CHAPTER 1

INTRODUCTION

Realtime Operating System (RTOS) supports applications that meet deadlines in addition to providing logically correct results. Workstation and personal computers are increasingly used for applications with realtime characteristics to suit human computer interaction, media computations and often concurrently executed with traditional non-realtime workstation. In realtime applications, scheduling algorithms are used for systems that are extremely constrained. Most of the realtime systems are designed using priority based preemptive scheduling and worst case execution time is estimated to guarantee the execution of high priority tasks. To meet realtime constraints for scheduling a task, different scheduling algorithms are used in multitasking. Although sharing of resources among multiple tasks simplify the exchange of information, each task should have exclusive access to the resource in order to remain robust against security threats and avoid data corruption and contention. Protecting resources in computing environments has long been recognized as a difficult and daunting problem. Three important factors influencing optimized resource scheduling are:

- Time driven resource management to meet scheduling requirements
- Problem specific OS facilities
- Integrated system wide scheduling support

Hardware implementation of shared resource protection with security service enhancement remains unexplored and active research work is focused only on effective handling of priority based scheduling algorithms to meet desired requirements. This research employs schemes of interrupt enabling/disabling, global variable usage using priority based preemptive scheduling algorithms and central queue management model to give optimal efficiency for multiple tasks.
1.1 OBJECTIVES OF RESEARCH

i. Dynamic Deadline Identification: An RTOS should be able to dynamically identify the task with the earliest deadline. To handle deadlines, deadline information is converted to priority levels and used for resource allocation.

ii. Proposes a Central queue based earliest deadline first (EDF) scheduler for conveying information available in FIFO queue.

iii. To propose kernel activity schemes suitable for handling critical section under realtime environment.

iv. To enlarge the research work for surveying compatible best known EDF and RM algorithms and to perform evaluation study.

v. Doing specific extension in the above area as a future scope to redesign new algorithms. Researchers and algorithms would make fundamental advances on above problems, fostering greater collaborations with realtime community.

vi. Combining a low overhead algorithm for securing information, error detection schemes and a realtime scheduler without sacrificing performance.

vii. To propose a hardware mechanism to track changes in task priority and adjusting the same during its runtime.

viii. For multiple tasks to communicate among themselves in a timely fashion, predictable inter-task communication and synchronization mechanisms are employed that are used to achieve data integrity by locking or unlocking resources.

ix. A new algorithm that can switch automatically between EDF algorithm and another scheduling algorithm to deal overloaded and under loaded conditions.

x. To develop a mathematical model for EDF and RM algorithms to minimize the fraction of jobs that misses their deadlines under very general workload assumptions and this model is based on queuing system.
1.2 BENEFITS OF RESEARCH WORK

i. Supports Priority based preemptive policies that allow concurrent task running without degrading system performance, protecting against high level and low level vulnerabilities, allows flexibility by redesigning hardware.

ii. Supports µc/os-II, allowing pre-emptive algorithms capable of attaining deterministic task switch latencies with task predictability makes it suitable for a large number of diverse applications. It is ported to vast variety of architectures, implemented in diverse designs, easy to integrate into a small project and gets into a simple application for running.

iii. Supports inter task communication through shared data structure using kernel provided calls and APIs to avoid priority inversion and thus eliminating shared resource problem.

iv. Comprehensive experimental evaluation of RM and EDF algorithms for different task sets indicating optimum processor utilization for realizing low cost hardware and software.

v. Supports preemptive model of execution-a currently executing job may be interrupted, and its execution resumed later, at no additional cost or penalty.

vi. OS services are compared and identified by parameters to form efficient operating system for an embedded system. This work proposes a realtime secure scheduling strategy for data transmission to enhance the communication throughput and reduce the overheads.

vii. Full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines.
1.3 PROBLEM STATEMENT AND PROPOSED SOLUTION

Problem 1

i. To have efficient hardware implementations for resource protection with inter process communication techniques.

Proposed Solution

i. For realtime implementation, best features of ARM processor such as multi-parameter acquisition, multi-level monitoring and networking are integrated featuring a high level of integration and low power consumption.

ii. In this context, LPC1768 hardware with customized \textmu c/os-II kernel and embedded C user space is used in this research. The software coding for the hardware functionality is written using Keil Software.

Problem 2

i. To design a data packet application system that requires packet data streaming, error detection layer, encryption layer and a real time packet scheduler for the selection of data packets for a particular time period.

Proposed Solution

i. Hardware set up is implemented using security protocol (Blowfish) and error detection coding schemes. This realtime application has been developed and run on cortex M3 LPC 1788 processor using \textmu c/os-II.

Problem 3

i. To address important metrics to characterize the behaviour of scheduling algorithms using theoretical and simulation based evaluations so that the results may be used to control and design realtime applications.
Proposed Solution

i. Two scheduling techniques have been used, analysed and compared based on best case response time, worst case response time, jitter and latency. Various issues have been presented on which there is still a need to work.

Problem 4

i. A missed deadline in hard real-time systems is catastrophic and it can lead to significant loss.

Proposed Solution

i. The scheduler temporarily assigns the earliest deadline from amongst the other tasks waiting for the resource, to the task while it is within its critical section to prevent the task with earlier deadlines miss their respective deadline, especially if the task within its critical section has a much longer time to complete and its exit from its critical section.

Problem 5

i. Shared resource problem arises while executing critical section due to the lack of mutual exclusiveness of resource for a task.

Proposed Solution

i. Disabling /Enabling interrupts using kernel provided calls to handle critical section. Two macros are used to enable/disable interrupts. This is the easiest and fastest way to prevent simultaneous access to a shared resource and μc/os-II uses this method to access internal variables and data structures. Care should be taken not to disable interrupts for too long. Doing so affects the response of the system and gives interrupt latency.

Problem 6

i. To provide scheme for avoiding data corruption due to lack of task synchronization.
Proposed Solution

i. Since, data variables can be shared among tasks; it is easier to move data from one task to other task. Two tasks need to have access to the same variable and these may synchronize their activities, i.e., tasks may activate other tasks and coordinate their activities using global variables. Reading a single global variable is a single execution step, without any chance that another task gets control and thus avoiding data corruption.

ii. To coordinate access to shared data, preemptive algorithm picks a task and lets it run for a maximum of some fixed time. If it is still running at the end of time interval, it is suspended and the scheduler picks another task to run. This method is the solution with the lowest overhead to prevent simultaneous access to a shared resource.

Problem 7

i. To provide simplest packet scheduling scheme for processing packets as they arrive, prohibition of waiting of packets for a long time to avoid their expiration.

Proposed Solution

i. Central queue based Earliest Deadline First (EDF) scheduling scheme finds a solution to this by scheduling packets based on their deadline. In this technique, realtime data packets are given higher priority and the delivery order of data packets in the ready queue may be changed by the preemptive priority based packet scheduler depending on their importance. This queuing implementation decomposes data into stream of packets, assigning priority and scheduling priority.

Problem 8

i. An issue with preemption is that continuous arrival of high priority data may lead to starvation of non-realtime data.
Proposed Solution

i. It can be solved by checking deadline and sorting packets according to remaining time to deadline.

Problem 9

i) To prove the effectiveness of realtime systems with respect to non realtime.

Proposed Solution

i. It can be proven by employing realtime systems for crucial applications like airbag system in a car.

Problem 10

i) To support various alarm signals for calculating runtime values.

Proposed Solution

i) Linux based alarm signals are implemented and their values are calculated using fibonacci series.

1.4 IMPLEMENTATION OF THE PROPOSED WORK

In this work, interrupt mechanism and synchronization schemes prove increasingly important in mutual exclusiveness of resource that recover and secure data under critical section and concurrent task running applications. Any critical section containing multiple tasks can be managed by RTOS. In order to manage the various tasks evaluated, Priority based Preemptive task Scheduling algorithm in µc/os-II Realtime Operating System is used. The algorithm proposed in this work is preemptive in nature and attempts to give fair CPU execution time by focusing on certain performance metrics of a task.

Allowing multiple tasks to have the same priority by adding a level of integration implies a pretty fundamental redesign of the ready list and scheduling algorithms, and probably the adoption of queue based scheduler. Queuing is a fundamental consequence of the statistical sharing that occurs in packet networks. The central queue algorithm (a unique one) is such one that supports true priority scheduling on a system-wide basis. The other algorithms do implement
priority scheduling within separate queues but on a system-wide basis. The primary benefit of using Central Queue scheduling is its adherence to pure priority scheduling.

The purpose of this work is to transfer packets with high access and transfer rate, to provide good scheduling algorithms, to provide good priority queues for storing packets temporarily and to provide secure way of managing database. Superiority of encrypted packet transfer is proven with others in terms of the throughput, response time, packet arrival rate and packet transfer speed. Scheduling algorithm in packet switching determines the packets that can be serviced at a time. Performance in terms of throughput, delays and fairness in delivering data packets depends on the scheduling decision. Maximizing performance requires the scheduler to solve some optimization problem, i.e., the scheduling decision must be taken in a very short time, not longer than the packet transmission time. μc/os-II could provide such a queue based scheduler as a compile time option. This scheduler will help in deciding which packet will be processed first according to their arrangement in priority queue. Packets will be arranged in the priority queue according to the priority that means high priority packets will be delivered first.

The i/p streamer consists of an input packet handler that can accept packets of variable size from multiple sources into a queue. A data stream is a sequence of digitally encoded coherent signals (packets of data or data packets) used to transmit or receive information that is in the process of being transmitted and Central queue based EDF scheduler is implemented for receiving and servicing data packets available in the FIFO queue.

To overcome priority inversion, μc/os-II supports priority ceiling and semaphore protocol mechanisms. Priority ceiling protocol is implemented in which a task owning the resource lock running at a higher priority than any other task that may acquire the resource. Each shared resource is initialized to a priority ceiling and whenever a task locks the resource, the priority of the task is raised to the priority ceiling. It works as long as the priority ceiling is greater than the priorities of any another tasks that may lock the resource. These resources are implemented using semaphores. Semaphores are added into the resource structure along with other information like priority ceiling of the resource and the link to the task that was currently holding the resource.
1.5 PERFORMANCE METRICS

Interrupt Latency

The most important specification of a realtime kernel is the amount of time interrupts are disabled. All realtime systems disable interrupts to manipulate critical section of code and re-enable interrupts when the critical sections have been executed. The longer interrupt are disabled, higher the interrupt latency. It is given by the equation,

\[ \text{Interrupt latency} = \text{Maximum amount of time interrupts are disabled} + \text{Time to start executing first instruction of ISR.} \]

\[ (1.1) \]

Interrupt Response

It is the time between the reception of the interrupt and start of the user code handles the interrupt. It is given by,

\[ \text{Interrupt response} = \text{Interrupt latency} + \text{Time to save CPU’s context.} \]

\[ (1.2) \]

Throughput

Throughput is the amount of data packets moved successfully from one place to another in a given time period.

Packet Loss

It is the fraction of packets not successfully received (i.e., passed CRC check) within some time window.

Mean Service Rate

It is the ratio between speed of the channel in bits per second to the mean packet length in bits.

Queuing Delay

It is the delay between the time the packet is assigned to the queue for transmission and the time it starts being transmitted.

Transmission Delay

It is the delay between the times that the first and last data bits of a packet are transmitted.
**Packet Arrival Rate**

Number of packets arriving into the queue per unit time.

**Response Time**

It is the time taken in an interactive program from the issuance of the command to the commence of the response of the command.

**Response Time Jitter**

Jitter is the size of the variation in the arrival or departure times of a periodic task. Jitter normally causes no problems as long as the actions all stay within the correct period. Response time jitter refers to maximum length of the interval in which a task can be released non-deterministically.

**Latency**

Latency is a measure of the time between a particular task and a system’s response to that task, and it’s quite often a focus for realtime developers. For a task $\tau_i$, the maximum input-output latency is defined as,

$$L_i = \max \left( finish-time - start-time \right) \quad \left(1.3\right)$$

### 1.6 CHAPTERWISE THESIS ORGANIZATION

**Chapter 2**

It presents the literature survey in three broad areas namely; (i) reported works on multitasking using RTOS (ii) reported works on queue based EDF packet scheduling (iii) reported works on resource allocation and critical section handling (iv) reported works on performance wise comparative analysis of scheduling algorithms.

**Chapter 3**

It discusses basic concepts of RTOS, task states, scheduling and comparative analysis of airbag inflation with respect to realtime and non realtime.
Chapter 4

This chapter presents comparative analysis/implementation of critical section handling/ADC suspend/resume/conversion applications using µc/os-II APIs.

Chapter 5

This chapter discusses block diagrammatic/ graphical representation of central queue based EDF scheduling, Metric wise realtime comparison of EDF and RM scheduling algorithms and implementation of Linux based interval timers using Fibonacci series.

Chapter 6

Results and discussion with hardware set up, comparison plots and tabulations are presented.

Chapter 7

The summary and conclusion with future scope is presented in this chapter.