A Comparative Study: RTOS and Its Application

Gaurav Rai[#1], Sachin Kumar [*2],

[1,2]Department of Electronics & Communication Engineering, Amity University
Amity University Uttar Pradesh, Lucknow India

***Abstract*— Over past few decades the idea and need of compatible Real time operating systems has emerged as one of the key factors in the development of Real time operating systems, because of 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. The primary role of an operating system 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, 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. This paper briefly discusses and describes various features of XENOMAI and RTAI in relation to compatibility, features, multitasking and resource management.**

Keywords: XENOMAI, RTAI, compatibility, features, multitasking, resource management

I. INTRODUCTION

All computer system is divided into four parts: Hardware, Operating system, Application software and User [1].All the above parts of the computer system depend on each other. User communicates with application software; application software depends on operating system for any support and resource. Operating system deals with hardware. Operating system works like resource manager. It used to do following operations for user and hardware: processes, threads, memory management, physical-memory management, dynamic memory management , scheduling, ipcs, interrupt management, I/O subsystems, device drivers, timer subsystems, file system, networking , system call API libraries APIs.

Real time operating system contains all core features available in GPOS. The difference mainly lies in the implementation and implementation is fine tuned to ensure low latency and time deterministic behavior. RTOS contains a combination of various modules, including the kernel, a file system, networking protocol stacks, and other components required for a particular application.

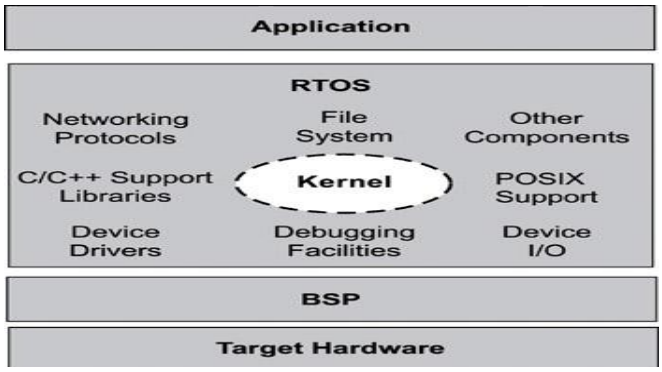

Fig1: High-level view of an RTOS, its kernel, and other components found in embedded systems. [2]

This paper briefly discusses and describes various features of XENOMAI and RTAI in relation to compatibility, features, multitasking and resource management.

A. RTAI (Real-Time Application Interface)

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 [15]. RTAI provides deterministic response to POSIX, interrupt, native RTAI real time task. Real time Application Interface consists mainly of two parts:

- The Linux kernel (patch with Adeos-based) which produce a hardware abstraction layer.
- A broad variety of services which make real-time programmer's lives easier [14].

RTAI versions over 3.0 use an Adeos kernel patch, slightly modified in the x86 architecture case, providing additional abstraction and much lessened dependencies on the 'patched' operating system. Adeos is a kernel patch comprising an Interrupt Pipeline where different Operating System Domains register interrupt handlers. This way, RTAI can transparently take over interrupts while leaving the processing of all others to Linux. Use of Adeos also frees RTAI from patent restrictions caused by RT Linux project.

Source model	Open Source
Latest stable release	RTAI3.9.2
Available programming language	C,C++
Kernel type	Linux Kernel

Supporting platform	ARM.MIPS (with FPU and TSC), X-86-64 PowerPc
Kernel type	Linux Kernel
Default user interface	CLI,GNOME shell, KDE Plasma Workspace Unity
License	--

Table1: Specifications RTAI 3.9.2

B. Xenomai:

There is a lack of common software framework for developing emulators, whereas the behavioral similarities between the traditional RTOS are obvious. The Xenomai technology aims at fulfilling this gap, by providing a consistent architecture neutral and generic emulation layer taking advantage of these similarities. It is also committed to provide an increasing set of traditional RTOS emulators built on top of this layer.

Xenomai relies on the common features and behaviors found between many embedded traditional RTOS, especially from the thread scheduling and synchronization standpoints. These similarities are exploited to implement a nucleus exporting a set of generic services. These services grouped in a high-level interface can be used in turn to implement emulation modules of real-time application programming interfaces, which mimic the corresponding real-time kernel APIs [17].

Source model	**Open Source**
Latest stable model	Xenomai 2.6.3
Available programming language	C,C++
Supporting platforms	X86,ARM,Power Pc IA-64,Blackfin,nois2
Kernel type	Linux kernel
Default user interface	CLI,GMOME shell, KDE, Plasma, Workspace, Unity
License	--

Table2: Specifications Xenomai 2.6.3

II. FEATURE COMPARISON BETWEEN RTAI AND XENOMAI

Out of various real time operating systems RTAI and XENOMAI are most recent and they have unique features which form the basis of comparison.

A. Multitasking and scheduling:

Scheduler is a component of the kernel - may be treated as a set of routines and a set of data-structures - managed using well defined rules and policies. Scheduler () routine/method may be invoked by the system, when appropriate one in the case of preemption point, second in the case of blocking a process inside a system call, one more such is when a process is terminated.

Scheduler may be divided into two components - one is the policy/ algorithm component and the other is the context switching component. Typically study of scheduler revolves around policy part. There are several policies proposed theoretically not all are used in practice. Even if used, there are modifications to theoretical policies and when they are implemented, these changes are for convenience. Several scheduling policies may be implemented in a given system. They may co-exist at the same time depending upon application's requirements; one or more policies may be used in a given system.

Scheduling policies can be broadly divided into two categories: Non real time scheduling policies and real time scheduling policies. In short, real time scheduling policies provide strict scheduling priorities. Non real time policies do not implement strict scheduling priorities. Their use is highly dependent on the applications and system's requirements. Typically non real time scheduling policies are more commonly used. Such policies are commonly used are time-slicing and time-sharing policies. A less common policy is first come first serve (FCFS).

In a multitasking system, there can be several processes existing in the system. Process control block or process descriptor is the name given to objects used to manage processes in the system. There is one process descriptor per process. Each process also has an assigned, unique process id maintained in the process descriptor (pd). Process manager subsystem is responsible for managing processes in the system. Process manager maintains process descriptors in a process descriptor table or process descriptor list.

A process may be understood from user-space/application perspective or system-space/kernel perspective. There is a process layout in user-space address-space of the process. User-space address space of the process is made up of logical addresses or virtual addresses.

Property	**RTAI**	**XENOMAI**
Priority scheduling	Fixed priority primitive	Configurable priority ,primitive but not block state
Same level scheduling	FCFS	Manual round-robin
Threaded	Single	Multi
Priority level	2^30	Time bound base priority level

Table 3: Multi-tasking and Scheduling

A newly created process is added to ready state meaning, state is changed to ready and pd is added to ready queue of the system. Ready queue is where are processes that are ready to execute are maintained. These processes have been allocated all other resources, but the processor scheduler scans ready queue, whenever it needs to select a new process to be allocated the processor.

Process scheduler is not a process. Scheduler may get invoked due to certain events in the system. One such event is a process termination, another such event is hardware interrupt and there are many other events due to which scheduler may

be invoked. It is a component of the operating system kernel and it is made up of several routines. One or more of which will be invoked during scheduling activities.

B. Synchronization

Synchronization typically involves co-ordination between processes and it may be achieved via operating system services explicitly or implicitly. In implicit case, co-ordination is a deliberate act of the developer. In explicit case, communication/data exchange is the real intention and system adds co-ordination to ensure certain natural rules are satisfied. This can be convenient to the developer or may interfere with the developer's work.

System also is involved in synchronizing the activities of the processes that are involved in exchanging the data meaning. There will be an implicit synchronization between the processes. If a process A is expecting data from another process B and initiating a receive call from a message queue, the corresponding process A will be blocked in the wait queue of the corresponding message queue instance. It is the responsibility of the system to wake-up the blocked process A, receiving process, when the sending process B has sent a message to the message queue instance.

Property	RTAI	XENOMAI
Constructs	Simulated with FIFO	semaphores, message queues, event flags or mailboxes and simulated with FIFO
Protocol	None	Priority inheritance protocol.

Table 3: Synchronization

C. INTERRUPT HANDLER AND SCHEDULING:

Deferred processing routines are pended/queued by I/O subsystems and device drivers of the system. Scheduler is invoked after interrupt handling. In reality, it is invoked after interrupt handling and processing of these deferred processing routines. There can be several of them depends on the load conditions. There can be a case, where a higher priority thread (that is currently interrupted or just woken up and ready for scheduling) may be delayed due to certain I/O deferred processing (that may benefit a lower priority thread). Latency involved in invoking the scheduler is not predictable and does not have an upper bound, deterministic behaviour.

Property	RTAI	XENOMAI
ISR	8259 RTHAL	
Signals	None	Signal used
Nested interrupt	No (queued)	Atomic

Table 4: Interrupt handler

D. Memory Management:

Virtual Memory (VM) of the system may lead to generation of page faults in process/threads of a process and this will lead to latencies and unpredictable performance. Due to VM, a kernel thread may be initiated any time in the system. Such unpredictable behaviour is also unacceptable. Subsystems point of view, driver's point of view and any activity in system space. Such memory requirements will be dynamic and require physical memory. Physical memory allocations are not predictable and latencies can also be higher.

This can be seen in 2 ways- one from the process perspective and another from kernel space perspective. Process perspective - mmap() and related facilities are vm based unpredictability with respect to VM applies here as well. mmap() may be used directly or indirectly via malloc(). IPCs , I/O subsystems and device drivers use physical memory manager for dynamic memory management and their inherent unpredictability is also passed on here.

PROPERTY	RTAI	XENOMAI
Virtual memory	No	For User-space application
Dynamic memory	No	Yes

Table 5: Memory Management

E. Inter Process communication:

Data exchange IPCs of GPOS use physical memory manager of the GPOS for dynamic memory requirements of buffers. The inherent unpredictability of PMM is passed on - there are no pre-allocations of buffers and mostly done as per run time requirements. These principles apply to pipes, message queues, sockets and any other such IPC mechanism supported by GPOS. Locks are also needed to manage processes and threads in many run time scenarios, priority inversion problems may arise meaning, a higher priority process/thread may be blocked by a lower priority process/thread and the worst part is that the lower priority process/thread may be pre-empted by other intermediate priority processes/threads. If locks do not support certain special features, problems such as priority inversions may be encountered in GPOS.

Property	RTAI	XENOMAI
Constructs	FIFO SHARED MEMORY	semaphores, message queues, event flags or Mailboxes.

Table 6: Inter Process Communication

III. Conclusion and Future Work

Operating Systems try to fit in different requirements of application. This can be achieved by using customization of the given operating system. This paper includes comparison between different features of Xenomai and RTAI. It also represent that using Xenomai with the developer point of view

is lot easier than any other RTOS. Any developer can change RTOS feature according to their application requirement.

IV. REFERENCES

[1] Operating system concept-Selberschatz, Galvin, Gagne 7th Edison

[2] Real time concept for Embedded system by Qing Li with Caroline Yao

[3] "Real Time Systems - by Jane W.S. Liu / yr of publ. 2000"

[4] "Programming for real world - by Gallmeister / yr of publ. 1993"

[5] http://en.wikipedia.org/wiki/LynxOS

[6] www.lynuxworks.com/rtos

[7] 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

[8] Real-Time Operating Systems: An Ongoing Review by Ramesh Yerraballi.

[9] http://www.t-engine.org/tron-project/itron

[10] http://en.wikipedia.org/wiki/ITRON_project

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

[12] http://www.enea.com/solutions/rtos/ose/

[13] http://en.wikipedia.org/wiki/Operating_System_Embedded

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

[15] 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*.

[16] http://en.wikipedia.org/wiki/Xenomai_vs._RTAI

[17] Xenomai – Implementing a RTOS emulation framework on GNU/Linux by Philippe Gerum1989.