プロセスのメモリオブジェクトを再利用する OS カーネルアップデート手法

寺田 献[†] 山田浩史[†]

Operating system (OS) kernel updates are a part of daily life in contemporary computer systems including high-end servers in data centers as well as desktop PCs and smartphones. Kernel updates are announced frequently because OS kernels are still being developed to improve their performance, add new functionality, and repair security vulnerabilities. Although announced updates should be applied as soon as possible since they often include critical vulnerability fixes, kernel updates usually require an OS reboot that involves the restart of not only the kernel but also all of the software, causing downtime that can disrupt software services running on the kernel. This negative impact is a big hurdle to readily conduct kernel updates.

Various researchers have tackled this issue so far. The dynamic kernel update (DKU) is a representative approach to applying patches to the kernels at runtime[1),3)]. Since DKU does not require an OS reboot in the kernel update, its downtime is almost zero. But, DKU applicability is inherently limited. Some DKU systems are difficult to update data structure types and non-quiescent kernel functions that are always on the call stack of kernel threads. Also, DKU sometimes requires the development of special patches from the original ones, which means OS kernel knowledge at the source-code level is necessary to use. This is non trivial since recent kernels are more complex and some are closed-source and/or proprietary.

The reboot-based kernel update is an attractive approach to efficiently manage an OS reboot for the update. The use of the process migration techniques moves the processes to another machine until the OS reboot completes[2),4)]. Another technique leverages virtual machine technology to generate a virtual machine memory image that is after the OS reboot[6)]. The other one is to preserve running process states across OS reboots[5)]. These approaches can deal with more types of kernel updates since the conventional kernel update procedure is conducted, but the reboot-based kernel update is not a perfect solution. The process migration approaches are resource-consuming; these requires redundant resources that include the same memory size of the updated machine. The other approaches cause non-negligible downtime; we have to restart processes or wait for the kernel boot.

Our goal is to overcome the drawbacks of the reboot-based kernel update. This paper presents *Nap*, a reboot-based kernel update approach whose downtime is shorter and resource consumption is much less than the conventional ones. Nap launches the newer kernel in the background on the same physical machine, and forces the kernel to inherit the running states of the older kernel. Nap makes downtime as short as possible by keeping the running states of processes and switching the newer kernel just after it becomes ready. In addition, Nap requires much less memory by efficiently moving the running process states from the older to the newer kernel.

Nap is a thin layer running between hardware and the OS kernel like a hypervisor. Our approach orchestrates Nap and the OS kernel. To boot the newer kernel in the background, Nap leverages memory and CPU virtualization. After a kernel has been patched, Nap boots the patched kernel while the older kernel is running. The kernel starts to inherit the running states of the older one after its initialization. To inherit the running process states, Nap receives *essential contexts* from the older kernel and passes them to the newer one. An essential context is a kernel object related to a process state necessary to restart the process, such as PID, CPU registers, memory maps, and so on. The newer kernel recreates processes based on the essential contexts. The user-level memory regions in the older kernel are reused. The newer kernel is resumed just after receiving all the essential contexts. At the same time, the attachment and detachment

† 東京農工大学
Tokyo University of Agriculture and Technology

of the devices in the newer and older kernel are done in parallel. The newer kernel attaches the detached devices one after another.

We are implementing a prototype of Nap on Linux 2.6.39.4 and Xen 4.5.0. The domU is attached to NIC and storages through PCI passthrough. We conducted a preliminary experiment with the prototype. We measured the save/restore time of the running processes whose memory size is different (16, 64, 256, 1024, and 4096 MB). The result shows that save/restore times are increased as the memory size is more. The save times are less than 500 ms in all the cases. The restore times except for the 4096 MB case are less than 600 msec. The restore time in the 4096 MB case is more than 1 second. Since our prototype is premature, we expect that this time becomes much shorter.

参 考 文 献

1) J. Arnold and M. F. Kaashoek. Ksplice: Automatic Rebootless Kernel Updates. In *Proc. of the 4th ACM European Conference on Computer Systems (EuroSys '09)*, pages 187–198, 2009.
2) D. E. Lowell, Y. Saito, and E. J. Samberg. Devirtualizable Virtual Machines Enabling General, Single-Node, Online Maintenance. In *Proc. of the 11th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '04)*, pages 211–223, 2004.
3) K. Makris and K. D. Ryu. Dynamic and Adaptive Updates of Non-Quiescent Subsystems in Commodity Operating System Kernels. In *Proc. of the 2nd ACM European Conference on Computer Systems (EuroSys '07)*, pages 327–340, Mar. 2007.
4) S. Potter and J. Nieh. Reducing downtime due to system maintenance and upgrades. In *Proceedings of the 19th USENIX Large Installation System Administration Conference (LISA '05)*, pages 47–62, Dec. 2005.
5) M.Siniavine and A.Goel. Seamless Kernel Updates. In *Proc. of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '13)*, pages 1–12, 2013.
6) H. Yamada and K. Kono. Traveling Forward in Time to Newer Operating Systems using ShadowReboot. In *Proc. of the 9th ACM International Conference on Virtual Execution Environments (VEE '13)*, pages 121–130, 2013.