Real Time Operating System used in Embedded System for Complexity Reduction

Arti Bajaj
MCA, B.Sc.(CA)
Tecnia Institute of Advanced Studies, New Delhi, India

Ashima Bhatnagar Bhatia
M Phil.(CS), MCA, B.Sc.(IC)
Tecnia Institute of Advanced Studies, New Delhi, India

Alka Batra
MCA, Tecnia Institute of Advanced Studies, New Delhi, India

Abstract - Embedded systems are the computing devices hidden inside a vast array of everyday products and appliances such as cell phones, toys, handheld PDAs, cameras. These intelligent processors have embedded themselves into all fields of our lives be it the kitchen (food processors, microwave ovens), the living rooms (televisions, air conditioners) or the work places (fax machines, pagers, laser printer, credit card readers). Embedded systems employing multiple tasks can have non-deterministic output, which complicates the determination of expected outputs for given inputs, and thus, complicates contrast, occur in a wide range of popular but less safety-critical systems such as consumer electronic devices, and tend not to have such rigorous temporal constraints As the complexities in the embedded applications increase, use of an operating system brings in lot of advantages, requirements and include safety critical systems such as those found in avionics and automotive systems. Soft real-time embedded systems, in The Operating System (RTOS) have strict constraints placed on it, and have to interface well with the hardware below For example, hard real-time embedded systems have strict temporal. Hence we move to another face of this dynamically changing environment, which is the capability of the Operating System to handle Real Time Processes. Most embedded systems also have real-time requirements demanding the use of Real time Operating Systems (RTOS) capable of meeting the embedded system requirements. Real-time Operating System allows real-time applications to be designed and expanded easily.

I. INTRODUCTION

1.1. Embedded System

An embedded system is a combination of hardware and software and perhaps other mechanical parts designed to perform a specific function. Microwave oven is a good example of one such system. This is in direct contrast to a personal computer. Though it is also comprised of hardware and software and mechanical components it is not designed for a specific purpose. Personal computer is general purpose and is able to do many different things.

An embedded system is generally a system within a larger system. Modern cars and trucks contain many embedded systems. One embedded system controls anti-lock brakes, another monitors and controls vehicle's emission and a third displays information on the dashboard. Even the general-purpose personal computer itself is made up of numerous embedded systems. Keyboard, mouse, video card, modem, hard drive, floppy drive and sound card are each an embedded system. Tracing back the history, the birth of microprocessor in 1971 marked the booming of digital era. Early embedded applications included unmanned space probes, computerized traffic lights and aircraft flight control systems. In the 1980s, embedded systems brought microprocessors into every part of our personal and professional lives. Presently there are numerous gadgets coming out to make our life easier and comfortable because of advances in embedded systems. Mobile phones, personal digital assistants and digital cameras are only a small segment of this emerging field.

One major subclass of embedded systems is real-time embedded systems. A real time system is one that has timing constraints. Real-time system's performance is specified in terms of ability to make calculations or decisions in a timely manner. These important calculations have deadlines for completion. A missed deadline is just as bad as a wrong answer. The damage caused by this miss will depend on the application. For example if the real-time system is a part of an airplane's flight control system, single missed deadline is sufficient to endanger the lives of the passengers and crew.

II. PROBLEMS WITH EMBEDDED SYSTEM

Although embedded systems platforms and software are tailored for specific application domains such as consumer electronics, automotive and avionics they all share common problems. Problem examples include timing overruns due to effects such as blocking, unexpected time dependent calculations, and difficulties in understanding the implications of changes. These and other issues can be traced to conflicts in functional decomposition of high level requirements into the existing capabilities of desktop operating system semantics adopted for the embedded systems domain.

The existing open problems are a concern as greater and greater demands are being placed on precision and reliability in the growing breadth of application domains within systems that are becoming larger and more complex. Some of the challenging new trends in designing embedded systems are: The existing open problems are a concern as greater and greater demands are being placed on precision and reliability in the growing breadth of application domains within systems that are becoming larger and more complex. Some of the challenging new trends in designing embedded systems are:

- **Complexity** - Greater levels of functionality together with legacy code and lack of abstractions. The complexity of these issues is derived from:
  – Consumer electronics systems with the convergence of storage requirements, connectivity, and increased integration of functionality (camera, mp3, connectivity for consumer electronics).
  – Automotive systems that are integrating more functionality to decrease cabling and numbers of processors.
  – Avionics sector weight is a major issue. Size and weight issues are driving the movement away from federated systems to integrating functionality on fewer units.
- **Flexibility** - late changes, software download, reuse.
- **Dependability** - the level of integrity required in both failure and non-failure cases have increased. This has been brought about not just due to the fear of losing valuable sales (e.g., Intel adopted more formal approaches after their floating point unit problems on the early version of the Pentium processor) but also because of legislative pressure.
- **Connectivity** - on the systems level we have system integration where there is greater pressure on systems to work together, e.g., mobile phones to communicate with laptop computers etc.
- **Modularity** - needed to help provide maintainability (see below) but also to support concurrent and multisite development of systems and subsystems. Concurrent and multi-site development is exacerbated as more projects are managed as partnerships and/or using global software development teams.
- **Maintainability** - there is a move away from monolithic development as it makes change difficult and does not support reuse strategies such as Product Line Architectures.
- **Upgradeability** - there is a need to be able to upgrade systems in the field. The upgrades need to be performed by both experts and naive users.
- **Size and power** - there is pressure towards smaller devices that can run over batteries for longer periods of times.

III. REAL TIME OPERATING SYSTEM

A real time system is one whose correctness involves both the logical correctness of outputs and their timeliness. It must satisfy response- time constraints or risk severe consequences including failure. As defined by Donald Gillies "A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time in which the result is produced. If the timing constraints are not met, system failure is said to have occurred."

These systems respond to a series of external inputs, which arrive in an unpredictable fashion. The real-time systems process these inputs, take appropriate decisions and also generate output necessary to control the peripherals connected to them. The design of a real-time system must specify the timing requirements of the system and ensure that the system performance is both correct and timely. There are three types of time constraints:

- **Hard**: A late response is incorrect and implies a system failure.
- **Soft**: Timeliness requirements are defined by using an average response time. If a single computation is late, it is not usually significant, although repeated late computation can result in system failures.
- **Firm**: Firm real-time systems have hard deadlines, but where a certain low probability of missing a deadline can be tolerated. Most real-time systems interface with and control hardware directly. The software for such systems is mostly custom-developed. Real-time Applications can be either embedded applications or non-embedded (desktop) applications. Real-time systems often do not have standard peripherals associated with a desktop computer, namely the keyboard, mouse or conventional display monitors. In most instances, real-time systems have a customized version of these devices.

3.1 *Characteristics of Real Time Systems*

- Multitasking: Provided through system calls. Priority based Scheduling: In principle of flexible concepts, but limited to number of priority levels.
- Ability to quickly respond to external interrupts.
- Basic mechanisms for process communication and synchronization.
- Small kernel and fast context switching.
- Support real time clock as internal time interface.

IV. FEATURES OF RTOS'S UPON EMBEDDED SYSTEM

Most of the times, a real time system will be an embedded system. The processor and the software are embedded within the equipment, which they are controlling. Typical embedded applications are Cellular phone, Washing Machine, Microwave Oven, Laser Printer, Electronic Toys, Video Games, Avionic controls etc. RTOS will be embedded along with the application code in the system. There is no Hard Disk or Floppy Drive from which the OS will be loaded the entire code remains in the Read Only Memory of the system. So it must be small in size. Various features are:

4.1 Task Synchronization & Intertask Communication:

There are several tools available in RTOS to enable inter task communication and task synchronization

Semaphores: Semaphores are intertask communication tools used to protect shared data resources. Tasks can call Take-Semaphore and Release-Semaphore functions. If one task has called Take-Semaphore and has not called the Release-Semaphore to release it, then any other task that calls Take-Semaphore will block until first task calls Release-Semaphore. A counting semaphore is used when more than one task uses the same resource like in the case of a buffer pool management. Using a different semaphore for highest priority tasks ensures better response. Multiple semaphores can be used to protect different shared resources.

4.2 Message Mailboxes

Messages are sent to a task using kernel services called message mailbox. Mailbox is basically a pointer size variable. Tasks or ISRs can deposit and receive messages through the mailbox. A task looking for a message from an empty mailbox is blocked and placed on waiting list for a time(time out specified by the task) or until a message is received. When a message is sent to the mail box, the highest priority task waiting for the message is given the message in priority-based mailbox or the first task to request the message is given the message in FIFO based mailbox.

4.3 Interrupt Latency

Probably the most important feature for evaluating the performance of a RTOS is its ability to respond to interrupts. The time taken by the interrupt handler to deal with an interrupt and get back to regular program execution is extremely important in systems governed by hard real time constraints.

4.4 Memory management

Each programs need to held in a memory generally in a ROM to be executed. The task data (stack and registers) and all variables must be stored in RAM. In a real-time system the main requirement is that the access time should be bound or predictable. The use of demand paging is not allowed since the systems providing virtual memory mechanisms use memory swapping which is not predictable. RTOS have fast and predictable functions to allocate and free fixed size buffers. RTOS allows to setup pools each of which consist of same number of memory buffers.

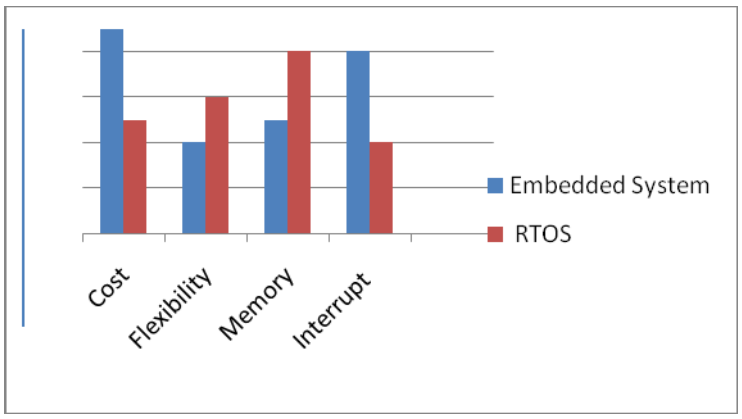

Figure 1: Comparison b/w Embedded System & RTOS

V. RTOS USED FOR EMBEDDED REAL TIME APPLICATIONS

5.1 QNX RTOS v6.1

The QNX RTOS v6.1 has a client-server based architecture. QNX adopts the approach of implementing an OS with a 10 Kbytes micro-kernel surrounded by a team of optional processes that provide higher-level OS services .Every process including the device driver has its own virtual memory space. The system can be distributed over several nodes, and is network transparent. The system performance is fast and predictable and is robust.

5.2 VRTX

VRTX has multitasking facility to solve the real-time performance requirements found in embedded systems. Pre-emptive scheduling is followed ensuring the best response for critical applications. Inter-task communication is by use of mailboxes and queues. Mailbox is equivalent to an event signal and events can pass data along with the event. Queues can hold multiple messages and this buffering facility is useful when sending task produces messages faster than the receiving task can handle them. Dynamic memory allocation is supported for allocation and release is in fixed size blocs to ensure predictable response times. VRTX has been designed for development and target system independence as well as real-time clock independence.

5.3 Windows CE 3.0

Windows CE 3.0 is an Operating system rich in features and is available for a variety of hardware platforms. It exhibits true real-time behavior most of the times. But the thread creation and deletion has periodic delays of more than 1 millisecond occurring every second. The system is robust and no memory leak occurs even under stressed conditions. CE 3.0 uses virtual memory protection to protect itself against faulty applications.

5.4 pSOSystem/x86 2.2.6

pSOS+ is a small kernel suitable for embedded applications. This uses the software bus to communicate between different modules. The choice of module to be used can be done at compile time making it suitable for embedded applications. System has a flat memory space. All threads share the same memory space and also share all objects such as semaphores. So it has more chances of crashing. Around 239 usable thread priority levels available making it suitable for Rate monotonic scheduling. pSOS has a multiprocessor version pSOS+m which can have one node as master and a number of nodes as slaves.

5.5 Windows NT

The overall architecture is good and may be a suitable RTOS for control systems that need a good user interface and can tolerate the heavy recourse requirements demanded for installation. It needs hard disk and a
powerful processor. Configuration and user interaction requires a dedicated screen and keyboard. The choice of selecting components for installation is limited and it is not possible to load and unload major components dynamically. Because of all these limitations Windows NT not suitable for embedded applications. It is neither suitable for other real time applications because of the following factors:

a) There are only 7 priority levels & there is no mechanism to avoid priority inversion.

b) The Queue of threads waiting on a semaphore is held in a FIFO order. Here there is no regard for priority, hampering the response times of highest priority tasks.

c) Though ISR responses are fast, the Deferred Procedure Calls (DPC) handling is a problem since they are managed in a FIFO order.

d) The thread switch latency is high (~ 1.2 ms), which is not acceptable in many real-time applications mission critical applications are involved. Selection of a particular RTOS for an application can be made only after a thorough study of the features provided by the RTOS. The choice of Operating System generally comes after the selection of the processor and development tools

VI. Conclusion

Real time Operating systems play a major role in the field of embedded systems especially for development tools. Hence the first step in choosing an RTOS must be to make the processor, real-time performance and the budget requirements clear. Then look at the available RTOS to identify the one which suits our application. Even though most of the current kernels (RTOS) are successfully used in today's real-time embedded systems, but they increase the cost and reduce flexibility. Next generation real-time operating systems would demand new operating systems and task designs to support predictability, and high degree of adaptability.

References

[1] David E Simon, An Embedded Software Primer. Reading, MA: Addison-Wesley, 1999.
[2] Gauthier L, Yoo S and Jerraya A, "Automatic generation and targeting of application-specific operating systems and embedded systems software," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20(11), pp.1293-1301, November 2005.
[3] Ghosh S., Mosse D. and Melhem R., "Fault-Tolerant Rate Monotonic Scheduling", Journal of Real-Time Systems, pp. 149-181, 1998.
[4] Michael Barr, Programming Embedded systems in C and C++. CA : O'Reilly & Associates, 1999.
[5] Brian Santo, "Embedded Battle Royale," *IEEE Spectrum*, Dec. 2001, pp.36-41.
[6] Dedicated Systems Experts, *RTOS Evaluation Project.* Brussels, Belgium: Dedicated Systems Experts, 2001.
[7] John A. Stankovic, Krithi Ramamritham, "The Design of the Spring Kernel," *Proc. IEEE- Real-Time Systems Symposium*, Dec.1987, pp.146-57