© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

BASICS OF REAL TIME OPERATING SYSTEM

Tushar Wason, Anubhav Chandra
Student, Department of Information Technology,
Dronacharya College of Engineering, Gurgaon, Hr, India

***Abstarct*- This article explains the concepts of kernel, threads, messaging (like events and mailboxes), scheduler and the main pieces to put an RTOS application together. To do this in a real application will require more details, for example, how to create mailboxes and events.The best way to get started, if you are new to using an RTOS, is to bring up and use one of the many examples that come from the different RTOS vendors. In the article above I have used Express Logic for the code snippets. Express Logic provides many example applications and has BSP's for many different boards and devices.**

I. INTRODUCTION

What is a real-time operating system?

A **real-time operating system** (**RTOS**) is an operating system (OS) intended to serve real-time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter.

A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task; the variability is *jitter*.[1] A *hard* real-time operating system has less jitter than a *soft* real-time operating system. An RTOS that can usually or *generally* meet a *deadline* is a soft real-time OS, but if it can meet a deadline it is a hard real-time OS.[2]

An RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency ; a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.[3]

II. HOW DOES AN RTOS WORK?

The core of an RTOS is known as the kernel. An API is provided to allow access to the kernel for the creation of threads, among other things. A thread is like a function that has its own stack, and a Thread Control Block (TCB). In addition to the stack, which is private to a thread, each thread control block holds information about the state of that thread.The kernel also contains a scheduler. The scheduler is responsible for executing threads in accordance with a scheduling mechanism. The main difference among schedulers is how they distribute execution time among the various threads they are managing. Priority-based, preemptive scheduling is the most popular and prevalent thread scheduling algorithm for embedded RTOSes. Typically, threads of the same priority execute in a round-robin fashion.

Most kernels also utilize a system tick interrupt, with a typical frequency of 10ms. Without a system tick in the RTOS, basic scheduling is still available, but time-related services are not. Such time-related services include: software timers, thread sleep API calls, thread time-slicing, and timeouts for API calls.

The system tick interrupt can be implemented with one of the hardware timers in the embedded chip. Most RTOS have the ability or extension to reprogram the timer interrupt frequency dynamically such that the system can sleep until the next timer expiration or external event. For example, if you have an energy sensitive application you might not want to run the system tick handler every 10ms if not necessary. Suppose for example the application is idle and the next timer expiration is 1000ms away. In this case, the timer can be reprogrammed to 1000ms and the application can enter low-power mode. Once in this mode, the processor will sleep until either another external event either case, when the processor resumes execution the RTOS adjust the internal time according to how much time has elapsed and normal RTOS and application processing is resumed. This way, the processor only executes when the application has something to do. During idle periods the processor can sleep and save power.

III. RTOS COMPONENTS

Let's see what features an RTOS has to offer and how each of these features might come in handy in different applications.

```
void ThreadSendData( void )
{
 while (1)
 {
 // Send the data…
 }}
```

IV. THREADS

Threads are like functions, each with its own stack and thread control block (TCB). Unlike most functions, however, a thread is almost always an infinite loop. That is, once it has been created, it will never exitA thread is always in one of several states. A thread can be ready to be executed, that is, in the READYstate. Or the thread may be suspended (pending), that is, the thread is waiting for something to happen before it goes into the READY state. This is called the WAITING state.

State	Description
Executed	This is the currently-running thread
Ready	This thread is ready to run
Suspended	This thread is waiting for something. This could be an event, a message or maybe for the RTOS clock to reach a specific value (delayed).
Completed	A thread in a completed state is a thread that has completed its processing and returned from its entry function.(A thread in a completed state cannot execute again.)
Terminated	A thread is in a terminated state because another thread

V. SCHEDULER

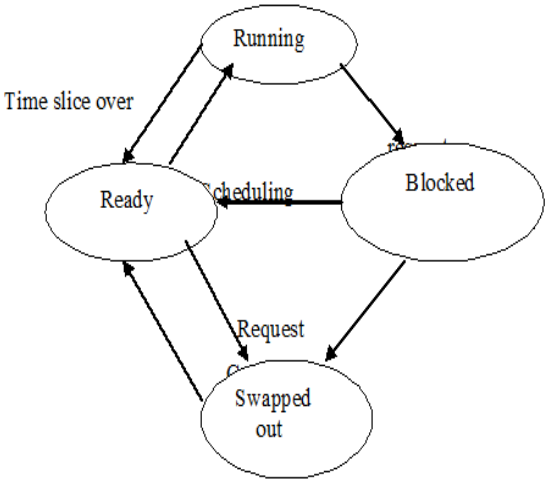
There are two major types of scheduler from which you can choose:

Event-driven (Priority-Controlled Scheduling Algorithm)

In event-driven systems, every thread is assigned a priority and the thread with the highest priority is executed. The order of execution depends on this priority. The rule is very simple: The scheduler activates the thread that has the highest priority of all threads that are ready to run

2. Time-sharing

The most common time-sharing algorithm is called Round-Robin. With round-robin scheduling, the scheduler has a list of the threads that make up the system and it uses this list to check for the next thread that is ready to execute. If a thread is READY, that thread will.execute. Associated with each thread is its 'time-slice'. This time-slice is the maximum time a thread can execute for each round the scheduler makes.



Some of the most widely used RTOS are :

- OSE
- QNX
- RTLinux
- VxWorks
- Windows CE

VI. CLASSIFICATION OF RTOS

RTOS can be classified into three types :

Hard RTOS : These type of RTOS strictly adhere to the deadline associated with the tasks. Missing on a deadline can have catastrophic affects. The air-bag example we discussed in the beginning of this article is example of a hard RTOS as missing a deadline there could cause a life.

Firm RTOS : These type of RTOS are also required to adhere to the deadlines because

missing a deadline may not cause a catastrophic affect but could cause undesired affects, like a huge reduction in quality of a product which is highly undesired.

Soft RTOS : In these type of RTOS, missing a deadline is acceptable. For example On-line Databases.

VII. CONCLUSION

Real-time systems have benefitted from a wealth of research in all areas of operating system design. Because of the temporal requirements of real-time tasks, traditional operating system design principles, algorithms, and techniques do not directly apply. As a result, every area of operating system research has needed extension into the real-time domain. Scheduling, memory management, process communication, file systems, networking, power management, garbage collection, fault tolerance, security, and many other aspects have been researched with real-time systems in mind, and a significant amount of progress has been made.

REFERENCES

[1] http://en.wikipedia.org/wiki/Real-time_operating_system

[2] http://www.iar.com/Global/Resources/Developers_Toolbox/RTOS_and_Middleware

[3] http://www.thegeekstuff.com/2012/02/rtos-basics/

[4] http://www.cs.uiuc.edu/class/sp05/cs523/prev_exams/2004spring/midterm/magee/node8.html