# GLOBAL SCHEDULING FOR THE EMBEDDED VIRTUALIZATION SYSTEM IN THE MULTI-CORE PLATFORM

Tsung-Han Lin, Hitoshi Mitake, Yuki Kinebuchi and Tatsuo Nakajima
*Department of Computer Science, Waseda University, Tokyo, Japan*

Keywords: Virtualization, Scheduling, Embedded System, Operating System, Real-time System.

Abstract: In this paper, we are trying to address the global scheduling issues in the embedded multi-core virtualization system. Virtualization system is widely being used in the embedded device nowadays, especially with the multi-core platform also shown in the embedded system world. Global scheduling in the virtualization system makes the real-time tasks in the GPOS (General Purpose Operating System) to have chance to be scheduled against the tasks in the RTOS (Real-Time Operating System). We proposed using GPOS/RTOS mix scheduling and VCPU migration to deal with global scheduling problem. Meanwhile, some issues like interrupt latency in the GPOS and the priority inversion problem in the RTOS are also discovered while doing this research. We would like to make more detail investigation about this issues to improve the quality of global scheduling so we can construct a more ideal virtualization environment.

## 1 INTRODUCTION

Virtualization technique is originally used in the data center or desktop environment to have multiple operating systems running on a single platform. System consolidation cost can also be reduced by using this technique. Famous works like VMWare[1] and VirtulBox[2] are all very mature systems.

Recently, it is been also widely used in the embedded system platform ,such as L4 microkernel (Klein et al., 2009). This development trend is also accelerated by the emerging of the multi-core platform in the embedded system because extra computing power can be used.

Installing a RTOS alongside with a GPOS in the embedded virtualization system is a common setup because some real-time jobs like radio transmission can use RTOS to achieve real-time guarantee and some normal desktop usage applications can leave to the GPOS. But with the growth of real-time handling ability in the GPOS, some tasks in it can also compete the resources with those in the RTOS. For example, with the addition of the rt-patch[3], Linux now can handle soft real-time jobs quite well. So even in the GPOS/RTOS co-existing environment, users may want to install some soft real-time jobs in the GPOS such as audio/video applications that may need to use some GPOS-provided features like GUI control and those applications and GUI supports are hard to be adopted to the RTOS. However, lots of embedded virtualization works tend to give the RTOS' tasks the highest priorities, so these tasks can preempt the executions of tasks in the GPOS, even if they may have more urgent real-time handling requirements than those in the RTOS. This is also because the virtualization layer has no way to know much about the inner information of the GPOS, so they cannot tell who is urgent or not. The real-time performance in the GPOS will therefore suffer serious degradation.

Motivated by this fact, we are developing a global scheduling framework to try to improve the real-time responsiveness in the GPOS in our multi-core virtualization work named SPUMONE. We proposed a global scheduling scheme to solve the real-time performance lost problem alongside with VCPU[4] migration to better utilize the resources. A new scheduling unit called *rt-VCPU* is being introduced in our work, and is trying to give the virtualization layer more clues about the scheduling information. During this investigation, we are facing some issues in the process of development, such as interrupt latency and real-time guarantee. More details of those issues are given in the following sections.

[1] VMware, Inc., http://www.vmware.com
[2] Oracle Corporation, http://www.virtualbox.org
[3] https://rt.wiki.kernel.org/
[4] Virtual CPU, which is the basic scheduling unit of the virtualization layer.


Lin T., Mitake H., Kinebuchi Y. and Nakajima T..
GLOBAL SCHEDULING FOR THE EMBEDDED VIRTUALIZATION SYSTEM IN THE MULTI-CORE PLATFORM.
DOI: 10.5220/0003906203780382
In *Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems* (PECCS-2012), pages 378-382
ISBN: 978-989-8565-00-6

The rest of the paper is organized as follows. A previous virtualization work will be introduced in the Section 2. Section 3 is talking about the proposed global scheduling approaches in the virtualization system. Some issues when dealing with global scheduling is discussed in the Section 4. Some related works and conclusion are given in the Section 5 and 6 respectively.

# 2 VIRTUALIZATION SYSTEM: SPUMONE

In our previous project, we have developed a virtualization system named SPUMONE for the embedded multi-core platform and is also shown in the Figure 1. It is a very small virtualization layer in which it only handles the VCPU scheduling and dispatch the interrupt to the proper guest OSes. In the multi-core version of SPUMONE, each CPU core has its own virtualization layer instance, as one can see in the Figure 1. This gives the overall system more scalability. More detail information about SPUMONE can be found in our previous work (Kinebuchi et al., 2009).

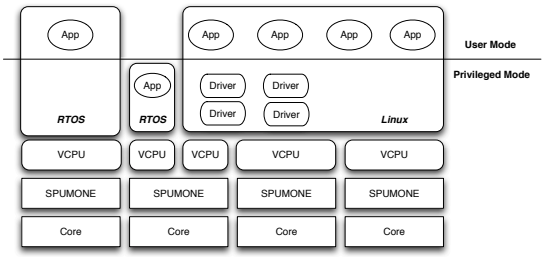


Figure 1: SPUMONE overall architecture in the multi-core platform.

# 3 GLOBAL SCHEDULING

## 3.1 Background of Global Scheduling

Here we would like to use global scheduling and VCPU migration techniques to improve the overall system real-time responsiveness, especially for the GPOS. Global scheduling in the virtualization environment means scheduling the VCPUs of the RTOS and the VCPUs of the GPOS at the same time, and using some criteria to control the scheduling behavior. In most of RTOS/GPOS co-existing embedded virtualization works, SPUMONE included, the tasks in the RTOS are always given the highest priorities in the virtualization platform scheduling. Therefore, the VCPUs that execute those tasks on behalf of RTOSes can always preempt the execution of VCPUs of GPOSes. This is because that traditionally when designers want to have this kind of RTOS/GPOS virtualization environment setup, they want to place all real-time urgent tasks in the RTOS and not bothered by the tasks in the GPOS, so the tasks in the RTOS can run without any restriction.

Recently, however, with the improvement of real-time handling in the GPOS, Linux for example, users may also try to install some soft real-time required tasks in the GPOS and use some utilities provided by the GPOS, such as *chrt* command in Linux, to manipulate the real-time attributes of the tasks and hope they can be executed even more urgent than some tasks in the RTOS. For instance, some streaming or graphic handling applications need GUI (Graphical User Interface) to control the behavior of them; meanwhile, they also require real-time responsiveness. So it is better for those users to install them in the GPOS, because the GPOS handles the GUI applications better than that of the RTOS, and hope the real-time handling ability of the GPOS can achieve what they desire to have.

## 3.2 Global Scheduling in SPUMONE

### 3.2.1 Basic Setup in SPUMONE

Trying to further investigate the global scheduling problem, we plan to use our SPUMONE work to examine the issue.

Default setting in the SPUMONE, which is also common to most embedded virtualization works, provides each guest OS some VCPUs, and SPUMONE simply schedules these VCPUs while giving RTOS' VCPUs the highest priorities to run and schedule GPOS' VCPUs using round-robin manner. As one can see in this kind of setting, RTOS' tasks always run before or preempt the execution of the tasks in the GPOS. The real-time tasks in the GPOS will therefore suffer serious performance lost and lead to considerable deadline miss problems. What causes the problem is that SPUMONE or even other virtualization works cannot identify the differences among a common task in the GPOS, a real-time task in the GPOS and a task in the RTOS. So SPUMONE treats all tasks in the GPOS as the same priority, and the real-time tasks in it have therefore no chance to be scheduled more promptly.

In order to address this phenomenon, here we introduce a new scheduling unit for SPUMONE in which the real-time tasks in the GPOS can use this unit to let themselves have chances to be scheduled against the tasks in the RTOS. A new type VCPU

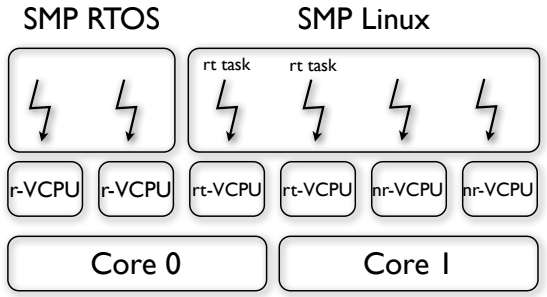


Figure 2: Using different VCPUs to schedule.

called *rt-VCPU* is being used for this purpose in the SPUMONE. This *rt-VCPU* can give SPUMONE a *hint* that this VCPU may contain some important jobs and has to be run in some degrees of urgency, so SPUMONE can alter the scheduling algorithm to reflect the setting change. For example, in the Figure 2 [5], some tasks which need real-time responsiveness in the GPOS are bound to some *rt-VCPUs* and when they are on the same physical core with *r-VCPU*, some scheduling algorithm can be applied to improve the real-time responsiveness of them.

By using a specific VCPU, *rt-VCPU* in this case, to handle specific tasks, this also means that we have to change those tasks' CPU affinities either by users or we have to modify the GPOS in order to get the information of who is urgent and has to be bound to that specific VCPU. Here we simply ask users to manually set the CPU-affinities of their desired tasks to a certain VCPU. It is also reasonable because users who want their tasks to run urgently by manually setting the real-time attribute of them can also easily set the CPU-affinities according to designer-provided system manual.

#### 3.2.2 VCPU Operation, Scheduling and Migration

Scheduling units of SPUMONE are now divided into three kinds and we have to define the relations among them and how they interact with each others.

We categorize possible scheduling situations into three kinds and try to apply some scheduling strategies to further analyze the effect of them:

1. *r-VCPU*s and *nr-VCPU*s share a PCPU[6].
2. *rt-VCPU*s and *nr-VCPU*s share a PCPU.
3. *rt-VCPU*s and *r-VCPU*s share a PCPU.

In the first situation, it is obvious that r-VCPU is the highest priority task for the virtualization layer, if there is no any *rt-VCPU* involved which will be discussed in the situation 3 later. So an *r-VCPU* can preempt the execution of *nr-VCPU* anytime it wants and therefore the deadline of those tasks on the *r-VCPU* can be guaranteed. Situation 2 is the same as that of situation 1. Because *rt-VCPU* contains real-time jobs which are critical than normal jobs in the same system, let the execution of *rt-VCPU* prior to that of *nr-VCPU* is reasonable.

But both situation 1 and 2 have a common issue which we still need to do more investigations in the future and it will be discussed in more detail in the next section.

Situation 3 is the main operation we need to deal with. Because different systems use different notations or approaches to assign real-time priority, we plan to use the method similar in the (Kinebuchi et al., 2008) to do the global scheduling of *rt-VCPU* and *r-VCPU*. Winner of this scheduling strategy will keep running its normal execution, but the loser cannot afford idle waiting because either an *r-VCPU* or an *rt-VCPU* is still a critical real-time handling job after all, otherwise it would make no difference between a real-time task and a non-real-time one. Luckily, in the multi-core platform, there are other available computing resources, PCPUs, for designer to use. Therefore, we will try to *migrate* the loser VCPU in this situation 3 to other PCPUs and schedule against the VCPUs on that target PCPUs using the previous mentioned strategies, situation 1 and situation 2. However, potential scheduling problem in the RTOS, especially for SMP RTOS, could also cause serious performance degradation. We will, again, discuss this issue in the following section.

## 4 DISCUSSION: GLOBAL SCHEDULING ISSUES

Further discussions about some issues while dealing with global scheduling in the embedded multi-core virtualization design will be given in this section.

In the situation 1 and 2 mentioned in the previous section, it is not the problem for most non-real-time tasks to be preempted by the real-time tasks, but this is not the case for the system interrupt handling tasks in the GPOS. Albeit users can selectively change their jobs' real-time priority, but they cannot change that of interrupt handling tasks. So there must be some ways for the system to help itself not to affect the execution and real-time responsiveness of them. For example, in the Figure 3, two tasks in the Linux use two different VCPUs, one uses *rt-VCPU* and the other uses

[5]Now a normal VCPU for the GPOS which handles non-real-time tasks is called *nr-CPU* and a normal VCPU for the RTOS is called *r-VCPU* in order to ease of discussion later.

[6]Physical CPU or Physical Core.

*nr-VCPU* while the one that uses *nr-VCPU* is responsible for handling interrupt. One possible serious performance degradation would be that if a *nr-VCPU* is to handle a frequently arrival interrupt source like networking device. If these interrupt requests are ignored too often because of VCPU scheduling strategy, overall system will also suffer considerable performance lost.

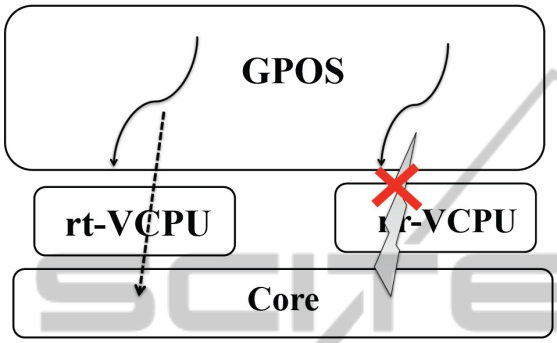


Figure 3: Interrupt handling delayed by rt-VCPU preemption.

We are trying to mitigate these kind of situations by using the following three kinds of scheduling approaches for the GPOS VCPUs and measuring the effects of them:

- Tasks using *rt-VCPU* still can preempt the execution of interrupt-handling *nr-VCPU*. But we are attempting to use a approach called *Interrupt Coalescence* to minimize the effect of preemption. It is an approach that bundles multiple interrupt requests and interrupts the system only once. It can also effectively reduce some interrupt handling overhead. Therefore, interrupt-handling *nr-VCPU* can use this scheme to deal with highly arrival interrupt request. *Interrupt Coalescence* itself, however, still have some other issues which are beyond the scope of this paper, and we are investigating some variations of it and trying to find a proper way to combine it with *rt-VCPU* scheduling.
- Because in portable or mobile devices, networking operations may play a very important role on them, it is better to handle them first. An alternative here is we are considering to let interrupt to be handled its *top-half* first for the *nr-VCPU* and remaining *bottom-half* of interrupt handling is deferred and scheduled against *rt-VCPU* and the loser of scheduling is also migrated. In the Linux, for example, top-half of interrupt handling only does some important operations like setting registers and quick response to the interrupt source. Some other not-so-important handling parts can therefore be deferred later. The real-time responsiveness of those tasks may therefore not suffer too many PCPU resource waiting.
- The third alternative is to simple change the second approach mentioned above by letting *rt-VCPU* always be scheduled before bottom-half handling VCPU. So the resulting priority order will be top-half interrupt handling first, then the *rt-VCPU* and the bottom-half of interrupt handling comes the last.

Meanwhile, there are some other issues when using our proposed virtualization setup.

- What is the best ratio of *rt-VCPU*/*nr-VCPU* combination a GPOS should have? If in a 4-core SMP Linux, for example, only one *rt-VCPU* is used, all the real-time loading will be put on this *rt-VCPU*. The real-time responsiveness of all these tasks will be affected. On the other hand, if too many *rt-VCPU*s are introduced into the system, it will make no difference between normal jobs and urgent jobs because users can assign all he wants to all available *rt-VCPU*s. So the ratio has to be carefully assigned and evaluated.
- There must be a situation in which two *r-VCPU*s in the RTOS are sharing a data, but the *rt-VCPU* won the scheduling battle. So one of the *r-VCPU*s has to be migrated to another PCPU if the scheduling strategy mentioned above is used here. *Priority Inversion* may therefore occurs. Hence, while dealing with SMP RTOS, the use of migration also has to be taken care of.

Using global scheduling and setting proposed in this paper in the embedded multi-core virtualization environment also has to consider those issues as well.

## 5 RELATED WORK

In most virtualization works using VCPU migration, they focused on the subjects other than improving the real-time responsiveness in the GPOS such as synchronization problem or resource utilization.

Because the virtualization layer doesn't know much about the inner details of the guest systems, manipulating or migration VCPUs could cause some serious synchronization problems, lock holder preemption for example. Authors (Mitake et al., 2011) addressed this problem by dynamically migrating VCPUs according to exception events. A balance scheduling algorithm is proposed (Sukwong and Kim, 2011) in which they dynamically measured the load,

checking whether the capacity is full or not in a runqueue, of a PCPU. Then dynamical VCPU assignment is performed to separate two related VCPUs to different runqueues to further improve the system performance because these two VCPUs might be sharing the same data structure.

For the time being, there is not a lot of works found which is relating to our GPOS-real-time improvement VCPU scheduling work in the multi-core platform. A similar global scheduling work (Kinebuchi et al., 2008) used mix prioirties from both RTOS and GPOS to calculate global priority values that the virtualization can use to globally schedule the tasks in both guest systems to improve the overall real-time responsiveness of the system. Its focus, however, is on the single-core platform and no migration-like operations are performed when resources conflict or synchronization problem occur.



## 6 CONCLUSIONS

In this paper, we focus on dealing with the global scheduling in the embedded multi-core virtualization system. We proposed some scheduling strategies by introducing a new scheduling unit called *rt-VCPU* to give virtualization layer a hint and also use VCPU migration to balance the loading and real-time responsiveness of the overall system. Some issues that we would like to address in more detail in this global scheduling work are also given such as interrupt handling latency in the GPOS and possible priority inversion in the SMP RTOS. These are important and commonly seen issues in the embedded virtualization system, so we have to consider them carefully and find out the best system consolidation approach.

## REFERENCES

Kinebuchi, Y., Morita, T., Makijima, K., Sugaya, M., and Nakajima, T. (2009). Constructing a multi-os platform with minimal engineering cost. In *Analysis, Architectures and Modelling of Embedded Systems: Third IFIP TC 10 International Embedded Systems Symposium*, pages 195–206.

Kinebuchi, Y., Sugaya, M., Oikawa, S., and Nakajima, T. (2008). Task grain scheduling for hypervisor-based embedded system. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 190–197, Washington, DC, USA. IEEE Computer Society.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. (2009). sel4: formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 207–220, New York, NY, USA. ACM.

Mitake, H., Kinebuchi, Y., Courbot, A., and Nakajima, T. (2011). Coexisting real-time os and general purpose os on an embedded virtualization layer for a multi-core processor. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 629–630, New York, NY, USA. ACM.

Sukwong, O. and Kim, H. S. (2011). Is co-scheduling too expensive for smp vms? In *EuroSys*, pages 257–272.