International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

REAL TIME OPERATING SYSTEMS: A COMPLETE OVERVIEW

Mrinal Parikshit Chandane

Former Assistant Professor, Dept. of E&TC, KJSCE, (India)

ABSTRACT

Telecommunication applications such as telephony, navigation and military signaling systems etc. are real time embedded systems. These systems must meet strict deadline constraints and handle different aspects of the application successfully. Real Time Operating Systems (RTOS) is most widely used software architecture for executing such embedded applications demanding strict deadlines and handling multiple tasks together. The important feature of RTOS is the timing considerations such that most urgent operations are taken care of by assigning higher priorities. RTOS also focuses on the communication and synchronization between different tasks to achieve the objective of the application. Our paper focuses on the discussion and overview for the quick understanding of design and implementing issues of RTOS for any microcontroller or microprocessor to be used in real time embedded systems.

Keywords: Latency, Multitasking, Priority Inversion, Priority Inheritance, RTOS, Semaphore.

I. INTRODUCTION

A Real Time Operating System (RTOS) is an operating system developed for real-time embedded applications evolved around processors or controllers. It allows priorities to be changed instantly and data to be processed rapidly enough that the results may be used in response to another process taking place at the same time, as in transaction processing [1]. It has the ability to immediately respond in a predetermined and predictable way to external events. Overall a mode of action and reaction by RTOS and application software handles the entire embedded application.

Let us consider the role of RTOS in the mobile phone. A cell phone has several features like call processing, notifying a waiting call, maintain a phone directory, messages and other utilities like web browsers, calculator, games, apps etc. RTOS handles each of these features as a separate task. Suppose a user is playing game on his cell phone and a call arrives, immediately the caller's ID starts flashing on the screen. After completion of the call, the user can resume the game from the level/ point it has got suspended. It is observed that in this case RTOS handled the tasks using priorities and performed multitasking using context switching and scheduler.

This paper gives quick overview of complete architecture of RTOS. The paper also discusses the various aspects and issues of design and implementation of RTOS for controllers and processors.

II. DESIGN ASPECTS

To develop an RTOS for a particular application, one has to consider the hardware architecture available, the resources to be handled, the available memory etc. to decide for the various aspects of the Real Time Operating System.

2.1 Task

A task also called a thread is a simple program that thinks it has the CPU all to itself. The design process for a real time application involves splitting the work to be done in different tasks. Each task is assigned a priority, its own set of CPU registers and its own stack area. Each task is typically in an infinite loop that can be in any of the five states: Dormant, Ready, Running, Waiting and Interrupted. The first step in designing of the RTOS is to decide on the number of tasks that the CPU of the embedded appication can handle.

2.2 Multitasking

It is the process of scheduling and switching the CPU between the several tasks. It maximizes the CPU utilization and makes programming efficient for designing and debugging [2]. Processors and Controllers with inbuilt pipe lining architecture and can further increase the speed of the execution of a particular task.

2.3 Context Switch

When a multitasking kernel decides to run different tasks, it simply saves the current task's context storage area (Task Stack). Once this operation is performed, new task's context is restored from its storage area (Stack) and execution of new task's code is resumed.

2.4 Task Priority

Priority is assigned to a task depending on its importance such as static priority. Static priority means priority of the task does not change during run time. Dynamic priority means the priority of the task can change during run time.

2.5 Kernel

In multitasking system, kernel is responsible for management of tasks and context switching. A kernel program will require its own code space (ROM area) and data structure space (RAM area) which will increase the system overhead [3]. So, kernel should be designed strictly to consume 2% to 5% of overall CPU time. Kernel can provide indispensable service such as semaphore management, mailboxes, queues, time delays etc. Kernel can be preemptive or non-preemptive.

2.6 Resource

It is any entity used by the task. A resource can thus be an input or output device, a variable, a structure or a set of registers etc. A shared resource can be accessed by more than one task. However, each task should gain exclusive access to the shared resource to prevent data corruption. This is called Mutual Exclusion. Most common methods to obtain exclusive access to shared resources are disabling interrupts, test and set, disabling scheduling and using semaphores.

2.7 Inter Task Communication

It is sometimes necessary for a task or an interrupt service routine has to communicate to another task. It can be done using global data under mutual exclusion. The other option available for Inter task communicatio is to send messages using either message mailbox or message queue.

2.8 Interrupts

Interrupts allows controller or processor to process events when they occur. In RTOS, interrupt service routine processes the events and upon completion returns to background for a foreground/ background system, interrupted task for a non-preemptive kernel and highest priority task to run a preemptive kernel.

2.9 Clock Tick

It is a special interrupt that occur periodically, generally in microseconds. It is a system's heart beat and provides time outs when tasks are waiting for event to occur. The faster the tick rate, higher the overhead imposed on the system. All kernels allow tasks to be delayed for certain number of clock ticks.

III. IMPLEMENTING AN RTOS

3.1 Basic Requirement

The most important aspect for writing RTOS is choosing the programming language. Assembly language, C, C++ or Embedded C can be chosen depending on the architecture of the processor/ controller under consideration, peripheral requirement, maximum number of tasks it can handle and the structure of the task stack area should be known. Writing the device drivers for peripherals is another major task of RTOS.

3.2 Main Software Architecture

The entire design of RTOS can be broken up in sub parts like Initialization of RTOS, creation and deletion of tasks, assigning priority to the tasks, scheduling the tasks, RTOS utility (Delays or device drivers), parmaeter passing, UART, debugging, application and finally compiling and linking the whole code [4] [5].

3.3 Task's Stack

Each stack has its own stack space. Stack space can be either allocated either statistically or dynamically. The size of the stack is application specific.

3.4 Task Control Block (TCB)

When a task is created it is assigned to a task control block. TCB is a data structure that is used by a RTOS to maintain the state of the task during context switch. When the task regains control of the CPU, the TCB allows the task to resume execution exactly where it is left off. All functions of TCB is initialized when a task is created. Thus a TCB can contain CPU register values, values of variables and task stack.

3.5 RTOS initialization

RTOS should be initialized before calling any of other services. This involves initialization of variables and data structures, task control blocks and task stacks [6]. After initialization, RTOS is ready to provide the services by creating different task.

3.6 Creating and Deleting task

Task is an infinite loop function. While writing RTOS three main things are to be considered such as creating the task, assigning priority to the task and deleting the task.

In order for system to manage the application it must be divided into different tasks. The tasks should be created by passing its address along with parameters from the main program to it. When the task is completed it can be deleted by removing the task number from the scheduler.

3.7 Assigning Priority

Assigning priority to task is required to know which task is more important than other to facilitate scheduling.

3.8 Ready Table

Each task is assigned a unique priority level between highest priority and lowest priority. Lowest priority is assigned to the idle task. Each task ready to run is placed in the ready list. Task priorities does not depend on maximum number of tasks. Tasks priorities can also be grouped. When task is ready to run, it sets its corresponding bit in the ready table declaring its ready status. Keeping ready list allows reducing the amount of RAM needed, when application requires few task priorities. Thus, by checking the ready status from the ready table and priority scheduler decides which task will run.

3.9 Scheduling

The DORMANT state of the task corresponds to a task which resides in program space but has not been able available to RTOS. A task is made available by making its entry in ready table. Thus when a task is created it is made READY to run. A task can be made DORMANT by deleting its entry from the ready table. Sending a task in DORMANT state means deleting that task. Scheduler will place the higher priority task that is READY to RUNNING state. Only one task can be running at any given time. A READY task will not run until all other higher priority tasks are either placed in the wait state or are deleted.

IV. RESULTS AND CONCLUSION

RTOS of 32 tasks in assembly language was designed and implemented for Philips 8052 microcontroller with the peripherals 4x4 Hex keypad, 2 line LCD display and an external RAM. The keyboard was interrupt driven, after every 10msec the keybaoard was scanned for any key press and then depending upon the key press, the particular task was made ready in the scheduler. If that task happens to be a higher priority task then context switching was performed by the sheduler. Applications such as Column Controller, Digital Calculator and Dancing LEDs were successfully implemented using same RTOS designed and implemented considering all the above apects discussed. The same concept of RTOS can be implemented in languages like assembly, C, C++, embedded C for any microcontroller or processor in a real time embedded system like cell phones, satellite communication, navigation etc [7] [8].

REFERENCES

[1] Stankovic, John A., and Raj Rajkumar. "Real-time operating systems." Real-Time Systems 28.2-3 (2004): 237-253.

[2] Jensen, E. Douglas, C. Douglas Locke, and Hideyuki Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems". Journal of IEEE Real time System Symposium,. Vol. 85. 1985.

[3] Buttazzo, Giorgio C., Marko Bertogna, and Gang Yao. "Limited preemptive scheduling for real-time systems. a survey." Industrial Informatics, IEEE Transactions on 9.1 (2013): 3-15.

[4] Ramamritham, Krithi, and John A. Stankovic. "Scheduling algorithms and operating systems support for real-time systems." Proceedings of the IEEE82.1 (1994).

[5] Laplante, Phillip, "A. REAL-TIME Systems design and analysis". IEE, 1993.

[6] Stallings, William, Goutam Kumar Paul, and Moumita Mitra Manna, "Operating systems: internals and design principles".Upper Saddle River, NJ: Prentice Hall, 1998.

[7] Reddy, K. Srinivasa., "New Real-time Operating Systems for Embedded Systems". Laxmi Publications, 2014.

[8] Cooling, Jim E., "Software design for real-time systems". Springer, 2013.