Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 9, September 2014, pg.297 – 306

RESEARCH ARTICLE

Real Time System Fault Tolerance Scheduling Algorithms

Ramita Mehta, Ms. Upasna
Department of Computer Science, GURU KASHI University, Talwandi Sabo (Bathinda)

ABSTRACT : The main objective of this paper is to implement the real time scheduling algorithms and discuss the advantages and disadvantages of the same. Task within the real time system are designed to accomplish certain service(s) upon execution, and thus, each task has a particular significance to overall functionality of the system. Scheduling algorithms in non-real time system not considering any type of dead line but in real time system deadline is main criteria for scheduling the task.
Keywords – SJF, FCFS, Scheduling, Round Robin, Preemptive, Non Preemptive

INTRODUCTION

Real Time System

The Oxford Dictionary of Computing [13] defines a real-time system as:

Any system in which the time at which 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.

The correct behavior of a real-time system depends as much on the timing of computations as it does on the results produced by those computations. Results delivered too late may be useless, or even harmful [14]. Real-time systems are in widespread use and can be found in such application domains as industrial automation, process control, C3 (Communications, Command, and Control), and multimedia. There are two distinct types of systems in this field: hard real-time systems and soft real-time systems. Hard real-time systems are those in which it is imperative that all computations are strictly performed within the specified time, regardless of the operating conditions. Failure to meet the timing constraints of even one task may invalidate the correctness of the entire system. Soft real-time systems, in contrast, are those in which strict adherence to the timing constraints of tasks are not always guaranteed.

The most important attribute of a task in a real-time system is its timing constraints. Such timing constraints must be expressed precisely. A deadline is the most widely used form of a timing constraint. It offers a binary view of the usefulness of a task's completion with respect to a single point in time: the firm deadline. The completion of a task is of no value beyond the deadline and, conversely, would yield full benefit any time prior to that deadline. Such deadline-based systems have a wide range of applications and sufficiently address the requirements of a large sector of the real-time industry. The notion of deadline is particularly well-suited for hard real-time environments where the modes of system operations are mutually exclusive and likewise binary in nature, the system operates correctly if all deadlines are always met, incorrectly otherwise, a task succeeds if it meets its deadline and fails otherwise. For example, the task of deploying the parachutes of the Martian Lander by its on-board control systems must be completed by a hard deadline, else the entire system catastrophically fails. Deadlines also may be used in soft real-time systems in which missed deadlines are to be expected. The operational optimality criteria for these systems can consequently be defined in terms of met or missed deadlines. The operational objective for such systems can be, for instance, to minimize the number of missed deadlines.

Hard Real Time System

In hard real time system a hard dead line is a completion time constraint, such that if the deadline is satisfied, i.e., the task's execution point reaches the end of the deadline scope before the deadline time occurs, then the time constrained portion of the task's execution is timely, otherwise that portion is not timely. E.g. in Autopilot system microprocessor must control the airbags etc. in case of out of control, nuclear plant control etc.

Firm Real Time System

Where the consequences are not severe, but result produced after deadline becomes useless, e.g. airline reservation, banking system etc.

Soft Real Time System

A Soft deadline is a completion time constraint, such that if the deadline is satisfied i.e., the task execution point reaches the end of the deadline scope before the deadline time occurs then the time constrained portion of the task's execution is more timely. Thus hard dead line is the special case of soft deadline. E.g. telephone switching it makes the connection before process execution, Image processing applications etc where utility of result decreases over time after deadline expires.

There are a large variety of real time systems but all have common characteristics, which differentiate them from non-real time systems.

1. Time Constraint: One very common form of time constraint is deadline associated with tasks. A task deadline specifies the time before which the task must complete and produce results. It is the responsibility of the RTOS, schedulers particularly, to ensure that all tasks meet their respective deadline.
2. Safety-Criticality: For traditional non-real time systems safety and reliability are independent issues. However, in many real time systems these two issues are intricately bound together making them safety-critical.

Applications of Real Time System

Embedded real-time systems, commercial transaction systems, transportation systems, and military/space systems - to name a few. The supporting research includes system architecture, design techniques, coding theory, testing, and validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing. These areas often involve widely diverse core expertise ranging from formal logic, mathematics of stochastic modeling, graph theory, hardware design and software engineering. Redundancy has long been used in fault-tolerant and adaptive systems. However, redundancy does not inherently make a system fault-tolerant and adaptive; it is necessary to employ fault-tolerant methods by which the system can tolerate hardware component failures, avoid or predict timing failures, and be reconfigured with little or graceful degradation in terms of reliability and functionality. Early error detection is clearly important for real-time systems; error is an abbreviation for erroneous system state, the observable result of a failure. The error detection latency of a system is the interval of time from the instant at which the system enters an erroneous state to the instant at which that states are detected. Keeping the error detection latency small provides a better chance to recover from component failures and timing errors, and to exhibit graceful reconfiguration. However, a small latency alone is not sufficient; fault-tolerant methods need to be provided with sufficient information about the computation underway in order to take appropriate action when an error is detected. Such information can be obtained during system design and implementation. In current practice, the design and implementation for real-time systems often does not sufficiently address fault tolerance and adaptive-ness issues. Proper task allocation and an effective uniprocessor scheduling can improve the performance of a RTS.

The ability to deliver service is called dependability. The schema of dependability computing Means to attain dependability has been grouped by researchers into four major categories:

- **Fault prevention/avoidance:**

Fault prevention aims at reducing the creation or occurrence of faults during the computing system life cycle. Means are used during the system design phase. Some of them have an impact on the created system. Others prevent faults occurring during its useful life. These means concern the system modeling tools (including implementation technologies), the system models and the processes used to obtain these models.

- **Fault tolerance:**

Fault tolerance aims at guaranteeing the services delivered by the system despite the presence or appearance of faults. Fault tolerance approaches are divided into two classes:

- Compensation techniques for which the structural redundancy of the system masks the fault presence, and,
- Error detection and recovery techniques, that is, detection and then resumption of the execution either from a safe state or after the operational structure modification (reconfiguration).Error recovery techniques are split into two sub-classes: Backward recovery aiming at resuming execution in a previously reached safe state and Forward recovery aiming at resuming execution in a new safe state.
- **Fault removal**: Fault removal aims at detecting and eliminating existing faults. Fault removal is older than those on fault prevention. Fault removal techniques are often

considered at the end of the model definition, particularly when an operational model of the system is complete.

- **Fault Evasion:** Means to estimate the present number, the future incidence, and the likely consequences of faults.

Scheduling Algorithms are used for fault tolerance as well as fault avoidance which may be classified as

1. *First Come First Serve*
2. *Shortest Job First*
3. *Preemptive*
4. *Non-Preemptive*
5. *Round-Robin Technique*

Assumptions

- Set of tasks **Ti** every task has attribute arrival time, Deadline, Worst case execution time.
- **Priorities** of the task, priorities are set according to the scheduling algorithm.
- **System_clock** is the clock set for the task i.e. deadline time to every task, system clock is set according to the task requirement.
- **arrival_queue** is used to store the arrival task, i.e. all the arrival tasks are inserted into the **arrival_queue** .
- **ready_queue** is the queue for inputting the ready task, i.e all the ready to execute tasks are store in ready queue.
- **remove** method is used to take the task from queue.
- **execute** method is used to process the task, i.e. according to the turn of the task they are get executed.
- **Compare method** function is used to compare the task for their deadline if deadline of the task **Ti** is greater than the task **Tk** then task **Ti** is executed otherwise **Tk** is executed using executed method if both having equal deadline then task with first arrival is executed.

First Come First Serve (FCFS)

Algorithm 1: GTFD_U (FCFS)

```
1. Input set of periodic tasks set S=T1, T2, T3………., Tn, Duration, Activation Time,
   Deadline in arrival_queue ;
2. Calculate priorities of the task according to their deadline.// Highest priority is given
   to critical delay i.e. smaller delay higher its priority.
3. for time 1,2,3………system_clock, system_clock do
4. remove task from arrival_queue and put all tasks Ti in ready_queue;
5. start with highest priority task  Ti according to their deadline;
6. while(ready_queue is not empty)
   {
      remove task Ti from ready_queue;
      execute task Ti on single CPU;
```

```
       compare(Ti,Tk)
        {
             if Deadline(Ti) > Deadline(Tk)
             execute Ti;
             else
             execute Tk on single CPU;
             if Deadline(Ti) = = Deadline(Tk)
             use first input of ready_queue;
                           }

     }

    end while
    end for
7.  end Algorithm
```

(Shortest Job First Algorithm)

Algorithm 2: (Shortest Job First)

```
1.  Input set of periodic tasks set S=T1, T2, T3………., Tn, Duration, Activation Time,
    Deadline ;
2.  Calculate priorities of the task according to their deadline.// Highest priority is given
    to critical delay i.e. smaller delay higher its priority.
3.  for time 1,2,3………system_clock, system_clock do
4.  remove task from arrival_queue and put all tasks Ti in ready_queue;
5.  start with highest priority task  Ti according to their deadline;
    while(ready_queue is not empty)
    {
       remove task Ti from ready_queue;
       execute task Ti on Multiple CPUs;
       compare(Ti,Tk)
        {
             if Deadline(Ti) > Deadline(Tk)
             execute Ti on Multiple CPUs;
             else
             execute Tk on Multiple CPUs;
             if Deadline(Ti) = = Deadline(Tk)
             use  first input of ready_queue;
             for T=1 To Ti //Loop to check for preemption
             if newly arrival task Tk  > currently executing task Tk
                      then
             preempt();
               }
    }           end while
    end for
    preempt ()
    {
                      Temp=Ti;
                      execute Tk on Multiple CPUs;
```

```
                assign priority to newly arrival task;
          }
8.  end Algorithm
```

(Round Robin Quantum Non Preemptive) Algorithm

Algorithm 3: RRQ_NP (Round Robin Quantum Non Preemptive) Algorithm

```
1.  Input set of periodic tasks set S=T1, T2, T3………., Tn, in arrival_queue Duration,
    Activation Time, Deadline,Quantum;
2.  Calculate priorities of the task according to their deadline.// Highest priority is given
    to critical delay i.e. smaller delay higher its priority.
3.  Set Quantum=n;
4.  for time 1,2,3………system_clock, system_clock do
5.  remove task from arrival_queue and put all tasks Ti in ready_queue;
6.  start with highest priority task  Ti according to their deadline;
7.  while(ready_queue is not empty)
    {
       remove task Ti from ready_queue;
       execute task Ti on Singlee CPU;
     if Quantum is complete
    then
    take another task Tk according to priority on single CPU;
    end if
    end while
    end for
8.  end Algorithm
```

(Round Robin Quantum Preemptive) Algorithm

Algorithm 3: RRQ_P (Round Robin Quantum Preemptive) Algorithm

```
1.  Input set of periodic tasks set S=T1, T2, T3………., Tn, in arrival_queue Duration,
    Activation Time, Deadline,Quantum;
2.  Calculate priorities of the task according to their deadline.// Highest priority is given
    to critical delay i.e. smaller delay higher its priority.
3.  Set Quantum=n;
4.  for time 1,2,3………system_clock, system_clock do
5.  remove task from arrival_queue and put all tasks Ti in ready_queue;
6.  start with highest priority task  Ti according to their deadline;
7.  while(ready_queue is not empty)
    {
       remove task Ti from ready_queue;
       execute task Ti on Multiple CPU;
     if Quantum is complete
    then
    take another task Tk according to priority on Multiple CPU;
          If task with higher priority than preempt currently executing task
          preempt ();
    end if
    end while
    end for
8.  end Algorithm
```

Problem Formulation: Need and Significance of Proposed Research Work

Real time fault tolerant scheduling algorithms cover both offline and online scheduling. In Offline scheduling algorithms periodic tasks (regular tasks) are used where the periods of tasks are harmonic with smallest period available in the tasks set and relative deadline of the task is not more than its period's. They used backup concept to schedule a task in fault tolerant real time system. Here if the primary copy of a task gets failed, its backup is guaranteed to complete by its deadline. In case primary copy succeeds the time slot reserved for its backup need not to be executed and adds to the slack in the system. Online scheduling allows execution of a part of backup copy along with its primary copy .The total time available before deadline is less than the sum of the time required to complete primary and backup copy of the task

Fault Tolerant Scheduling Problem

A set of n periodic tasks p = {T1, T2… Tn} is to be scheduled on a number of processors, task i will be represented as Ti until and unless specified. For each task Ti, there are a primary copy Priti and a backup copy Backti associated with it. The computation time of a primary copy Priti is denoted as ci, which is the same as the computation time of its backup copy Backti. The tasks may be independent or dependent of each other. The fault tolerant scheduling requirements are given as follows:

1. Each task is executed by one processor at a time and each processor executes one task at a time.

2. All periodic tasks should meet their deadlines.

3. Maximize the number of processor failures to be tolerated.

4. For each task Ti, the primary task Priti or the backup Backti is assigned to only one processor for the duration of ci and can be preempted once it starts, if there is a task with early deadline than the presently executed task

The processors are assumed to be not identical i.e. they are heterogeneous. Requirement (1) specifies that there is no parallelism within a task and within a processor.

Requirement (2) dictates that the deadlines of periodic tasks should be met, may be at the expense of more processors. Requirement (3) is a very strong requirement. The primary and backup tasks should be scheduled on different processors such that any one or more processor failure will not result in the missing of the deadlines of the periodic tasks. Requirement (4) implies that tasks are preemptive. A processor is informed the failure of other processors during the execution of a task. Also, require taking care to ensure that exactly one of the two copies of a task is executed to minimize the wasted work. Since no efficient scheduling algorithm exists for the optimal solution of the fault tolerant real-time multiprocessor scheduling problem as defined above, I will trying to solve this problem by taking new parameters with existing algorithms proposed by researchers, and check their effect by simulation.

Objectives

Whenever talking about the Fault tolerance we mean that how much load a system can bear with what type of scheduling would define the fault tolerance of the system.

Main objectives of my research are implementing the scheduling algorithms and also the fault tolerance of the system depending on the algorithms being used . Following algorithms are being implemented during the project

1. *FIRST COME FIRST SERVER (FCFS)*
2. *SHORTEST JOB FIRST (SJF)*
3. *PREEMPTIVE SCHEDULING*
4. *NON-PREEMPTIVE SCHEDULING*
5. *ROUND ROBIN ALGORITHM*

Research Methodology

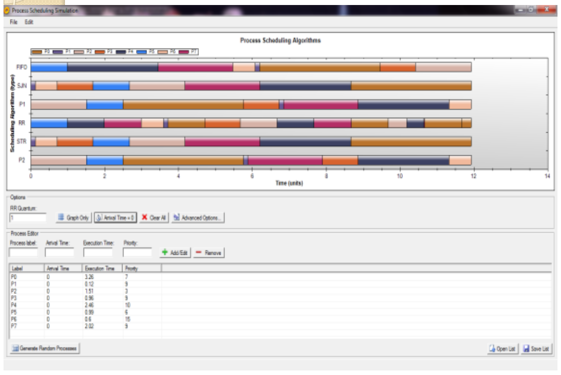
Real Time Applications is an enabling technology for many current and future application areas and becoming increasingly pervasive. The next generation real time system must be designed to be dynamic, predictable, flexible, and reliable and to be able to deal with non-deterministic fault prone environments under rigid timing constraints. So the main problem is to task missed their deadline, so I am trying to execute the task under their deadline, and finding CPU load and Power consumption with remaining time of the task.

The increasing complexity of the hardware multiprocessor architectures as well as of the real-time applications they support makes very difficult even impossible to apply the theoretical real-time multiprocessor scheduling results currently available so I am using simulation techniques to implement my algorithms.

Simulation takes the input as vb.net i.e. scheduling algorithms are is executed on simulation console which then produce the results in the form of Gantt charts of all the inputting tasks and concerned CPU, following diagram shows the architecture of simulation used in this dissertation.

FIGURE

STR

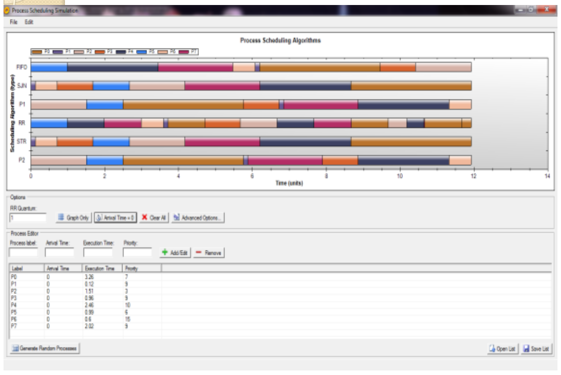


RR Quantum: 1

Process label: | Arrival Time: | Execution Time: | Priority:

Label	Arrival Time	Execution Time	Priority
P0	0	3.26	7
P1	0	0.12	9
P2	0	1.51	3
P3	0	0.96	9
P4	0	2.46	10
P5	0	0.99	6
P6	0	0.6	15
P7	0	2.02	9

RR



RR Quantum: 1

Process label: | Arrival Time: | Execution Time: | Priority:

Label	Arrival Time	Execution Time	Priority
P0	0	3.26	7
P1	0	0.12	9
P2	0	1.51	3
P3	0	0.96	9
P4	0	2.46	10
P5	0	0.99	6
P6	0	0.6	15
P7	0	2.02	9

References

1. Listman A.L., Campbell R.H.,"A fault tolerant Scheduling problem", IEEE Trans. On Software Engineering, Vol. 47,no.5, may 1986,pp.1089-1095
2. Pandya M., and Malek M., "Minimum achievable utilization for fault tolerant processing of periodic task," Technical report TR-07-94, University of Taxas at Austin,March 1994.

3. Ramos –Theul S., "Enhancing Fault Tolerance of Real Time System Through Time Redundancy" Carnegie Mellon University, 1993.

4. Krishna C.M.,and Shin Kang G.”On Scheduling Task with Quick Recovery From Failure” IEEE Trans. On Computers, vol.,c-35,no.5,pp448-455.

5. Tsuchiya T.,Kakuda Y.,and Kikuno T., ”Fault Tolerant Scheduling Algorithm for Distributed Real Time System”

6. Librato Frank, Melhaem Rani, and Moss Deniel,” Tolerant of Multiple Transient Faults For Periodic Tasks in hard Real Time System”, IEEE Trans on Software Engineering, Vol., Se-6

7. Mosse Daniel, Meihem Rani , and Ghosh Sunondo.”Non Preemptive Real Time Scheduler with Recovery From Transiant faults and its and its Implementation “, IEEE Trans on software Engineering, vol.29, no. 8, Augest 2003

8. Oh.Y., and Son S.H. ,”Enhancing Fault Tolerance in Rate Monotonic Scheduling in Real Time System “, International Journal of Time Critical Computing Systems ,vol.7, no.3.

9. Dhall S.K., and Liu C.L., “On Real Time Scheduling Problem” International journal on Operation research vol.26, pp127-140.

10. Liu C.L., and Layland J.W.,”Scheduling Algorithm for Multiprogramming For Hard Real Time System” Journal ACM, Vol.20, no.1, pp 40-61.

11. Lehoczky J.P.,Sha L., and Ding Y., “The Rate of Monotonic Scheduling Algorithms Exact Characterization and average case Behaviour,” IEEE Real Time System Symposium,pp. 166-177