International Journal of Computer Applications (0975 – 8887)
Volume 61– No.1, January 2013

Real-time Linux using RTAI

Laxmikantha.K
Department of Information Science Engineering
PDIT, Hospet, India

Kishore M
Department of Electronics and Communication Engineering
PDIT, Hospet, India

H. M. Guruprasad
Department of Electronics and Communication Engineering
PDIT, Hospet, India

ABSTRACT

To support the hard real-time requirement for mission critical applications, we enhance the real-time ability in Linux kernel through some kernel mechanisms. First, we focus on new technique called normal task delayed locking technique can be used to reduce the OS latency. Second, because of the coarse-grained timer mechanism can not satisfy the microsecond-level timer resolution required by real-time tasks, we present a new microsecond-level timer mechanism, which is based on UTIME technique. The simulation and analysis shows that the design can improve the real-time performance of the Linux system efficiently, which could be used to most of the embedded hard real-time systems.

Keywords

Real-time systems, UTIME technique, OS latency, coarse-grained timer, mission critical applications

1. INTRODUCTION

Real-time systems are computing systems that must react within precise time constraint to events in the environment, and applied especially for embedded equipments. Linux is an operating system that becomes increasingly popular. Its flexibility and its adherence to the POSIX standard [1] make it a good candidate for general adoption in the embedded systems market, however, the Linux, designed to be a traditional monolithic unpreemptable kernel, is not suitable for real-time applications for the following issues. (1) Lacking for effective real-time scheduling. Although Linux has limited POSIX real-time support after version 1.3, which allows a process to be specified as a real-time process, the scheduler simply gives real-time processes higher priorities than normal processes. (2) Unpreemptible kernel, Linux 2.6 has realized a preemptable kernel, but it doesn't resolve the problem of uncertain interrupt latency. (3) Coarse-grained timer mechanism can't satisfy the microsecond-level timing constrains required by real-time jobs. (4) Frequent interrupt events greatly increase the indeterminate execution delay of real-time jobs. These inabilities makes the standard Linux kernel unusable in hard-real time and mission critical systems where missing a deadline may endanger the whole system or even lives. So we adopt two new kernel mechanisms to improve the hard real-time of the Linux. The rest of the paper is organized as follows: In section 2 we describe the standard solutions devised up to the present to cope with the lack of hard real-time capabilities of the Linux kernel; The proposed approach, relying on embedding a standard Linux kernel, which includes the method so-called normal task delayed locking technique, section 3 describes proposed approach and implementation ; In section 4 includes simulation and analysis Finally, in section 5 we provide some concluding remarks.

2. STANDARD SOLUTION

In the past, much work has been done to improve the real-time capability in Linux kernel. However, all of these solutions adopt one of these two approaches: 1)A sub-kernel is a very small and minimal operating system that provides a very good real-time performance in terms of determinism, response time, preemptibility and timing guarantees. patching kernel approach Modify and patch the Linux Kernel to provide a behavior that is as close as possible to the one of a full fledged real-time kernel, as initially proposed by the KURT project to make good use of strong function of Linux kernel, we find out the second approach is a good choice, which can either raise system real-time performance and is easy to realize, and also fit for development of embedded systems.

3. PROPOSED APPROACH AND IMPLEMENTATION

The most important reason because Linux is not a native real time system is the presence of unpredictable delays caused by non-preemptible operations running in kernel space. Version 2.6 of the Linux kernel introduces the possibility to preempt some system calls. However there still are non-preemptible operations that can introduce unresponsiveness. In this paper we focused on the unresponsiveness introduced by non preemptible sections, because it is the source of the largest peaks. To reduce the unresponsiveness that may be introduced by non-preemptible sections inside the main kernel and the modules that may be loaded into the kernel to add functionalities, we designed a method called normal task delayed locking technique.

3.1 Real time Preempt Modules

Task response time is the time interval from the coming of a certain event to the task response to the event and execution of it. So we give the definition of the OS latency: If the process that requires execution at time t is actually scheduled at time t', we define the OS latency as KLT = t' - t. Imaging the following scene:

There are two tasks: task A and task B. Task A is a normal task and task B is a real-time task. When task A is running in critical section S, an interrupt occurs and wakes up a higher priority task B. Suppose that task B also want to enter the critical section S, which is currently held by task A. The real time task B's kernel latency time KLT (B) stand show in the (1), and its max kernel latency time show in the (2):

KLT (B) stand=T (A) execute +T(B) schedule (1)

KLT (MAX) stand=MAX {T (A1) execute, T (A2) execute

…… (AN) execute} + {MAX {T (B1) schedule,

T (B2) schedule……T (BN) schedule} (2)

Texecute is the task execution time, and Schedule is the task schedule time. If the probability, which is TAexecute <T(B) schedule, is Φ, the average kernel latency time AKLTstand show in the (3):

AKLTstand=Φ*(T (B) schedule) + (1-Φ)*(T (A) execute+

T (B) schedule) (3)

If Schedule and Execute are assumed to be constant as in Linux2 .6, AKLT only depends on the probability Φ, if we can reduce the probability Φ, we can minimize the kernel average delay time caused by critical section, so propose a new technique, called a normal task delayed locking technique, to improve the real-time performance of embedded Linux. The proposed technique employs the rule that entering a critical section is allowed only if the normal task does not disturb the future execution of the real-time application. The decision process is illustrated in Figure 1

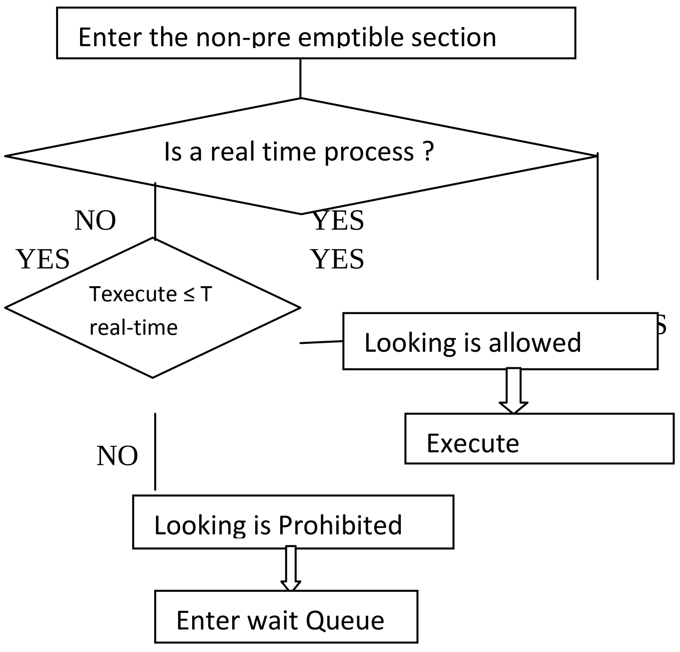

Figure1. Decision process

When one thread that belongs to normal task enters non preemptible section, it compares Texecute with the next real timer interrupt Treal_time. If execute is larger than Treal_time, locking is prohibited and the thread enters a wait queue for future execution. Otherwise, locking is allowed and the thread starts the execution as usual. The decision process guarantees that the execution of a non-preemptible section will not delay scheduling the real-time process.

And after using normal task delayed locking technique, the AKLT can be minimized because the conflict probability Φ between task A and B is very low. The real time task B's kernel latency time KLT (B)delay show in the(4), its max kernel latency time KLT(MAX)delay show in the (5), and its average kernel latency time AKLTdelay show in the (6):

KLT (B) delay=T (B) schedule (4)

KLT (MAX) delay=MAX {T (B1) schedule,.T (B2) schedule T (BN) schedule} (5)

AKLTdelay=T (B) schedule (6)

From formulas (4), (5), (6) we can see the changes between the standard Linux and the modified Linux through formulas (7), (8), (9):

ΔKLT=KLT (B) stand- KLT (B) delay =T (A) execute
(7)

ΔKLT (MAX) stand = MAX {T (A1) execute, T (A2) execute …T (AN) execute} (8)

ΔAKLTstand= Φ*(T (B) schedule) + (1-Φ)* (T (A) execute +T (B) schedule)- T (B) schedule = (1-Φ)*T (A) execute (9)

From above, we can see that the normal task delayed locking technique greatly reduce the indeterminate kernel delay. This model shows the principle: The longer the execution of the critical section, the more the average delay of the high priority task B caused by the critical section can be reduced by the normal task delayed locking technique.

3.2 Avoiding Starvation

Since our approach is to defer the execution of a non-real-time process conditionally, the starvation problem can arise if an urgent timer interrupt occurs frequently enough to cause the process to sleep indefinitely. More specifically, the condition under which the starvation problem may arise is as follows:

T (A) execute>MAX {T (B1) real-time,

T (B2) real-time, T (BN) real-time} (10)

One possible solution to avoid such a starvation is that if the non-real-time process is blocked by an urgent timer interrupt more than, for example, 100 times, then the process ignores the locking decision and forces itself to acquire the spin lock. At that very moment, the real-time process may experience a longer latency than before, due to the long non-preemptible section. However, this situation occurs rarely, because even in the worst case, such a long latency will not occur for the next 99 times. In addition, starvation is avoided, since non-real time process will have already passed the obstacle, a non-preemptible section

3.3 Fine-grained Timing Resolution

In general, an operating system providing time-sharing mechanism uses a periodic timer to dispatch the CPU time to every task. It can achieve a good balance between the task responsiveness and context switching overheads through selecting a proper timer frequency, and usually, millisecond level timer resolution can be achieved. However, millisecond level timer resolution can't satisfy the microsecond-level timer resolution required by real-time tasks. Thus providing a microsecond level timer resolution is the basic and most important feature in real-time system. While reducing the period of the timer interrupt can get microsecond-level timer, it would increase the overhead to handle the interrupt. In this paper, we implement a new timer scheme based on the UTIME to support microsecond-level timing resolution. Firstly, two independent timer managers are implemented in the kernel: (1) standard timer manager: it provides coarse-grained timing resolution, i.e. 10ms in the general case, to normal tasks; (2) high-resolution timer manager: it offers microsecond-level timing resolution, it can be generated by using the RTC, a particular clock usually built into the chipset of all PCs and PC-derived computers, for real time tasks. In the default case, timers from normal tasks are added to the first manager, while timers from real-time tasks are added to the second manager. In this way, the real-time response can be better guaranteed by our timer mechanism. Besides, more flexible different-grained timing requirements are supported in our system. Because microsecond-level timing resolution can be available to non-real-time tasks through extended programming interfaces, which usually need coarse-grained timing resolution in the default case.

3.4 Implementation

There are two Linux modules, a spin lock monitor and a spin lock manager, are added to original Linux OS. The spin lock monitor continuously tracks the lock hold time of each non-preemptible section, and if necessary, if updates the Texecute Value. The spin lock manager determines whether it will allow the locking or not , by comparing Treal time from the Urgent timer and Texecute from the spin lock monitor, as explained in section 3.1. Whenever one process tries to a spin lock, it first has to get permission has been issued by the spin lock manager, the process acquires the spin lock and starts the execution of the non-preemptible section.

```
#defines spin_lock (Texecute, Treal_time) {
if (real_timer_off){
do {
If (Texcute >Treal_time){
sleep_this_thread ();
}
measure_current_time ();
preempt_disable ();
_raw_spin_lock (lock);
add timer ()}}
#defines spin unlock () {
do {update_lock_hold_time ()
raw_spin_unlock (lock);
 preempt enable ();
}
```

4. DISCUSSION

Experiment environment in this article is based on CPU: Intel® Pentium® 4 CPU 1.80GHz Processor, 504 MB of RAM. We make a test program to test standard Fedora Linux and modified Fedora Linux 2.6.23. In order to compare the real time performance. The tests consist to run selected applications on the system, generate interrupt and measure the Linux kernel latency response and interrupt latency. One was a real-time process and the others were background processes. The real-time process model is a periodic real-time application, such as video player, and we let it repeat executions for some times measure the interrupt latency and jitter for normal task it will take time in terms of millisecond and for real-time task it will take time in terms of microseconds

5. CONCLUSION

The Linux kernel version 2.6 has made some progress towards a better real-time performance but still it contains long critical Secondly we present a new microsecond-level timer mechanism, which is based on UTIME, but provides Sections that cause long interrupt latency and pre-empt Latency. In this paper, we enhance the real-time ability in Linux kernel through two kernel mechanisms. Firstly, We propose a normal task delayed locking technique; For improving the real time performance of the embedded Linux preferential support for microsecond-level timers. These solutions can reduce the OS latencies, and limits the interrupt latency and jitter to a constant time.

6. REFERENCES

[1] C.L. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46-61, 1973

[2] L. Abeni, A. Goel, C. [2] M. Barabanov and V. Yodaiken. Real-time Linux –Linux journal, March 1996

[3] P. B. Sousa, K. Bletsas, E. Tovar, and B. Andersson, “On the implementation of real-time slot-based task-splitting scheduling algorithms for multiprocessor systems,” in *Proc. of the 13th Real-Time Linux Workshop (RTLWS’13)*, Prague, Czech Republic, 2011, pp. 207–218.

[4] E. Bianchi, L. Dozio, Some Experiences in fast hard realtime control in user space with RTAI-LXRT, 2nd Realtime Linux Workshop, Orlando, Florida (USA), November 27-28, 2000.

[5] Krasic, J. Snow and J. Walpole, “A Measurement-Based Analysis of the Real-Time Performance of Linux”, In The 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2002) , SanJose, September 2002

[6] Y.C. Wang and K.J. Lin. Enhancing the real-time capability of the Linux kernel. In *Proc. of 5th RTCSA’98*, Hiroshima, Japan, Oct 1998

[7] M. Tim Jones, “Anatomy of real-time Linux architectures From soft to hard real-time”, IBM 15 Apr 2008

AUTHORS PROFILE

Prof.Laxmikantha K, currently working at Proudhadevaraya Institute of Technology, Hospet, India. His area of interest includes C&UNIX, Embedded Systems, RTOS, System software, Device drivers.

Mr.Kishore M, Currently working as an Assistant Professor at Proudhadevaraya Institute of Technology (PDIT), Hospet, India. His area of interest includes wireless communication and networking, antenna theory and design, smart antenna and its application. Integrated Circuits. He as various national and international publications into his credit.

Prof. H M Guruprasad, Currently working at Proudhadevaraya Institute of Technology, Hospet, India. His area of interest includes VLSI, analog and digital communication, real time embedded systems. Member of Indian society of Technical Education, New Delhi.