FTRR has its own advantages like, easy implementation, starvation free scheduling etc., but FTRR is still not as good as FTDM when it comes to reducing the number of missed deadlines. The **main disadvantage of FTDM** is **that it is computationally expensive and difficult to implement compared to FTRR**.

We can observe from figure 9 that number of missed deadlines for FTDM increases drastically as utilization U moves towards 100%. This is due to lesser room available for the re-execution of tasks. Generally fixed priority scheduling policy misses more deadlines as the utilization approaches towards 100% which is necessary but not a sufficient condition for schedulability. If there was no fault tolerance implemented, number of missed deadlines will be even higher as all the faulted task will add up to the missed deadlines. To have an efficient FTDM algorithm it is a good idea to maintain a balance between MTTF and utilization.

We can see from figure 9 that increasing MTTF doesn’t improve the results much for FTRR as if the performance depends mostly on task utilization. Developing a realistic variant of FTRR can be a challenging work for future.

**References**

1. Mehta, Mr. Upasna, “Real Time System Fault Tolerance Scheduling Algorithms.”
2. Reza Ravanmehr, “A Survey on Scheduling Approaches for Hard Real-Time Systems”
3. Yaashuwanth .C & R. Ramesh. “Intelligent Time Slice For Round Robin In Real Time Operating Systems”
4. <https://github.com/atl87/task_gen>