ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

Advanced Interrupt Mechanism for Hybrid Operating System

Prakash S. Prasad[1], Professor Dr. Akhilesh R. Upadhyay[2]

Research Scholar, CSE Department, Bhagwant University, Ajmer, Rajasthan.INDIA[1].

Director,Sagar Institute of Research,Technology and Science,Bhopal–462 041 (M.P.) INDIA[2]

Abstract: Real time performance analysis is critical during the design and integration of embedded software to guarantee that application time constraints will be met at run time. To select an appropriate operating system for an embedded system for a specific application, OS services needs to be analyzed. These OS services are identified by parameters to form Performance Metrics. From five performance parameters of real-time operating system, scheduling latency and interrupt latency are the fundamental constraints for improving real-time performance of Linux OS. The proper handling of Interrupt is given in this paper. This paper analyzed the performance metrics in order to select right OS for the specific embedded application and also suggested the mixed kernel architecture for advantage. Another way to handle interrupt mechanism is also discussed for Hybrid System.

Keywords: Operating system, real-time system, Hybrid system, Linux kernel, Interrupt Mechanism.

I. INTRODUCTION

COMBINING both a real-time and a time-sharing subsystem, hybrid operating systems can provide both predictable real-time task execution and non-real-time services with well-known interfaces and lots of existing applications. In order to achieve relatively low development and maintenance costs, the time-sharing subsystem of a hybrid system is often based on commodity operating systems, such as Linux[1].

Real-time systems are widely used in the construction of national defense, aerospace, industrial control and many other fields. As the most critical characters which may affect the whole system, real-time character is required to guarantee the performance of these systems. With the development of computer technology, electronic information technology, many systems are controlled by computer system currently. As the majority of general-purpose computers are using Microsoft's Windows or Linux which is open-source as operating system, which are not real- time operating system, so these systems have not enough real-time characters. At present, the vast majority of real-time systems are embedded systems while built through the embedded processors with embedded operating system. In this paper, “real-time systems” mentioned in the following text are embedded systems.

Combining both a real-time and a time-sharing subsystem, hybrid operating systems can provide both predictable real-time task execution and non real-time services with well known interfaces and lots of existing applications. In order to achieve relatively low development and maintenance cost the time-sharing subsystem of a hybrid system is based on commodity operating system such as Linux[1]

ENERGY consumption is an important design concern for mobile embedded systems that are battery powered and thermally constrained. Displays have been known as one of the major power consumers in mobile systems . Conventional liquid crystal display (LCD) systems provide very little flexibility for power saving because the LCD panel consumes almost constant power regardless of the display content while the external lighting dominates the system power consumption [3]

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 [8].

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.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

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[7].

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 [5].

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 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.

Characteristics

Its real time characteristic-Response to events in time and complete tasks within the limited time

The scheduling objective is letting high priority task go first. The tasks running on real-time operating system should be certain.

Some data are highly sharing in real-time operating system.

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.

Scheduling of tasks

It is crucial for the real-time operating system to adapt preemptive scheduling kernel, which is based on task priority. The µC/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 characteristics. However, it also consumes a certain degree of system resources. Thus, time complexity of scheduling algorithm, in turn, has an impact on the real-time characteristic.

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 [6]. 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

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

and restore different number of registers; some processors have a single special instruction which is able to achieve all the register's preserve and restore job; some processors provide a number of registers group, the context switching required only need to change the register group pointer [9]. Operating system data structures will also affect the efficiency of context switch.

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.

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. [3]. 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

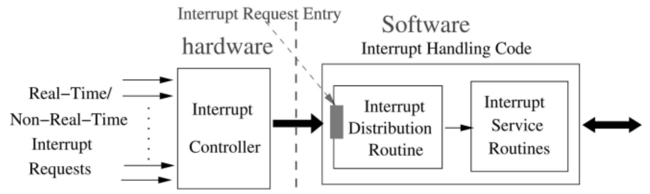

Fig. 1. A typical interrupt request handling procedure in a hybrid system

Figure shows a typical interrupt request handling procedure

in a hybrid system. Basically, real-time and non-realtime

interrupt requests are passed to the interrupt handling code through the interrupt request entry. The interrupt handling code can be separated into two parts, the interrupt distribution routine and interrupt service routine (ISR). First, the interrupt distribution routine determines the entry point for an interrupt request. Next, a specific interrupt service routine is called, and then the interrupted task/program/interrupt is resumed or a new task/interrupt is rescheduled to be executed before exiting from the ISR. Although the interrupt handling in hybrid systems looks like that in general-purpose OS, it has to be changed a lot with the structure shown in Fig. 1, in which real-time and non-real-time interrupts are passed through the same interrupt request entry. First, in the interrupt distribution code, in order to satisfy the predictability of the real-time subsystem, we need to separate real-time and non-real-time interrupts since they will be processed differently. Second, we have to solve the interrupt disabling problem when dealing with non-real-time interrupts[2].

The interrupt disabling problem is caused as follows: The time-sharing subsystem of a hybrid system is usually treated as the task with the lowest priority. With the lowest priority, the time-sharing subsystem task cannot block realtime interrupts nor can it prevent itself from being preempted. On the other hand, in a time-sharing operating system, such as Linux, interrupt disabling is frequently used in interrupt handlers, critical sections, and so on. And in most processors, interrupt disabling is achieved by masking the interrupt disabling/enabling bit in the Program Status Word (PSW) register, and all interrupt requests will be disabled if the bit is set. In hybrid systems, we cannot really set the interrupt disabling/enabling bit for interrupt disabling from the time-sharing subsystem task.

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.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

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.

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.

I. 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 [1, 2]. 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 other words 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. Therefore Interrupts should be properly handled and tackled for the scheduling of the Tasks and Handling the Interrupts.

D. 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.

II. Mixed Kernel Architecture

The Mixed kernel is the combination of Monolithic and Micro Kernel. As the Monolithic kernel is generally associated with the desktop Operating System and Micro Kernel is with Embedded or the Real Time Operating System. So whenever the task is supplied to the Kernel it will first determine if it is to be submitted to the Monolithic Kernel or the Micro Kernel, and then accordingly scheduling strategy will be applied to the tasks so that the execution of tasks can be delivered at the real time constraints.In this system some tasks are required to be suspend and wait for the sharing resource. The real-time and non-realtime

interrupt requests are passed to the interrupt handling code through the interrupt request entry. The interrupt handling code can be separated into two parts, the interrupt distribution routine and interrupt service routine (ISR). First, the interrupt distribution routine determines the entry point for an interrupt request. Next, a specific interrupt service routine is called, and then the interrupted task/program/interrupt is resumed or a new task/interrupt is rescheduled to be executed before exiting from the ISR. Although the interrupt handling in hybrid systems looks like that in general-purpose OS, it has to be changed a lot with the structure shown in Fig. 1, in which real-time and non-real-time interrupts are passed through the same interrupt request entry. First, in the interrupt distribution code, in order to satisfy the predictability of the real-time subsystem, we need to separate real-time and non-real-time interrupts since they will be processed differently. Second, we have to solve the interrupt disabling problem when dealing with non-real-time interrupts[2].

The interrupt disabling problem is caused as follows: The time-sharing subsystem of a hybrid system is usually treated as the task with the lowest priority. With the lowest priority, the time-sharing subsystem task cannot block realtime interrupts nor can it prevent itself from being preempted. On the other hand, in a time-sharing operating system, such as Linux, interrupt disabling is frequently used in interrupt handlers, critical sections, and so on. And in most processors, interrupt disabling is achieved by masking the interrupt disabling/enabling bit in the Program Status Word (PSW) register, and all interrupt requests will be disabled if the bit is set. In hybrid systems, we cannot really set the interrupt disabling/enabling bit for interrupt disabling from the time-sharing subsystem task

VI CONCLUSION

The selection of right operating system for a specific application has a great impact on performance of real- time system. In the embedded application the improvement of real-time performance of Linux kernel has far-reaching significance. The Interrupt Mechanism for hybrid System is to be dealt with proper care so that the basic cause of Hybrid system is to be maintained. The complex relationship between the tasks may cause heavy system consumption on internal communication between tasks. The Scheduling and separating the tasks in the beginning will also give us flexibility to apply different algorithms for different task queues. Synchronization mechanism between tasks will decline the real-time performance of system. The Interrupt should be activated at proper time so that the race condition among the tasks can be prohibited and the interrupt should not mask the hard real-time tasks. At last, how to use a real-time operating system to implement an actual application system is the key for all embedded system developer.

REFERENCES

[1] Yodaiken and M. Barabanov, "Real-Time Linux," Proc.Applications Development and Deployment Conf. (USELINUX), Jan.1997

[2] Miao Liu et.all, "On Improving Real Time Interrupt Latencies of Hybrid Operating Systems with Two-Level Hardware Interrupts",IEEE Transactions On Computers, Vol. 60, No 7, July 2011, pp. 978-991.

[3] Mian Dong and Lin Zhong, "Power Modeling and Optimisation for OLED Displays", IEEE Transactions on Computers,Vol. 11, No 9, September 2012, pp. 1587-1599.

[4] 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.

[5] 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.

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

[7] J. S. Xing, J. X. Liu, and Y. J. Wang, "Schedule ability test performance analysis of rate monotonic algorithm and its extended ones", Journal of Computer Research and Development, vol. 42(11), Nov. 2005, pp. 2025-2032.

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

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

[10] Chen Han Fei, "Research of Key Problem about Real-Time Operating System", Doctor's Dissertation of Zhejiang University,2009.

[11] YuZhaoAn, "Research of Real-Time Performance and Software Reliability based on Embedded Industrial Control System with Windows CE", Master's dissertation of Northwest University,2009.

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

BIOGRAPHY

Prakash S. Prasad has published more than 10 papers in National and International Conferences. He is Member Of IEEE, ISTE and IACSIT. He has completed his bachelors degree in 1997, and Masters Degree in 2007. He is currently working as assistant professor at Priyadarshini college of Engineering and Head of The Department of Information Technology. He is having 14 Years of Teaching experience and his interests include network security, Operating System and System Software.

Prof. (Dr.) Akhilesh R. Upadhyay obtained Ph.D. degree in Electronic Engineering from the Swami Ramanand Teerth Marathwada University, Nanded in 2009, M.E. (Hons.) and B.E. (Hons.) in Electronics Engineering from S.G.G.S. Institute of Engineering & Technology, Nanded [M.S.] in year 2004 and 1996 respectively. He is currently working as Professor EC Dept. and Vice Principal at Sagar Institute of Research and Technology, Bhopal, India since October 2009. He has more than 13 years teaching and 3 years of industry experience. He is Associate Editor of Journal of Engineering, Management & Pharmaceutical Sciences, Ex-Editor of International Journal of Computing Science and Communication Technologies and member of Editorial Boards/Review Committee of various reputed Journals and International Conferences of various Countries. He has more than 54 research publications in reputed International/National Journals and Conferences (e,g, IEEE, ACM, Springer, IJCTEE, JICT). He also authored more than 16 text/reference books on electronics devices, instrumentation and power electronics for various Universities. He is recognized Ph.D. Supervisor for various Universities in India and presently guiding 10 Ph.D. scholars. He visited Malaysia in July 2007 as overseas representative of APIIT and San Francisco California, U.S.A. in January 2012 for attending International Conference "Electronic Imaging 2012" to present his latest research work.