International Journal of Science Engineering and Advance Technology

Consideration And Evolution of Real Time Operating Systems: RTOS

[1]Srinivas Pithani, [2]B.V.V.S.R.K.K.Pavan
Electronics And Communication Department,[1,2]kiet+, Korangi, Eastgodavari Dist., A.P, India
[1]sreenivaspithani@gmail.Com, [2]bhavaraju.Pavan5@gmail.Com

Abstract

This context deliberates the literature evolution of RTOS (Real Time Operating Systems) and its contributions to the embedded world. RTOS is is an operating system (OS) intended to serve real-time application 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. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.It is often associated with few misconceptions & we have tried to throw some light on it. We have specified few commercial RTOS of uncommon categories of real-time applications and have discussed its real-time features. A comparison of the commercial RTOSs' is presented. We conclude by discussing the results of the survey and comparing the RTOS based on performance parameters.

Keywords- μC/OS-II, RTAI,RTOS

I .Introduction

In general, an operating system (OS) is responsible for managing the hardware resources of a computer and hosting applications that run on the computer. An RTOS performs these tasks, but is also specially designed to run applications with very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard.

We realize its contribution in making our life comfortable and safe, for that it has to satisfy time and memory constraints. The performance of these systems depends on the OS which are used. Most of these systems require RTOS for such a precise task. The first RTOS was produced more than 20 years ago by DEC for the PDP family of machines, which then undergone the process of evolution. Real time operating System (RTOS), as the name suggests provides a deadline associated with tasks and an RTOS adheres to this deadline as missing a deadline can cause affects ranging from undesired to catastrophic. RTOS must be deterministic and pre-emptive. An RTOS is effective and allows the real-time applications to be designed and expanded more easily whilst meeting the performances required.

Abbreviations and Acronyms:
IPC – Interprocess communication
MPU – Microprocessor unit
DEC – Digital Equipment Corporation
LoC– Lines of code

An operating system generally consists of two parts: kernel space (kernel mode) and user space (user mode).The basic part of any OS and which acts as a bridge between applications and the actual data processing at the hardware level, is a kernel. A kernel can provide the lowest-level abstraction layer for the resources (especially processors and I/O devices)

The types of kernel are discussed below:

A. Monolithic Kernel :-

- Monolithic systems are often known as "The Big Mess" or spaghetti code.
- This type of kernel was prominent in the early days.
- Here, the system is a collection of procedures.
- Each module calls any other module.
- No information hiding (as opposed to modules, packages, classes which are used now)

With millions of LoC and 1-16 bugs per 1000 LoC monolithic systems are likely to contain many bugs. This type of OS has the problem of being difficult to debug. From a high-level reliability perspective, a monolithic kernel is unstructured.

B. Micro-kernel :-

- Here, code moves as much as possible from the kernel into "user" space.
- Communication takes place between usermodules using message passing.
- It is easier to extend a microkernel and to porttheoperating system to new architectures.
- It is more reliable (less code to run in kernel mode) and more secure as well.

C. Exo-kernel : -

- Here kernel separates hardware from application.
- Kernel allocates physical resources to application. It is based on conceptual model (files systems, virtual address space, schedulers, sockets)
- Application decides what to do with these sources. It can link to a lib OS to emulate a conventional OS.
- Usually used where strong hardware interaction is required.
  RTOS can be defined as "The ability of the operating system to provide a required level of service in a bounded response time."[2] RTOSs' are broadly classified into two categories, namely, hard real time and soft real time as described below:

A. Hard RTOS:These types of RTOS strictly adhere to the deadline associated with the tasks. These systems can't tolerate any delay, otherwise the system will break. For ex, break system in automotives, pacemakers.

B. Soft RTOS:In these types of RTOS, missing a deadline is acceptable. These systems can tolerate delay but is highly undesired. e.g. on-line Databases. In addition, they are classified according to the types of hardware devices (e.g. 8-bit, 16-bit, 32-bit MPU) supported.

II. 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 [3]

A. Task Priority:

Preemption defines the capability to identify the task that needs a resource the most and allocate it the control to obtain the resource. In RTOS, such capability is achieved by assigning individual task with the appropriate priority level. Thus, it is important for RTOS to be equipped with this feature.

B. *Reliable and Sufficient Inter Task CommunicationMechanism:*

For multiple tasks to communicate in a timely manner and to ensure data integrity among each other, reliable and sufficient inter-task communication and synchronization mechanisms are required.

Other features of RTOS have been discussed during performance analysis done later in the paper. Here the RTOS are chosen from different categories which are:-

- µC/OS-II: Most compatible RTOS
- EMBOS : Priority-controlled multitasking system
- QNX Neutrino: Commonly used for multiple node systems.
- RTAI : Real Time Application Interface for Linux

III. MISCONCEPTIONS REGARDING RTOS and THEIR CLARIFICATION:-

- RTOS is very fast: This is not true.
  An RTOS should have a deterministic behavior in terms of deadlines but it's not true that the processing speed of an RTOS is fast. This ability of responsiveness of an RTOS does not mean that they are fast.
- All RTOS are same.
  As already discussed there are two types of RTOS (Hard & soft).
- RTOS uses considerable amount of CPU overhead. This is not true. Only 1%-4% of CPU time is required by an RTOS [4]
- There is no science in RTOS system design. But most good science grew out of attempts to solve practical problems faced by different embedded systems.
- RTOS always consumes lot of energy.

Energy consumption of any system is mainly dependent on hardware used & architecture followed.

IV. LIMITATIONS of RTOS:

- It can be Costly.
- RTOS are generally complicated and can consume a non-trivial amount of processor cycles.
- RTOS doesn't support multitasking with absence of round-robin scheduling.

V. RTOS UNDER COMPARISON:

A. µC/OS-II: µC/OS-IIis a MicroC/OS kernel was originally published in a three-part article in Embedded Systems Programming magazine and the book "µC/OS The Real-Time Kernel" by Jean J. Labrosse. based on the source code written for µC/OS, and introduced as a commercial product in 1998, µC/OS-II is a portable, ROM-able, scalable, pre-emptive, real-time, deterministic, multitasking kernel for microprocessors, and DSPs. It manages up to 255 application tasks, and its footprint can be scaled (between 5 Kbytes to 24 Kbytes) to only contain the features required for a specific application..

B. EMBOS:EMBOSS is well established. It is used in demanding production environments reflecting the maturity of the code base. A major new stable version is released each year. For those who need the latest code, the current source code tree can be downloaded via CVS. There have been many thousands of downloads including site-wide installations by administrators across the world,

catering for hundreds or even thousands of users. Many interfaces to EMBOSS are available including easy to use web interfaces and powerful workflow software, enabling applications to be combined into analysis pipelines..[6]

C. QNX Neutrino:It is a commercial Unix-like RTOS, originally developed in 1982 by Canadian company Quantum Software Systems. QNX was one of the first commercially successful microkernel O.S and is used in a variety of devices including cars and mobile phones.[7]

D. RTAI:RTAIstands for Real-Time Application Interface. It is a real-time extension for the Linux kernel - which lets you write applications with strict timing constraints for Linux. It's used to control robots, data acquisition systems, manufacturing plants, and other time-sensitive instruments and machines. It also enhances the hard real-time scheduling capabilities and primitives for applications to use it.[8]

VI. PERFORMANCE ANALYSIS PARAMETERS:

A. Deadlock:

It is a situation in which two or more competing processes are each waiting for other to finish and thus neither ever does. It usually results into resource starvation or lag in execution. As we know any delay in result from RTOS can be catastrophic, so we expect RTOS to preferably avoid situation of deadlock or handle it efficiently.

Conditions for deadlock:

- Mutual exclusion
- Circular wait
- Hold and wait
- Less preemption

These four conditions are known as the Coffman conditions from their first description in a 1971 article by Edward G. Coffman, Jr. [9] Probability of RTOS of getting into a deadlock condition depends on the degree of preemption.

1) Avoiding deadlock: Best approach is to prevent one of the four Coffman conditions from occurring, especially the fourth one. Also methods like acquiring additional information in advance about the process and deciding whether a process should wait or not can also be used by a RTOS to prevent deadlock.

2) Deadlock handling: If deadlock occurs, RTOS should be able to recover from it, as early as possible. Usually solution for recovery is process termination, which is implemented as:

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated. B. Memory footprint:

Memory footprint is an estimate of RAM and ROM requirements of an RTOS on a specific embedded platform. Effective code, read-only data of the kernel, and any runtime library code are all collectively part of the ROM size. RAM requirements on the other hand are a sum of data structures and global variables and temporary programs. Memory footprint values depend upon architecture of hardware platform, compiler settings (optimizations) & most importantly OS configurations which include kernel size & size of run-time libraries. Footprint metrics are often an important decision factor when considering an RTOS solution, especially in situations where devices have limited on-chip memory and no possibility of interfacing with external memory. [10] Lower footprint value of RTOS may reduce cost dedicated for storage hardware while planning a project and can also further increase scope of adding more functions in the system keeping the memory size same.

C. Portability/ Compatibility:

Often, a current application may outgrow the hardware it was originally designed for, as the requirements of the product increases. An RTOS with such a capability can thus be ported between processor architectures. Thus, giving flexibility to choose hardware according to project equirements.

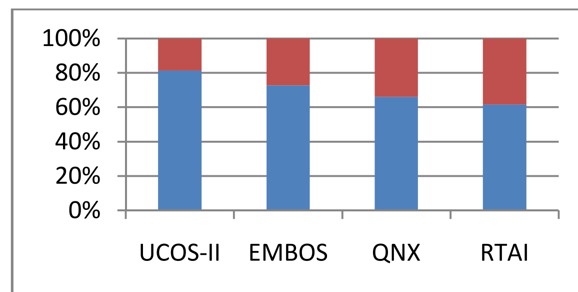

Fig1. No. of Platforms supported by RTOS under survey

D. Development tools provided:

A sufficient set of development tools including debugger; compiler and performance profiler might help in shortening the development and debugging time, and improve the reliability of the coding. Commercial RTOSs usually have a complete set of tools for analyzing and optimizing the RTOSs' behavior whereas Open-Source RTOSs may not have the same, with them.

E. Security provided:

Until last decade, RTOS had to serve its typical features, but from last few years, due to attacks of

malicious software on important systems & network it has to provide complete security. Here security refers to protection which secures system from any unauthorized access inside the system and as well as outside the system. They are expected to follow POSIX standards of security to serve current embedded applications.

F. Run-time performance:

Run-time performance of an RTOS is generally governed by the interrupt latency, context switching time and few other metric of kernel performance. This consideration is useful if the performance assessment of the application on a given RTOS is to prototype its performance-critical aspects on standard hardware [12].

1) Latency: Latency is analyses externally taking the RTOS under test in conjunction with the hardware as a black box. The latency consists of the time difference between the moment that an interrupt is generated and the moment that the associated interrupt handler generates an external response.

2) Jitter: Jitter is indirect information obtained from several latency measures, consisting of a random variation between each latency value. In a RTOS, the jitter impact could be notorious, as it is analyzed by Proctor when trying to control step motors.

3) Worst Case Response Time: Worst Case Response Time is obtained using the method proposed by ISA that was discussed above analyzing the maximum interrupts frequency that is handled by the RTOS with reliability. The worst case response time is the inverse of the maximum frequency obtained. (The test proposed by the A&CS consists in setting a system that copies the input signal directly to an output port, and measuring how many pulses was generated in the input and how many were copied to the output. Theoretically, while the system is stable, the accumulated number of pulses in both ports should be equal. Later, the input signal frequency should be slowly incremented until the input and output pulses count starts to diverge. In this moment, the frequency should be reduced until the maximum system operation frequency is found.)[13]

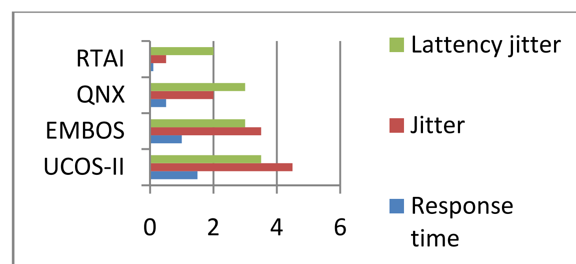

Fig3. Show worst case response time

A: Response Time (1/max sustained frequency), B: Latency, C: Latency Jitter

4) Scheduling algorithms: Scheduling algorithms used in an O.S, have a great impact on their run-time performance. Problems regarding multi-tasking & Task-priority are addressed by these algorithms. Frequency of context switching is also decided by these algorithms.

The µC/OS-II Scheduler:

Base Scheduling algorithm: - Priority Pre-emptive the tasks are done in order of their priorities. A particular task will be pre-empted, only if a task having higher priority is requested. Tasks that have higher periodic requirements should be given higher priority; tasks that make intensive computations should be given lower priority. µC/OS-II has an option to enable the Round-Robin mode, in which every task gets equal CPU burst turn by turn. Starvation is possible as the low priority tasks may not get CPU.

RTAI Scheduler: FPPS (Fixed Priority Preemptive Scheduling) is used in RTAI, in which higher priority jobs are completed first. Priority is again decided on basis of time constraints. Preemptive scheduling suffers problem of context switching. More the preemption more are the context switches. Fixed priority Scheduling with deferred pre-emption (FPDS) has been proposed instead of FPPS. FDPS is a mid-ground between FPPS and FPNS(Fixed Priority non-Preemptive Scheduling) and gives benefits of both the techniques.(gives advantage over context switching overhead and resource access control) Each job of FPDS contains sub-jobs and pre-emption is possible only between sub-jobs. Lesser the preemptions, lesser is the context switch.[14]

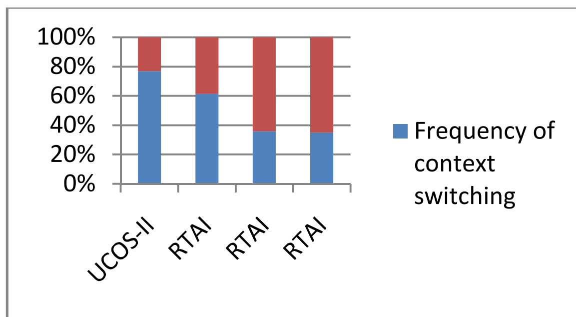

Fig.4 This Figure shows Frequency of context switching of RTOS under survey.

5) Interrupt Latency Interrupt Latency is defined as the sum of interrupt blocking time during which the kernel is pending to respond to an interrupt, saving the tasks context, determining the interrupt source, and invoking the interrupt handler. For a particular interrupt, the latency also includes the execution time of other nested interrupt handlers. Since most embedded systems is interrupt- driven, low interrupt latency will drastically increase system throughput.(In this experiment, we configured the MPC8260 hardware timer with a period of 50MHz to generate a timer interrupt every 20us)[15]

Table Number	Calculation of Interrupt Latency	
Interrupt latency (µS)	µCOS-II	RTAI
	125	132(1.2)

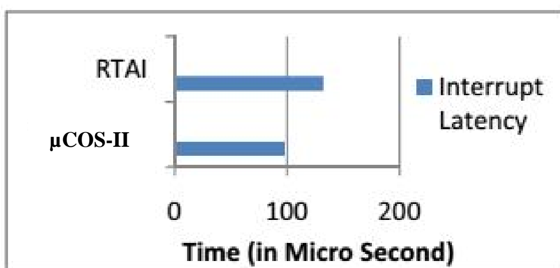

Fig5.Shows Interrupt Latency of RTAI and µCOSII

It is not surprising that VxWorks has much lower interrupt latency (35%) than RTAI. Traditional Linux is known for having high interrupt latency. It appears that even though RTAI had been added with real-time capability, it still exhibits some non-real-time behavior.

6) Inter-Process Communication

Modern real-time applications are constructed as a set of independent, cooperative tasks. Along with high-speed semaphores, µC/OS-II and RTAI also provide message queue as higher-level synchronization mechanism to allow cooperating tasks to communicate with each other. Because of the implementation complexity, using this service imposes the greatest amount of latency and thus is a key metric to operating system study.[16]

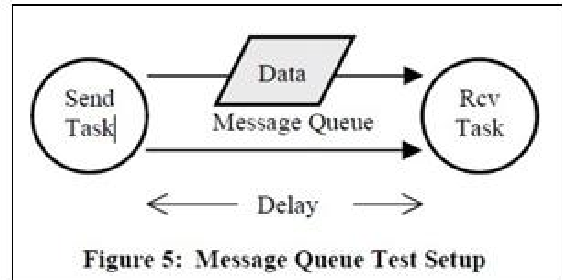

Fig.6 This Figure shows Message Queue Test Setup

This test is done by creating and activating (or open) a message queue. Next, spawning a receiving task from which the messages receive function is invoked. The
receive system call blocks the receiving task and put it in the wait state (since the message queue is empty). While the receiving task was waiting for the message, we spawned a sending task to send a message via the same message queue. The time between the sending task to call the message send function and the receiving task to receive message notification is given in Table below [17]:

Table Number	Calculation of message Queue Delay	
Message Queue Delay (µS)	µCOS-II	RTAI
	118(0.9)	113(1.8)

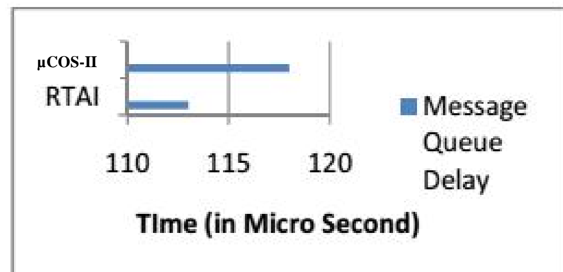

Fig. 7 This Figure shows Message Queue Delay in µCOS-IIs and RTAI.

VII. CONCLUSIONS:

Real time applications have become popular these days due to the complexity in the system. To meet those complexities, the developers are given the invariable task of making the real time software. There are quite large numbers of RTOS available in the market and one does get confused as to which one to select; such that it provides the efficient embedded systems design in terms of cost, power consumption, reliability, speed etc. Ranking RTOS is a tricky and difficult because there are so many good choices are available in the market. The developer can choose either commercial RTOS (44% developers are using) or open- source RTOS (20) or internally developed RTOS (17 %). From comparison and study of these selected commercial RTOS, we can infer that –

- RTAI and µC/OS-II provide a wide range of supported platforms.
- RTAI, an Open-source RTOS is most suited for small applications, such as robotics and medical devices.
- Whereas, µC/OS-II is dominant in highly complex and high performance applications.
- When we have many interrupts µC/OS-II is the better pick & it is faster than RTAI. Applications which are getting number of interrupts continuously in such case µC/OS-II should be preferred. Interrupts will tend to increase when number of inputs is applied to the RTOSsimultaneously.

REFERENCES

[1]Real Time Operating Systems by-KarteekIrukulla [Online]. Available: http://www.scribd.com/doc/40401275/RTOS

[2] POSIX Standard 1003.1

[3]Features of RTOS [Online]. Available: http://www.thegeekstuff /2012/02/rtos-basics/

[4]Deadlock[Online].Available: en.wikipedia.org/wiki/Deadlock

[5] http://en.wikipedia.org/wiki/µC/OS-II

[6]http://en.wikipedia.org/wiki/EMBOS

[7] http://en.wikipedia.org/wiki/QNX

[8] http://en.wikipedia.org/wiki/RTAI

[9]Coffmanconditions[Online]. Available:http://www.mentor.com/embedded-software/sources/overview/measuring-rtos-performancewhat-why-how

[10]A.Silberschatz, P.B.Galvin and G.Gagne, “Operating System Concepts”

[11]RTOS Selection Guide [Online].Available: ttp://www.zembedded.com/rtos-basic-selection-guide/

[12] Rafael V. Aroca, GlaucoCaurin – “A Real Time Operating Systems (RTOS) Comparison”

[13] Rafael V. Aroca, GlaucoCaurin – “A Real Time Operating Systems (RTOS) Comparison”

[14] Mark Bergsma, Mike Holenderski, Reinder J. Bril and Johan J. Lukkien–“Extending RTAI/Linux with Fixed-Priority Scheduling with Deferred Preemption”

[15]Jun Sun, “Interrupt Latency”, Monta Vista Software[Online].Available http://www.mvista.com/realtime/latency/

[16] Performance Analysis of µC/OS-II and RTAI [Online].Available: Performance Analysis of VxWorks and RTLinux by Benjamin Ip http://www.scribd.com/doc/3792519/Vxworks-Vs-RTLinux.

[17] J.A.Stankovic and K.Ramamritham, -“The spring kernel: A new paradigm for real-time operating systems