Survey on RTOS: Evolution, Types and Current Research

Nandana V.
Dept.of Computer Science
LBS Institute of Technology for Women
Trivandrum, India

Jithendran A.
QuEST Global Engineering Services Pvt. Ltd.
Trivandrum, India

Shreelekshmi R.
Dept.of Computer Science
LBS Institute of Technology for Women
Trivandrum, India

ABSTRACT
Traditionally in embedded systems, real time tasks are implemented using a simple scheduling algorithm. Embedded systems are mostly constrained in size and resource requirements, hence scheduling algorithm is preferred. Due to the remarkable advancement in the embedded area, numerous real time operating systems (RTOS) have been developed in the recent years. This paper presents the literature survey which gives an overview about the evolution of real time systems and its current scenario. Differences between RTOS and General Purpose Operating System (GPOS) are listed. The challenges faced by developers while using an RTOS are also explored.

Keywords
RTOS, Evolution, GPOS

1. INTRODUCTION
RTOS is an OS that produces results in real time. Efficiency of the system depends on the logical correctness as well as the time at which result is produced. Unlike GPOS which focuses on the amount of work done within a given time frame, an RTOS is more focused on the criticality of timeliness. If the result is delayed even for a millisecond, it is considered as a system failure. Thus RTOS is mostly used for time critical applications which require minimum buffering delays.

This paper is organized as follows. Initially the evolution of real time systems is discussed with a description of rate monotonic scheduling. This is followed by differences between RTOS and GPOS and different types of RTOS. Finally challenges and current research are listed followed by conclusion. A number of surveys have been done before in this area, but this is the first wherein the evolution, challenges and the current research work in the area of RTOS is consolidated.

2. EVOLUTION
The concept of real time systems was introduced almost seventy years back. James Martin who was a famous British information technology consultant and author, proposed one of the first definitions for real time systems – A real time computer system may be defined as one which controls an environment by receiving data, processing them and taking action or returning results sufficiently quickly to affect the functioning of the environment at that time [1]. In the initial years, there was no operating system developed that was inherently real time. Whenever there was a requirement for a real time task, the whole system, both the hardware and software was designed in such a way that it is customized to that particular real time task alone. The origin of real time systems can be traced back to two major areas; operations research and queuing theory [2, 3]. The roots of real time systems are deeply embedded in these two areas. Operations research is mainly involved in decision making process. By the use of advanced analytical models, it helps in strategic decision making. In the context of real time systems, it helps us to decide which task should be run next by an operating system. Queuing theory, as the name implies, deals with the learning of queues. The amount of time an element should wait in the queue, the average length of the queue etc is studied in this area. Queues play a major role in real time systems as a number of real time tasks will be ready to run in a particular instance, all of which will be waiting inside a queue. Operating system should use data structures in such a way that it minimizes the waiting time of the tasks, and no high priority task should be kept waiting while a lower one is being executed.

In the past, computers used to execute tasks in batch mode. Inputs are predefined and the system produces the complete set of results after a particular amount of time. This model was not suited in situations where inputs are constantly changing and cannot be predicted beforehand. There came a need to develop systems that produce time critical as well as safety critical results. This led to the concept of real time systems.

Whirlwind I was the first system that produced results in real time [4]. It was developed in the late 1940s. During the cold war, U.S. Navy required a flight simulator to train bomber crews. The requirement was that the aerodynamics model used by then system should be adapted to any plane. An analog computer was developed initially which was huge and inflexible. Later focus was shifted to digital systems. It took three years to build the vacuum tube computer, which went online on April 20, 1951. The computer takes control input from pilots and updates a simulated instrument panel.

After three years, Navy lost interest but Air Force took over. Air Force tried to use computers to help the task of ground controlled interception, and Whirlwind was the prospective candidate. Thus Whirlwind I directly led to the development of Whirlwind II design which was used in Semi-Automatic Ground Environment (SAGE). United States Air Force SAGE air defense system was developed in 1957 [5]. SAGE helped in capturing the bird's eye view of the target area so that bombers can easily target with less amount of time. Numerous small images were captured from various radar sites and unified by the real time system to produce one large image of the area. Communication between sites was carried out through teleprinters.

After Navy and Air Force, American Airlines took interest in real time systems. Initially airline ticket booking was a tedious task. Entire process was done manually were eight operators

used to sit around a table. Whenever a ticket was booked, the operator placed a mark on the side of the corresponding card. A fully booked flight will have its entire card marked. Thus the availability of a flight was seen visually. The process was not time consuming due to the number of flights, but the manual process of checking the availability, marking the card, and printing out the ticket at times took more than an hour. Also due to space constraints, no more than eight operators can be placed around a file. This called for the need of a real time system and SAGE design was considered the most appropriate candidate.

In 1959 Semi-automated Business Research Environment (SABRE) was developed [6]. It automated the American airline reservation system. Ticket availability was checked automatically by the system and tickets were printed out instantly. The teleprinters used in SAGE were used here to receive request and send responses.

In 1960s non military interests in real time systems were developed and the first commercial real time operating systems were developed for mainframe computers. IBM developed Basic Executive in 1962 which provided diverse real time scheduling. It was followed by Basic Executive II. In 1970s focus changed to mini computers. RT-11 was developed which was a small, single-user real-time operating system for the Digital Equipment Corporation (DEC) PDP-11 family of 16-bit computers. Real-time systems, process control, and data acquisition were the most important applications of RT-11. Later RSX-11 was developed. It was designed for and much used in process control and program development.

2.1 Rate Monotonic Scheduling

Before real time systems came into existence, round robin and time sharing algorithms were the prominent ones used. Since those were not suitable for time critical and safety critical applications, rate monotonic scheduling was introduced [7]. It was invented by Liu and Leyland in 1973. This scheduling formed the core kernel of the initial real time systems. In rate monotonic scheduling, the run time modeling of threads are studied. Using the previous history, the amount of time required by a thread to execute is computed and it is checked whether the thread will complete its execution within the prescribed deadline. Accordingly static priorities are assigned. Job having the smallest cycle duration will have the highest priority.

The main drawbacks of rate monotonic scheduling were priority inversion and deadlock. This was caused due to resource sharing. Deadlock is a situation where each process is waiting for the other process in a cycle. At least one resource required by each process is currently acquired by the next process in the cycle. Thus every process waits for each other in an infinite loop. Priority inversion is a situation where a higher priority process is preempted by a lower priority process since a resource required by the former is held by the latter. One of the solutions to prevent the above drawbacks is to disable preemption. As there are only limited resources available for use, it is impossible to avoid resource sharing in real life. Hence the above solution was not efficient. Instead of preventing resource sharing, methods were devised to prevent or control the drawbacks caused by it. Many algorithms were developed for the same including deadlock avoidance, deadlock recovery etc. Priority inheritance protocol is a prominent one among them in which a lower priority process will temporarily acquire the higher priority of the process it is currently blocking. Until 1980s, there was no particular language dedicated to the development of real time applications. Numerous programming languages were used by different programmers depending on their individual comfort level and knowledge. A need for unified language was recognized and ADA was developed [8]. It was initially licensed to United States Department of Defense (DoD). Due to its safety critical features it was later adapted to commercial applications in embedded systems were missing even a single deadline could result in catastrophic failures including human loss e.g. air traffic control. 1980s also witnessed another major change in trend. Microprocessors started finding a place in real time systems. Versatile Real-Time Executive (VRTX) is one among the prominent ones that was developed initially. VRTX runs the Hubble Space Telescope.

Since then the embedded technology has come a long way. With the advent of ICs, microprocessors, microcontrollers, SoC etc embedded technology found application in large number of areas. Initially focus was only on military and space applications mostly funded by the government. Later with cheaper technologies, embedded systems found place in a large no of consumer products. It plays a prominent role in our day to day life now e.g. smart homes. Technology is so advanced that day by day the hardware is becoming smaller and processing power is becoming higher. Currently numerous microcontrollers and processors are available in market optimized for a variety of applications. With the increase in the number of microprocessors and controllers, numerous RTOS started being developed, each customized to one or more no of the processors. A variety of RTOS is currently available in market. Developers choose RTOS based on the license, availability of ports, availability of development tools, familiarity with the language etc.

3. RTOS vs. GPOS

RTOS and GPOS are distinct from each other because of the following characteristics.

Time criticality vs. throughput: Main goal of RTOS is to achieve deterministic behavior. Results have to be produced within strict deadlines. Usually RTOS is used only for customized applications. Hence focus is more on the timeliness rather than the amount of work done, whereas in GPOS main aim is to achieve high throughput. GPOS is dedicated to a large number of tasks. Hence maximum amount of work has to be completed within a given amount of time.

Scheduling Algorithms: GPOS has the liberty to use any scheduling algorithm as long as the throughput is met. Whereas in RTOS, algorithm is always priority based. A higher priority task is never made to wait. Hybrid algorithms can be used but priority based algorithm must be one among them.

Latency: In GPOS there is unbounded dispatch latency. The more number of threads to schedule, more latency will get added. In RTOS, processes and threads in it has got bounded latencies due to the application of queuing model.

Hardware: RTOS is light weight and small in size compared to a GPOS. A GPOS is made for high end, general purpose systems. An RTOS is usually designed for a low end, stand alone device. It is economical to port an RTOS to an embedded system of limited expectations and functionalities.

4. TYPES OF RTOS

One of the important characteristic of RTOS is jitter. Jitter is the consistency in the amount of time with which an OS produces the result for a given input. Main goal of RTOS is to achieve minimum amount of jitter. An input should always

produce results within the exact same time, no matter how many times it is executed or when it is executed. This characteristic contributes to the deterministic behavior of an RTOS. RTOS can be broadly classified as follows [9].

4.1 Hard RTOS

Hard RTOS has the least amount of jitter among all types of RTOS. Missing even a single deadline is considered as total system failure. It might even cause human loss. Design of such RTOS requires considerable effort and it must undergo rigorous testing before put into use. E.g. RTOS used in airplane control systems. Failure of even a single task or delay of even a microsecond in enabling a function might result in plane crash.

4.2 Firm RTOS

In firm RTOS, missing a deadline can cause catastrophic results, but not human loss. With more number of failures, performance degrades heavily. E.g. RTOS used in ATM machines. The delay in execution of function results in displeasure of customer. Crashing of system midway might even result in monitory loss of the customer.

4.3 Soft RTOS

Soft RTOS has the least amount of jitter among all the RTOS. The performance level required is determined beforehand, and the system is expected to satisfy only the prescribed requirements. Failures are tolerated if the performance is above the prescribed level. E.g. RTOS used in live video streaming. Delay or loss in connection for a small period of time is tolerated as long as the speed and clarity requirements are met for the remaining time.

5. CHALLENGES

Before choosing an RTOS, developer needs to confirm which hardware platform he is going to work on. Changing the hardware midway might force a change in the RTOS as the new hardware might not be optimized for the previously chosen RTOS. Also the developer should be familiar with the development tool used. Not all development tools will be supported by every RTOS. Certain RTOS has wide range of developmental support provided by different communities and choice of such RTOS results in easier development of programs. Peripheral support and stack availability also varies widely from one RTOS to another. Jerry Krasner has enumerated some of the RTOS selection challenges as follows [10].

Importance of time to market: Competition is very high in the embedded market. It is not only important to develop quality products, but also to deploy it within the specified timeline so that we are ahead of our market competitors. Hence it is better to invest more in the beginning of the project rather than identifying the bottlenecks at a later time thus getting far behind in the competition.

Comparing design outcomes: Different RTOS have distinct characteristics of its own. Some will be more suited to certain applications than the other. Determining the correct RTOS has a direct influence on the time to market.

Avoiding overqualified RTOS: Avoiding overqualified RTOS is as important as choosing the correct RTOS. It might result in over complexity. Once the performance requirements are met, providing extra amount of memory or higher speed is of no use. Also unnecessary training will be required to understand the functionalities. Sometimes overqualified RTOS can even lead to misuse.

Problem of delays in embedded applications: Unlike normal projects embedded projects have problems of its own. The choice of hardware is of utmost important. It should have the capability of executing the functionalities implemented by the software. If care is not taken in the design phase, project will face difficulties in midway. A change in hardware might lead to change in software and the development tool used and at times the project has to be begun from scratch.

6. CURRENT RESEARCH

Table 1 details some of the prominent RTOS that currently exist in the market [11].

Table 1. List of RTOS

Name	License	Platforms
FreeRTOS	Modified GNU GPL	ARM, AVR, AVR32, ColdFire, HCS12, IA-32, Cortex-M3, MicroBlaze, MSP430, PIC, PIC32, Renesas H8/S, 8052, STM32, EFM32
LynxOS	Proprietary	Motorola 68010, x86/IA-32, ARM, Freescale PowerPC, PowerPC 970, LEON
RTLinux	GNU GPL	Same as Linux
TI-RTOS	BSD license	Primarily Texas Instruments: MSP430, MSP432, C2000, C5000, C6000, TI's ARM families (Cortex M3/4F, Cortex R4, Cortex A8, Cortex A15), SimpleLink Wireless MCUs (CC2xxx, CC3xxx)
ThreadX	Proprietary	ARC, ARM/Thumb, AVR32, BlackFin, 680x0-ColdFire, H8-300H, Luminary Micro Stellaris, M-CORE, MicroBlaze, PIC24-dsPIC, PIC32, MIPS, V8xx, Nios II, PowerPC, SH, SHARC, StarCore, STM32, StrongARM, TMS320C54x, TMS320C6x, x86/x386, XScale, Xtensa/Diamond, ZSP
VRTX	Proprietary	ARM, MIPS, PowerPC, RISC
VxWorks	Proprietary	ARM, IA-32, Intel 64, MIPS, PowerPC,SH-4, StrongARM, xScale
WindowsCE	Proprietary	x86, MIPS, ARM, SuperH

Extensive amount of research work is carried out in the field of RTOS. Flexible and energy aware scheduling is one main area. Real Time virtualization focuses on virtual machines with real-time performance requirements. Improving the design and development methods for safety-critical embedded systems is another area of focus. Electronics designers and

manufacturers are also focusing on real time systems. Certain semiconductor companies are developing RTOS on their own to be in par with the emerging trends and gain advantage over their market competitors. TI-RTOS is one such RTOS developed by Texas Instruments (TI) intended only for TI microcontrollers.

There are many research groups solely focused on real time systems. .DIstributed and Real-Time systems (DiRT) is a research group maintained by Department of Computer Science at University of North Carolina. They mainly focus on single and multiprocessor real-time operating systems. University of Waterloo has a Real-time Embedded Software Group that concentrates on research on real-time embedded software systems. Real-Time Systems Research Group at York has been conducting research in this area since 1990. Several other universities such as University of Pennsylvania, University of Texas etc. also has groups dedicated to real time systems alone.

Performance of RTOS is measured using various metrics. One such metric is provided by ThreadX, known as Thread-Metric benchmark suite. The suite contains distinct tests that checks most commonly used RTOS features. The number of RTOS events that can be processed within a given time interval is calculated. The more the number of events, higher is the efficiency. Table 2 details the performance comparison of ThreadX and FreeRTOS using Thread-Metric benchmark suite [12]. The tests were run on a Microchip 40 MIPS PIC24HJ256GP610 PIC24 16-bit microcontroller.

Table 2. Performance comparison of ThreadX and FreeRTOS

Test	ThreadX	FreeRTOS
Cooperative scheduling	11,847,800	Not supported
Preemptive scheduling	4,870,885	3,717,913
Interrupt processing	6,918,050	1,881,892
Interrupt preemption processing	3,052,151	2,400,967
Message processing	6,928,383	484,691
Synchronization processing	15,337,354	1,989,999

7. CONCLUSION

RTOS has evolved to a great extent in the recent years. Every RTOS has a distinct feature of its own. But the growth in this field is not fully utilized by the developers yet as most of the RTOS is licensed and highly expensive. Peripheral support and stack availability also varies widely from one RTOS to another. Out of the few available free/open source RTOS, support for proprietary protocols is minimal. Developers do not have the luxury of time to adapt RTOS to project requirements. Hence they are unwilling to change from their current working condition. It would be greatly beneficial if all the project requirements are listed in the beginning itself, so that the developer can ensure that the RTOS chosen has support for all the necessary drivers and protocols.

8. REFERENCES

[1] Phillip A. Laplante, Seppo J. Ovaska, 2011., "Real-Time Systems Design and Analysis: Tools for the Practitioner", Edition 4

[2] https://en.wikipedia.org/wiki/Operations_research.

[3] https://en.wikipedia.org/?title=Queueing_theory.

[4] https://en.wikipedia.org/wiki/Whirlwind_I.

[5] https://en.wikipedia.org/wiki/SemiAutomatic_Ground_Environment

[6] https://en.wikipedia.org/wiki/Sabre_(computer_system)

[7] https://en.wikipedia.org/?title=Rate-monotonic_scheduling

[8] https://en.wikipedia.org/wiki/Ada_(programming_language)

[9] https://en.wikipedia.org/wiki/Real-time_operating_system

[10] Jerry Krasner,2007, RTOS Selection and Its Impact on Enhancing Time-To-Market and On-Time Design Outcomes

[11] https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

[12] http://rtos.com/news/detail/Express_Logics_ThreadXMCU_RTOS_Scores_Top_Marks_in_Microchip_Technologys_PIC24_Benchmarks/