Design and Implementation of Dual Kernel for Considering Hard Real-Time Constraints.

Seung-mo Yang
Dept. of Computer Science
Semyung University
San 21-1, Shinwol-Dong, Chechon
Chungbuk Korea
ysm3333@kebi.com

Chi-ho Lin*
Dept. of Computer science
Semyung University
San21-1,Shinwol-Dong,Chechon
Chungbuk Korea
ich410@venus.semyung.ac.kr*

Hi-seok Kim**
Dept. of Electronic Engineering,
Chongju University
36 Naedik-Dong Sangdang-Gu
Chongju Chungbuk Korea
khs8391@chongju.ac.kr**

Abstract

Because of the great variety of demands on real-time scheduling, real-time kernel should be small, fast and predictable. In this paper, we present that Real-time applications should be split into small and simple parts with hard real-time constraints.

Following this concept, we have designed and implemented to have the properties of both hard real-time kernel and general kernel.

And, to prove be useful the proposal kernel, we compare and analyze the performance with RT-Linux 0.5a

1. Introduction

A real-time computer system can be define as a system that performs its functions and responds to external, asynchronous events within a specified amount of times[1]. Most control systems fall into this category. A real-time operating system capable of guaranteeing timing requirement of the processes under its control. That is, Correct timing is the key feature. There are hard and soft real-time systems[1]. Soft real-time systems are those in which timing requirement are statistically defined[2].

An example can be a video conferencing system: it is desirable that frames are not skipped, but it is acceptable if a frame or two is occasionally missed. In hard real-time system, the deadlines must be guaranteed [2]. For example, if during a rocket engine test this engine begins to overheat, the shutdown procedure must be completed in time. One key issue is the need to provide predictability.

In this paper, to satisfy the above requirement, real-time kernel has designed and implemented the following properties; predictability, fast, low interrupt latency, simple scheduler.

2. Design and Implementation

2.1 Design and Implementation of hard Real-time Kernel

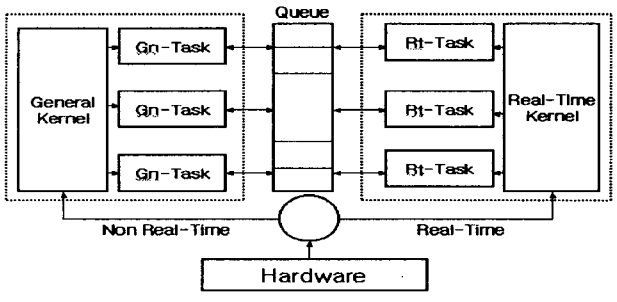


Figure 2.1 Total structure of kernel.

To stability of kernel through efficient distribution of total work, the proposal kernel was designed and

implemented independent between general kernel and hard real-time kernel. real-time kernel and general kernel, that have different properties as shown in Figure 2. 1.

Real-time kernel has the following properties. Hard real-time task and interrupt service routines are executed in that part. Real-time task are running in kernel mode because the overhead of system call can be reduced dramatically and the fast context switch is possible. the scheduler of real-time kernel is a simple priority-based preemptive scheduler. It is implemented as a routine that chooses among the ready tasks with one with the highest-priority and marks it as a next task to execute. Tasks give up the processor voluntarily or are preempted by a higher-priority task when its time to execute comes. There are no tasks with the same priority, that is, one priority can be assigned to only one task. As a result, the maximum number of tasks executed at the same time is 256. But the 2 priority, priority 0 and priority 1, have been reserved. priority 0 is reserved for real-time IDLE task that is executed when there is no task to run. IDLE task does not usually work anything, but the real-time IDLE task starts the switcher of the general part and wait forever for next real-time event. Priority 1 is reserved for interrupt service associated with tasks in general kernel. it is executed by real-time kernel when there is no real-time tasks in ready state.

General kerenl has the following properties. Task are running in user mode and scheduler of this part works like the general purpose time shared operating system and is non-preemptive, optimize the average case and maximize the task throughput. There are tasks with the same priority, that is one priority can be assigned to many tasks. The tasks with the same priority level are scheduled round-robin. priority 0 has been reserved for general IDLE task that is executed when there is no tasks to run.

2.2 Task

This paper proposal kernel support the general tasks and three kinds of real-time task; "interrupt task", "periodic Task" and "asynchronous task".

Real-time interrupt task is designed to be used for controlling or giving/taking the service to/from the device that informs the service request non-periodically by the interrupt. for example, when the device has an error or a fault, or request/service the data to process, it requests the recovery or service to the real-time interrupt task.

Periodic task is designed to be used for controlling or giving/taking the service to/from the device periodically. For example, the video frame must be transferred to the memory at every 1/30sec and processed. In this case, the associated periodic task must be waked up at every 1/30sec and transfer the data.

Asynchronous task is designed to be used for command or data processing in real-time. If you need control the device command or process the data and transfer the result to any real-time task or general tasks in real-time, use asynchronous task.

2.3 Scheduling

The main task of real-time scheduler is to satisfy timing requirements of tasks[3]. Scheduler have been implemented a priority-based preemptive scheduler. The scheduler

directly supports periodic tasks. The period and the offset is specified for each of them. An interrupt-driven(sporadic) task can be implemented by defining an interrupt handler that wakes up the needed task. for periodic tasks with deadlines equal to periods a natural way to assign priorities is given by yhe rate monotonic scheduling algorithm[4]. According to this algorithm tasks with shorter periods get higher priorities. A set of n independent periodic tasks scheduled by the rate monotonic algorithm is guaranteed to meet all deadline if

$$\frac{C1}{T1}+\frac{C2}{T2}+\frac{C3}{T3}+\cdots\frac{Cn}{Tn}\leq n(2^{\frac{1}{n}}-1)$$

where Ci is the worst-case execution time of task I, and Ti is the period of task I. Sporadic tasks can often be treated as periodic ones for priority assignment[5].

The scheduler treats general kernel as the lowest priority real-time task. General kernel only runs when the real-time kernel has nothing to do.

2.4 Timing

Precise timing is necessary for the correct operation of the scheduler. Timing inaccuracies cause deviations from the planned schedule, resulting in so-called task release jitter[5]. In most applications task release has an undesirable effect. It is important to minimize it. In this paper kernel manages time by receiving non-periodic interrupts which is viewed periodically on the point of view of one task from system time base. The time generator must provide an interrupt at a fixed interval.

In this paper, the kernel has generalized time management using one-shot and cyclic timers on conjunction with semaphore[6]. Multiple timers are managed simultaneously using an ordered list of pending timer event.

2.5 Interrupt handling

One of the problems with doing hard real-time system is the fact that kernel uses disabling interrupts as a means of synchronization[7].

In this paper, interrupt is happened hardware, regard as one of external events. as happened hardware interrupt can to be separated RT-interrupt and GN-interrupt(general event). RT-Interrupt means that immediately process requested service call from hardware by sensitive time. GN-Interrupt means that happened hardware needless concept of real-time. Here is because interrupt regard to event, RT-Interrupt is doing wakeup task as send interrupt event to connected task, In this point, It defined interrupt task, which received Interrupt event.

2.6 Interprocess Communication.

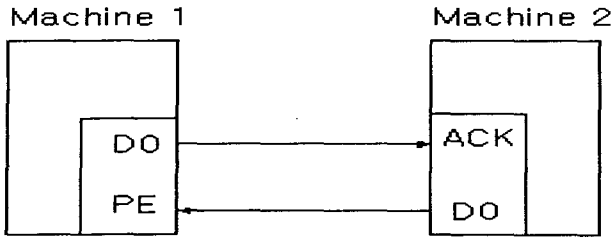
Communication between both parts split out is via queue that can be only created by real-time tasks and connected to by the general tasks. The QUEUE is only one directional that is determined on creation[7].

That is data is moved from real-time kernel to general kernel or from general kernel to real-time kernel. Also it is determined whether tasks are blocked or not when real-time kernel is created or connected to.

3. Experimental Results

In order to measure the performance of the proposal kernel, We have conducted several experiments with respect to interrupt latency and predictability[7]. The experiments

were performed on two performed on two IBM PC compatible computers running Real-time Linux version 0.5a, and the proposal kernel.



Machine A: Pentium 133MHz, 32MB of RAM.

Machine B: Pentium 166MHz, 32MB of RAM

Figure 3.1Measuring Interrupt Latency

To measure the maximum interrupt latency, an additional machine running the proposal kernel (Machine 1) was used to send interrupt requests to the machine being tested (Machine 2) and to measure the response time of the latter(Figure 3.1).

system	Interrupt latency(us)	Scheduling precision(us)
Machine A RT-Linux kernel	84	155
Machine A, the proposal kernel	80	151
Machine B RT-Linux	34	64
Machine B, the proposal kernel	30	61

Table 3.1. Performances Measurements Results

To measure scheduling precision a periodic real-time task was run. On each wake-up the current time was obtained and compared to the estimate. Maximum deviation were recorded. The result was seen form Table 3.1.

Overall, the results show that the proposal kernel is a viable platform for hard real-time processing.

4. Conclusion

In this paper, we was designed and implemented small and efficient kernel on real-time control system, and to minimized overhead of total system through efficient distribution of total work, we was implemented independent between general kernel and hard real-time kernel, not only small but also to be compatible application field of high speed real-time control system.

Toward, real-time control systems will be ask for fast response time, minimum of interrupt latency and scheduling precision. Future, expected that will be assistance efficient expansibility of real-time control systems.

5.Reference

[1] Borko Furht, Dan Grostick, et al. Real-time UNIX systems: design and application guide. Kluwer Academic Publishers Group, Norwell, MA, USA, 1991

[2] Alan Burns. Scheduling hard real-time systems: A review. Software Engineering Journal 6(3): 116-128, 1991

[3] J. A Stankovic, "Microconception about Real-Time Computing," IEEE Comput. vol.21, no. 10, Oct. 1988

[4] C.L Liu and J.W Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM,20(1):44-61, January 1973.

[5] Sang H. Son, editor. Advances In Real-Time Systems , chapter 10, pages225-248. Prentice, 1984.

[6]] JEAN J. LABROSSE, "A Portable Real-Time Kernel in C." p40-53, EMBEDDED SYSTEMS PROGRAMMING, MAY. 1992.

[7] Daniel Stodolsky, J. Bradley Chen, and Brian Bershad. Fast interrupt priority management in operating system kernels. In proceedings of the 2nd USENIX Symposium on Microkernels and Other kernel Architectures. USENIX, September 1993.

[8] Michael Barabanov, " A Linux-based Real-Time Operating System" , New Mexico Institute of Mining and Technology Socorro, New mexico, June 1, 1997.