CHAPTER 1

INTRODUCTION

1.1 REAL TIME COMPUTING

A real-time computing system is expected to respond to stimuli within some small upper bound on response time and any late result is as bad as a wrong one. Thus, correctness of a real-time system could be stated true with logical perfection in the computational result and its timeliness. A soft real-time system is a system that has timing requirements, but occasionally missing the task deadlines have negligible effects. Hard real-time should meet the timing requirements of system, computations must always be met or the system will fail. Determinism, guaranteed worst-case interrupt latency and guaranteed worst-case context switch times characterize real-time operating systems, the software employed in real-time systems. Given these characteristics and the relative priorities of tasks and interrupts in the system, it is possible to analyze the worst-case performance of the software and hence the real-time characteristics of the system.

1.2 REAL-TIME SYSTEMS

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced. Examples of real-time system are command and control systems, process control systems, flight control systems, Space Shuttle avionics system, future systems
such as the space station, space-based defense systems, and large command and control systems. Such safety critical applications often involve the execution of several activities, each of these tasks having hard inviolable timing constraints. Hence, real-time system is a system for which the time at which output is produced is significant. The lag from input time to output time must be sufficiently small for acceptable timeliness. Real-time system requires determinism. The real-time systems (RTS) are broadly classified into two main categories.

**Soft Real-Time Systems:** A soft Real-time system is a system that has timing requirements, but occasionally missing the task deadlines have negligible effects. Examples of Soft Real-Time Systems include Multimedia, where dropping a few frames might be acceptable with the degradation in the performance being acceptable, and washing machine in which washing 10 seconds over time might occur once in a while.

**Hard Real-Time Systems:** Hard real-time systems are used to control physical processes that range in complexity from automobile ignition systems to controllers for flight systems and nuclear power plants. Such systems are referred to as hard real-time systems because the timing requirements of system computations must always be met or the system will fail. The scheduler for these systems must coordinate resources to meet the timing constraints of the physical system. This implies that the scheduler must be able to predict the execution behavior of all tasks within the system. This basic requirement of real-time systems is predictability. Unless the behavior of a real-time system is predictable, the scheduler cannot guarantee that the computation deadlines of the system will be met.

The requirement of predictability differentiates real-time systems from conventional computing environments and makes the scheduling solutions for conventional systems inappropriate for real-time systems.
Conventional computing environments, such as time sharing systems, assume that tasks and their resource and computation requirements are not known in advance. As such, no effort is made in these systems to determine, a priori, the execution behavior of the tasks. It is also the case in these systems that an unknown waiting time for computation results is acceptable. None of these assumptions hold for a real-time system. For a real-time system to be predictable, all tasks and their resource and computation requirements must be known in advance. Also, for hard real-time systems, unknown delays waiting for computation results are unacceptable. A scheduling solution that captures the timing behavior of the system is needed for a real-time system.

1.3 REAL-TIME TASKS

A task set can be defined as the collection of tasks that need to execute on a particular processor. Each task executes with its own processing and resource usage requirements. Tasks with regular arrival times are called periodic tasks. A common use of periodic tasks is to process sensor data and update the current state of the real-time system on a regular basis. Periodic tasks typically used in control and signal processing applications have hard deadlines. Tasks with irregular arrival times are aperiodic tasks. Aperiodic tasks are used to handle the processing requirements of random events such as operator requests. An aperiodic task typically has a soft deadline. Aperiodic tasks that have hard deadlines are called sporadic tasks. It is assumed that each task has a known worst case execution time.

- **Hard and Soft Deadline Periodic Tasks:** A periodic task has a regular interarrivial time equal to its period and a deadline that coincides with the end of its current period. Periodic tasks usually have hard deadlines, but in some applications the deadlines can be soft.
- **Soft Deadline Aperiodic Tasks**: An aperiodic task is a stream of jobs arriving at irregular intervals. Soft deadline aperiodic tasks typically require a fast average response time.

- **Sporadic Tasks**: A sporadic task is an aperiodic task with a hard deadline and a minimum inter-arrival time. Note that without a minimum inter-arrival time restriction it is impossible to guarantee that a sporadic task’s deadline would always be met.

In addition, various other parameters define a task such as its priority, deadline requirement, computation time, and resource usage specifications.

### 1.3.1 Task Characteristics

Two characteristics of tasks are considered to be of primary interest are criticality or importance and timing. However overall performance is also significantly affected by a number of other factors, including communication and synchronization of tasks mutual exclusion requirements, and fault tolerance.

Task importance is frequently a subjective issue, whereas timing is objective (at least, the values are objective even if they have been arrived at by subjective reasoning). These points apply to both periodic and aperiodic tasks. The essential timing attributes of tasks, as shown in Figure 1.1, are:

- Deadline ($T_D$)
- Worst-case computation time ($T_{cw}$) and for synchronous or periodic tasks
- Period ($T_p$)
Deadline is a relative information with respect to a task’s arrival. A task must finish its execution before its deadline. Worst-case computation time could be defined as the maximum time the task utilizes the processor execution for its completion. Figure 1.1 shows timing attributes for tasks.

![Diagram of task timing attributes]

**Task basic timing attributes**  **Periodic task – timing attributes**

**Figure 1.1 Timing attributes of real-time task**

### 1.3.2 Scheduling Criteria

Different CPU scheduling algorithms have different properties and may favor one class of processes over another. The properties of various algorithms are needed to be considered while selecting which algorithm to be used. Many criteria have been suggested for comparing CPU scheduling algorithms. The characteristic used for comparison can make a substantial difference in the determination of the best algorithm. The criteria includes the following
**CPU Utilization:** CPU utilization may range from 0 to 100 percent. In real-time systems it should be loaded from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).

**Throughput:** One measure of work is the number of processes completed per time unit called throughput. Throughput may be 1 process per hour for long transactions and 10 process per second for short transactions.

**Turnaround time:** The interval from the time of submission of the process to time of completion of that process is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into the memory, waiting to get into the ready queue, executing on the CPU and doing I/O.

**Waiting time:** The CPU algorithm doesnot affect the amount of time during which the process executes or does I/O it affects the amount of time that the process spends waiting in a ready queue. Waiting time is sum of the periods spent waiting in the ready queue.

**Response time:** In an interactive system turnaround time may not be the best criterion. Often a process can produce output fairly early, and can continue computing new results while the previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. The response time is the amount of time it takes to start responding, but not the amount of time it takes to output the response. The turn around time is generally limited to the speed of output device.
1.3.3 The Scheduler Object

The purpose of task scheduling is to organize the set of tasks ready for execution by the processor system (more precisely, to organize them so that performance objectives are met). Thus it is essentially an optimization problem. The order or arrangement of these tasks is called a schedule. For real-time embedded systems the primary criterion is to ensure that all tasks meet their deadlines. A schedule can be feasible or optimal. A feasible schedule orders tasks so that they all meet their deadlines. An optimal schedule is one which ensures that failures to meet task deadlines (i.e. late results) are minimized. A feasible task schedule is easier to attain with periodic rather than aperiodic tasks. Unfortunately scheduling optimization is particularly difficult to achieve in the presence of a mix of periodic and aperiodic tasks. Even more unfortunately this is the normal situation in hard embedded systems. Wherever possible, it makes sense to turn aperiodic tasks into periodic tasks. This can be done by replacing interrupt-driven operations with signal polling. Clearly there is a limit to what can be done because of the polling overhead, but this is a matter of individual system design. It is not sufficient to consider only the task timing attributes, task interactions must also be allowed. Thus communication and synchronization constraints must be taken into account owing to precedence and exclusion relations between the application tasks. Precedence relations exist between tasks when some tasks require information produced by others. Exclusion relations exist when some of them exclude others to prevent errors caused by simultaneous access to shared resources.

The scheduler is responsible for coordinating the execution of several tasks on a uniprocessor system and the scheduler may be preemptive or non-preemptive. To meet the timing constraints of the system, a scheduler
must coordinate the use of all system resources using a set of well understood real-time scheduling algorithms that meet the following objectives:

- Guarantee that tasks with hard timing constraints will always meet their deadlines.

- Attain a high degree of schedulable utilization for hard deadline tasks (periodic and sporadic tasks). Schedulable utilization is the degree of resource utilization at or below which all hard deadlines can be guaranteed. The schedulable utilization attainable by an algorithm is a measure of the algorithm’s utility. The higher the schedulable utilization, the more applicable the algorithm is for a range of real-time systems.

- Provide fast average response times for tasks with soft deadlines (aperiodic tasks).

- Ensure scheduling stability under transient overload. In some applications such as radar tracking, an overload situation can develop in which the computation requirements of the system exceed the schedulable resource utilization. A scheduler is said to be stable if under overload it can guarantee the deadlines of critical tasks even though it is impossible to meet all task deadlines.

One of the most common ways of ordering tasks in software is to connect them as a linked list. Their relative importance determines their position within the list, ie at the top having highest priority. When a context switch takes place, this one is loaded (dispatched) into the processor system. The scheduling problem is to decide exactly where tasks should be placed within the ready list. These decisions are made using a scheduling algorithm or policy which takes into account various task characteristic.
1.3.4 Task Scheduling Issues in OS and RTOS Environment

To better understand that the commonly seen bottlenecks in RTOS, some of the details are presented regarding their nature what they are, how they are implemented and how often they are invoked.

**Task Scheduling:** It is the process of determining which task should be running at any given time. The most commonly implemented approach for scheduling in commercial RTOS or other embedded operating systems is based on task priorities. Priority-driven schedulers assign each task a priority and execute the task with the highest priority that is ready.

**Time Management:** The RTOS gets its sense of time by a periodic interrupt generated every clock tick by a hardware timer. The rate of the generation of this clock tick is the system clock frequency which is not to be confused with the CPU clock frequency. Typically the system clock frequency is on the order of kHz while the CPU clock is on the order of MHz. Time management refers to the RTOS’s ability to allow tasks to use this mechanism to be scheduled at specific times, i.e, block the tasks for a specific time before becoming ready.

**Event Management:** Most real-time operating systems provide services for communication and synchronization between tasks known as inter-process communication (IPC). Examples are semaphores, message queues, etc. Event management involves keeping track of which tasks are blocked i.e. waiting for IPC and which tasks should be released, i.e., have to receive IPC on resource availability. Table 1.1 shows the overheads of RTOS.
Table 1.1 Overheads of RTOS

<table>
<thead>
<tr>
<th></th>
<th>Implementation/Overhead</th>
<th>Performed When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling</td>
<td>• Unsorted ready list – task selection has linear overhead</td>
<td>• System calls which change task status, e.g., priority Change. scales with workload size.</td>
</tr>
<tr>
<td></td>
<td>• Sorted ready list – Task selection has constant small overhead, task insertion on ready queue has linear overhead</td>
<td>• Preemptive System – After time interrupt. Scales with system clock frequency.</td>
</tr>
<tr>
<td></td>
<td>• Bit-vectors – Constant overhead implementation for systems with unique and static priorities.</td>
<td>• NonPreemptive System – Task Completion Points, after Timer Interrupt if System is idling. Scales with workload size.</td>
</tr>
<tr>
<td>Time Management</td>
<td>• Each task has individual delay counter containing its absolute delay. Timer Update requires updating each counter and has a linear overhead</td>
<td>• Preemptive System – After timer interrupt. scales linearly with system clock frequency.</td>
</tr>
<tr>
<td></td>
<td>• UNIX callout table. Delay counters hold delays relative to that of previous element. Updates have variable overhead.</td>
<td>• Non-Preemptive System – Task Completion Points, After Timer interrupt if system is idling. scales with workload size.</td>
</tr>
<tr>
<td>Event Management</td>
<td>• Similar to task scheduling with each event having its own Blocked Queue</td>
<td>• Tasks use IPC. dependent on extent to which workload uses IPC.</td>
</tr>
</tbody>
</table>

Note that the overhead of the RTOS due to these operations is proportional to the product of frequency and complexity. Since both of these components may increase linearly with the number of tasks, there may be a
quadratic relationship between the number of tasks in the system and the RTOS overhead due to them. In addition, the overhead of task scheduling and time management increases linearly with the system clock frequency. The longest time that it takes to perform any critical section of code (including task scheduling, event or time management) adds to the maximum response time in preemptive systems. However, in non-preemptive systems where interrupts are polled, a response task is delayed more by executing workloads than by RTOS operations. This is because workloads execute for longer periods than RTOS operations and thus disable interrupt response for longer periods.

1.4 BASICS OF HARDWARE

In the early days of real-time computing, the fundamental requirements of timeliness and simultaneity were realized by the user. Within the user application software, user explicitly synchronized the execution schedules of the various tasks with a basic clock cycle. To this end, user usually wrote the own organization program, a so-called cyclic executive. Thus predictable software behavior was realized, and the observation of the time conditions was guaranteed. Later, this method of cycle synchronized programming was replaced by the more flexible approach of asynchronous multiprogramming, which was based on the task concept. Tasks could be activated and run at any time, asynchronously to a basic cycle. The flexibility and conceptual elegance of the method were generally gained at the expense of renouncing predictability and guaranteed time conditions.

During the last decade the discipline of real-time systems design considerably gained research interest in many of its domains. However the question of internal organization of process control computer systems which would provide a necessary basis to fulfill the requirements set upon real-time systems has not been consistently dealt with until recently. Rather, it was
considered advantageous and appropriate to employ conventional Von Neumann computers. The only adaptations to the real-time application area were manifested by including process peripherals and externally available interrupt lines. All other real-time requirements were met by software, viz, namely by operating systems and by careful application programming. The problems identified in real-time systems design, however, could generally not be solved in this way.

RTOS and Embedded OS have become extremely important to the development of real-time systems as reflected by the growing market for RTOS and Embedded OS. RTOS and Embedded OS provide services for better hardware abstraction, multitasking, task synchronization, etc. However, the benefits that RTOS and Embedded OS provide do not come for free. The use of RTOS and Embedded OS has several important effects on various performance parameters of real-time systems.

Processor Utilization: The fraction of processing power that an application is able to use is processor utilization. In order to provide services like multitasking, preemption and numerous others, the RTOS introduces an overhead which lowers the processing power available to the application.

Response Time: The time it takes for a real-time system to response to external stimuli is response time. This delay is highly dependent on several factors, including whether or not the RTOS is preemptive and whether polling or interrupts are used to sense the stimulus. The effects of RTOS and Embedded OS on response time vary widely, but, in general, RTOSes and Embedded OS increase response time to varying degrees.

Much of the performance degradation caused by RTOS and Embedded OS can be traced to their core operations, namely task scheduling, time management, and event management. Analysis of these functions reveals
that they are executed frequently, that they perform inter-related actions and that these actions exhibit parallelism. This parallelism cannot be exploited by software-based implementations. However, hardware-based solutions can exploit this parallelism and lower the performance degradation. Also, the rapidly dropping cost of logic has made it possible to put custom hardware for enhancing frequently executing operations on embedded processors, without increasing costs significantly.

These factors suggest that RTOS and Embedded OS would greatly benefit from a custom hardware solution. This is the motivation for the Realtime task Scheduler co-processor, a memory-mapped on-chip peripheral designed to optimize task scheduling, time management, and event management. Measurements are taken of the performance impact of the RTM on models of realistic real-time systems. These measurements show that processor utilization and maximum response time are both reduced by an order of magnitude.

1.5  PROBLEM FORMULATION

1.5.1  Need for Scheduler Simulator

From design perspective, real-time systems can be approached from different views. As an example, engineers prefer to deal with hardware control while computer scientists prefer to deal with the system modeling. The system modeling will explain how to model task interactions and how to allocate processor time for each task. The system modeling is burdensome because there are many different scheduling policies and the scheduling problem is known to be very complex. As a consequence of complexity, most learners feel that the scheduling theory is only a collection of rules that have to be memorized. Therefore much of the attention is not paid to the fact that the most important concept is not the exact description of a rule but what kind
of conditions and problems are better suited for each rule. The aforesaid misunderstanding can be solved, assigning jobs that require not only the resolution of a schedule but the experimentation with the problem. Although this can be done by hand, it has limitations due to the exponential growth in the resolution time with the problem size. The use of simulation technique would thus help circumvent this issue.

1.5.2 Review of Literature

Barbara Korousic-Seljak (1994) explained a variety of task scheduling policies for use with real-time embedded system, including commonly used practical strategies as well as new ideas from the research community. The performance of these policies is compared using simulation techniques and also multitasking system, Real time executives and RTOS are explained.

Bini and Buttazzo (2004) explored the factors that affect the schedulability of fixed priority task and proposed the way for improving to increase the schedulability of task in fixed priority system using polynomial time test and exact time test.

Important classical scheduling theory results for real-time computing were identified by Stankovic et al (1995). Implications of these results from the perspective of a real-time system designer were discussed. Uniprocessor and multiprocessor results were addressed and important issues such as future release times, precedence constraints, shared resources, task value, overloads, static versus dynamic scheduling, preemption versus nonpreemption, multiprocessing anomalies and metrics were discussed.

Sha et al (2004), using an example consisting of a single server and a single hard real-time task showed that existing worst-case response time
analysis can be improved for deferrable servers and sporadic servers when a
server is exclusively used for hard realtime task. A detailed description about
fixed priority and dynamic priority, soft real-time and feedback scheduling
was explained by the authors. Ramamritham and Stankovic (1994) discussed
four paradigms underlying the scheduling approaches and presented several
examples of each. The four paradigms were: static table driven scheduling,
static priority preemptive scheduling, dynamic planning-based scheduling,
and dynamic best effort scheduling. In the operating system context, the most
of the proprietary commercial kernels as well as real-time extensions to time-
sharing operating system kernels do not fit the needs of predictable real-time
system. They also discussed several research kernels that are currently being
built to explicitly meet the needs of real-time applications.

The problem of jointly scheduling both hard deadline periodic task
and soft a periodic task has been the subject of considerably research in real-
time system. One of the most widely accepted solutions for this problem are
slack stealing algorithm. However, these algorithms are rather impractical,
since they all imply a considerably scheduler overhead. Sergio et al (1999)
introduced a complete hardware architecture that implements slack stealing in
hardware using an optimal algorithm that had been completely redesigned to
perform efficiently in hardware. The proposed solution was a circuit that
behaves as a kind of sophisticated interrupt controller that takes the task
workload and the interrupts as inputs, and provides an output to inform the
CPU which is the highest priority task. Middonet et al (2010) demonstrated
an approximate slack stealer algorithm for real-time Embedded System.

Real-time operating system kernels in embedded system need to be
configurable. Unfortunately many of today's commercial real-time kernels are
monolithic. These optimized code packages are difficult to change and
maintain. This was motivated mainly to achieve short response time and easy
access of debugging information. To solve the drawbacks of the monolithic real-time kernels, the microkernel structure was introduced. The microkernel approach has been discussed for having disadvantages regarding performance. Nordstrom et al (2005) developed and explained a prototype that showed how a modular microkernel architecture in hardware could be used to speed up task management, semaphores and flags for a commercial real-time kernel. Liu and Layland (1973) discussed the problems involved in multi-program scheduling on a single processor and showed that full processor utilization could be achieved by dynamically assigning priorities on the basis of the tasks deadlines.

Lehoczky et al (1989) presented an exact characterization of the ability of rate monotonic scheduling algorithm to meet the deadline of periodic task set. They also presented a stochastic analysis that gives the probability distribution of the breakdown utilization of randomly generated task sets.

Generalized rate-monotonic scheduling theory is a recent development that has had large impact on the development of real-time system and open standards. Sha et al (1994) provided an up-to-date and self-contained review of generalized rate-monotonic scheduling theory. This theory can be applied in practical system development to facilitate concurrent development by geographically distributed programming teams. The existing hardware and software components can also be reused. Manabe and Aoyagi (1995) discussed feasibility decision for a given real-time task system by the rate monotonic scheduling algorithm. Naghibzadeh (2002) established modified version of rate monotonic algorithm called delayed rate monotonic algorithm that used two ready states. He also compared the rate monotonic and delayed rate monotonic (DRM). It is shown that there DRM algorithm is safer than RM algorithm. Beitollahi and Deconinck (2006) presented a
scheme to add enough and efficient time redundancy to the Rate Monotonic (RM) scheduling policy for periodic real-time task. This scheme can be used to tolerate transient fault during the execution of task.

Earliest deadline first (EDF) has become one of the most promising scheduling schemes for providing quality-of-service differentiation over high speed networks. Zhi Quan and Chung (2003) developed an analytical framework for estimating the loss probabilities for the aggregated traffic and the individual flows. This enabled them to determine whether a given flow could meet its deadline with the required loss probability. The problem of scheduling periodic time critical task on a monoprocessor system and some of the results obtained are discussed by Chetto and Chetto (1989). Zhang and Burns (2007) presented an exact and efficient schedulability test for the application task based on the capacity demand criterion when the global scheduler could be Fixed Priority or EDF. For embedded system with hard real-time deadlines, it is important to be able to verify that all of the tasks will meet their deadlines is presented by Dong-zhi He et al. A missed deadline could result in catastrophic behavior of the system. Morton and Loucks (2007) presented a way to represent tasks using accelerators. The critical section is used to extend processor demand analysis to tasks that use an accelerator during execution. Dey et al (1996) focussed on the problem of on-line scheduling of a random arrival sequence of IRIS task on a single processor with the goal of maximizing the average reward accrued per task and per unit time. They described and evaluated several policies for this system through simulation. Further the authors observed that the best performance is exhibited by a two-level policy where the top-level algorithm is responsible for allocating the amount of service to task. They also noted the bottom-level algorithm, using the earliest deadline first rule, is responsible for determining the order of tasks executed.
Audsley et al (1993) discussed the application of Deadline Monotonic Scheduling Theory (DMST). This theory is an extension of the more familiar approach based on rate monotonic priority assignment. The model presented can accommodate periodic and sporadic processes, different levels of criticality, process interaction and blocking, precedence constrained processes and multi-deadline processes. It is particularly well integrated with the use of Immediate Priority Ceiling Inheritance for control over process blocking. A basic pseudo-polynomial schedulability test is outlined and then supplemented by the introduction of offsets to control jitter, and period transformation to enable critical (hard) processes to be "protected" during potential transient overloads. These mathematical techniques derived within DMST can help designers experiment with alternative formulations and prove essential properties of system before they are deployed. Deadline monotonic scheduling is used to schedule a set of tasks in a multiprocessor system, Fault-tolerance is achieved by using a combined duplication technique where each task scheduled on a processor has either an active or a passive copy scheduled on a different processor. Bertossi et al (1997) conducted simulation experiments that reveal a saving of processors with respect to those needed by the usual approach of duplicating the schedule of the non-fault-tolerant case.

Roehi (1995) proposed a new way of scheduling that implements a new priority queue in the round robin algorithm that gives priority to tasks with short central processing unit burst thereby improving the performance of the task with less CPU burst. Fair scheduling with tunable latency by Ecker (1996) is a round robin approach that proposed an alternative and lower complexity approach to packet scheduling, based on modifications of the classical round robin scheduler. He showed that appropriate modifications of the Weighted Round Robin service discipline can, in fact, provide tight fairness properties and efficient delay guarantees to multiple sessions. Round robin approach has its applications on networks by allowing the network
devices to have a free share of network resources. Various types of scheduling algorithm such as Deficit Round Robin by Shreedhar and Varghese (1996), Deficit Round Robin alternated by Arco J.M(2005) and credit round robin do and Yun (2003) have been implemented. Kunis and Rünger (2010) proposed a block identification algorithm which identified suitable blocks of tasks in arbitrary layer-schedules and, thus, allowed the application of the movement of blocks to a wide range of layer-based scheduling algorithms.

Scheduling theory generally assumes that real-time systems are mostly composed of activities with hard real-time requirements. Many systems are built today by composing different applications or components in the same system, leading to a mixture of many different kinds of requirements with small parts of the system having hard real-time requirements and other larger parts with requirements for more flexible scheduling and for quality of service. Hard real-time scheduling techniques are extremely pessimistic for the part of the application, and consequently it is necessary to use techniques that let the system resources be fully utilized to achieve the highest possible quality. Aldea et al (2006) provided the ability to compose several applications or components into the system, and to flexibly schedule the available resources while guaranteeing hard real-time requirements.

Diaz et al (2007) described Realtss, an open source real-time scheduling simulator to simulate real-time scheduling algorithm without the need of implementing them in a RTOS. Sarkar (2011) presented a mechanism on Partition-Oriented ERfair Scheduler that achieves lower overhead using an online partitioning/merging mechanism. This enables the scheduler to retains the optimal schedulability of a fully global scheduler by merging processor groups as resources become critical while using partitioning for fast scheduling at other times.
An ADA framework called cheddar which provides tools to check if the real time application meets its temporal constraints was explained by Singhoff et al (2004). The framework is based on real time scheduling theory and is mostly for educational purposes.

The growing complexity of embedded applications and pressure on time-to-market have resulted in the increased use of embedded real-time operating system. Unfortunately, RTOSes can introduce a significant performance degradation. Kohou et al (2003) presented Real-Time Task Manager (RTMba) processor extension that minimizes the performance drawbacks associated with RTOS. The RTM accomplishes this by supporting, in hardware, a few of the common RTOS operations that are performance bottlenecks: task scheduling, time management, and event management. By exploiting the inherent parallelism of these operations, the RTM completes them in constant time, thereby significantly reducing RTOS overhead. Manacero et al (2001) exposed a scheduler simulator for real-time tasks, RTsim, that can be used as a tool to teach real-time scheduling algorithm. It simulates a variety of preprogrammed scheduling policies for single and multiprocessor system and introduces simple algorithm variants.

Cooling and Tweedale (1997) presented the design development and use of a dedicated microcontroller based task scheduler. Jeahwan Lee et al (2003) showed the performance comparison and analysis result among three RTOS.

A new scheduling framework called YASA has been presented by Golatowski et al (2002). They worked towards a rapid prototyping system for hard real-time system focusing on scheduling algorithm and scheduler implementations. The developed framework aims at speeding up the decision making process during selection of a suitable scheduling algorithm for a real-
time application. The framework supports various kinds of realtime scheduling algorithm that can be simulated for evaluation purposes.

The dynamic behavior of multitasking system is complex and hard to visualize. De Beer and Fidge explained how a simple multitasking simulator was constructed by representing computational model used by the simulation theory in a commercial simulation toolkit. The resulting simulator allows complex multitasking behaviors to be displayed graphically. Kravetz et al (2001) examined the scalability of the Linux 2.4.x scheduler as the load and number of CPUs increases. They showed that the current scheduler design involving a single runqueue and lock from lock contention problems limits its scalability. They presented an alternative design using multiple runqueues and priority levels that can reduce lock contention while maintaining the same functional behavior as the current scheduler. Bahga and Madisetti (2011) proposed the design of resource manager and master scheduler for the open close environment that allows efficient realization of multiple applications within a multi tasked platform.

Vroey et al (1996) described a language for defining scheduling algorithm for hard real-time system and a tool to simulate the behavior of such system on a predefined task set. Cheng and Ras (2007) presented some of these features, which are implemented using the Priority Ceiling Protocol (PCP). PCP is an extension of the Priority Inheritance Protocol (PIP), with the added features of preventing deadlocks and priority inversions. Priority inversion occurs when a high priority task is blocked for an unbounded time by lower priority task. They implemented PCP in ADA 2005. Unfortunately, PCP is not known to be supported by any operating system. A detailed discussion of the protocol, and other related issues, are presented by authors. A common approach to characterizing hard real-time task with repetitive requests is the periodic task model. In the periodic task model, every task
needs to be executed once during each of its periods. Han et al (1996) proposed the preemptive distance-constrained task system model which can serve as a more intuitive and adequate scheduling model for “repetitive” task executions. They designed an efficient scheduling scheme for the model, and derived a schedulability condition for the scheduling scheme.

Impulse C is a high level language widely used in Hardware/Software Co-Design. It provides users with various Hardware Software Communication mechanisms. Xi Jin et al. (2010) presented an improved implementation of shared memory communication in Impulse C. Sabry et al (2010) proposed a compilation flow that utilizes the register windows to optimize the thermal profile and to reduce the hot spots.

Jian Huang and Jooheung Lee (2011) developed a reconfigurable Architecture to support zero quantized discrete cosine transform (ZQDCT) and a hybrid model based quality priority algorithm to reduce power consumption, required hardware resources, and consumption time. Shenoy et al (2010) implemented several profiling methods for dynamically monitoring sensor-based platforms and analyzed the associated network traffic, energy and code impacts. LI Jie et al (2010) analyzed distributed real-time scheduling algorithms such as GRMS and DSr and pointed out the future direction of real-time scheduling research.

Semantic Web Services propose the use of ontologies that provide semantic descriptions to traditional Web Services. Thus, it becomes possible to create mechanisms that can reason about these ontologies and interpret the service goals and how to use it. Therefore, the use of Semantic Web Services allows automating the processes of discovery, composition and invocation of services. However, service consumers must be concerned with some factors that may compromise the reliability of the system, as an unavailable service or a service that produces incorrect results. So applications that use services
should provide mechanisms that can ensure reliability in face of these problems. Barros (2011) proposed a middleware that provided the reliable use of semantic services, providing a mechanism for fault tolerance and support for restrictions to evaluate the correctness in the results of services.

Denial of Service (DoS) attacks represent, in today’s Internet, one of the most complex issues to address. A session is under a DoS attack if it cannot achieve its intended throughput due to the misbehavior of other sessions. Many research studies dealt with DoS, proposing models and/or architectures mostly based on an attack prevention approach is presented by Ming Luo et al (2008). Cotroneo et al (2002) analyzed the fundamental requirements to be satisfied in order to protect hosts and routers from any form of Distributed DoS (DDoS). Then they proposed a network signaling protocol, named Active Security Protocol (ASP), that satisfied most of the defined requirements. ASP provides an active protection from a DDoS attack, being able to adapt its defense strategies to the current type of violation.

Khan Suraiya and Traore Issa (2005) qualitatively and quantitatively analyzed the impact of DOS attacks on three simple system parameters - request arrival rate, queue growth-rate, and response time. Swing tries to protect servers from attacks by using a new strategy. In the mechanism, when a DoS attack is detected, the server will automatically change its address to get rid of the attack. Meanwhile, existing connections from normal clients will be kept using an address-switch protocol like shim6. A p2p network is included in the mechanism to help clients establish new connections to the server under attack situations, and side equipments are deployed near the server to monitor and reshape the network flow.

Yao Zhao et al (2009) identified a practical way to launch DoS attacks on security protocols by triggering exceptions. Through experiments, the authors showed that even the latest strongly authenticated protocols are vulnerable to these attacks.
Denial of Service (DoS) attacks constitute one of the major threats and among the hardest security problems in today's Internet. Distributed Denial of Service (DDoS) attacks, whose impact can be proportionally severe. With little or no advance warning, a DDoS attack can easily exhaust the computing and communication resources of its victim within a short period of time. Because of the seriousness of the problem many defense mechanisms have been proposed to combat these attacks.

Karthik et al (2008) provided an understanding of the existing attack methods, tools and defense mechanisms, the goal of the work is to simulate an environment by extending Network Simulator2, setting attacking topology and traffic to evaluate and compare the methods of DDoS attacks and tools. Based on the simulation and evaluation results, more efficient and effective algorithm, techniques and procedures to combat these attacks may be developed. Aye (2009) explored the use of overlay proxy networks to protect applications against DoS attacks.

Tackling the challenge of distinguishing legitimate traffic from attack would aid in the detection of Denial of Service (DoS) / Distributed DoS (DDoS) attacks. Spoofing of source address would further harden the detection of such attacks. Malliga et al (2008) proposed a flow based scheme to detect the DoS attacks. This scheme adapt itself to the changes trends of the current traffic. Their system weeds out most of the spurious traffic at the source end, thus avoiding clogging of the target and the network. Their scheme distinguishes itself from other source end defenses, which use statistics to gather profiles. Information entropy, a measure to find correlation among traffic flows, is then used. Maciá-Fernández, et al. (2009) proposed a mathematical model for the LoRDAS attack. This model allowed them to evaluate its performance by relating it to the configuration parameters of the attack and the dynamics of network and victim. The model is validated by
comparing the performance values given against those obtained from a simulated environment. Finally, the experience of the model suggests some recommendations for the challenging task of building defense techniques against this attack.

TCP is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP). TCP has drawbacks in its connection establishment for possible DOS attacks. TCP maintains the state of each partly established connection in its fixed size queue until it is fully established and accepted by the application. The attackers can make the queue full by sending connection requests very much and not completing the connection establishment steps for those requests. Zin-won Park et al (2003) designed and implemented an extended TCP for preventing the DOS attacks. In the extended TCP, the state of each partly established connection is not maintained in the queue. The authors modified the 3-way handshake procedure of TCP, and implemented the extended TCP in Xinu operating system. It has been proven in theory that a low-rate TCP-targeted Denial-of-Service (DoS) attack is possible by exploiting the retransmission timeout mechanism of TCP. In contrast to most DoS attacks, this exploit requires periodic, low average volume traffic in order to throttle TCP throughput. Consequently this attack is hard to detect and prevent, since most DoS detection systems are triggered by high-rate traffic.

Efstathopoulos (2009) studied the results of the attack on a real system (Linux), and evaluated the effectiveness of RTO randomization in defending against low-rate TCP targeted DoS attacks, showing that the method can prevent a TCP flow from being throttled from attack traffic. Meigen Huang et al (2009) developed an improved send protocol against DoS attacks in mobile IPV6 environment.
Firewalls are considered as the first line of defense in any network. An attacker may use probing to learn a firewall’s policy, and then launch a DoS attack that floods the firewall with traffic targeting the rules at the bottom of this policy. Al-Haidari et al (2009) proposed a countermeasure that enables the firewall to endure the attack attempts without denying service to legitimate clients. The goal of their work is to use an entropy-based scheme to distinguish between the legitimate and attack traffic. Then, the legitimate traffic will be placed in a queue with a higher priority than the queue holding the attack traffic. Their results show that the proposed scheme improves on the performance of the firewall under a DoS attack. Defending against DoS attacks is extremely difficult; effective solution probably requires significant changes to the Internet architecture. Habib and Roy (2009) presented a series of architectural changes aimed at preventing most flooding DoS attacks, and making the remaining attacks easier to defend against. Haining Wang and Srinivasan (2010) proposed a hybrid mechanism to effectively mitigate DoS attack.

The growing popularity of the 802.11-based Wireless LAN (WLAN) also increases its risk of security attacks. Liu and Yu (2008) presented an analysis and solution to two DoS attacks: Deauthentication flooding (DeauthF) and disassociation flooding (DisassF) attacks. They conducted experiments to understand the impact of the attacks, and applied the Markov chain model to study the transition probabilities under attacks. They then followed the newly proposed 802.11w standard and implemented a solution to prevent the attacks and their results showed that 802.11w is effective for low rate deauthentication and disassociation attacks but fails to protect against the flooding attacks because it takes significant resources to authenticate frames. They also proposed an integrated approach to applying traffic pattern filtering (TPF) over 802.11w to resolve DeauthF and DisassF DoS attacks. Their simulation results yielded satisfactory performance for the integrated approach.
Voice over IP (VoIP) telephony is vulnerable to a range of attacks, since its operation relies on the underlying IP network. Huang et al (2009) proposed a history-based IP filtering layer to defeat these DoS attacks by blocking the Session Initiation Protocol (SIP) packets from previously unseen sources. Their empirical evaluation shows that their approach achieves significant improvement in CPU utilization under DoS attacks. Users of IP multimedia subsystems (IMS) usually access the system from IMS providers own fixed or mobile IP access network but the providers also allow the users to access the IMS system from the Internet. With connection to the Internet the IMS system is exposed to DoS attacks that represents the real security threat to the IMS systems. DoS attacks can be carried out with the various types of packets that produce different impact on the targeted IMS systems. Priselac et al (2010) suggested a protection mechanism on reduction of Dos attacks on IMF systems.

As more and more critical services are provided over the Internet, the risk to these services from malicious users is also increasing. Several networks have witnessed DoS attacks over the recent past. Garg and Reddy (2002) reported their experience of building a Linux-hared prototype to mitigate the effect of such attacks. Their prototype provides an efficient way to keep track of server and network resources at the network layer and allows aggregate resource regulation and their scheme provides a general, and not attack specific, mechanism to provide graceful server degradation in the face of such an attack. They reported on the rationale of their approach, the experience in building the prototype, and the results from real experiments.

DoS attacks play a significant role among all the network security issues today. Xiangbin Cheng et al (2008) presented a mechanism to limit the effectiveness of DoS attacks. Inspired by the address-switch conception of the newly proposed shim6 protocol, Swing tries to protect servers from attacks by
using a new strategy. In the mechanism, when a DoS attack is detected, the server will automatically change its address to get rid of the attack. Meanwhile, existing connections from normal clients will be kept using an address-switch protocol like shim6. A p2p network is included in the mechanism to help clients establish new connections to the server under attack situations, and side equipment are deployed near the server to monitor and reshape the network flow. Li (2003) investigated the statistical anomaly detection of DoS Computer network attacks Using MIBII supplied traffic parameters of the SNMP, as carried out by MAID. MAID is a hierarchical, multi-tier, multi-observation-window, anomaly-based network intrusion detection system, prototyped in the laboratory for the US Army's Tactical Internet.

Li Wang et al (2004) presented a robust mechanism called charge point monitoring (CPM) to detect denial of service (DoS) attacks. The core of CPM is based on the inherent network protocol behaviors and is an instance of the Sequential Change Point Detection. To make the detection mechanism insensitive to sites and traffic patterns, a nonparametric Cumulative Sum (CUSUM) method is applied, thus making the detection mechanism robust, more generally applicable, and its deployment much easier. Stephen Frechette (2005) examine the effectiveness of a circulant graph proxy-network overlay topology. The author evaluate the selected topology’s fault-tolerance and its effectiveness in the resistance of DoS attacks.

1.5.3 Objectives of the Thesis

Even though a lot of scheduling algorithms have been developed, just a few of them are available to be implemented in real-time applications. In order to use, test and evaluate scheduling policy it must be integrated into an Operating System environment complex task. Simulation is another alternative to evaluate a scheduling policy. Unfortunately, just a few real-time scheduling simulators have been developed and most of them require the use of a specific simulation language.
The motivation of this thesis is to develop a new Real-Time Scheduler Simulator and a Task Scheduler Co-Processor are proposed in this research which can extensively be used as a Teaching Tool. The objectives of this research are:

- To simulate the scheduling of real-time tasks and to execute various real-time scheduling algorithms for tasks in real-time environment
- To study the performance of various traditional real-time scheduling policies and to submit a performance analysis report to ease the selection of real-time scheduler for a given system and to create a graphical user interface.
- To develop a hardware setup to practically demonstrate the scheduling of real-time tasks.
- To develop the architectural model for web-based simulator to prevent DoS attacks
- To develop new scheduling algorithms for real-time environment and make a comparison with the existing algorithm

1.6 OUTLINE OF THE THESIS

The thesis titled “Design and Development of Web based Real Time Scheduler Simulator for Real Time Embedded Systems” delineates the development of a Web- enabled model that can be used as teaching tool for learning the execution patterns of various scheduling algorithms used and interaction with the user in web environment. This dissertation is divided into seven chapters, including the introduction.
Chapter 1 deals with the basic concepts, literature survey and the objectives of the research. The need for the web-based simulator is also explained in this chapter.

Chapter 2 discusses various types of schedulers and types of scheduling algorithms available in operating systems and real-time systems. It also explains the existing simulators and drawbacks of existing simulators, various types of scheduling algorithms in operating systems and real-time operating systems. Further, it gives the details about various types of DoS attacks that can be performed in the server

Chapter 3 presents with the development of scheduler simulator and the basis co processor. The simulator is designed in both LaBVIEw and Java environment. The proposed simulator is implemented in distributed environment through web. This chapter also makes the comparison between existing and proposed simulator and highlights the advantages of the proposed simulator when implemented in a distributed environment.

Chapter 4 describes the backend hardware architecture. The backend is made up of both 8051 and ARM LPC 2148 processors. A detailed step by step hardware implementation is given in this chapter

Chapter 5 is concerned with DoS attacks. This attack uses all the system resources thereby enabling the files on the website becoming inaccessible to the legitimate user. To avoid such type of attacks, a secured architecture which uses location hiding approach to protect the server thereby preventing DoS attacks is developed and explained in this chapter.

Chapter 6 explains the development of new scheduling algorithms for Real-time systems. A comparison is made between the existing algorithms and the proposed algorithms. The advantages of proposed algorithms over the existing algorithms are listed in this chapter.
In chapter 7 a review of work reported, major conclusions reached and contributions made are dealt with. Recommendations for further research are also stated.

1.7 CONCLUSION

A brief survey of the techniques and methods available in the literature for effective implementation of Design and Development of Web-based Real Time Scheduler Simulator for Real Time Embedded Systems has been discussed. The motivation for the present work has been brought out. The organization of the thesis has also been presented briefly.