Real Time Operating System (RTOS) With Its Effective Scheduling Techniques

Panini A. Trivedi

V.V.P. Engineering College
Rajkot, Gujarat, India
Pahinitrivedi21@gmail.com

Abstract **- A Real Time Operating System (RTOS) comprises of two components, viz., "Real-Time" and "Operating System". Real-time systems are 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. RTOS are those which must produce correct responses within a definite time limit. A RTOS is any information processing system that has to respond to externally generated signal within a finite and specified period. It is a computer system where the correct functioning of the system depends on the results produced and the time at which they are produced. Scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth).By use of Proper techniques of scheduling, we can perform multiple task in a given time.**

Keyword - RTOS (Real Time Operating System),HRT(Hard Real Time), Orchestration, RMS(Rate Monitoring Scheduling).

I. INTRODUCTION

RTOS is an operating system that supports real-time applications by providing logically correct result within the deadline required. Basic Structure is similar to regular OS but, in addition, it provides mechanisms to allow real time scheduling of tasks. An Operating system (OS) is nothing but a collection of system calls or functions which provides an interface between hardware and application programs. It manages the hardware resources of a computer and hosting applications that run on the computer. An OS typically provides multitasking, synchronization, Interrupt and Event Handling, Input/ Output, Inter-task Communication, Timers and Clocks and Memory Management. Core of the OS is the kernel which is typically a small, high optimized set of libraries.

Though real-time operating systems may or may not increase the speed of execution, they can provide much more precise and predictable timing characteristics than general-purpose OS. RTOS is key to many embedded systems and provides a platform to build applications. All embedded systems are not designed with RTOS. Embedded systems with relatively simple/small hardware/code might not require an RTOS. Embedded systems with moderate-to-large software applications require some form of scheduling, and hence RTOS.

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.

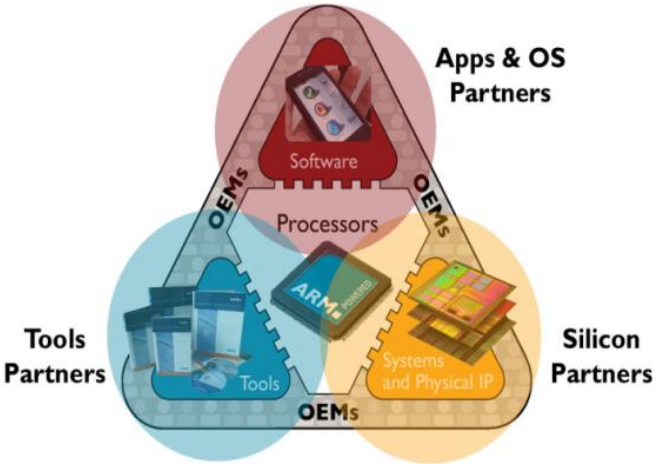


Fig. 1 RTOS Image

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*.

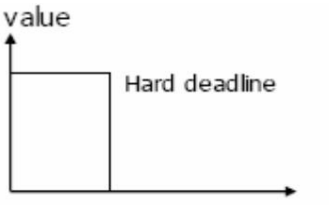
II. RTOS CLASSFICATION

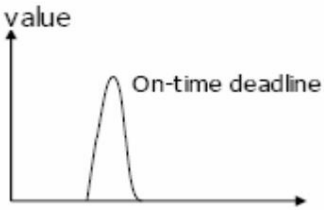
RTOS specifies a known maximum time for each of the operations that it performs. Based upon the degree of tolerance in meeting deadlines, RTOS are classified into following categories.

Hard real-time

Degree of tolerance for missed deadlines is negligible. A missed deadline can result in catastrophic failure of the system. An Hard Real Time (HRT) system is a system where not meeting a deadline can have catastrophic effects. HRT systems require a much more strict definition and could be described as follow:

"*An hard real time system is a system where the programmed reaction to a stimulus is guaranteed to be completed within a known finite time*".

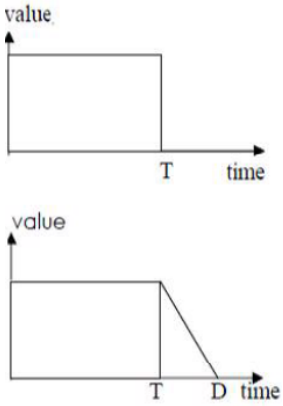




Graph 1. Hard Real-Time

Firm real-time

Missing a deadly ne might result in an unacceptable quality reduction but may not lead to failure of the complete system. A single system may have both hard and soft real-time Subsystems. In reality many systems will have a cost function associated with missing each deadline.

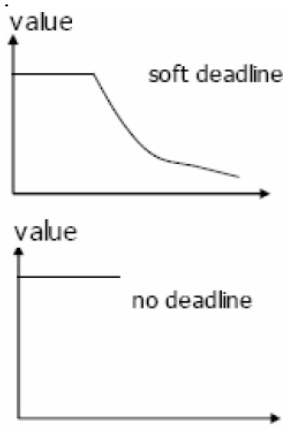


Graph 2. Firm Real-Time

Soft real-time

Deadlines may be missed occasionally, but system doesn't fail and also, system quality is acceptable. A Soft real time system is a system where not meeting a deadline can have undesirable but not catastrophic effects, performance degradation for example. SRTs could be described as follow:

"*A soft real time system is a system where the programmed reaction to a stimulus is almost always completed within a known finite time*".



Graph 3. Soft Real-Time

III. FEATURES OF RTOS

An RTOS must be designed in a way that it should strike a balance between supporting a rich feature set for development and deployment of real time applications and not compromising on the deadlines and predictability.

The following details describe the features of an RTOS (Note that this list is not exhaustive). Context switching latency should be short. This means that the time taken while saving the context of current task and then switching over to another task should be short.

The time taken between executing the last instruction of an interrupted task and executing the first instruction of interrupt handler should be predictable and short. This is also known as interrupt latency. Similarly the time taken between executing the last instruction of the interrupt handler and executing the next task should also be short and predictable. This is also known as interrupt dispatch latency.

Reliable and time bound inter process mechanisms should be in place for processes to communicate with each other in a timely manner.

An RTOS should have support for multitasking and task preemption. Preemption means to switch from a currently executing task to a high priority task ready and waiting to be executed.

Real time Operating systems but support kernel preemption where-in a process in kernel can be preempted by some other process.

IV. SCHEDULING

Scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). 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 and minimal thread switching 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.

The scheduler is concerned mainly with:

- Throughput - The total number of processes that complete their execution per time unit.
- Response time - amount of time it takes from when a request was submitted until the first response is produced.
- Waiting Time - Equal CPU time to each process (or more generally appropriate times according to each process' priority). It is the time for which the process remains in the ready queue.

V. RTOS TASK SCHEDULING TECHNIQUES

A. Preemptive Scheduling

Each Task has a priority relative to all other tasks. The most critical Task is assigned the highest priority. The highest priority Task that is ready to run gets control of the processor .A Task runs until it yields, terminates, or blocks, Each Task has its own memory stack. Before a Task can run it must load its context from its memory stack (this can take many cycles).

In this primitive scheduling model a task must be in one of four states:

- Running – the task is in control of the CPU.
- Ready – the task is not blocked and is ready to receive control of the CPU when the scheduling policy indicates it is the highest priority task in the system that is not blocked.
- Inactive – the task is blocked and requires initialization in order to become ready.

Blocked – the task is waiting for something to happen or for a resource to become available.

Typical RTOS Task Model

Typical RTOS based on fixed-priority preemptive scheduler, Assign each process a priority. At any time, scheduler runs highest priority process ready to run. Process runs to completion unless preempted.

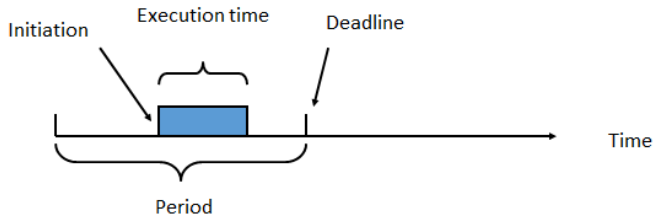


Fig. 2 Typical RTOS Task Model

Priority-based Preemptive Scheduling always runs the highest-priority runnable process.

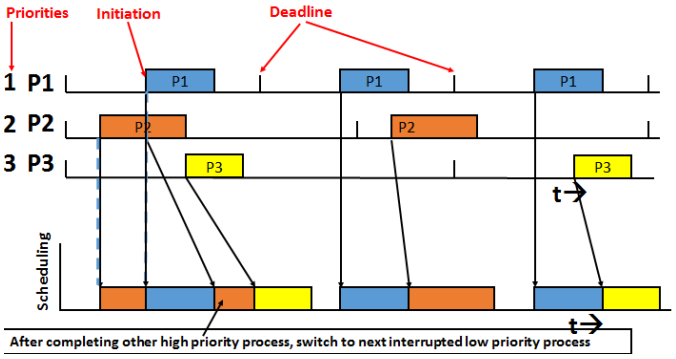


Fig. 3 Priority-based Preemtive Scheduling

A. Cyclic Scheduling

This is an important way to sequence tasks in a real-time system. Cyclic scheduling is static computed offline and stored in a table.

Non-periodic work can be run during time slots not used by periodic tasks, implicit low priority for non-periodic work. Usually non-periodic work must be scheduled pre-emptively.

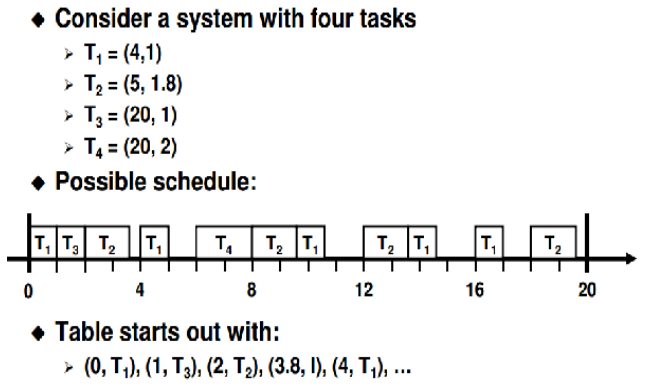


Fig. 4 Example of Cyclic Scheduling

B. Round Robin (Time Sliced) Scheduling

Round robin means that each ready task runs turn by turn only in a cyclic queue for a limited time slice,

$$T_{slice} = \frac{T_{cycle}}{N}$$

Where,

$$T_{slice} = Limited\ time\ slice$$
$$T_{cycle} = Time\ cycle$$
$$N = Number\ of\ tasks$$

Round robin is a hybrid model of clock-driven model (for example cyclic model) as well as event driven (for example, pre-emptive).A real time system responds to the event within a bound time limit and within an explicit time.

A round-robin rotation can happen because of the following events:

The currently executed thread voluntarily invokes the chThdYield () API in order to allow the execution of another thread at the same priority level, if any.

The currently executed thread voluntarily goes into a sleep state, when the thread is awakened it goes behind any other ready thread at the same priority level.

The currently executed thread is preempted by a higher priority thread; the thread is reinserted in the ready list behind any other thread at the same priority level.

If the CH_TIME_QUANTUM configuration constant is set to a value greater than zero and if the specified time quantum expired and if a thread with equal priority is ready then the currently executing thread is preempted and reinserted in the ready list behind any other thread at the same priority level.

Round-Robin Analysis

If there are n processes in the ready queue and the time slice is q, then each process ideally would get $\frac{1}{n}$ of the CPU time in chunks of q time units, and each process would wait no longer than nq time units until its next quantum. A more realistic formula would be **n(q+o)** where o is the context switch overhead. So, for practical purposes, it is desirable that the context switch be negligible compared to the time slice.

The performance of the Round-Robin algorithm depends heavily on the size of the quantum. If the quantum is very large, the Round-Robin algorithm is similar to the First-Come, First-Served algorithm. If the quantum is very small, the Round-Robin approach is called *processor sharing*.

C. Rate-Monotonic Scheduling

Rate Monotonic Scheduling is Common way to assign priorities.Processes with shorter period given higher priority.A set of *n* independent tasks scheduled by the rate monotonic algorithm. will always meet its deadlines, for all task phasing's, if

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots\ldots\ldots\ldots\ldots \frac{C_n}{T_n} \leq U(n)$$

Where

- C_n = deterministic computation time of each task
- Tn = period of particular task
- $U(n)$ = *Scheduling bound*, the maximum fraction of processor utilization allowable for *n* tasks: U(n) = 1 or 100%

The utilization bound (UB) test allows schedulability analysis by comparing the calculated utilization for a set of tasks and comparing that total to the theoretical utilization for that number of tasks:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots\ldots\ldots\ldots\ldots \frac{C_n}{T_n} \leq U(n) = 1$$

If this equality is satisfied, all of the tasks will always meet their deadlines. If the total utilization calculates to greater than 100%, the system will have scheduling problems.

Utilization Bound Test

Assumes rate monotonic priority assignment, Task with smaller period is assigned higher priority, Guaranteed to be schedulable if test succeeds.

$$\mathbf{U(n) \equiv \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1}$$

In computer science, Rate-Monotonic Scheduling is a scheduling algorithm used in real-time operating systems with a static-priority scheduling class. The static priorities are assigned on the basis of the cycle duration of the job. The shorter the cycle duration is, the higher is the job's priority.

These operating systems are generally preemptive and have deterministic guarantees with regard to response times. Rate monotonic analysis is used in conjunction with those systems to provide scheduling guarantees for a particular application.

Liu & Layland (1973) proved that for a set of *n* periodic tasks with unique periods, a feasible schedule that will always meet deadlines exists if the CPU utilization is below a specific bound (depending on the number of tasks). The schedulability test for RMS is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

Where,

C_i is the computation time,

T_i is the release period (with deadline one period later), and

n is the number of processes to be scheduled.

For example $U \leq 0.8284$ for $n = 2$. When the number of processes tends towards infinity this expression will tend towards:

$$\lim_{n \to \infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 0.693147\ldots$$

So a rough estimate is that RMS in the general case can meet all the deadlines if CPU utilization is 69.3%. The other 30.7% of the CPU can be dedicated to lower-priority non real-time tasks. It is known that a randomly generated periodic task system will meet all deadlines when the utilization is 85% or less, however this fact depends on knowing the exact task statistics (periods, deadlines) which cannot be guaranteed for all task sets.

VI. CONCLUSIONS

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. The benefits of using real-time models are very real, and absolutely necessary for some tasks.

VII. REFERENCES

[1] Cyprian F. Ngolah, Yingxu Wang, and Xinming Tan "IMPLEMENTING TASK SCHEDULING AND EVENT HANDLING IN RTOS+" Univ. of Calgary, 2500 University Drive ***NW,*** Calgary, AB, Canada T2N 1N4.

[2] Arezou Mohammadi and Selim G. Akl "Scheduling Algorithms for Real-Time Systems" School of Computing, Queen's University, Kingston, Ontario, Canada K7L 3N6, July 15, 2005.

[3] Jiangtao Wu, Xiang Long, and Lei Wang "Safety mechanism of RTOS on multi-core processor" State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing, China. 2011 International Conference on System Science, Engineering Design and Manufacturing Informatization.

[4] Jonathan L. Herman, Christopher J. Kenna,Malcolm S. Mollison,,James H. Anderson and Daniel M. Johnson "RTOS Support for Multicore Mixed-Criticality Systems *" The University of North Carolina at Chapel Hill, Northrop Grumman Corp. 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium.

[5] Xiongli Gu , Peng Liu , Mei Yang , Jie Yang , Cheng Li , Qingdong Yao "An efficient scheduler of RTOS for multi/many-core system" Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, United States.