Journal of Computer Science 10 (11): 2349-2357, 2014
ISSN: 1549-3636

doi:10.3844/jcssp.2014.2349.2357 Published Online 10 (11) 2014 (http://www.thescipub.com/jcs.toc)

RTOS IMPLEMENTATION OF NON-LINEAR SYSTEM USING MULTI TASKING, SCHEDULING AND CRITICAL SECTION

[1]Sujitha, M., [2]V. Kannan and [3]S. Ravi

[1,3]Department of ECE, Dr. M.G.R. Educational and Research Institute University, Chennai, India
[2]Jeppiaar Institute of Technology, Kanchipuram, India

Received 2014-04-21; Revised 2014-07-19; Accepted 2014-12-16

ABSTRACT

RTOS based embedded systems are designed with priority based multiple tasks. Inter task communication and data corruptions are major constraints in multi-tasking RTOS system. This study we describe about the solution for these issues with an example Real-time Liquid level control system. Message queue and Mail box are used to perform inter task communication to improve the task execution time and performance of the system. Critical section scheduling is used to eliminate the data corruption. In this application process value monitoring is considered as critical. In critical section the interrupt disable time is the most important specification of a real time kernel. RTOS is used to keep the interrupt disable time to a minimum. The algorithm is studied with respect to task execution time and response of the interrupts. The study also presents the comparative analysis of the system with critical section and without critical section based on the performance metrics.

Keywords: Critical Section, RTOS, Scheduling, Resource Sharing, Mailbox, Message Queue

1. INTRODUCTION

Realtime Operating System (RTOS) is specially used to meet the real time constraint and to support the sophisticated facilities required by an embedded system. In this study a non realtime liquid level control system is transformed to real time using multiple tasks. In order to manage the various tasks, Priority based Preemptive Task Scheduling algorithm in RTOS is used. Each task in an application is assigned a priority, with higher priority values representing the need for quick response. In modern RTOS multitasking is a technique used for enabling multiple tasks to share a single processor. It is simply the ability to run two or more independent tasks on one CPU (appears to be at the same time) and running concurrently.

A realtime kernel like (FreeRTOS) supports multitasking. It is a priority based pre-emptive realtime multitasking kernel for processors, written mainly in C programming language. RTOS design has been designed for very small embedded systems and implements essential set of functions, easy to integrate into small embedded systems. RTOS services a very basic set of handling tasks and memory management, just sufficient API Synchronization, drivers for external hardware (or) access to a file system. It also allows an unlimited number of tasks to run as long as Hardware and memory can handle it. Finally it implements queues, binary and counting semaphores and mutex. (Liu, 2000).

1.1. Task

A task is an independent thread of execution that can complete with other concurrent tasks for processor execution time. A task is schedulable. A task can be in one of two simple states: “Running” or “not running”. Suppose

Corresponding Author: Sujitha, M., Department of Electronics and Communication Engineering, Dr. M.G.R. Educational and Research Institute University, Chennai, India

there is only one core, at a time only one task can run; all other tasks are in the "not running" task.

Fig. 1 gives a simplified representation of this life cycle. When a task changes its state from "Not running" to running, the "Not running" state can be expanded as shown in **Fig. 2**. A task can be preempted because of a higher priority task (or) it has been delayed (or) it waits for an event. When a task can run but is waiting for the processor to be available, its state is said "Ready". The features of RTOS system are listed in **Table 1**. (Melot, 2014).

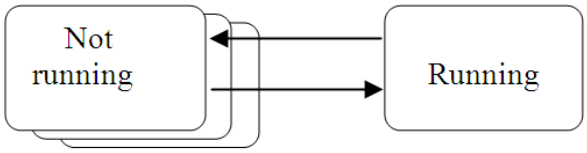

Fig. 1. Simplified life cycle of a task

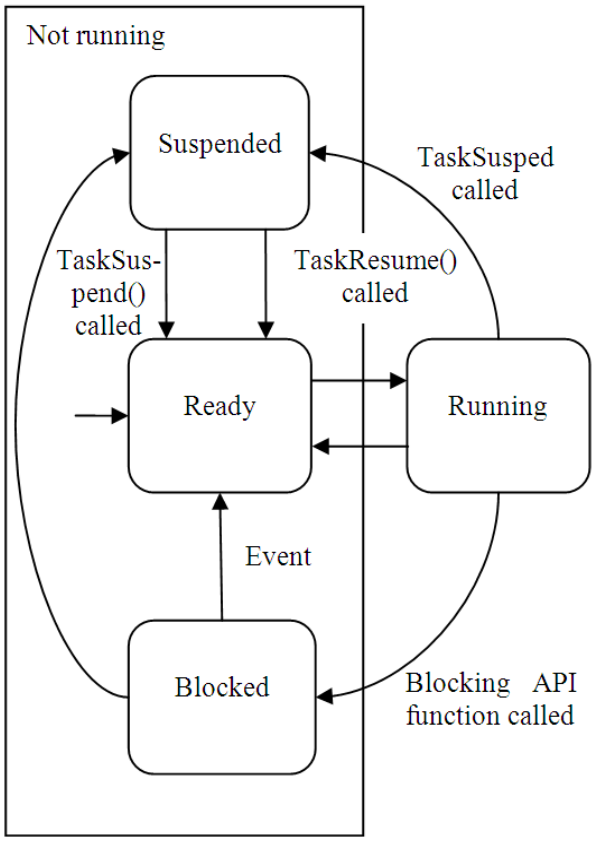

Fig. 2. Life cycle of a task

1.2. Scheduler

A scheduler is the part of the kernel responsible for deciding which task should be executing at any particular time. Operating systems may be distinct to three types based on schedulers features, a long, a medium and short term scheduler. The long term scheduler selects a process from the job pool and loads them into memory for execution. The short term scheduler or CPU scheduler selects from among the processes from memory and reduces the degree of multiprogramming results in the scheme of swapping. Swapping is the scheme which is performed by dispatcher that gives control of the CPU to the process selected by the short term scheduler. In the RTOS based embedded system CPU scheduling plays an important role which always has a time constraint on computation. The realtime task should be scheduled to be completed before their deadlines because real time system is one whose applications are mission critical. Most realtime systems control unpredictable environments, to handle unknown and changing tasks, it need operating systems (Rajput and Gupta, 2012).

1.3. Inter Task Communication

RTOS provides a variety of mechanisms for communication and synchronization between tasks. These mechanisms are necessary in a multi task preemptive environment, since without them the tasks might corrupt information or otherwise interfere with each other. Information can be communicated between tasks in two ways:

- Through sharing global data
- Sending messages

Message sending is the main mechanism which is used for that purpose. Message sending methods include Mailbox and Queue. Through kernel services message can be sent to a task. A message mailbox also called a message exchange. It is typically a pointer size variable through a service provided by the kernel. A task (or) an ISR can deposit a message into the mail box. Through a service provided by the kernel, one or more tasks can receive messages. Both the sending task and receiving task will agree as to what the pointer is actually pointing to. Each mail box is associated with a waiting list, in case more than one task desires to receive message through the mail box (Labrosse, 2002).

To send one (or) more messages to a task, Message Queue is used. A message queue is an array of mail boxes through a service provided by the kernel.

Table 1. RTOS Features

Features of RTOS	
Realtime	RTOS is a hard realtime operating system
Preemptive or cooperative operation	Process pass control from one task to another task by yielding in cooperative scheduling. The scheduler interrupts at regular frequency simply to increment the tick count
Dynamic scheduling	Scheduler decision points occur at regular clock frequency
Scheduling algorithm	The highest priority process is scheduled first by scheduler algorithm. Where more than one task exists at the highest priority, tasks are executed in round robin fashion.
Double linked list	Multiple tasks can exists with the same priority assigned. Tasks of the same priority are organized in a double-linked list.
Inter-process communication	Tasks within RTOS can communicate with each other through the use of queuing, mail box and synchronization mechanisms
Blocking and deadlock avoidance	In RTOS, tasks are either block with a fixed period of time or non-blocking.
Critical section processing	Critical section processing can be handled by semaphore or mutex or disabling of interrupts.
scheduler suspension	When exclusive access to the CPU is required without jeopardizing the operation of ISRs, the scheduler can be suspended.
Memory allocation	RTOS provides multiple heap models as part of the distribution.

Table 2. RTOS mail box and message queue function

Typical OS functions	Purpose
OSMboxPend()	Retrieving message from a mailbox
OSMboxPost()	Posted message to a mail box
OSMboxcreate()	Creating message
OSMboxAccept()	Accepting message
OSMboxQuery()	Information about mailbox
OSMboxDel()	Deleting a mailbox
OSMboxPostOpt()	Message is posted in mailbox

A message pointer can be deposited into a message queue by a task or an ISR. Through a service provided by the kernel one (or) more tasks can receive messages. Both the sending and receiving task will agree as to what the pointer is actually pointing to. Generally First in First out methodology is used to extract a message from queue (Abt and Thomas, 2012). **Table 2** shows the mailbox and message related functions.

In this article, section 2 gives related work. Section 3 describes hardware implementation, section 4 describes the performance metrics and section 5 describes about real time implementation of critical section.

2. RELATED WORK

Kolhari and Bhopale (2012) implemented the porting of MicroC/OS-II kernel in LPC 2148 microcontroller for the implementation of features like multitasking, time scheduling, mailbox and, mutex. Here a real time kernel is the software that manages the time of a micro controller to ensure that all time critical events are processed as efficiently as possible. Different interface modules of ARM7 microcontroller like UART, ADC and LCD are used. Data acquired from these interfaces is tested using µC/OS-II based real time operating system.

Indersain *et al*. (2013) implemented porting of kernel in ARM powered microcontroller for the implementation of multitasking and time scheduling. Here the real time kernel is the software that manages the time of a micro controller to ensure that all time critical events are processed as efficiently as possible. Different type of interface modules of ARM7 microcontroller like UART, ADC, DAC, KAYPAD, LCD, USB are used and data acquired from these interfaces is tested using µC/OS-II based real time operating system. The steps involved in porting the RTOS and final implementation details are provided.

Rajput and Gupta (2012) implemented priority based round robin CPU scheduling Algorithm for realtime systems where there is more than one task with same priority to share CPU time, so the burden is on the user to proxy out the time slicing code to a high level mechanism of their own design. Allowing multiple tasks to have the same priority by adding a level of integration implies a fundamental redesign of the ready list and scheduling Algorithms and probably the adoption of queue based Scheduler. In its state µC/OS-II is an optimal solution of embedded real time software engineering problems.

3. HARDWARE IMPLEMENTATION

Realtime liquid level system consists of a conical tank, Host controller and zigbee module. Conical tank has a liquid level sensor, inflow and outflow rotary sensor. These sensor outputs are connected to controller

in host. Host has controller and a ZigBee module. ZigBee module is used to communicate with server. Controller calculates final action values and drive to valve control inputs in conical tank.

The hardware implementation of the realtime control system is shown in **Fig. 3 to 5**.

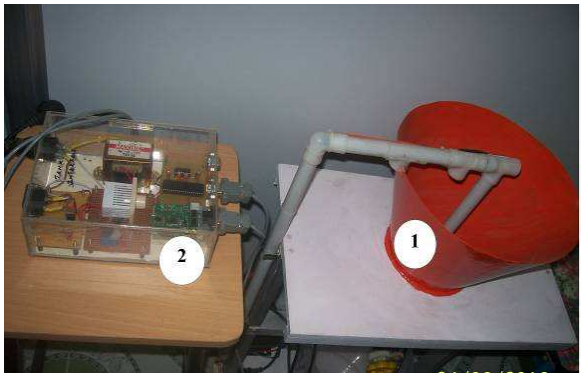

Fig. 3. Host system and conical tank setup 1. Conical Tank 2. Host controller

Fig. 4. Host Controller module 1. Host ZigBee 2. Controller

Fig. 5. Server ZigBee Interface 1.Server ZigBee Module

4. PERFORMANCE METRICS

4.1. Throughput

Throughput refers to the number of task execute in a unit of time. The higher number Throughput indicates more work done by the system.

4.2. Context Switch

A context switch is the process of storing and restoring state of a preempted process, so that execution can be resumed from same point at a later time. Context switching is usually resource intensive, so the design of operating system is to optimize only these switches. More number of context switch lead to wastage of time and memory, which in turn deccreases the system performance (Rajput and Gupta, 2012).

4.3. Turnaround Time

Turnaround time refers to the amount of time taken to complete the process and is how long it takes the time to execute that process.

Turnaround time = Time of process completion-time of process submission

Total turnaround time is the sum of the periods spent waiting to get into memory, waiting time in the ready queue, execution time on the CPU and doing I/O (Rajput and Gupta, 2012).

4.4. Waiting Time

Waiting time is the total time a process has been waiting in ready queue.

Waiting time = Time of process scheduled-time of process ready to executes

The CPU scheduling algorithm does not affect the amount of time during which a process executes or does input-output; it affects only the amount of time that a process spends waiting in ready queue (Rajput and Gupta, 2012).

4.5. Response Time

In an interactive system, turnaround time may not be best measure. Often, a process can produce some output fairly early and can continue computing new results while previous results are being produced to the user.

Response time = Time of response received-time task submitted (Kamal, 2008).

So we can conclude that a good scheduling algorithm for real time and time sharing system must possess following characteristics (Rajput and Gupta, 2012).

- less context switches
- high throughput
- low turnaround time
- low waiting time
- low response time

5. REALTIME IMPLEMENTATION

The following tasks are identified:

- Tracking input to Controller (Task1)
- ZigBee Communication (Task2)
- Error handling (Task3)
- Monitor process value (Task4)

In this study four tasks are created, namely Task1 (uctsk_setpoint), Task2 (uctsk_zigbee), Task3 (uctsk_error) and Task4 (uctsk_display). The system initialization flow chart is shown in **Fig. 6**. The function of Task1 is getting the setpoint values from server and writes these data into controller. Task1 flowchart is given in **Fig. 7**.

Task2 access the process status values namely inflow, level and outflow. The collected process status values are send to the server through Zigbee communication. Task2 flowchart is given in **Fig. 8**.

The function of Task3 is processing the error and deciding the control output for action. When this task is started it gets the process value such as level from server then it wait for an interrupt from server. Once it receives an interrupt from the server, reads the process value from plant for final action. Task3 flowchart is given in **Fig. 9**.

Task 4 is used to display the different data in monitor. When this task is started it will wait for error signal interrupt. Once it receives the interrupt it will display process status data. Task4 job is considered critical as it should not be preempted, when it is reading and displaying process value. . A critical section of code is called a critical region. This section of code should treat indivisibly. The section of code must not be interrupted once it starts executing. To ensure this, interrupts are typically disabled before the critical code is executed and enabled when the critical code is finished. Disabling the interrupts before a critical section starts executing and enabling interrupts is a powerful option for solving shared resource problems. For critical section implementation RTOS has interrupt disabling and enabling functions that execute at entering and exiting section respectively (OS_ENTER_CRITICAL and OS_EXIT_CRITICAL).

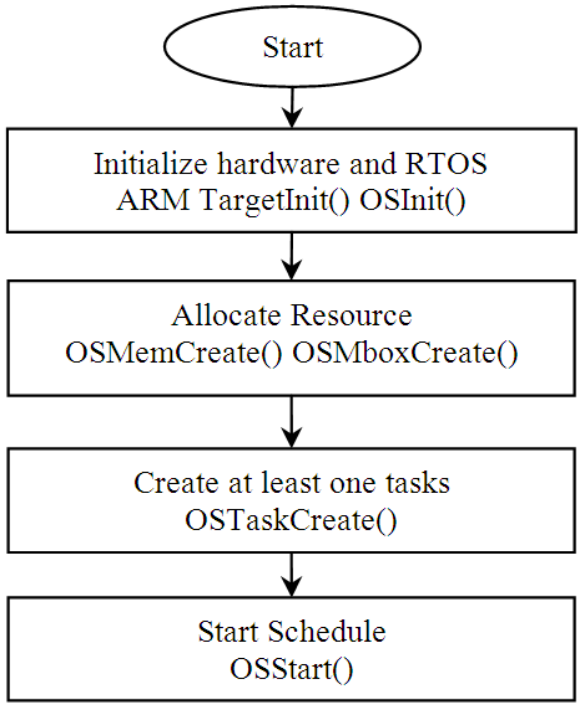

Fig. 6. Flowchart for iinitialization

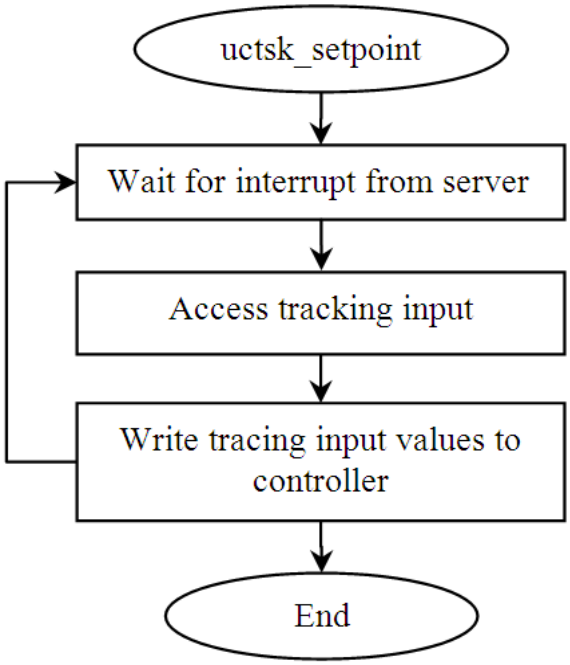

Fig. 7. Flowchart for set point tracking

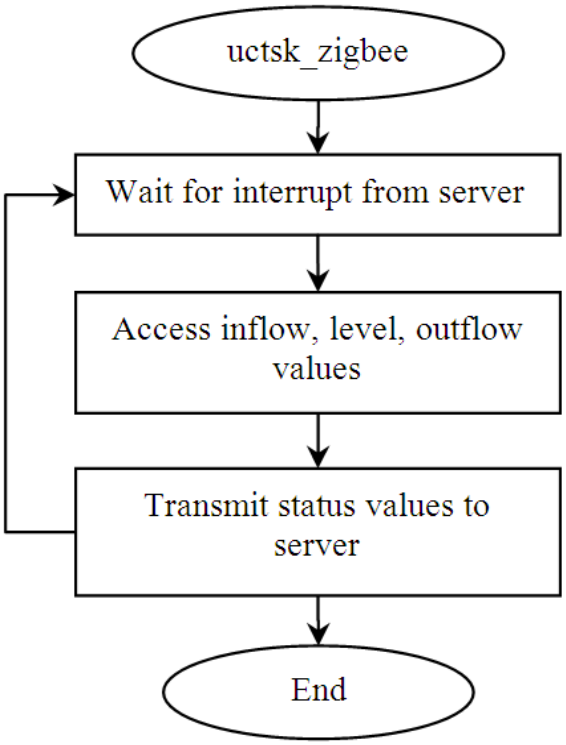

Fig. 8. Flowchart for Zigbee communication in Task2

To achieve significant enhancement in the task execution and prevention of other task to run in between, these application programming interfaces are incorporated in RTOS under critical environment that provides the interface between the application software and system software. Task4 without critical implementation flowchart is given in **Fig. 10**.

Task4 with critical implementation flowchart is given in **Fig. 11**. **Table 3** shows the related OS functions used for this implementation.

Task2 and Task4 has highest and equal priority task and Task1 has low priority. Task3 has medium level priority. A mailbox is created using OSMboxCreate() function to communicate process status values to Task2 and Task4. In Task3 read inflow, level and outflow values from sensors in plant and post a message to mailbox using OSMboxPost() function. Task2 and Task4 are in waiting state for message. Once mailbox receive message from Task2, OSMboxPend() resume execution immediately after call OS_Sched(). Now Task2 and Task4 moved to ready state from blocked state. Task4 will execute first because Task4 has critical section which will block all tasks by disabling interrupt. Once Task4 is completed then Task2 will execute.

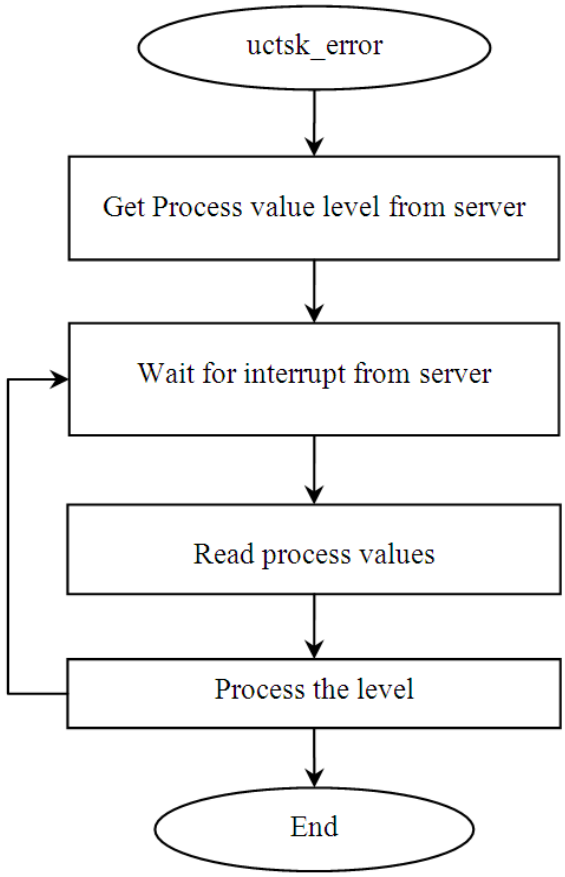

Fig. 9. Flowchart for error tracking

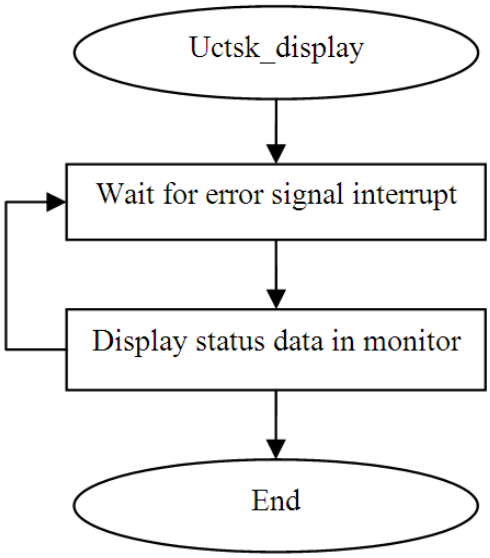

Fig. 10. Flowchart for display parameters without critical section

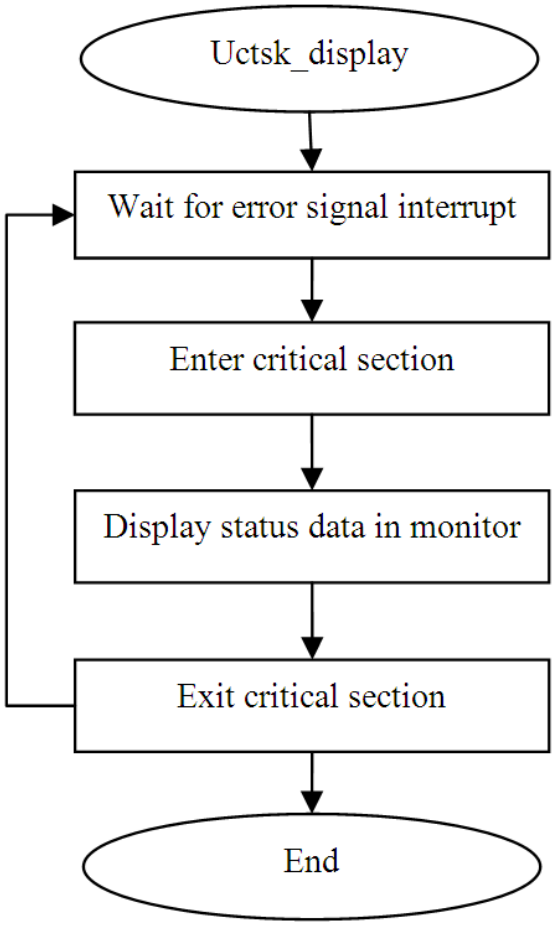

Fig. 11. Flowchart for display parameters with critical section

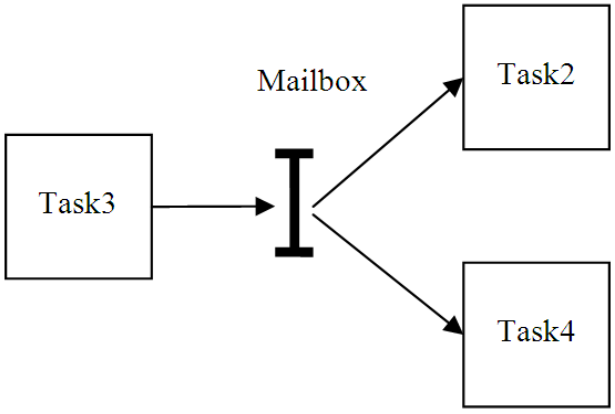

Fig. 12. Relationship between Mailbox and Tasks

Communication from Task3 to Task2 and Task4 using mailbox is shown in **Fig. 12** (Inam *et al*., 2011).

6. RESULT

Table 4 shows the arrival and burst time for various tasks. For Analysis the Time quantum is taken as.

Figure 13 and **14** shows Gantt chat (Matarneh, 2009) for system without critical section and with critical section. System without critical section average response time and average waiting time are calculated below:

- Average Response Time = (0+0+2+0) = 2/4 = 0.5
- Average Waiting Time = (0+0+2+6)/4 =2
- Number of Context switch: 8

System with critical section average response time and average waiting time are calculated below:

- Average Response Time = (0+2+2+0) = 4/4 = 1
- Average Waiting Time = (0+2+2+0)/4 =1
- Number of Context switch: 6

Table 5 and **Fig. 15** show performance comparison between system with critical section and without critical section.

Table 3. Critical Function of RTOS

Typical µC/OS functions	Purpose
OS_ENTER_CRITICAL	Enter to Critical
OS_EXIT_CRITICAL	Exit From Critical
OS OS_FLAG_CONSUME	Consume the Arguments
OS_Task_Create()	Create a Task
OS_Task_delete()	Delete a Task
OS_Task_Suspend()	Suspend a particular Task
OS_Task_Resume()	Resume the Task
OS_FLAG_PEND	Arguments are Waiting
OS_FLAG_Wait_SETALL	Set all Values While pending
OS_Task_Name_EN()	Enable the Task
OS_Task_Name_SET	Set Task Name

Table 4. CPU Burst time for tasks

Task	Arrival time (ms)	Burst time (ms)
Task1	2.0	2.0
Task2	7.0	3.0
Task3	3.0	2.0
Task4	6.0	2.0

Table 5. Performance comparison

	AWT (ms)	ART (ms)	No. context switch
Critical section	1	1.0	6
Without critical section	2	0.5	8

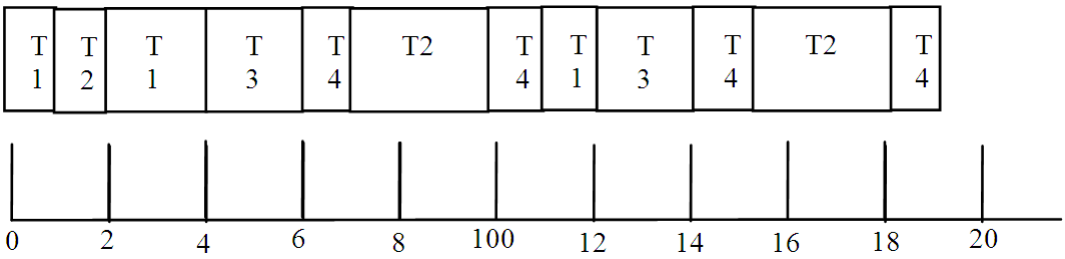

Fig. 13. Gantt chart for RTOS operation without critical section

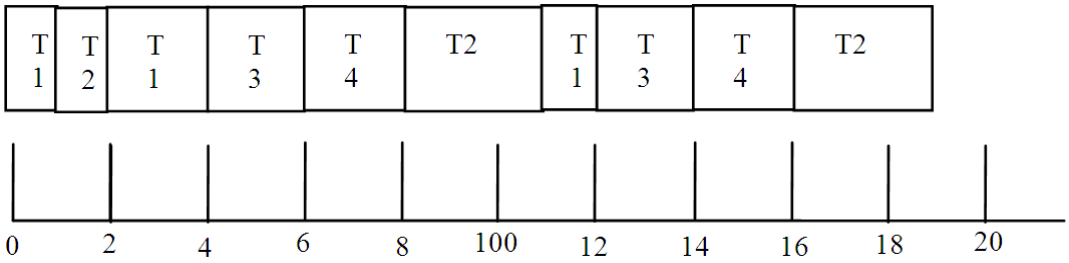

Fig. 14. Gantt chart for RTOS operation with critical section

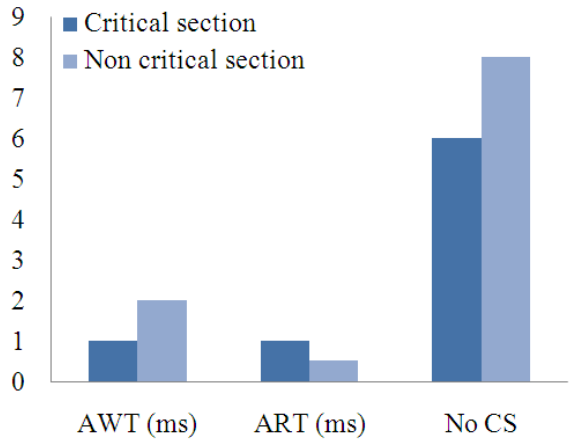

Fig. 15. Comparison between system with Critical section and without critical section

7. DISCUSSION

From the results given, it is observed that the average waiting time is reduced in system with critical section. But response time is increased little due to disable interrupt during execution of critical section. Since critical section designed with minimum burst time, it will not affect overall performance of system. Number context switch is reduced in system with critical section. This will reduce memory use and task execution time.

From the given results it is observed that nonlinear system is protected from the tasks entering simultaneously from multiple tasks or ISR. Tasks synchronization is handled very well by preemptive based task scheduling and the overall system performance is improved.

8. CONCLUSION

In this study scheduling multiple tasks along with critical section and the realtime performance was analyzed. From the results it is concluded that RTOS based embedded system can manage any critical section containing multiple tasks. Event based and time based planning, scheduling can be implemented by RTOS stack and timer management, for quick and efficient functioning of a Control System.

For future scope, by properly redesigning the scheduler to improve average response time enhancing features of scheduler, controllability over the execution can be easily achieved.

9. ACKNOWLEDGEMET

The researcher are grateful to Dr. S. Ravi, professor, Head of the Department, ECE, Dr. MGR Educational and Research Institute his encouragement

and guidance in the successful completion of the implementation of this system.

10. REFERENCES

Abt, A.R. and K.Thomas, 2012. ARM based embedded web servers for industrial applications. Int. J. Comput. Applic., 44: 28-35.

Inam, R., J. Mäki-Turja, M. Sjödin and M. Behnam, 2011. Hard real-time support for hierarchical scheduling in FreeRTOS. Mälardalen University.

Indersain, N. Sharma and D. Singh, 2013. Design and implementation of µc/Os II based embedded system using arm controller. Int. J. Eng. Technical Res., 1: 1-4.

Kamal, R., 2008. Embedded Systems 2E. 1st Edn., McGraw-Hill, Education, New Delhi, ISBN-10: 0070667640, pp: 681.

Kolhari, N.R. and N.I. Bhopale, 2012. Porting and Implementation of features of ?C/OS II RTOS on Arm7 controller LPC 2148 with different IPC mechanisms. Int. J. Eng. Res. Technol.

Labrosse, J.J., 2002. MicroC/OS-II: The Real Time Kernel. 2nd Edn., CMP Taylor and Francis, San Francisco, ISBN-10: 1578201039, pp: 605.

Liu, J.W.S., 2000. Real-Time Systems. 1st Edn., Pearson Education India, Upper Saddle River, ISBN-10: 8177585754, pp: 624.

Matarneh, R.J., 2009. Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes. Am. J. Applied Sci., 6: 1831-1837. DOI: 10.3844/ajassp.2009.1831.1837

Melot, N., 2014. Study of an operating system: FreeRTOS. Operating systems for embedded devices.

Rajput, I.S. and D. Gupta, 2012. A priority based round robin CPU scheduling algorithm for real time system. Int. J. Innovation Eng. Technol., 1: 1-11.