Volume 3, No. 3, May-June 2012

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Comparison of Scheduling Mechanisms in commercial Real-Time Operating Systems

Mir Ashfaque Ali
Department of Information Technology
Govt. Polytechnic
Amravati, India.
realtime.ali@gmail.com

Dr. S. A. Ladhake
Principal
Sipna's College of Engg. & Technology
Amravati, India.
sladhake@yahoo.co.in

Abstract: **A** number of commercial real-time operating system are available. A real-time operating system (RTOS) supports real-time applications and embedded systems. Real-time applications have the requirement to meet task deadlines in addition to the logical correctness of the results. This paper focuses on pre-requisites for an RTOS to be POSIX compliant and discuss task scheduling in RTOSs. We survey the prominent commercial RTOSs and present a comparison of their scheduling properties. We conclude by discussing the results of the comparison and suggest future research directions in task scheduling in real-time operating systems.

Keywords: Task scheduling, real-time operating systems, commercial RTOS, POSIX.

I. INTRODUCTION

A real-time system is one whose correctness involves both the logical correctness of the outputs and their timeliness [11]. A real-time system must satisfy bounded response-time constraints; otherwise risk severe consequences, including failure. Real-time systems are classified as hard, firm or soft systems. In hard real-time systems, failure to meet response-time constraints leads to system failure. Firm real-time systems are those systems with hard deadlines, but where a certain low probability of missing a deadline can be tolerated. Systems in which performance is degraded but not destroyed by failure to meet response-time constraints are called soft real-time systems. A real-time system is called an embedded system when the software system is encapsulated by the hardware it controls. The microprocessor system used to control the various operations of many automobiles is an example of a real-time embedded system. An RTOS differs from common OS, in that the user when using the former has the ability to directly access the microprocessor and peripherals. Such an ability of the RTOS helps to meet deadlines.

II. RTOS FEATURES

The kernel is the core of the OS that provides task scheduling, task dispatching and inter-task communication. In embedded systems, the kernel can serve as an RTOS while commercial

RTOSs like those used for air-traffic control systems require all of the functionalities of a general purpose OS. The desirable features of an RTOS include the ability to schedule tasks and meet deadlines, ease of incorporating external hardware, recovery from errors, fast switching among tasks and small size and small overheads. In this section we discuss the basic requirements of an RTOS and the POSIX standards for an RTOS.

A. Basic Requirements of RTOS:

The following are the basic requirements of an RTOS.

a. Multi-threading and preemptibility:

To support multiple tasks in real-time applications, an RTOS must be multi-threaded and preemptable. The scheduler should be able to preempt any thread in the system and give the resource to the thread that needs it most. An RTOS should also handle multiple levels of interrupts i.e., the RTOS should not only be preemptible at thread level, but at the interrupt level as well.

b. Task priority:

In order to achieve preemption, an RTOS should be able to determine which task needs a resource the most, i.e., the task with the earliest deadline to meet. Ideally, this should be done at run-time. However, in reality, such a deadline-driven OS does not exist. To handle deadlines, each task is assigned a priority level. Deadline information is converted to priority levels and the OS allocates resources according to the priority levels of tasks. Although the approach of resource allocation among competing tasks is prone to error, in absence of another solution, the notion of priority levels is used in an RTOS.

c. Predictable task synchronization mechanisms:

For multiple tasks to communicate with each other, in a timely fashion, predictable inter- task communication, and synchronization mechanisms are required. The ability to lock/unlock resources to achieve data integrity should also be supported.

d. Priority inheritance:

When using priority scheduling, it is important that the RTOS has a sufficient number of priority levels, so that applications with stringent priority requirements can be implemented [13].

Unbounded priority inversion occurs when a higher priority task must wait on a low priority task to release a resource while the low priority task is waiting for a medium priority task. The RTOS can prevent priority inversion by giving the lower priority task the same priority as the higher priority task that is being blocked (called priority inheritance). In this case, the

CONFERENCE PAPER
National Conference on Information and Communication Technology for Development
Organized by PRMITR, Amravati (MS) India
http://mitra.ac.in/forthcoming.html

blocking task can finish execution without being preempted by a medium priority task. The designer must make sure that the RTOS being used prevents unbounded priority inversion [10].

e. Predefined latencies

An OS that supports a real-time application needs to have information about the timing of its system calls. The behavior metrics to be specified are:

- ***a) Task switching latency:*** Task or context-switching latency is the time to save the context of a currently executing task and switch to another task. It is important that this latency be short.
- ***b) Interrupt latency:*** This is the time elapsed between the execution of the last instruction of the interrupted task and the first instruction in the interrupt handler, or simply the time from interrupt to task run [6]. This is a metric of system response to an external event.
- ***c) Interrupt dispatch latency:*** This is the time to go from the last instruction in the interrupt handler to the next task scheduled to run. This indicates the time needed to go from interrupt level to task level.

B. POSIX compliance:

IEEE Portable Operating System Interface for Computer Environments, POSIX 1003.1b provides the standard compliance criteria for RTOS services and is designed to allow application programmers to write applications that can easily be ported across OSs. The basic RTOS services covered by POSIX 1003.1b include:

- ***a) Asynchronous I/O:*** The ability to overlap application processing and application initiated I/O operations [8].
- ***b) Synchronous I/O:*** The ability to assure return of the interface procedure when the I/O operation is completed [8].
- ***c) Memory locking:*** The ability to guarantee memory residence by storing sections of a process that were not recently referenced on secondary memory devices [24].
- ***d) Semaphores:*** The ability to synchronize resource access by multiple processes [19].
- ***e) Shared Memory:*** The ability to map common physical space into independent process specific virtual space [8].
- ***f) Execution Scheduling:*** Ability to schedule multiple tasks. Common scheduling methods include round robin and priority-based preemptive scheduling.
- ***g) Timers:*** Timers improve the functionality and determinism of the system. A system should have at least one clock device (system clock) to provide good real-time services.
  The system clock is called CLOCK_REALTIME when the system supports Real-time POSIX [13].
- ***h) Inter-process Communication (IPC):*** IPC is a mechanism by which tasks share information needed for a particular application. Common RTOS communication methods include mailboxes and queues.
- ***i) Real-time files:*** The ability to create and access files with deterministic performance.
- ***j) Real-time threads:*** Real-time threads are schedulable entities of a real-time application that have individual timeliness constraints and may have collective timeliness constraints when belonging to a runnable set of threads [13].

III. SCHEDULING ALGORITHMS FOR RTOS

In this section, the various scheduling schemes adopted in RTOSs is discussed. For small or static real-time systems, data and task dependencies are limited and therefore the task execution time can be estimated prior to execution and the resulting task schedules can be determined off-line. Periodic tasks typically arise from sensor data and control loops, however sporadic tasks can arise from unexpected events caused by the environment or by operator actions. A scheduling algorithm in RTOS must schedule all periodic and sporadic tasks such that their timing requirements are met.

The most commonly used static scheduling algorithm is the Rate Monotonic (RM) scheduling algorithm of Liu and Layland [12]. The RM algorithm assigns different priorities proportional to the frequency of tasks. RM can schedule a set of tasks to meet deadlines if total resource utilization is less than 69.3%. If a successful schedule cannot be found using RM, no other fixed priority scheduling system will avail. But the RM algorithm provides no support for dynamically changing task periods and/or priorities and tasks that may experience priority inversion. Priority inversion occurs in an RM system where in order to enforce rate-monotonicity, a non-critical task with a high frequency of execution is assigned a higher priority than a critical task with lower frequency of execution. A priority ceiling protocol (PCP) can be used to counter priority inversion, wherein a task blocking a higher priority task inherits the higher priority for the duration of the blocked task. Earliest deadline first (EDF) scheduling can be used for both static and dynamic real-time scheduling. Its complexity is O(n2), where n is the number of tasks, and the upper bound of process utilization is 100% [11]. An extension of EDF is the time-driven scheduler. This scheduler aborts new tasks if the system is already overloaded and removes low-priority tasks from the queue. A variant of EDF is Least Slack Time (LST) scheduling where a laxity is assigned to each task in the system and minimum laxity tasks are executed first. LST considers the execution time of a task, which EDF does not. Another variant of EDF is the Maximum

Urgency First (MUF) algorithm, where each task is given an explicit description of urgency. The cyclic executive is used in many large-scale dynamic real-time systems [3]. Here, tasks are assigned to a set of harmonic periods. Within each period, tasks are dispatched according to a table that lists the order to execute tasks. No start times need be specified, but a prior knowledge of the maximum requirements of tasks in each cycle must be known.

One disadvantage of dynamic real-time scheduling algorithms is that even though deadline failures can be easily detected, a critical task set cannot be specified and hence there is no way to specify tasks that are allowed to fail during a transient overload.

CONFERENCE PAPER
National Conference on Information and Communication Technology for Development
Organized by PRMITR, Amravati (MS) India
http://mitra.ac.in/forthcoming.html

IV. COMMON FEATURES OF COMMERCIAL RTOSS

a. Speed and efficiency: Most RTOSs are microkernels that have low overhead. In some, no context switch overhead is incurred in sending a message to the system service provider.
b. System calls: Certain portions of system calls are non-preemptable for mutual exclusion. These parts are highly optimized, made as short and deterministic as possible.
c. Scheduling: For POSIX compliance, all RTOSs offer at least 32 priority levels. Many offer 128 or 256 while others offer even 512 priority levels.
d. Priority inversion control: Many operating systems support resource access control schemes that do not need priority inheritance. This avoids the overhead of priority inheritance.
e. Memory management: Support for virtual memory management exists but not necessarily paging. The users are offered choices among multiple levels of memory protection.

V. GENERAL PURPOSE RTOS IN EMBEDDED INDUSTRY

This section focuses on VxWorks [25], the most widely adopted RTOS in the embedded industry. It is the fundamental run-time component of Tornado II, a visual, automated and integrated development environment for embedded systems. VxWorks is a flexible, scalable RTOS with over 1800 APIs and is available on all popular CPU platforms. It comprises the core capabilities of network support, file system, I/O management, and other standard run-time support. The micro kernel supports a full-range of real-time features including 256 priority levels, multitasking, deterministic context switching and preemptive and round robin scheduling. Binary and counting semaphores and mutual exclusion with inheritance are used for controlling critical system resources. VxWorks is designed for scalability, which enables developers to allocate scarce memory resources to their application rather than to the OS. Portability requires a distinct separation of low-level hardware dependent code from high-level application or OS code. A Board Support Package (BSP) represents the hardware-dependent layer. A BSP is required for any target board that executes VxWorks. TCP, UDP, sockets and standard Berkeley network services can all be scaled in or out of the networking stack as necessary. VxWorks supports ATM, SMDS, frame relay, ISDN, IPX/ SPX, AppleTalk, RMON, web-based solutions for distributed network management and CORBA for distributed computing environments. VxWorks [25] supports virtual memory configuration. It is possible to choose to have only virtual address mapping, to have text segments and exception vector tables write protected, and to give each task a private virtual memory upon request.

VI. COMPARISON OF COMMERCIAL RTOS

The following table lists the most widely used commercial RTOSs and their main features with respect to the five basic requirements of an RTOS as explained above:

Table: 1

RTOS, Vendor	Scheduling	Task priority levels	Task Synchronization mechanisms	Priority inversion Prevention provided
AMX KADAK Products Ltd.	Preemptive	N/A	Mailbox or Message exchange manager; wait-wake requests	Yes
C Executive JMI Software Systems	Prioritized FIFO, time slicing	32	64 system calls; Messages, dynamic data queues	Yes
CORTEX Australian Real-time Embedded Systems.	Prioritized FIFO, prioritized Round robin, Time slicing	62	Recursive Resource locks, mutexes, monitors and counting semaphores	Yes, uses Priority ceiling
Delta OS CoreTek Systems	Prioritized Round robin	256	Semaphores, timers and message queues	Yes
ECos RedHat, Inc.	Prioritized FIFO, Bitmap	1 to 32	Rich set of synchronous Primitives including timer and counters	Yes, uses Priority ceiling
embOS SEGGER MicrocontrollerSyst	Prioritized Round robin	255	Mailbox, binary and counting semaphore	No
eRTOS JK Microsystems,Inc.	Prioritized Round robin	256	Inter-thread Messaging (messages and queues), semaphores	No
INTEGRITY Green Hills Software, Inc.	Prioritized Round robin, ARINC 653	255, but configur able	Semaphores; break points can be placed any where in the system including ISRs.	Yes, using mutex, highest locker semaphore
IRIX SGI	Prioritized FIFO, Round robin	255	Message queues	Yes
Nuclear Plus Accelerated Technology, Inc.	Prioritized FIFO	N/A	Mailboxes, pipes and queues can be created dynamically as required	Yes

CONFERENCE PAPER
National Conference on Information and Communication Technology for Development
Organized by PRMITR, Amravati (MS) India
http://mitra.ac.in/forthcoming.html

OS-9 Microware Systems Corpn.	Prioritized	64K	Uses Active Queue, Event, Semaphore, Wait and Sleep Queue	Yes
OSE OSESystem	Prioritized FIFO	32	Message-based Architecture	Yes
RT-Linux Finite State Machine Labs	Prioritized FIFO, uses an Extensible scheduler	1K	Real-time tasks in RT Linux can Communicate with Linux processes via either shared memory or through a file like interface.	Yes, uses lock free data structures and priority ceiling
ThreadX Express Logic Inc.	Prioritized FIFO, Preemption -Threshold	32	Event flags, Mutexes, couting Semaphores and message services	Yes, using Preemption -threshold (Disables Preemption over ranges of priorities instead of disabling preemption of entire system) and priority inheritance
QNX Neutrino QNX Software Systems Ltd.	Prioritized FIFO,prioritized Round robin	64	Message-based Architecture	Yes, using Message based priority inheritance

VII. CONCLUSION

Although a variety of commercial real-time operating systems are available in the market, selection of a particular RTOS is crucial for a specific application. In this paper, a review of the basic requirements of an RTOS including the POSIX 1003.1b features is discussed. Use of RTOS is beneficial in most real-time embedded design projects as it provides a deterministic framework for code development and portability. To meet the needs of commercial multimedia applications, low code size and high peripheral integration is needed. RTOSs should be API compatible. Code reuse considerations are also important.

VIII. REFERENCES

[1]. S.R.Ball, Embedded Microprocessor Systems, Second edition, Butterworth-Heinemann, 2000.

[2]. M.Bunnell, "Galaxy White Paper," http://www.lynx.com/lynx_directory/galaxy/galwhite.html

[3]. G.D.Carlow, "Architecture of the Space Shuttle Primary Avionics Software System," CACM, v 27, n 9, 1984.

[4]. CompactNET, http://www.compactnet.com.

[5]. H.Gomaa, Software Design Methods for Concurrent and Real-time Systems, First edition, Addison-Wesley, 1993.

[6]. S.Heath, Embedded Systems Design, First edition, Butterworth-Heinemann, 1997.

[7]. http://www.rtos4voip.com/index.html

[8]. IEEE. Information technology--Portable Operating System Interface (POSIX)-Part1: System Application: Program Interface (API) C Language, ANSI/IEEE Std 1003.1, 1996 Edition.

[9]. Jbed RTOS, http://www.esmertec.com

[10]. E.Klein, "RTOS Design: How is Your Application Affected?," Embedded Systems Conference, 2001.

[11]. P.A.Laplante, Real-Time Systems Design and Analysis: An Engineer's Handbook, Second edition, IEEE Press, 1997.

[12]. C.L.Liu and J.W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," Journal of the ACM, v 20, n 1, pp. 46-61, 1973.

[13]. Jane.W.S.Liu, Real-time Systems, Prentice Hall, 2000.

[14]. LynxOS, http://www.lynuxworks.com

[15]. Microsoft Windows NT workstation resource kit.

[16]. C.Muench and R.Kath, The Windows CE Technology Tutorial: Windows Powered Solutions for the Developer, First edition, Addison-Wesley, 2000.

[17]. Rainfinity: http://www.rainfinity.com/index.html

[18]. H.Y.Seo, and J.Park. "ARX/ULTRA: A new real-time kernel architecture for supporting user-level threads," Technical Report SNU-EE-TR1997-3, School of Electrical Engineering, Seoul National University, 1997.

[19]. A.Silberschatz, P.B.Galvin and G.Gagne, Operating Systems Concepts, Sixth edition, John Wiley, 2001.W.Stallings, Operating Systems: Internals and Design Principles, Third edition, Prentice-Hall, 1997.

[20]. J.A.Stankovic and K.Ramamritham, "The Spring Kernel: A New Paradigm for Hard Realtime Operating Systems," ACM Operating Systems Review, v 23, n 3, pp. 54-71, 1989.

[21]. M.Stokely and N.Clayton, FreeBSD Handbook, Second edition, Wind River Systems, 2001.

[22]. M.Teo, "A Preliminary Look at Spring and POSIX 4," Spring Internal Document, 1995.

[23]. The Open Group, http://www.opengroup.org/

[24]. VxWorks, http://www.windriver.com

[25]. C.Walls, "RTOS for Microcontroller Applications," Electronic Engineering, v 68, n 831, pp. 57-61, 1996.

CONFERENCE PAPER
National Conference on Information and Communication Technology for Development
Organized by PRMITR, Amravati (MS) India
http://mitra.ac.in/forthcoming.html

[26]. K.M.Zuberi and K.G.Shin, "EMERALDS:A Microkernel for Embedded Real-Time Systems," Proceedings of RTAS, pp.241-249, 1996.

[27]. Baskiyar and N. Meghanathan, "A Survey of Contemporary Real-Time Operating Systems," Informatica pp. 233–240, 2005.

CONFERENCE PAPER
National Conference on Information and Communication Technology for Development
Organized by PRMITR, Amravati (MS) India
http://mitra.ac.in/forthcoming.html