Real-Time Operating Systems: An Ongoing Review

Ramesh Yerraballi
Dept. of Computer Science and Engineering
University of Texas at Arlington

Abstract

Real-time operating systems have evolved over the years from being simple executives using cyclic scheduling to the current feature-rich operating environments. The standardization of POSIX 1003.1, ISO/IEC 9945-1 (real-time extensions to POSIX) has contributed significantly to this evolution, however, the specification leaves plenty of room for individual implementations to both interpret and specialize their RTOSs. Accordingly, there has been a proliferation of both commercial and free RTOSs, notably, the μITRON OS, the OSEK-VDX OS specification, commercial RTOSs like VxWorks, VRTX, LynxOS, OSE and QNX, and free RTOSs like RT-Linux (RTAI), and Windows CE.The goal of the work reported in this paper is to draw the real-time systems practitioner and researcher's attention to these choices and bring out the similarities and differences among them. Work is underway to, install, test and benchmark the aforementioned OSs to draw a more objective assessment.

1 Introduction

The primary role of an operating system (OS) is to manage resources so as to meet the demands of target applications. Traditional timesharing operating systems target application environments, that demand *fairness* and *high resource utilization*. Real-time applications on the other hand demand *timeliness* and *predictability* , and the operating systems targeting these applications meet these demands by paying special attention to a host of OS features like: (i) Multitasking (ii) Synchronization (iii) Interrupt and Event Handling (iv) Input/Output (v) Inter-task Communication (vi) Timers and Clocks (vii) Memory Management.

The design of a real-time operating system (RTOS) is essentially a balance between providing a reasonably rich feature set for application development and deployment and, not sacrificing predictability and timeliness. In addition to timeliness and predictability some other desirable characteristics have been identified in the various standards [10, 7, 6] specifications. In this paper, we attempt to demonstrate that the various RTOSs implementing these standards, differ in their implementation choices and strategies. This demonstration should allow a practitioner to choose the right RTOS for a particular application.

The specific real-time operating systems that are considered in this paper are: **LynxOS**: A UNIX-compatible, POSIX-conformant real-time operating system for embedded applications from Lynx Real-Time Systems Inc. It is scalable, fully re-entrant, preemptible, and ROMable [2].
μITRON: An open RTOS specification for embedded systems resulting from the TRON (The Real-Time Operating System Nucleus) project. Participant companies that have implemented the specification include, Fujitsu, Hitachi, Mitsubhishi, Miyazaki, Morson, Erg Co., Firmware Systems, NEC, Sony Corp., Three Ace Computer Corp., and Toshiba [10].
OSE: A commercial RTOS from Enea Data Systems that boasts to have bridged the gap between applications and the kernel by providing a rich set of features inside the kernel. It's message based architecture allows for efficient IPC and synchronization [5].
OSEK-VDX: The "Open Systems in Automotive Networks" RTOS specification that has been adopted by the following organizations in their embedded systems: Adam Opel AG, BMW AG,DaimlerChrysler AG, University of Karlsruhe - IIIT, PSA, Renault SA, Robert Bosch GmbH, Siemens AG, Volkswagen AG [6].
QNX: A real-time, extensible POSIX compliant OS with a lean micro-kernel and a team of optional cooperating processes. [9].
RTAI: It evolved from NMT_RTLinux (New Mexico Institute of Technology's Real-Time Linux), and takes a unique approach of running Linux as a task (lowest priority) that competes with other real-time tasks for the CPU [3].
VRTX: A highly reliable RTOS from Mentor Graphics that is the first to be certified under the US FAA's stringent RTCA/DO-178B level A standard for mission-critical aerospace systems. It is based on a Nanokernel running on top of a Hardware Abstraction Layer to provide fast and predictable response [11].
VxWorks: The most popular (and complete) commercial RTOS (from Wind River Systems) in the embedded industry with ports for virtually all CPUs in the market [12].
Windows CE: Microsoft's embedded operating system for handheld PCs and small embedded processors. Though it's current version (2.0) does not really qualify as an RTOS, both feature-wise and performance-wise, Microsoft promises to fix these shortcomings in version 3.0 [13].

The above list is in no way exhaustive but is a reasonable subset of more than 50 commercial, academic (research-based) and free RTOSs currently available. We have not included several excellent academic RTOSs which have been very well reviewed in [4]. Further, we only picked one of several (Lineo, ecos, Lynx Bluecat etc.) new RTOS derivatives of the Linux operating system. We refer to the

POSIX standard extensively in this paper. The POSIX standards related to real-time OS's (already ratified) consist of: 1003.1 for OS, process, filesystem and device API, 1003.2 for utilities, 1003.1b for real-time, and 1003.1c for threads. POSIX 1003.1d, which defines additional real-time extensions is not yet ratified.

The rest of the paper is organized as follows: Section 2 presents the motivation behind the study and reviews some related studies. In Section 3, several desirable characteristics of an RTOS are discussed in turn with a comparison of how the various RTOSs meet these desirables. We compare the performance of the various RTOSs in Section 4 with conclusions and future work in Section 5.

2 Motivation

A recent survey conducted by ITRON [10], among Japanese real-time systems developers showed (refer to Figure 1 in [10]) that one of the major concerns was the lack of personnel familiar with real-time systems (and embedded systems) technology. One major factor that is contributing to this is the lack of a common standard and the abundance of incompatible real-time operating systems in the market, each targeted towards a specific segment of the industry. There is a need therefore to draw the similarities and differences between these operating systems, so that a real-time system developer can make an intelligent choice for the application at hand.

Another related survey conducted by the TRON association reported that among Japanese companies using RTOSs for embedded systems, on an average 30-40% of them use an RTOS based on the ITRON specification. Other commercial OSs together amount to less than 30%, with the rest using either in-house RTOSs or no OS at all. It is interesting to note that very few American (Tandem Computers is one of select few) and European companies use RTOSs based on the ITRON specification. In particular the Automotive industry in Europe has adopted the OSEK RTOS [6] specification, while the rest of the world RTOS market is split between popular commercial OSs like LynxOS, QNX, VRTX and VxWorks.

Another motivation for the work reported in this paper is the lack of such information in the public domain. The only known source of such (Evaluation of RTOSs) information is through the Real-Time Consult (RTC) group[1] which charges a price that is only affordable by large corporations. It is the understanding of this author that the RTC report goes into finer details of RTOS performance perhaps at the level described by the Annex G of the POSIX 1003.1 standard. While, this is the eventual goal of our project, we have already made some progress in this direction to share the results with the academic community.

3 RTOS Feature Comparison

Figure 1 gives a functional diagram of an RTOS with its various components. The following discussion delves into these components, and their desirable functionality.

The main components in the functional diagram are the hardware and the kernel of the RTOS running on top of it

[1] http://www.realtime-consult.com

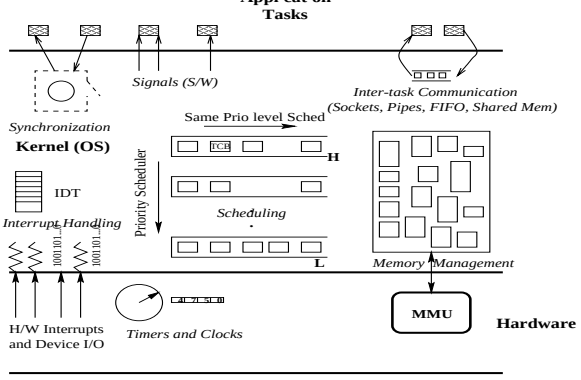

Figure 1. Real-Time Operating System: A Functional Diagram)

and servicing tasks and interrupts that comprise the real-time application. The OS has to provide, (i) task management (scheduling, dispatching, creation and termination of tasks etc.), (ii) synchronization (for resource sharing) (iii) interrupt handling (manipulate and monitor the interrupt descriptor table-IDT) to service hardware interrupts (iv) memory management (virtual memory and dynamic memory allocation) (v) programmable clocks and timers, and (vi) inter-task communication (sockets, pipes, FIFO, shared memory etc.). The following sub-sections will describe the desirable functionality from each of these components and how the various RTOSs compare.

Multitasking

It is essential for an RTOS to clearly distinguish between *schedulable* and *non-schedulable* entities. Schedulable entities are typically characterized by a context (a control block) and can make explicit requests for resources (CPU, memory, I/O), further they are scheduled by a *scheduler*. The scheduler itself and such entities like interrupt handlers, and most system calls are non-schedulable by nature. Often they are characterized by the fact that they can execute continuously, periodically or in response to events. Further, their use of the CPU is implicit.

Multi-tasking involves fast switching between tasks allowing multiple tasks to be in a state of execution yet only one task is executing at any instant. A RTOS must provide (at a minimum) a multi-tasking mechanism that is priority-based and preemptive in nature. It should provide sufficient number of priority levels to be of practical use. For example Windows CE provides only 8 priority levels making it rather impractical for use in a majority of real-time scenarios. All of the reviewed RTOSs support a priority-based

	OSE	OSEK-VDX	LynxOS	μITRON
Prio. Sched.	Preemptive and Cyclic	Non-Preemptive and Preemptive	Fixed. Prio. Preemptive	Fixed. Prio. Preemptive
Same-level Sched.	RR	FCFS	RR/Quantum/FCFS	FCFS
Threaded	Single	Single	Multi	Single
Priority Levels	32	≥ 8	$256 + 3 \times 256$	–

QNX	RTAI	VRTX	VxWorks	Win-CE
Fixed. Prio. Preemptive	Fixed. Prio. Preemptive	Fixed. Prio. Preemptive	Fixed. Prio. Preemptive	Fixed. Prio. Preemptive
RR/Adaptive	FCFS	RR	RR	RR (1-7) FCFS (0)
Single	Single	Single	Single	Single
32	2^{30}	256	256	8

Table 1. Multi-Tasking and Scheduling

preemptive scheduling mechanism with OSE and OSEK providing cyclic and non-preemptive scheduling in addition (see Table 1). A typical number of priority levels sufficient for most real-time applications is 32, however VRTX and VxWorks have 256 levels and LynxOS has 256 priority levels with another 256 levels each for the RR, Quantum and FCFS schedulers. RTAI allows for 2^{30} potential priority levels with Linux operating at priority level 2^{31}. OSEK being a specification (with several implementations), requires that any implementation provide at least 8 priority levels (μITRON does not specify any such limit). Lastly, Windows CE as its major shortcoming provides only 8 priority levels.

Synchronization

Synchronization is necessary for real-time tasks to share mutually exclusive resources (devices, memory areas, buffers etc.), which is also needed for implementing task-dependence (execute statement x in task B after task statement y in task A). Traditional solutions using semaphores (and related constructs like monitors, critical regions) can result in unbounded priority inversion. Priority inversion is said to occur when a higher priority task is temporarily forced to wait for a lower priority task. Such inversion of priority can go unbounded when medium priority tasks preempt the lower priority task (due to lack of resource conflicts).

Classical solutions to the problem are the simple *priority inheritance protocol*(PIP) and the complex *priority ceiling protocol*(PCP) (popularly implemented as the highest locker protocol(HLP)). Both protocols prevent unbounded priority inversion where PCP provides a better (lower) bound at a higher cost (implementation-wise). It is desirable therefore that an RTOS provide at least PIP. Table 2 gives the synchronization mechanisms provided by the various RTOSs. While a majority of the RTOSs support

	OSE	OSEK-VDX	LynxOS	μITRON
Constructs	Signals(with buffers) 1 fast Sem Bin Sems	Semaphores Events Preemptive	Semaphores Cond. Vars Queued signals	Semaphores Event Flags Mailboxes
Protocol	None	HLP	PIP	None

QNX	RTAI	VRTX	VxWorks	Win-CE
Semaphores Message Passing Signals	Simulated with FIFOs	Semaphores	Interrupt Locks Preemptive Locks Semaphores	Critical Section Events Mutexes
PIP	None	PIP	HLP	PIP

Table 2. Synchronization

semaphores, some of them have other preferred methods of synchronization (see Table 2). For example, OSE has extensive support for signals which (unlike POSIX signals) carry a buffer with them. Mutexes (used in Windows CE) are similar to semaphores but less generic in that they are used specifically to guard critical sections. RTAI does not support semaphores, instead a programmer can use FIFOs (file in file out) to simulate semaphores because they provide a similar functionality.

Interrupt and Event Handling

For maximum productivity (and performance) it is important to allow application developers to, specify and write interrupt handlers (Interrupt Service Routines -ISRs) for Hardware Interrupts. A significant part of a embedded real-time system development is writing device drivers, therefore the RTOS should provide low level control of interrupts through interrupt handlers. Software interrupts like signals (POSIX) are also desirable. OSE is the exception to the rule that interrupt handlers be non-schedulable entities (see Table 3). In OSE, interrupts are associated with interrupt processes that are assigned high priority. OSEK has elaborate support for interrupts (its primary target being automotive environments with numerous remote sensors and actuators). LynxOS, QNX, VRTX and VxWorks provide preemptible ISRs. RTAI allows very primitive interrupt handling which involves programming the 8259 interrupt controller through the RT Hardware Abstraction Layer. Lastly, though Windows CE supports writing ISRs (non-schedulable) the technical documentation by Microsoft [13] describes the preferred method to be the use interrupt service threads (IST) that run at priority level 0 (highest). Nesting of interrupts is allowed in all but RTAI and Windows CE.

	OSE	OSEK-VDX	LynxOS	μITRON
ISR	Interrupt Processes	3 types of ISRs	Preemptible Int. Handlers	Implementation specific
Signals	Non-POSIX	Non-POSIX	POSIX	None
Nested Interrupts	Yes	Yes	Yes	Yes

QNX	RTAI	VRTX	VxWorks	Win-CE
Preemptible Int. Handlers	8259 RTHAL	Preemptible Int. Handlers	Special Context	Interrupt Service Threads
POSIX	None	Queued POSIX	Queued POSIX	-
Yes	No (queued)	Yes	Yes	No

Table 3. Interrupt and Event Handling

Communication

Inter process communication (IPC) in RTOSs is primarily to exchange data on the same processor, however with an increasing number of real-time systems taking a more distributed (networked) form of operation some RTOSs allow process communication between processes resident on different processors. Popular forms of IPC include, shared memory, message queues, pipes, FIFOs (file in file out) and sockets. Desirable properties of IPC mechanisms in the context of an RTOS include, provision for non-blocking communication, bounded operation (r/w) latency and asynchronous communication. All the RTOSs that provide an IPC mechanism provide the above properties. While

	OSE	OSEK COM	LynxOS	μITRON
Constructs	Signals with buffers, Shared Mem	Messages, Shared Mem	Msg. Queues, Pipes,Sockets, Shared Mem	Mailboxes, Data Queues

QNX	RTAI	VRTX	VxWorks	Win-CE
Messages, Pipes,Queues, Socklets	FIFOs Shared Mem	Msg. Queues, Pipes,Sockets, Shared Mem	Msg. Queues, Pipes,Sockets, Shared Mem,RPC	Shared Mem

Table 4. Inter-Process Communication

shared memory (physical addrs) is often an obvious mechanism for IPC it can be cumbersome and unsafe unless the RTOS provides an API for it, which is the case with all the studied RTOSs (see Table 4). Popular IPC mechanisms in traditional OSs like sockets while being supported, have been totally rewritten to provide real-time response. For example, QNX provides a mechanism called *Socklet* that is a less memory intensive version of a traditional socket. Though Linux itself has a rich IPC set, RTAI provides only FIFOs which are used to communicate between real-time tasks and also between real-time tasks and Linux tasks. VxWorks supports RPC (remote procedure calls) for distributed system implementation.

Timers and Clocks

All the RTOSs reviewed in this paper provide good support for timers, time-triggered tasks and clocks. All of them allow access of clocks at nanosecond resolutions when supported by the hardware.

Memory Management

Most older RTOSs did not see the need for supporting virtual memory, due to the lack of an MMU (memory management unit) on the processor and, due to the non-determinism introduced by it. However, most modern processors (with the exception of small embedded processors) come with a programmable MMU. Dynamic memory allocation allows programming flexibility but introduces the overhead of garbage collection. Therefore, calls to `malloc` can block due to unavailability of memory. Several of the RTOSs allow restricted use of dynamic memory allocation, for example (see Table 5)almost all of them disallow dynamic memory allocation calls in interrupt service routines.

	OSE	OSEK-VDX	LynxOS	μITRON
Virtual Memory(MMU)	Segmented	No	Demand Paged	No
Dynamic Allocation	Pools/Blocks	No	Yes	Fixed Pool

QNX	RTAI	VRTX	VxWorks	Win-CE
Yes	No	Yes	Optional	Segmented
Yes	No	Yes	Yes	Yes

Table 5. Memory Management

Providing support for virtual memory is often a very difficult choice to make if the processor has an MMU because, not supporting VM would amount to a waste of the MMU, while supporting it would have the downside of non-determinism. VxWorks has dealt with this issue by providing virtual memory as an optional add-on to the core RTOS.

4 Performance Comparison

Several researchers and practitioners have passionately argued [8, 1] that the oft-quoted performance metrics of *context switch latency* and *interrupt latency* are not the sole measures of merit of an RTOS. However, almost all RTOS publish these metrics as primary figures of merit. The POSIX 1003.1 (optional Annex G) standard specification goes to great lengths by elaborating a list of performance metrics of interest in an RTOS, however, being an optional component leaves no incentive for RTOS manufacturers to adhere to it.

In addition to the context switch time and the interrupt latency it is desirable to know the maximum time taken by *every* system call. Such measures should be predictable and independent of the current state of the operating system. Context switch time is the delay incurred in saving the context of the current running process and restoring the context of the next process chosen by the dispatcher to run. Interrupt latency is the time elapsed between the occurrence of an interrupt and the execution of the first instruction of the corresponding interrupt handler. The figures reported in Table 6 have been extracted from various sources [3, 13] of literature and therefore are incomparable (due to the different environments under which they were

	OSE	LynxOS	QNX
Environment Disabled	M 68360 25 MHz	M 68030 25 MHz MMU,Cache	Pentium 200 MHz
CS Lat.(μs)	136	180	2
Interrupt Latency	-	13	-

RTAI	VRTX	VxWorks	Win-CE
Pentium 100 MHz	M 68030 25 MHz MMU,Cache	M 68030 25 MHz MMU,Cache	Pentium 200 MHz
4	80	110	34.4
-	4	3	9.5

Table 6. Performance Comparison

observed). However, some inferences can be drawn from these numbers: VRTX is a clear choice (in terms of performance) in that it clearly outperforms OSE, LynxOS and VxWorks in its context switch time and comes close to the best in its interrupt latency. The comparison between QNX and Windows CE shows that Windows CE needs improvement which may be expected from the next version(3.0).

Note that several entries have "-"'s in them implying unavailability of information. Further, we skipped both OSEK-VDX and μITRON from the table because there are several implementations.

5 Conclusions and Future Work

In conclusion we have shown in this study that the world of RTOSs is less chaotic than it appears on the surface. Though only 9 RTOSs were covered in this study they are a good representative set giving sufficient breadth to the observations made. The performance comparison reported gives some insight into the relative quality of these RTOSs.

We are currently continuing to investigate a few more RTOSs of interest and have a detailed performance study underway. The eventual goal of this study is to report performance figures at the level of detail suggested in Annex G of the POSIX 1003.1 standard.

References

[1] Garcia-Martinez, A.; Conde, J.F.; Vina, A., "A Comprehensive Approach in Performance Evaluation for Modern Real-Time Operating Systems", *Proceedings of the 22nd EUROMICRO Conference*, 1995, pp. 61 -68.

[2] LynxOS, "The scalable, reliable and highly deterministic operating system for real-time and embedded applications", *http://www.lynx.com/pdfs/lynxos.pdf.*

[3] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, and D. Beal,, "RTAI: Real-Time Application Interface", *Linux Journal Magazine, Issue No. 72, April 2000.*

[4] B. Mukherjee, K. Schwan and K. Ghosh "A Survey of Real-Time Operating Systems – Preliminary Draft" *Technical Report Georgia Tech. College of Computing, Report No. GIT-CC-93-18.*

[5] OSE, "OSE Realtime Kernel", *http://www.ose.com/PDF/rtk.pdf.*

[6] OSEK-VDX, "Open Systems and the Corresponding Interfaces for Automotive Electronics", *http://www.osek-vdx.org/os20r1a.pdf.*

[7] POSIX 1003.1, "International Standard for Real-Time: POSIX 1003.1;ISO/IEC-9945-1" *IEEE Standards Publication.*

[8] Sacha, K.M., "Measuring the Real-Time Operating System Performance", *Proceedings of Seventh Euromicro Workshop on Real-Time Systems*, 1995, pp. 34 -40.

[9] M. Schoenbrun, "QNX Faq", *Micro Business Applications: http://www.schoenbrun.com/mba/faq.htm.*

[10] H. Takada, Y. Nakamoto, and K. Tamaru, "The ITRON Project: Overview and Recent Results", *5th Intl. Conference on Real-Time Computing Systems and Applications (RTCSA)*, pp.3-10, Oct. 1998.

[11] VRTX, "VRTX Real-Time Operating System", *http://www.mentorgraphics.com/embedded/vrtxos/.*

[12] VxWorks, "VxWorks Programmers Guide", *http://www.wrs.com/pdf/vxworks_guide.pdf.*

[13] Microsoft Technical Document, "Real-Time Systems with Microsoft Windows CE", *Available at, http://www.eu.microsoft.com/windows /embedded/ce/resources/howitworks/realtime.asp.*