Coexisting Real-Time OS and General Purpose OS on an Embedded Virtualization Layer for a Multicore Processor

Hitoshi Mitake, Yuki Kinebuchi, Alexandre Courbot, Tatsuo Nakajima
Department of Computer Science
Waseda University
{mitake, yukikine, alex, tatsuo}@dcl.info.waseda.ac.jp

ABSTRACT

Porting operating systems to a virtualization layer produces a semantic gap because the assumptions that guest OSes rely on may not be ensured. On multi-core environments, this gap can cause the fatal performance degradations. The lock holder preemption (LHP) problem is a well known example of the sources of the performance degradation. It occurs when a thread holding a spin lock in an OS kernel is preempted by other OS kernels.

Some previous proposals can avoid this problem, but none of them cares about the real-time responsiveness of guest OSes. Therefore the approaches are not suitable for embedded systems. We have developed a new technique for avoiding the LHP problem. The approach can ensure both the real-time responsiveness of RTOS and the high throughput of GPOS that supports shared memory multi-processors.

This paper introduces the basic approach of our new technique and its experimental results. The results show that our new technique can make RTOS and GPOS coexist without degrading the real-time latency and is suitable to be applied to modern high performance multi-core processor based real-time embedded systems.

Categories and Subject Descriptors

C.3 [Computer System Organization]: Special-Purpose and Application-Based Systems—Real-time and embedded systems; D.4.7 [Software Operating Systems]: [Organization and Design][Real-time systems and embedded systems]

General Terms
Operating System

Keywords
Embedded system, Virtualization, Lock Holder Preemption

1. INTRODUCTION

The trend of multicore is coming to the world of embedded systems. As the combination of multi-core processors and virtualization technologies brought the benefits to the server side computing, they can also solve the problems of embedded systems.

As GPOS (General Purpose Operating System) cannot guarantee the real-time responsiveness, RTOS (Real-Time Operating System) cannot provide advanced high level functionalities such as VFS operations required by functional rich applications like a web browser. So modern embedded systems like smart phones are constructed mainly in one of the following two ways: 1) Assigning special hardware for the real-time functionality 2) executing both of RTOS and GPOS using the hybrid kernel approach, where executing GPOS as one task of RTOS. The first approach is not only inefficient from the economical perspective, but also the waste of hardware resources because a special purpose hardware like a base band processor for a mobile phone cannot be used for general computation. The second approach is efficient approach, but it introduces a large amount of software engineering cost.

When we consider a multi-core processor, RTOS can be coexist with SMP GPOS. Now, Linux already supports a multi-core processor. Thus, coexisting RTOS and SMP GPOS on a multi-core processor will be common in various embedded systems. However, there is a possibility to cause a fatal performance problem. In an SMP OS kernel, there are many shared resources that are protected using a spin lock. When the SMP OS is used on a virtualization layer, the lock holder preemption (LHP) problem occurs when RTOS preempt the execution of an SMP GPOS kernel when the kernel holds a spin lock in the kernel. Of course, there are previous solutions to solve the problem, but the solution increases the latency of real-time activities. Therefore, the solution is not suitable for embedded systems, where real-time constraints are usually important.

In this paper, we propose a new technique for avoiding the LHP problem which realizes both the real-time responsiveness of RTOS and the high throughput of GPOS. We also show the effectiveness implemented in SPUMONE [1].

SPUMONE is a virtualization layer for combining RTOS and GPOS in one system. It provides virtual CPUs (vCPU) to its guest OSes and schedule them with the fixed priority scheduling algorithm. In a typical case, the vCPU of RTOS has a higher priority than the one of GPOS. And SPUMONE provides the vCPU migration mechanism. So the the mapping between vCPUs and physical CPUs (pCPUs) can be
changed dynamically.

2. DESCRIPTION OF LOCK HOLDER PRE-EMPTION

Uhlig et al. [2] introduced the performance degradation caused by the LHP problem under the virtualized environment. In general, LHP is caused by the preemption of the vCPU which is executing a thread holding a spin lock in the kernel by other OS kernels. LHP causes the waste of processor time because when other vCPUs owned by the same OS kernel try to acquire the spin lock that the preempted vCPU currently hold. Thus, the vCPU tries to acquire the lock cannot continue to enter into the critical section protected by the spin lock. Therefore, other vCPUs trying to acquire the lock will spin in vain until the blocked vCPU is scheduled again and finally releases the lock. As we will describe in Section 4, LHP causes the fatal performance degradation.

The method to avoid LHP is also described in [2]. The technique is called delayed preemption mechanism. The principle of the technique is to prohibit the preemption of vCPUs which executes threads holding a spin lock temporarily and making these vCPUs yielding pCPUs later.

The technique improve the LHP problem significantly, but when the technique is used for embedded systems, increasing the interrupt latency becomes serious problem.

3. A PROPOSED NEW TECHNIQUE BASED ON VCPU MIGRATION

A new technique proposed by us is based on the vCPU migration mechanism of SPUMONE. By using this mechanism, guest OSes can change the pCPUs which executes their vCPUs. Under our method, the vCPU of GPOS which is sharing one pCPU with the vCPU of RTOS will be migrated to another core when the load of RTOS become higher than the threshold.

4. EVALUATION

We evaluated the three configurations of SPUMONE: raw SPUMONE, the version implemented the delayed preemption mechanism, and vCPU migration enabled. We used TOPPERS, implementation of TRON specification, as guest RTOS and Linux 2.6.16 as guest GPOS. Evaluation environment is RP1 prototype board with 4 core SH4A processors running with 600MHz clock frequency and 128MB DRAM.

This graph shown in Figure 2 shows that the delayed preemption mechanism realizes the best score of hackbench. But the delayed preemption mechanism fatally harms the latency of TOPPERS. The worst case delay of TOPPERS is 222.268ms. So this method is not suitable for embedded systems. The worst score is produced by the previous SPUMONE. And the second score is produced by our new technique based on the vCPU migration. This is worse than the delayed preemption mechanism, but nearly equal to the situation that Linux dominates 3 cores. And worst case delay of TOPPERS is 0.16 $\mu$s under this configuration. This graph also shows that the performance of Linux is not affected by the load of TOPPERS under our new technique. Even if we apply the delayed preemption mechanism, the performance of Linux is affected by the load of TOPPERS. Once the vCPU of TOPPERS is scheduled, the situation means that a thread running on the preempted vCPU of Linux has no acquired a lock, LHP does not occur but CPU time is consumed by TOPPERS. With our new technique, the performance isolation is established without static CPU partitioning. The result shows that our approach utilizes the CPU resources efficiently if the load of RTOS is less than 50%. And we choose this as the threshold triggering migration. Therefore, when the CPU utilization of RTOS becomes more than 50%, it is better to allocate one no shared CPU to RTOS.

5. CONCLUSION

The result described in this paper showed that our approach can satisfy both the real-time responsiveness of RTOS and the high throughput of GPOS.

6. REFERENCES
