Design of Hybrid Kernel and the Performance Improvement of the Operating System

Prakash S. Prasad and Akhilesh R. Upadhyay

***Abstract*—Embedded system application is a hot topic in today's date & Linux gradually becomes the most important operating system for embedded applications. Embedded real-time system must be able to response and deal with system events within the pre-defined time limitation. In real-time multi-tasking system, a lot of events and multiple concurrent tasks are running at the same time. Therefore, to meet the system response time requirement, we must ensure that each mission can be achieved within the required time frame.**

Current Operating Systems includes a graphical user interface that is widely used. Due to the absence of Real-Time ability, current Operating Systems has not been suitable for all industrial applications. On the other hand normal operating system has the advantage of having both widespread applications and broad user acceptance. Moreover lot many low priced user programs are available. This is an attempt to create a way to make operating system useful for industrial real-time applications eliminating its disadvantages without giving up its advantages of popular user applications.

***Index Terms*— Real time operating system, hybrid kernel, performance parameters.**

I. Introduction

The Hybrid Kernel combines the Desktop OS and RTOS so that they can run concurrently on the same PC and the user can get best of both worlds. To make this possible we have developed a software only, real-time extension technology for desktop OS. The new technology guarantees deterministic response on interrupts that are targeted at Desktop OS. Any PCI or ISA PC plug-in board controlled by RTOS can generate these interrupts and interrupts aimed at RTOS always receive a higher priority than those aimed at Desktop OS [1].

As long as at least one RTOS task is active, the processors execution time is available exclusively for RTOS. Desktop OS will be reactivated only if all the RTOS tasks have given up their execution time and RTOS has entered into the idle mode. The RTOS idle mode controls the reactivation of General purpose OS. This makes it possible for the programmer to control processor sharing between two operating systems according to the application requirements.

Contra posing the basic principles and mechanism of the real-time operating system, the paper has compared general operating systems with real-time operating system, and made a good effort to analyze the key factors which may affect the real-time characteristics of operating system, and then given an assessment method to evaluate the real-time character of operating system.

Real-time systems are specific application systems in general, because specific characteristics could ensure their real-time characters on a certain extent. Early real-time systems have no operating system supported. To implement multi-task management, engineers must program code for specific practical application. Therefore, these particular software developments are less inheritance for code reuse, maintenance and upgrades which brought a lot of trouble. The emergence of real-time embedded operating system provides a powerful tool for real-time systems design and development because of its real-time kernel, multi-task, scheduling and fast interrupt response mechanism and so on. Such real-time characteristics can significantly reduce the workload of developers, improve development efficiency, and bring a lot of convenience for the maintenance and upgrading systems.

However, a system that uses real-time operating is not necessarily a real-time system. Real-time operating system is just only provide a basis for the real-time system, and the most essential elements for a real-time system are to meet the system requirements of task-critical time, which means the system must response to events in time and complete tasks within the limited time[2].

II. Real Time Operating System: its Components and Characteristics

Real-time operating system is a subtype of operating system. It has a lot of characteristics which are similar to common operating system in many respects. It is mainly responsible for the control and management of variety of hardware resources to enable the hardware system to become available, and provides upper level applications with rich system calls. It schedules execution in a timely manner, manages system resources and provides a consistent foundation for developing application code [3].

A. Components of RTOS

Most of the RTOS kernels consist of following components:

- Scheduler - The scheduler is at the heart of every kernel. A scheduler provides the algorithms needed to determine which task executes when.
- Objects- The most common RTOS kernel objects are tasks, semaphores and message queues.
- Services- Most kernels provide services that help developers create applications for real time embedded systems. These services

Manuscript received March 5, 2011, revised March 30, 2012.

Prakash S Prasad is with Information Technology,Priyadarshini College of Engineering, Nagpur, India (e-mail: prakashsprasad@gmail.com).

Akhilesh R. Upadhyay is with Dept. E.C.,Sagar Institute of Research and Technology, , Bhopal, India.

comprise sets of API calls that can be used to perform operations on kernel objects or can be used in general to facilitate following services:

- Timer Management
- Interrupt Handling
- Device I/O
- Memory Management

Embedded systems are used for various applications. These applications can be proactive or reactive dependent on the requirements like interface, scalability, connectivity etc. Choosing the OS for an embedded system is based on the analysis of OS itself and the requirements of application.

B. Characteristics

1. Its real time characteristic-Response to events in time and complete tasks within the limited time
2. The scheduling objective is letting high priority task go first
3. The tasks running on real-time operating system should be certain
4. Some data are highly sharing in real-time operating system

III. Factors affecting Real-Time Characteristics of Operating System

There are varieties of factors impacting a system's real-time. Among these factors, operating system and its own factors play crucial roles, including process management, task scheduling, context switching time, memory management mechanism, the time of interrupt handle, and so on.

A. Scheduling of Tasks

It is crucial for the real-time operating system to adopt preemptive scheduling kernel, which is based on task priority. The uC/OS-II operating system uses this method to implement its scheduling. In an operating system with nonpreemptive scheduling mechanism, must have no strict real-time characteristic.

Preemptive scheduling provides a good foundation for real-time system. In order to maximize the efficiency of scheduling systems, the operating system should run with certain real-time scheduling algorithm.

There are some common real-time scheduling algorithms, such as the Liu and Layland Rate-Monotonic (RM) scheduling algorithm and the earliest deadline priority (EDF) algorithm. The RM scheduling algorithm is a type of static scheduling algorithm, in which the priority of tasks are determined by the length of the cycle of task, and the shorter cycle of task has a higher priority. The EDF algorithm is one of the most popular dynamic priority scheduling algorithms that define priority of tasks according to their deadlines. Clearly, an excellent task scheduling algorithm can improve the operating system's real-time characteristic. However, it also consumes a certain degree of system resource. Thus, time complexity of scheduling algorithm, in turn, has an impact on the real-time characteristic.

B. The Context Switching Time

In a multi-tasking system, context switch refers to a series operation that the right of using CPU transferring from one task which is running to another ready for running one [4]. In preemptive scheduling systems, there are a lot of events that can cause context switches, such as external interrupt, or releasing of resource which high priority tasks wait for. The linkages of tasks in an operating system are achieved by the process control block (PCB) data structure. When context switches occurred, the former tasks information was saved to the corresponding PCB or stack PCB specified. The new task fetches original information from corresponding PCB. The time switching consumed depends on the processor architecture, because different processors need to preserve and restore different number of registers; some processors have a single special instruction which is able to achieve all the registers' preserve and restore job; some processors provide a number of registers group, the context switching required only need to change the register group pointer [5]. Operating system data structures will also affect the efficiency of context switch.

C. The Time of Kernel Prohibiting Interrupt

To ensure the atomic of operating to some critical resource, the operating system kernel has to prohibit all of interrupt sometimes. Interrupt will break the sequence of instructions, and may cause damage of data. Prohibiting interrupt always delay the response of request and context switching. In order to improve real-time performance of operating system, noncritical operations can be inserted between the critical areas. Setting reasonable preemptive points in critical areas can reduce the prohibition time of interrupt.

D. Efficiency and Treatment Methods of Interrupt

As the driving force for operating system scheduling, interrupt provides approaches of interaction between external events and operating system. The interrupt response speed is one of the most important ingredients which impact the real-time performance of system. At the end of each instruction execution, CPU will detect the status of interrupt. If there is an interrupt request and the interrupt is not prohibited, the system will execute a series of interrupt treatments: pushing values of CPU registers to stacks, obtaining the interrupt vector and getting the procedures counter register value, then jumping to the entrance of ISR and beginning to run, etc. [6]. What have mentioned above requires some system consumption. For a specific system, the consumption is identifiable, that is to say: it is possible to calculate the time delay of interrupt treatment caused by this part of work.

As interrupt management strategy, allowing interrupt nesting can further improve the response of high-priority incident's real-time, but relatively low-priority interrupt handling will be suffer negative impact. It should be considered under certain situation.

Non-emergency interruption may cause delay to important and urgent tasks, because interrupt handling is executed before task and thread. In order to reduce the delay, the handle process should be divided into two parts, just like Linux divided it into the top half and bottom half. Also

Windows CE's interrupt handling is divided into two parts: ISR and IST. They tried to keep ISR as a short program, while allowing tasks do more work, and make full use of the task scheduling mechanism.

E. Memory Management Mechanism

Generally, a real-time operating system uses the most efficient unified physical address space. Every task runs in the same address space. This management method can avoid the address space switching caused by the process scheduling that will occupy a lot of system resources. Because converting virtual address to physical address will lower the system performance, real-time operating systems use physical address directly, although it may bring security and stability problems. One of the most popular embedded operating systems-Vxworks uses the mechanism.

Real-time operating systems never use virtual memory, because it is hard to estimate the time of fetching data from external storage medium. When a page miss occurs, memory management should swap pages between internal memory and external memory. This process will suspend current running task. So the execution of real-time task cannot be assured.

F. The Race Condition among Tasks

The tasks of the system may compete for sharing resources. It will definitely cause some tasks to suspend and wait for the sharing resource. In preemptive scheduling kernel, priority inversion is a serious problem caused by race condition. A low-priority task which occupies critical resources has no right to implement, while a high-priority task has to wait a middle-priority task to release CPU to low-priority task. So the high-priority task is affected seriously and the task scheduling will become unstable and unpredictable. The real-time performance of system deteriorates rapidly. After all, the high-priority task can only seize the CPU from the low-priority task. It can't seize the resources. At this condition, it is necessary to use priority inheritance and priority ceiling to resolve the problem.

IV. Analysis of Linux Kernel's Real Time Performance and how it is restricted

It's well known that an operating system's real-time performance is evaluated by the following five technologic parameters: Deterministic, Preemptive, Context Switching, Interrupt Latency and Scheduling Latency [7, 8]. Context Switching is relative with specific CPU and Deterministic is determined by the remaining three aspects. So in this paper Linux kernel's real-time performance is discussed from Preemptive, Interrupt Latency and Scheduling Latency.

A. Preemptive

In general there are two modes in Linux kernel which are user state and core state. When a process operates at user state, preemptive scheduling is possible to happen if there is no shared data. But at core state the kernel is non-preemptive [4] and the tasks ready to run must be done in sequence. When a critical section of code is executed or Preempt disable command is used, the task cannot be preempted. In a word Linux kernel's preemptive performance still doesn't meet the need of hard real-time performance.

B. Scheduling Policy

Scheduling latency is the time that it takes for a high priority task ready to run caused by an event to wait to be done and is determined by interrupt latency, non-preemptive time and scheduling algorithm. In general Linux kernel scheduling algorithm is an O (n) algorithm indicating scheduling time is relative with the task scale, which is caused by concentrated computing time slices. Scheduling time is certain independent of task scale because Active queue and Expired queue are set so that it is unnecessary to compute time slices concentrated and scan the whole queue before scheduling switch. Thus easily resulting in that non-real-time task blocks real-time one by disabling interrupt.

C. Interrupt Latency

An interrupt has the highest priority and can preempt any task. It is common to disable interrupt for safety in Linux kernel process. If lower priority tasks disable interrupt there will be uncertain latency time for real-time task's response, which is not allowed for real-time system.

Improvement on Linux Kernel Real-Time performance

It takes long time for Linux kernel to develop and its performance to increase. However for the standard Linux kernel its real-time performance is always a problem unable to be solved completely. It is not because the designers are not excellent for many top programmers and engineers in the world take part in developing Linux kernel, but the standard Linux kernel needs to take into account fairness, balance and scale compatibility, and many other factors so that real-time performance has to give in. The real-time performance of Linux kernel is improved by improving both scheduling strategy and interrupt latency which block real-time task.

The hybrid Kernel gives the flexibility to select the system according to the application. The choices are as follows:

1) Desktop Operating System
2) Embedded Operating System
3) Embedded Operating System with Soft Real time requirements
4) Embedded Operating System with hard real time requirements

The Input given by the Application Program interface will be submitted to the kernel. Microkernel layer takes control, which are special for Interrupt Handler mechanisms and Specific schedulers. Micro Kernel deals with real time tasks and gives them main priority. Monolithic Kernel Deals with non real time applications and tasks. However the intermediate layer of Micro kernel deals with the applications, but the non real applications will be scheduled by the monolithic kernel. Thus the advantages of both the kernels will be achieved and make the system General purpose System.

V. Conclusion

The Flexibility to Use the Desktop OS and RTOS simultaneously for the flexibility and portability can be

achieved by the implementation of Hybrid System. At the same time the Power of Both the Os can be attained for the Specific Applications, and we can build the application custom so that more options are available to the User.

REFERENCES

[1] Z. A. Yu. Research of Real-Time Performance and Software Reliability based on Embedded Industrial Control System with Windows CE. *Master's dissertation of Northwest University*, 2009.

[2] H. F. Han, Research of Key Problem about Real-Time Operating System. *Doctor's Dissertation of Zhejiang University*, 2009.

[3] Q. Li and C. Yao. Real-Time Concepts for Embedded Systems. *CMP Books*, 2003.

[4] Tangyin. Real-Time Operating System Application development Guide. *China Electric Power Press*, July 2002.

[5] Milan Milenkovic. *OPERATING SYSTEMS: Concepts and Design*. (Second Edition), Tata McGraw-Hill Publishing Company Limited, 22nd Reprint 2007. Pp-403.

[6] S. Andrew, Tanenbaum, S. Albert and Woodhull. *Operating Systems Design and Implementation* (Third Edition), Prentice Hall, January 04, 2006.

[7] L. I. Bing and L. I. Zhong-wen. Analysis of Linux Real-time Mechanism. *Computer Technology and Development*, vol. 17(09), Sep. 2007, pp. 41-44.

[8] B. J. Wang, M. S. Li and Z. G. Wang. Uniprocessor static priority scheduling with limited priority levels. *Journal of Software*, vol. 17(03), March 2006, pp. 602-610.