Performance Comparison of RTOS

Shahmil Merchant, Kalpen Dedhia
Dept Of Computer Science.
Columbia University

***Abstract**: Embedded systems are becoming an integral part of commercial products today. Mobile phones, watches, flight controllers etc are just a few of the products that one sees at a regular basis. The Philosophy of such Technologies is simple :One needs to have strict Real Time constraints such that one doesn't miss on crucial deadlines, which could prove to be very detrimental to a system. In order to ensure that, we need a very strong and compatible relationship between the hardware that is being used for these systems and the software, which is running on top of it, primarily the Operating System. The Operating System has strict constraints placed on it, and has to interface/communicate well with the hardware below it to prevent casualty. Hence we move to another facet of this dynamically changing environment, which is the capability of the Operating System to handle Real Time Processes. Due to the strict dependence of an Operating System on hardware one must have a Real Time Operating System, which would ensure fast interrupt response and real time scheduling.*
Our Projects looks into 2 such Operating Systems Ecos a Free Product released by Red Hat and RtLinux released by Finite State Machine Labs. The core aspect of our Analysis would be simple to check the real time attributes of these Operating Systems by means of simple applications. Due to these RTOS being POSIX 1.0 compliant and having a generic gcc compiler, one could check various latency issues involved in threading, management of File System and Interrupts. Through this Paper we do hope to put forth a better understanding of the finer intricacies of an RTOS and its subtle differences with a regular OS, and also provide a thorough study of the pros and cons of RtLinux and Ecos.

1. Introduction

Real time applications have become a very common phenomenon these days. Developers are given the enviable task of making software with real time constraints. Many a times they are made to take decisions on whether they would prefer to work on a RTOS or a regular OS.Due to the presence of a large number of RTOS available in the market one does get confused as to which one to use such that it provides the best over all benefits in terms of cost and operability. There could be a set of certain benchmarks, which one could examine the RTOS in terms of latency, susceptibility to different loads.

We do realize that we are faced with strict time constraints and lack of sufficient free ware tools and hardware, and hence the main goal of out project would be one of providing a larger picture of the job at hand in the form of an evaluation rather than providing a detailed study of measurement. We have restricted our selves to the above 2 OS due to the fact that they are free ware and are stable Linux ports with POSIX and gcc compliance.

After a thorough study of the above we have decided to approach our project in the following way:

- Study the typical Characteristics of an RTOS.
- Study the details of the 2 RTOS at hand in terms of the scheduling policies etc.
- See what aspects each RTOS satisfy and not satisfy.
- Do a comparison between the 2 based on the above results with carefully formulated applications.

2. Background

2.1 What is a RTOS?

A RTOS is a system, which satisfies the following conditions:

- Responds in a predictable way to unpredictable external events
- Strict timing constraints

2.2 Minimum Features of an RTOS

- Multi-tasked (threads) and pre-emptive priority driven
- Mechanism for thread synchronization (semaphores, mutexes, etc)
- Mechanism to avoid Priority Inversion
- Sufficient priority levels

2.3 Characteristics of an RTOS

- Model- Models for Multi Tasking based:
    - Non Defined process which is subdivide into tasks.
    - POSIX model with processes subdivided into threads
- Priority Levels- A way for scheduling processes based on their priority which could be defined either by there execution time or weight etc
- Dispatch time: Should be independent of number of threads on list.
- Number of Tasks: Defined by way of number of processes and threads and their corresponding memory utilization.
- Scheduling Policies: The sequence by which a process is run by a scheduler depends on the scheduling policy. Different scheduling policies exists such as
    - FIFO: First process that comes is run first

- STF: Shortest time first such that the process with smallest time required running is run first.
- EDF: Earliest deadline first, where the process with the earliest deadline is run so that it meets the deadline.
- Round Robin: Processes are run in a round Robin fashion
- Weighted Scheduling: Process is run in a round robin fashion based on their weight
- Priority scheduling: Is of 2 types
    - Preemptive Priority: Process with lower priority can be preempted by process with higher priority.
    - Non-Preemptive Priority: Process run according to Priority.

- Number of Documented states: Like running, runnable, waiting, etc
- Minimum RAM per Task
- Maximum Addressable memory space.
- Memory is typically defined by 2 terms
    - Logical Address: Which is defined by pages and provide an abstraction to the user of an entity for storage of data.
    - Physical Memory: The actual mapping of the Logical Address onto the physical Hard Disk managed by the Memory Management Unit.

The above ensure the facility of Virtual Memory in Operating Systems where process is not confined to issues related to contiguous regions in memory and running out of main memory (RAM) space. Virtual memory is supported by dynamic paging wherein pages that are required or would be required are the only pages that are brought into memory. The numbers of pages brought into memory depend on the number of frames and new pages would replace old pages by one of the following schemes:

- LRU: Least recently used pages are replaced
- Second Chance: Based on Flag setting.
- MFU: Most Frequently used pages are removed.
- LFU: Least Frequently pages are removed.

- FCFS: First come First serve wherein pages that were brought in first into memory are removed first.

Great care should be taken to avoid Belady's Anomaly where in the increase in the number of frames causes an increase in the number of page faults.

- Interrupt Handling: This is done by the following

  - An interrupt causes program to transfer to a certain region in memory which contains the address of the interrupt handler so as to deal with an interrupt
  - Interrupts are in the form of hardware interrupts or software interrupts in the form of system calls.

- A process and a thread can be distinguished by the fact that a process has a program counter whereas as thread does not. A process does not share address space like a thread.

- A process and a program can be distinguished by the fact that the program is a passive entity and a process is an active entity such that a Process is a program in execution.

3 Test Metrics

- Thread Switch Latency

  A thread is defined by different states such as waiting, running, runnable etc. The time taken for the thread to move from different states is a parameter for testing RTOS performance. The context switch overhead in switching from one thread to another is lesser than a process as a thread is a lightweight process.

- 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.

- Thread creation and destruction

The RTOS under study have POSIX 1.0 compliabilty. One could use the time to create the thread and destroy the thread as a good metric due to the simple reason it would show how well memory management would work in the system under consideration.

- File System management

File systems provide an abstraction to the higher levels of software code the way as to which programs are stored in disk. Files on disk could be stored in the following manner

  - Contiguous Allocation: Where processes are stored one after the other in the form of a heap and termination of a process could result in holes being created on the File system. Defragmentation is a solution.

  - Allocation table: A directory like structure wherein the directory has a pointer to each process on disk. FAT used in Window is an example.

  - Indexed Allocation: Where indexes to a linked list of processes exist. Best method allocation.

Hence by testing the time needed to create and open close a file could test the feasibility of different allocation schemes.

- Synchronization

  Shared resources form an integral part of an Operating system. Great care has to be taken when dealing with resources that can be used by different objects. Semaphore and Monitor implementation takes care of the Critical Section Problem.

- Load
  The way an RTOS behaves to different system loads is an integral part of testing of an operating system.

4 RTOS under consideration

4.1 eCOS

eCos, developed by Redhat, is an embedded configurable operating systems. eCos is an open source real-time operating system which is useful for deeply embedded applications. The core eCos system consists of a number of different components such as the kernel, the C library, an infrastructure package. Each of these provides a large number of configuration options, allowing application developers to build a system that matches the requirements of their particular application. To manage the potential complexity of multiple components and lots of configuration options, eCos comes with a component framework: a collection of tools specifically designed to support configuring multiple components. Furthermore this component framework is extensible, allowing additional components to be added to the system at any time.

- Environment
  - Processor: x86, ARM, MIPS, SPARC
  - Hosts: Windows, Redhat Linux
  - Compiler: GCC
- RTOS supplied as open source

4.2 RTLinux

RTLinux (RealTime Linux) is a small POSIX 1003.13/PSE51-compatible hard realtime operating system that runs Linux as its lowest priority thread. Linux runs as the lowest priority thread of the RTLinux kernel, and it is always pre-emptible. RTLinux realtime applications consist of realtime threads and signal handlers that run in the RT environment, and processes that run in Linux user space.

The RTLinux programming model is that anything that has strict timing requirements should be written as threads or signal handlers (interrupt handlers), and whatever does not need hard realtime should go into Linux. This allows us to keep the RT side small, deterministic and as fast as the hardware will permit, while still drawing on Linux for sophisticated services and applications.

- Propriety of Finite State Machine Labs
- Environment
  - Processor: x86, Power PC, MIPS, Alpha
  - Hosts: Linux
  - Compiler: GCC
- RTOS supplied as open source

5 Mechanism

5.1 Tools for Evaluation

5.1.1. Rdtsc instruction available on Intel x86 processors for strict timing measurements

- Metrics
  - Threads
  - Interrupts
  - File Systems
  - Synchronization

5.1.2 EL/IX

- EL/IX is an application programming interface that allows us to write applications on embedded systems
- Using EL/IX, we can write, test, analyze, and even simulate applications

5.1.3 PERTS (A prototyping environment for real-time systems)

- Build by UIUC
- Tool for testing and modeling Systems

Conclusion and Future Work

Studying the differences between the RTOS and OS we have come to the following conclusions

- Strict methods of handling interrupt in an RTOS as compared to a regular OS.

- Scheduling policies strictly dependent on priority and with built in priority inversion avoidance.
- Smaller size of kernel in RTOS enables faster loading onto memory.
- Strict and strong dependence of the RTOS on the underlying hardware.
- Synchronization and Communication of threads is handled at the Application level and Interrupt handling, Thread management and API's handles memory management, I/O.
- Lesser features (with respect to the above 2 RTOS) as compared to regular OS.

In the near future we intend to install both the operating systems and based on portability issues run corresponding applications.

Related Work

A large number of websites available on the internet deal with real time operating systems. Here are some of them that we visited:
http://www.redhat.com/embedded
http://www.rtlinux.com
http://cs-www.bu.edu/pub/ieee-rts/Home.html
http://pertsserver.cs.uiuc.edu/software/
http://www.tripac.com/html/prod-fact-rrm.html

Acknowledgements

We would like thanking Prof Edwards for giving us the opportunity to implement such an exciting project and giving us valuable guidance as and when required.