Real Time Event Management And Coordinating System

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Computer Science)

by

Ipsita Minj

(Roll No 211CS1272)

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

June 2013

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping this thesis. At the outset, I would like to express my sincere thanks to all the professors for giving me the opportunity to work on this thesis. They have been a source of knowledge. I extend my thanks to our HOD, Prof. A.K Turuk for his valuable contribution.

I am really thankful to my all friends specially Anand Tirkey, Anand Ekka, Biren Oram and Ashish Tirkey for providing me with kind words, a welcome ear, new ideas, useful criticism, or their invaluable time, I am truly indebted.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to the Aspire to Inspire Family Foundation, for their true inspiration for the motivation towards this thesis. I am also thankful to my family for their love, patience, and understanding.

Ipsita Minj

Roll No: 211CS1272

Abstract

Analysis and prediction of real time event managing is very important and interesting as this helps experts in managing events , making decisions and working more efficiently . This thesis Event Managing And Coordinating system (RT-EMaCS) model is initially considered for proper managing of time and task, and resulted in funtioning in both system field as well as practical world.

A EMaCS model can fit into any Java based platform such as laptops, desktops and any mobile device supporting Java, specially like android phones or tablets. The link between them can be done via Wi-Fi. In this thesis, the event organizers and the event coordinators communcate with better facilities in event management. It provides an easy, simple and better means of communication among one another. It prevents loss of time.

Contents

List of Figures

Chapter 1

Introduction

Chapter 1

Introduction

1.1 Introduction

In todays busy world, where even every micro second is considered as very significant, almost all existing systems can be considered as real time systems because time has become a very important factor for execution of any system. Or precisely, it can be said that any system which needs time ranging from seconds to fractions of milliseconds is a real time system.

Examples of real time systems are, consumer products such as digital cameras, camcorders, cell phones, microwave ovens, video game sets; office products such as fax machines, laser printers and security machines; industrial production such as chemical and nuclear plant control, industrial plant automation system, space missions, immternet routers, flight control systems, and so on. Real time system are those in which the time of production of result is as important as the correctness of the result itself. Real-time systems have strict timing requirements that must be met. Real-time systems guarantee that all the timing requirements can be met by the theory of real-time scheduling and schedulability analysis. The computational demands of most applications require the use of multiple processors. The use of the real-time systems in multiple processors has extended not only because of the high computing requirements of these applications, but also because of the faster response times and fault tolerant features of such type of systems. Also, a

reason for the popularity of multiprocessor systems is the drop in their prices. The increasing availability of multi-core and other multiprocessor systems has resulted in renewed interest in multiprocessor real-time scheduling. With the advent of multi-core architectures in particular, multiprocessor platforms can be expected in the future to be the standard computing platform in many settings, including settings where real-time constraints must be supported. In order to support such constraints on multiprocessor platforms, the development of useful real-time scheduling approaches and associated analytical results is crucial.

1.2 Dependability in RTS- a case study:

Trade offs among system performance and with respect to reliability are becoming increasingly important. Hence, reliability measurement in Real Time System is of an important use.The list of reliable Real Time Systems is very vast. We have therefore restricted our list to only a handful of applications where incapability to deal with faulty components leads to hazardous results.

(1) Computer On-Board an Aircraft: In many modern air craft the pilot can select an auto pilot option. As soon as the pilot switches to this mode, a non-board computer takes overall control of the air craft including navigation, take-off, and landing of the aircraft. In theautopilot,the computer periodically samples velocity and acceleration of the aircraft. The on-board computer computes X, Y,and Z co-ordinates of the current aircraft position from the sampled data, and compare them with pre-specified track data.It computes the deviation from the specified track values and take any corrective actions that may be necessary. In this case, the sampling of various parameters, and their processing is required be completed within a few microseconds.

(2) Missile Guidance System: A guided missile is one that is capable of sensing the target and homes onto it.In a missile guidance system, missile guidance is achieved by a computers mounted on the missile. The mounted computer computes the deviation from the required trajectory and effects change in track of the missile to guide it onto the target. The time constraint on the computer based

guidance system is that sensing and the track correction tasks must be activated frequently enough to keep the missile from straying from the target. Tasks are typically required to be completed within few hundreds of microsecond or even lesser time. If the computer on the missile goes faulty then means should integrated by which this faulty systems can be compensated, which is clearly possible by means of good fault tolerant technique.

(3) Internet and Multimedia Applications: Important uses of real-time systems in mul -timedia and Internet applications include : video conferencing and multimedia multi- cast, Internet router and switches. In a video conferencing application,video and audio signals are generated by cameras and microphones, respectively. The data are sampled at certain pre-specified frame rate.These are then compressed and sent as packets to the receiver over a network. At the receiver-end, packets are ordered, decompressed and then played. The timeconstraint at the receiver end is that the receiver must process and play the received frames at pre-determined constant rate.

1.3 Literature Review:

Research has been done on various aspects of realtime systems such as real time scheduling algorithms in uniprocessors and multiprocessors and its schedulability analysis, real time simulators, real time operating systems and real time communication system, etc. Most commonly used algorithms for real-time processor scheduling in uniprocessors are (i) Earliest-deadline first (EDF), (ii) Rate monotonic (RM), and (iii) Least Laxity First (LLF) with the objective to maximize the throughput [1] . In Earliest Deadline First scheduling, the task which has the shortest deadline is taken up for scheduling. EDF has been proven to be an optimal uniprocessor scheduling algorithm. This means that if a task set is schedulable under EDF, then then no other scheduling algorithm can feasibly schedule this task set [2]. EDF is a dynamic priority scheduling algorithm; however Rate Monotonic algorithm is a static priority algorithm which assigns priority to tasks based on the rate of occurrence [3]. In LLF the laxity value of every task in the

system is computed at each scheduling point, and the task which has least laxity is taken up for execution. However EDF and RMA scheduling are not optimal on multiprocessor system [4]

Multiprocessor real-time scheduling theory also has its origins in the late 1960s and early 1970s. Behera et al. [5] proposed the least switch and laxity first scheduling algorithm, which improves the least laxity first algorithm for periodic task by searching out an appropriate common divisor along with Modified Least Laxity First (MLLF).Baruah et al. [6] [7] presented two Pfair (Proportionate Fairness) scheduling algorithms called Pseudo Deadline (PD) and PF which differ in the way in which ties are broken when two sub-tasks have the same deadline. Recent research on proportionate fair (Pfair) scheduling has shown considerable promise in that it has produced the only known optimal method for scheduling periodic tasks on multiprocessors.

The EDZL (Earliest Dead-line first until Zero Laxity) [8] algorithm as proposed by Lee combines the EDF and LLF algorithms. EDZL schedules jobs based on their deadlines and laxities. When all jobs have positive laxities, EDZL schedules jobs according to EDF. Whenever the laxity of a job becomes zero, EDZL schedules the job with the highest priority. Kato and Yamasaki [9] presented an algorithm, named Earliest Deadline Critical Laxity (EDCL), for scheduling sporadic task systems on multiprocessors. EDCL reduces runtime overhead and implementation cost as compared with EDZL, but still strictly dominates G-EDF in schedulability. LLREF [10], is based on the fluid scheduling model, which executes all tasks at a constant rate. This algorithm divides the schedule into Time and Local execution time planes (TL-planes), which are determined by task deadlines. The algorithm schedules tasks by creating smaller local" jobs within each TL-plane.

D-EDF [11] is a combination of both of EDF and DM algorithm. During under-loaded condition, the algorithm uses EDF algorithm and priority of the job is decided dynamically depending on its absolute deadline. During overloaded condition, the algorithm uses DM algorithm and priority of the job will be decided statically depending on its relative deadline. D-EDF scheduling algorithm over-

comes the limitations of dynamic algorithm during overloaded conditions. Kim and Cho [12] proposed a scheduling algorithm, called PL ,a laxity based algorithm which ensures execution of a task with approximate proportional fairness at period of each task. Existing optimal algorithms on multiprocessors may cause excessive scheduling decisions and preemptions or may not be applied in a discrete environment. The PL algorithm can be applied in a discrete environment and reduce the number of scheduling decisions and preemptions compared with a Pfair algorithm.

1.4 Motivation:

Since time is the most important factor in todays generation, enhancement in real time technologies will help mankind to utilize every fraction of time. Application of real time systems has been found in simple devices around us such as consumer products, telecommunication domain products and applications, office products, medical equipments and imaging systems and industrial applications, etc. One such application in our daily lives is the events or functions we are invited to attend with family and friends or in schools and colleges. A yet to be registered organisation named Aspire to Inspire Family Foundation which conducts various programmes such as Christmas gathering, Family Day, and other charitable events faced problems on handling their events on the run time. There arose a need for such a system that would take care of the proper rescheduling and management and coordination of events on the run time. Hence, this thesis takes up the work of building a model that would ensure to provide the efficiency of scheduling and rescheduling in real life event management through real time computing.

1.5 Problem Statement:

Research and work has been done in various aspects of life using real time systems. One such aspect is event management. Many softwares already exist to manage the pre-event and post-event requirements. However there is a vital need

of managing any event going onstage. If the actual event does not go as planned then all pre-event arrangements go vain. This will, then, also affect the post-event planning. Hence this thesis takes up the work of applying real time systems in onstage event management so that the schedule of event may be satisfactorily maintained even in unforeseen circumstances.

1.6 Thesis Outline:

The thesis has been divided into five chapters and has been arranged as follows. Chapter 1 gives a brief description Real Time System and related works by various researchers in the area of real time scheduling. Chapter 2 describes model and performance metric in real time systems. Chapter 3 discusses the scheduling of tasks along with the major concepts in real time systems. Chapter 4 describe the proposed model for better event management on the run time. Chapter 5 finally concludes the thesis. Future work has been mentioned in chapter 6.

Chapter 2

RTS: MODEL AND METRIC

Chapter 2

RTS: MODEL AND PERFORMANCE METRIC

2.1 Workload Model:

Real time tasks get generated in response to some events that may either be external or internal to the system. The model consists of a set of n periodic tasks S = T1, T2, T3 . . . . . Tn, each Ti is released periodically with a period pi and has a deadline di. Each time a task recurs it is called an instance of the task. The jth instance of task Tj would be denoted as Tj(j). Each task ti has the following attributes [9, 13]:

$$T_i =< a_i, r_i, d_i, c_i, p_i > \tag{2.1}$$

Where a_i = arrival time, r_i = ready time, d_i = deadline, c_i = worst case, execution time, p_i = period

2.2 Performance metric:

The criterion mostly used to measure the performance of scheduling algorithms for hard real-time applications is their ability to find feasible schedules of the given application system whenever such schedules exist. An important parameter is the

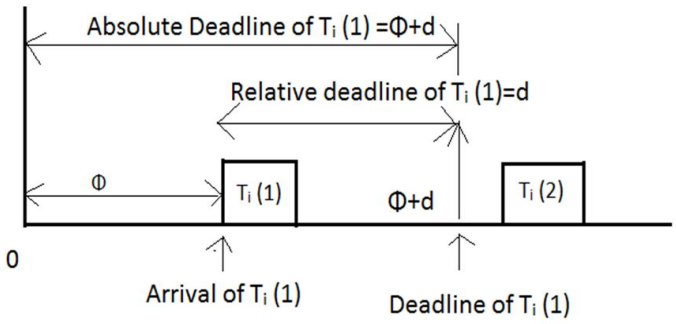

Figure 2.1: Relative and absolute deadline of a Task.

Success Ratio [9]

$$\frac{The\ number\ of\ task\ sets\ successfully\ scheduled}{The\ number\ of\ totalscheduled\ task\ sets} \tag{2.2}$$

Other commonly used performance measures include the maximum and average tardiness, lateness, and response time and the miss, loss, and invalid rates. The right choice of performance measure depends on the objective of scheduling [1].

- The *tardiness* of a job measures how late it completes respective to its deadline. Its tardiness is zero if the job completes at or before its deadline; otherwise, if the job is late, its tardiness is equal to the difference between its completion time(i.e., the time instant at which it completes execution) and its deadline.

- *lateness* of a job is the difference between its completion time and its deadline. Unlike the tardiness of a job which never has negative values, the lateness of a job which completes early is negative, while the lateness of a job which completes late is positive.

- *Response time* is the length of time from the release time of the job to the instant when it completes.

- *Miss rate* is the percentage of jobs that are executed but completed too late and *loss rate* give the percentage of jobs that are discarded, that is, not executed at all.

- *Invalid rate* is the sum of the miss rates and loss rates and gives the percentage of all jobs that do not produce a useful result.

Other performance metric as mentioned in [14] are utilization bounds, approximation ratios, resource augmentation or speedup factors, and empirical measures. The worst-case utilization bound U for a scheduling algorithm A is defined as the minimum utilization of any implicit-deadline task set that is only just schedulable according to algorithm A. The resource augmentation or speed up factor for a scheduling algorithm A is defined as the minimum factor by which the speed of all m processors would need to be increased such that all task sets that are feasible (i.e., schedulable according to an optimal scheduling algorithm) on m processors of speed 1 become schedulable under algorithm A.

Chapter 3

TASK SCHEDULING IN RTS

Chapter 3

Task Scheduling in RTS

3.1 Introduction

Real time systems comprises of one or more real-time tasks. These tasks get generated when some specific event, may be external or internal to the system, occurs. Hence a task is said to have arrived or got released when it gets generated. Real time tasks need quantitative expressions of time to describe its behaviour [1]. As real time systems need both logical correctness as well as timeliness in the execution of tasks, proper scheduling of these tasks becomes an important criterion. The scheduling problem in real time distributed systems can be conceptually separated into two parts. As there are many nodes where a task can be executed, the first question to be answered is how to assign the tasks to them. This is known as task allocation or global scheduling. Once the tasks are allocated, the problem is minimized to that of local scheduling for each node which is equivalent to scheduling problem in uniprocessor systems.

3.2 Concepts and terms

- *Job*: Each unit of work that is scheduled and executed by the system is called a job.

- *Task*: A set of related jobs which jointly provide some system function a task.

- *Processors*: Every job executes on some resources such as disk, CPU, network etc. These resources are called Processors in terms of real time systems literature.

- *Release Time*: The release time of a job is the instant of time at which the job becomes available for execution. The job can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met.

- *Response Time*: It is the time interval between the time of release of the job and the time at which the job completes.

- *Deadline*: Deadline is the instant of time by which the execution of a job needs to be completed.*Relative daedline* is the maximum response time that is allowed for a job, i.e. the time interval between the start of the task and the occurrence of deadline. *Absolute deadline* is the sum of release time and relative deadline, i.e. time interval between time 0 and the occurrence of deadline.Deadlines consist of run-ability constraints only,i.e, each task must be completed before the next request for it occurs [15].

- *Scheduling*: Scheduling is determining the sequence in which various tasks are to be assigned to the operating system. This is done by a scheduler. A scheduler is a module that allocates processors and resources to jobs and tasks [1]. Each task scheduler is characterised by the scheduling algorithm it employs [2]. The result produced by the scheduler is called a schedule. A schedule is intended to be optimal with respect to some criteria (such as timeliness).

- *Valid Schedule: A valid schedule of a set of tasks is a schedule which satisfying the following properties*

  Each process can only start execution after its release time.

All the precedence and resource usage constraints are satisfied.

The total amount of processor time assigned to each task is equal to its maximum or actual execution time.

- *Feasible Schedule*: A valid schedule is a feasible schedule if all the tasks complete without missing their timing constraints.

- *Optimal Scheduler*: A scheduler is called optimal scheduler if it always produces a feasible schedule on a set of tasks for which a feasible schedule exists. If an optimal scheduler cannot produce a feasible schedule for any task set then no other scheduler can.

3.3 Timeliness Specification

In real time systems the time is qualified which determines the type of real time system. The timeliness specification can be of the following type.

1. Deadline: A deadline is a completion time constraint which specifies that the timeliness of the tasks transit through the deadline scope depends on whether the tasks execution point reaches the end of the scope before the deadline time has occurred, in which case the deadline is satisfied. In other words, the instant of time by which the execution of a job needs to be completed.

2. Hard Deadline: A timing constraint or deadline is hard if the consequences of its failure(i.e. not meeting its deadline) is a fatal fault. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences.

3. Soft Deadline: Soft deadline is a completion time constraint where although the late completion is not desirable, yet a few misses of soft deadlines are tolerable as they do no serious harm. But the systems overall performance becomes poorer if more and more jobs with soft deadlines complete late.

3.4 Hard Real Time

Hard real-time is the case where: for the schedulable entities, some time constraints are hard deadlines, and the timeliness component of the scheduling optimization criterion is to always meet all hard deadlines (additional components may apply to any soft time constraints); for the non-schedulable entities, some upper bounds are hard, and the system has been designed and implemented so that all hard upper bounds are always satisfied (other non-schedulable entities may have soft upper bounds). Thus, the feasible schedules (with respect to those schedulable entity time constraints) are always optimal, and the predictability of that optimality is maximum (deterministic).

Hard real time systems can be hence defined as. systems that are based on deadline schemes, usually using priority as well. Such systems typically have a worst case requirement. Failure to meet timing requirement leads to fatal fault and failure to meet a deadline, in such systems, requires automated handling,

3.5 Soft Real-time

Soft real-time represents all cases which are not hard real-time (soft real-time is the general case, of which hard real-time is a special case). Time constraints are soft (which may include the hard deadline special case), such as the classical lateness function.

Any scheduling optimization criteria may be used (including the hard real-time special case), such as minimizing the number of missed deadlines, or minimizing mean tardiness, or maximizing the accrued utility. Predictability of schedule optimality (and thus thread timeliness) is generally sub-optimal, but may be deterministic (including but not limited to the special hard real-time case).

Upper bounds are soft, and predictability of non-schedulable entity timeliness is generally sub-optimal. Soft real time systems can be hence defined. systems that

have an average case timing requirement. Failure to meet the timing requirement is not critical in such systems. Such systems are often based on priority schemes Unlike in hard real time systems where it becomes important to ensure all tasks are completed by their deadline in soft real-time systems where meeting the deadlines of all the tasks is not essential. Critical Task, Efficient scheduling.

3.6 Type of Tasks

The three main categories of real time tasks based on recurrence of tasks are:

3.6.1 Periodic Tasks:

Periodic tasks are those that are repeated after certain fixed interval of time [1]. They are also called clock driven tasks because the fixed interval is generally notified by clock interrupts. Such fixed interval of time is called the period of the task. A formal way of representing a periodic task Ti is as (i ,pi , ei , di) where i is the occurrence of the instance of Ti, pi is the period of task, ei is the worst case execution time of task, and di is the relative deadline of the task. Sporadic Tasks: Sporadic tasks are those that are repeated at random instant of time. However there is a minimum separation between two consecutive instances of task. A sporadic task Ti can be represented by three tuple (ci, gi, di) where ci is the worst case execution time of an instance of the task, gi denotes the minimum separation between two consecutive instances of the task, di is the relative deadline.

3.6.2 Aperiodic Tasks:

An aperiodic task is similar to a sporadic task except that the minimum separation gi between two consecutive instances can be 0.

The basic task model, considered in this thesis work is which is modeled by a set of N periodic tasks Ti:

$T_i = (p_i, d_i, c_i) \mid 1 <= \text{i} <= \text{N} \mid$

3.7 Taxonomy of RT scheduling Algorithms

The vast majority of scheduling problems on systems with more than two processors are NP omplete [16]. The problem of sheduling in multiprocessor and distributed systems is reduced to that of uniprocessor scheduling [1]. In general, tasks may have data and ontrol dependencies and may share resources on different processors.

At the highest level, a distinction is drawn between hard and soft scheduling, depending on the timing constraint defined as hard and soft real time systems.

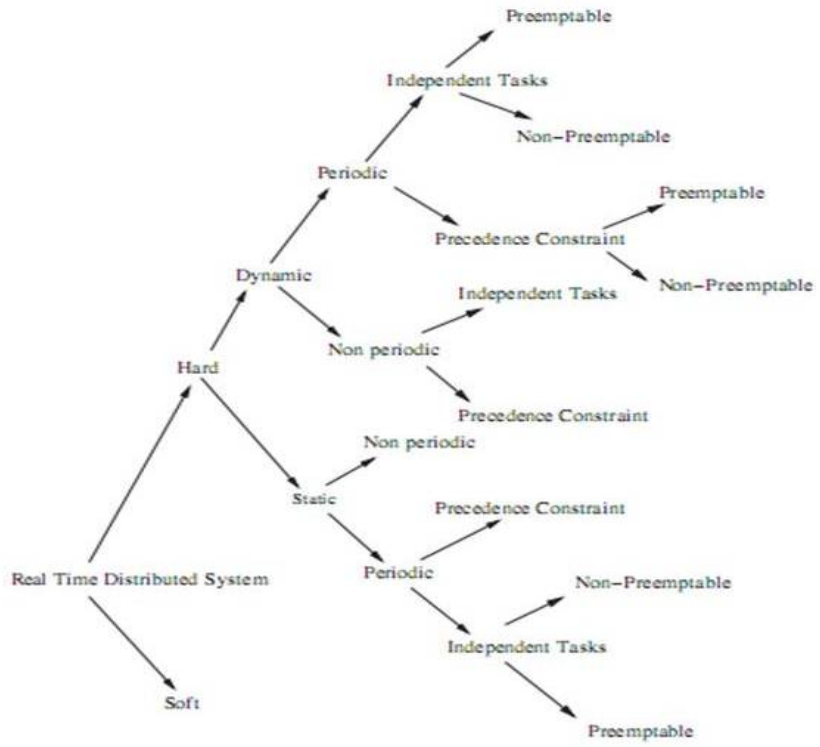

Figure 3.1: Taxonomy of Real Time scheduling

3.8 Task assignment

The optimal assignment of tasks to processors is, in almost practical cases, an NP complete problem. Hence applying heuristic techniques. These heuristic cannot guarantee that an allocation will be found that permits all tasks to be feasibly

scheduled. All that can be done is to make an allocation, check its feasibility, if the allocation is not feasible, modify the allocation to render that the allocation is feasible. Sometimes the allocation uses the communication cost as the part of the allocation scheme.

3.9 Task Scheduling

Classification of real time task scheduling algorithms can be done using various schemes. There are three main types of scheduling algorithms based on the scheduling points defined:(i) Clock Driven, (ii) Event Driven and (iii) Hybrid. [2]

3.9.1 Clock Driven Approach:

In this approach the decision of scheduling the tasks is done only at clock interrupts. It is also known as offline scheduler or static scheduler as the schedule is fixed before the execution in the system starts. Hence the run time overhead remains low. However these are not suitable for aperiodic and sporadic tasks since their arrival time cannot be predicted. Clock driven schedulers are simple and efficient. Table driven and cyclic schedulers are important examples of Clock driven schedulers.

3.9.2 Event Driven Approach:

In this approach, the scheduling decisions are taken at events such as task arrival or task completion. They are also called priority driven schedulers because of the pre-emptive property. The higher priority tasks can pre-empt the lower priority tasks. Event driven schedulers are more flexible than Clock driven schedulers because they can handle both aperiodic and sporadic tasks along with periodic task whereas clock driven schedulers can handle only periodic tasks.

3.9.3 Hybrid Approach:

As predictable by the name, the hybrid approach is a combination of the clock driven approach and event driven approach, i.e. the scheduling point are at clock interrupts as well as occurrence of event. Weighted Round-Robin Scheduler is a popular hybrid scheduler. Here every ready task is put in a First-in-First-out queue and gets a chance to execute for a time-slice. If it does not complete in that time slice it is put back in the queue. Such schedulers are not suitable for critical tasks.

3.10 Scheduling in Multiprocessors

Scheduling in multiprocessor can be viewed as attempting to solve two problems [14] :

1. The allocation problem: on which processor a task should execute.

2. The priority problem: when, and in what order with respect to jobs of other tasks, should each job execute.

Scheduling algorithms for multiprocessor systems can be classified according to when changes to priority and allocation can be made (referred to as migration-based and priority-based classifications [17]).

3.10.1 Allocation

1. No migration: Each task is assigned to a processor and no migration of task to any other processor is permitted.

2. Task-level migration: The jobs of a task are permitted to execute on different processors; but each job can only execute on a single processor.

3. Job-level migration: A single job is allowed to migrate to and execute on different processors; but parallel execution of a job is not permitted. [18]

3.10.2 Priority

1. Fixed task priority: A single fixed priority is applied to all the jobs of each task.

2. Fixed job priority: Different priorities may be applied to the jobs of a task, but each job has a single fixed priority. An example of this is Earliest Deadline First (EDF) scheduling.

3. Dynamic priority: Different priorities may be applied to a single job at different times, for example Least Laxity First (LLF) scheduling.

3.10.3 Best-known heuristics for the allocation problem

The best-known heuristics found in the literature that solve the allocation problem are [19]

1. First-Fit (FF): FF allocates a new object to a non-empty bin with the lowest index, such that the weight of the new object along with the weights of the objects already allocated to that bin, do not exceed the capacity of the bin. If the new object exceeds the capacity of the bin, then FF allocates the object to an empty bin.

2. Best-Fit (BF): If an object cannot be allocated to a non-empty bin then BF puts the object into an empty bin. Otherwise, BF will allocate the object to the non-empty bin with smallest capacity available, in which the object can be allocated. If there is more than one bin with the same capacity, then BF will choose the bin with smallest index.

3. Next-Fit (NF): After allocating the first object to the current bin, NF allocates the next object to the same bin, only if it fits into this bin. Otherwise, NF allocates the object to the next empty bin. Note that NF does not check if the object can be allocated in previous bins.

4. Worst-Fit (WF): This algorithm is similar than BF, with the difference that WF allocate objects into the bins with the greatest capacity available, in which they can be feasibly allocated.

3.11 Approaches to Real time Multiprocessor Scheduling:

Two basic approaches for scheduling real-time tasks on multiprocessor platforms are:

Partitioned Approach: In the partitioned approach, each task is assigned to a single processor statically and migration is not allowed. The set of tasks is partitioned into m subsets; each set is then assigned to a unique processor. As the tasks are not allowed to migrate, the multiprocessor scheduling is transformed into many uniprocessor scheduling problems.

Global Approach: In the global approach, tasks are allowed to freely migrate and execute on any processor. Store the tasks ready in one queue which is shared among all the processors. At every moment, the m highest priority tasks of the queue are selected for the m processors.

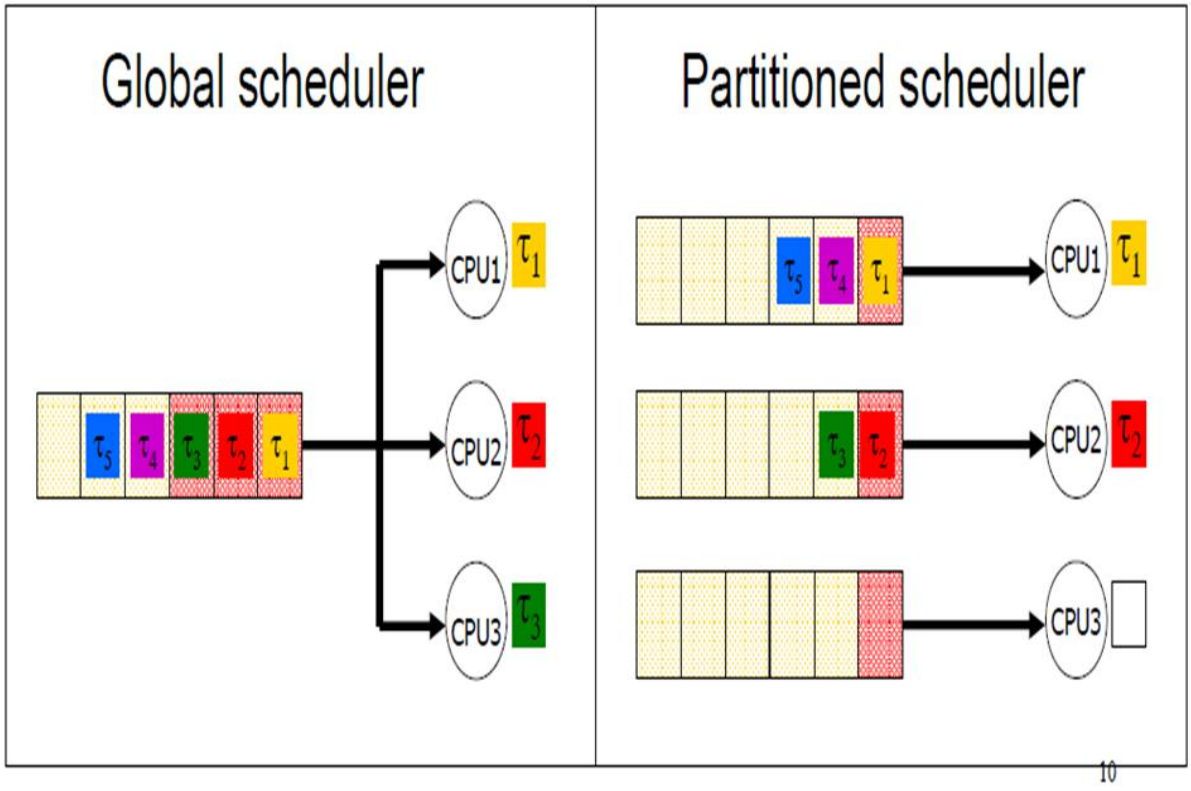

Figure 3.2: Global and Partitioned Scheduler in Multiprocessors.

Chapter 4

Proposed Work on RT-EMaCS :

Chapter 4

Proposed Work on RT-EMaCS

4.1 Introduction :

The goal of Event management and Coordinating system is to maximize the efficiency and reliability of the events scheduled even in cases when some changes in schedule are required. Static resource allocation is performed when applications are mapped in an offline planning phase i.e. mapping is used when a system is first started to ensure that all real time constraints will be met for a given initial system workload. Static mapping or resource allocation ensures that all real time constraints will be met for a given system workload before the system is put in operation . However real time systems operate in an environment that undergoes unexpected changes. Hence there is a need for a system that takes care of such unexpected changes.

4.2 Proposed Work- Real Time Event Management and Coordinating System (RT-EMaCS):

The objective of this system is to take care of the unexpected changes that occur in the real time environment. RT-EMaCS is an application that checks the schedulability of real time events in any real life onstage function or programme. It checks the feasibility of the events/programmes (i.e. real time tasks) even af-

ter the static scheduling is done once, i.e. it goes for a dynamic schedulability check during the run time after the initial static scheduling is done. RT-EMaCS is basically developed for the proper execution of the plans in any onstage function. Often it is seen that real life programmes do not execute as the way it is planned. Usually some or the other unexpected and unforeseen situations occur hampering the normal schedule of the programme. These may cause delay or need skipping of some events of the programme which might look chaotic or haphazard to the audience. RT-EMaCS is the appropriate assistance that helps avoiding such unforeseen situations.There systems that are often referred to as hard real-time systems, where real-time reflects the fact that they must directly interact with a changing physical environment and hard refers to the fact that at least some system functions must be performed within specific timing constraints [20]. However here, managing events in such real time occasions will be a soft real time system.

4.3 RT-EMaCS Model :

RT-ESaCS Model provides a platform to simulate the scheduled events of the programme, inquire whether the next event is ready to be executed or not and make the necessary changes in any case when the schedule cannot be met as planned. Since real time time tasks are those where both correct result and the time at which it is produced are important, programs that are decided in any real life function or programme are also real time tasks because the time by which the programs need to be complete is also important. Faults need to be identified before being tolerated [13].This model is an attempt to avoid any occurrence of fault by checking the details of upcoming programs rather than allowing a fault to occur and then going for fault tolerance. Again, this model can even help in fault tolerance if any fault occurs, by immediatedly resheduling.

RT-EMaCS is a Java based application which can run on any Java based platform such as laptops, desktops and any mobile device supporting Java, specially like android phones or tablets. The link between them can be done via Wi-Fi. In every programme different event responsibilities are been given to separate event

coordinators. During the running of the programme, there is need for the event coordinators to coordinate among themselves and adapt to various circumstances, usually unexpected ones. Hence if all event coordinators are provided with RT-EMaCS enabled devices, they get the advantage of the following features during their event coordination.

RT-EMaCS involves both static scheduling and dynamic scheduling. In the the beginning of programme,complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints, and future release times are known. As in dynamic scheduling, algorithm has complete knowledge of currently active tasks,but new task activations, not known to the algorithm when it is scheduling the current set, may arrive. Similarly, dynamic scheduling is required once the programme starts [21].

4.3.1 Features in RT-EMaCS :

Add new program:

The event coordinator has the option to add any new program to the schedule as and when required. Hence this makes it very user friendly and flexible.

Add sub-program:

The event coordinator can also make any changes or addition in the sub-programs.

Arranging Feature (Up and Down)

: Before the actual execution starts, the event coordinators have the facility to move the programs up or down in the schedule.

Postponeprogram

: If the event coordinator finds that if there is be some sort of delay, i.e. program execution is not ready, then the event coordinator can use the postpone feature

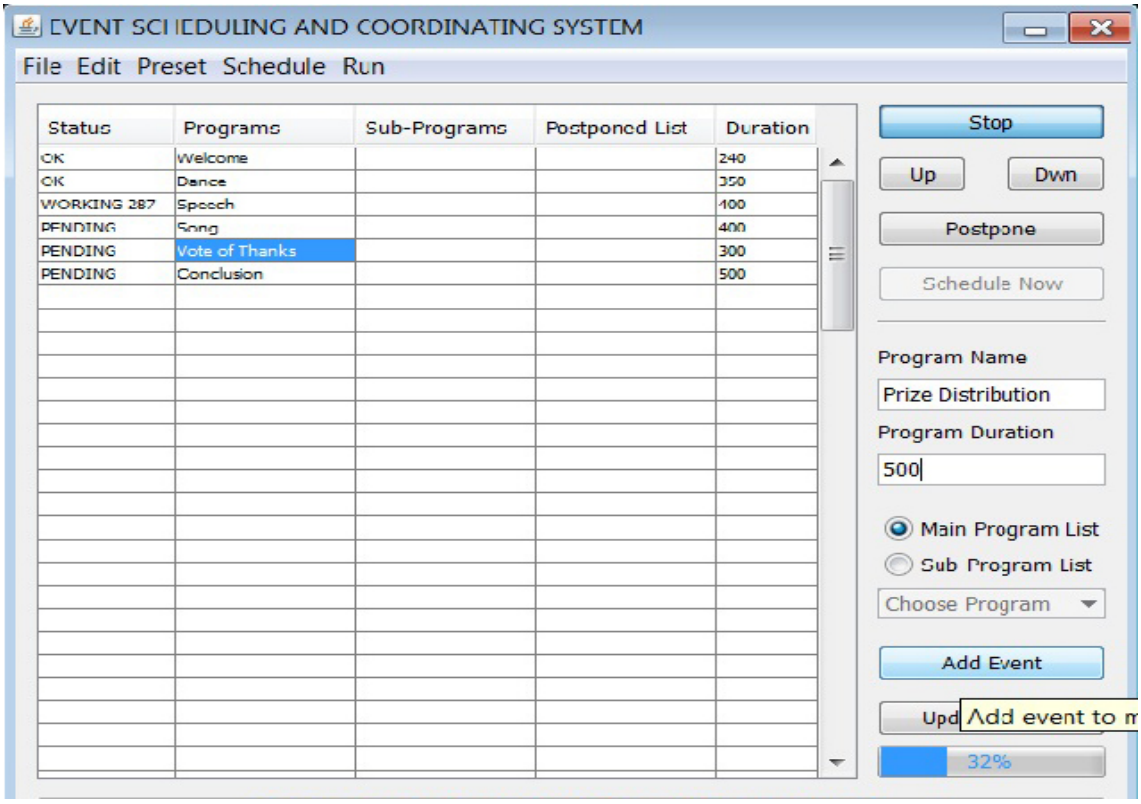

Figure 4.1: To Add A New Program And A Sub-Program.

the postpone that event.

Reschedule program:

The postponed programs can be rescheduled at the befitting time slot. The event coordination has options to reschedule them using real time mode (i.e. postponed program is scheduled immediately after the current running program.), End of List mode(i.e. the postponed program is scheduled as the last program), Default mode (i.e. there is no change in the schedule, program is added to the same slot as before.)

Skip program:

whenever there is need of completion of programme on time, the less priority events can be skipped.

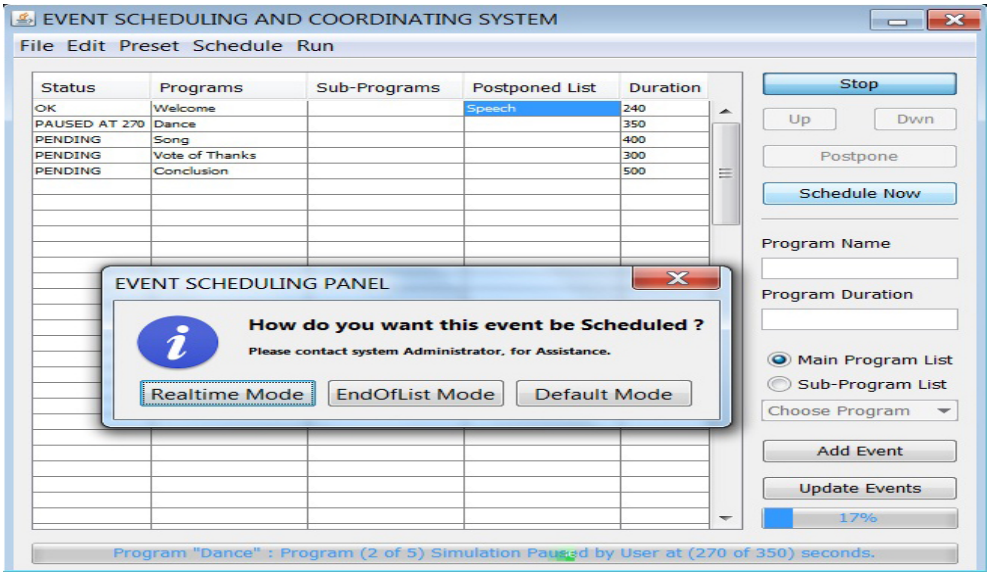

Figure 4.2: Reschedule By Pausing.

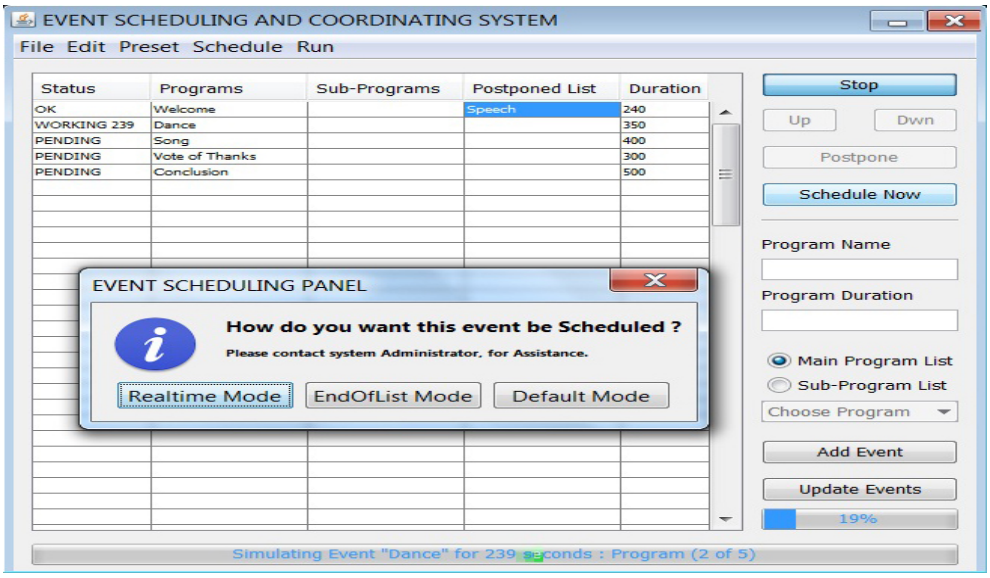

Figure 4.3: Reschedule while Simulation.

Pause simulation :

User can also pause the simulation in case any alteration of scheduling of programme is needed . Hence, it facilitates re-scheduling which prevents chaos.

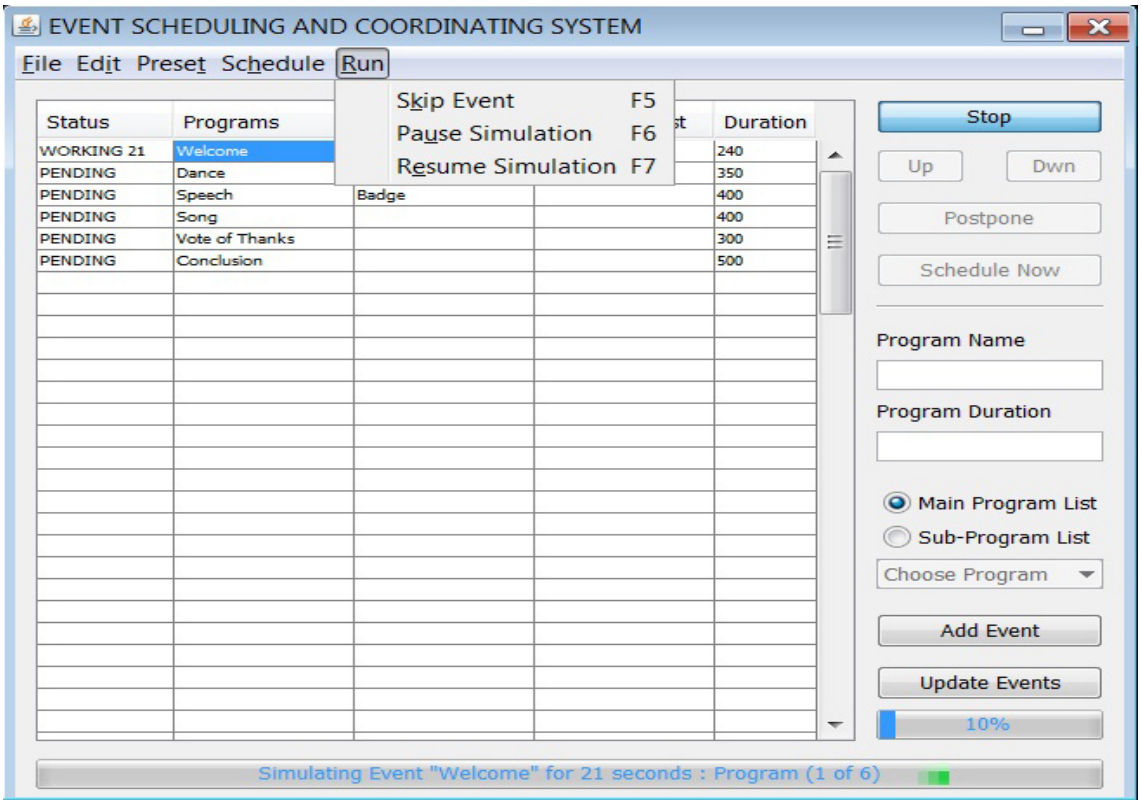

Figure 4.4: Skip a Program , Pause Simulation , Resume Simulation.

Chapter 5

Chapter 5

Conclusion and Future Work :

5.1 Conclusion

Use of RT-EMaCS will provide the event organizers and the event coordinators with better facilities in event management. It provides an easy, simple and better means of communication among one another. It prevents loss of time that would otherwise occur if any update had to be conveyed from one coordinator to another by means of physical movement. It also prevents from chaotic situations due to sudden changes in schedule. Use of RT-EMaCS does not require any complicated devices or instruments. Commonly available devices that support Java are suitable for running RT-EMaCS. Hence, it is cost efficient as compared to other methods. The features such as adding programs, removing programs, postponing and rescheduling are simple and easy to operate because of the friendly GUI.

5.2 Future Work

Our RT-EMaS has been modeled as distributed system, i.e. any event coordinator can made the changes or updates and a new schedule will then be circulator to all coordinators. This model can also be converted to a centralized system where all necessary updates are sent to a main coordinator and only he makes the

changes in the schedule and circulates to all other coordinators. Future work can be done in the field of automating the rescheduling of the programs. If a priority rule set is already fed in the application regarding the basic responses to any expected circumstances during the programme, then the application might need manual assistance only in cases when the interrupt(i.e. request for change) does match any rule provided in the rule set.

Bibliography

[1] J. W. Liu, "Real-time systems. 2000."

[2] R. Mall, *Real-Time Systems: Theory and Practice*. Prentice Hall, 2007.

[3] G. C. Buttazzo, "Rate monotonic vs. edf: judgment day," *Real-Time Systems*, vol. 29, no. 1, pp. 5–26, 2005.

[4] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *Software Engineering, IEEE Transactions on*, vol. 15, no. 12, pp. 1497–1506, 1989.

[5] H. Behera, S. Khuntia, and S. Nayak, "An improved least-laxity-first scheduling algorithm for real-time tasks," *International Journal of Engineering Science*, vol. 4, 2012.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[7] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Parallel Processing Symposium, 1995. Proceedings., 9th International*, pp. 280–288, IEEE, 1995.

[8] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, pp. 607–611, IEEE, 1994.

[9] S. Kato and N. Yamasaki, "Global edf-based scheduling with laxity-driven priority promotion," *Journal of Systems Architecture*, vol. 57, no. 5, pp. 498–517, 2011.

[10] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pp. 101–110, IEEE, 2006.

[11] D. Thakor and A. Shah, "D_edf: An efficient scheduling algorithm for real-time multiprocessor system," in *Information and Communication Technologies (WICT), 2011 World Congress on*, pp. 1044–1049, IEEE, 2011.

[12] H. Kim and Y. Cho, "A new fair scheduling algorithm for periodic tasks on multiprocessors," *Information Processing Letters*, vol. 111, no. 7, pp. 301–309, 2011.

[13] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 272–284, 1997.

[14] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[15] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[16] M. T. C. S. JIS, "Computers and intractability a guide to the theory of np-completeness," 1979.

[17] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," *Handbook on Scheduling Algorithms, Methods, and Models, pages*, pp. 30–1, 2004.

[18] A. Hangan and G. Sebestyen, "Rtmultisim: A versatile simulator for multiprocessor real-time systems," in *Proceedings of The 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, Pisa, Italy*, p. 15, 2012.

[19] O. U. P. Zapata and P. M. Alvarez, "Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation," *Seccion de Computacion Av. IPN*, vol. 2508, 2005.

[20] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Stress: A simulator for hard real-time systems," *Software: Practice and Experience*, vol. 24, no. 6, pp. 543–564, 1994.

[21] J. A. Stankovic, M. Spuri, M. Di Natale, and G. C. Buttazzo, "Implications of classical scheduling results for real-time systems," *Computer*, vol. 28, no. 6, pp. 16–25, 1995.