The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors

Authors: Gernot Heiser Ben Leslie

1 Summary

Hypervisors aim to mimic a real system in the most efficient way possible. A microkernels aim to have a minimal code size, by keeping the kernel to a bare minimum and moving most features to user space. The paper aims to reconcile the difference in objectives of the two and presents a solution (named OKL4 microvisor) which tries to borrow the good aspects of both. OKL4 microvisor tries to expose the virtualization primitives of a hypervisor (vCPU,vMMU,etc.) while keeping the codebase to a minimum (9.8 KLOC which compiles to about 35KiB). The micorvisor requires paravirtualized guests and is intended to be used on embedded processors (Presently only ARM processors are supported, and virtualization extensions are not required).

2 Detailed notes

Objectives of a microvisor

- Minimal codebase (and hence minimal TCB[1]), most features pushed to user space (separation of policy and mechanisms).
- General enough to support contruction of arbitrary systems on top.

Since in a micorvisor most features reside in user space hence a very fast IPC is needed, which may prove to be a bottleneck if not implemented efficiently.

Objectives of a hypervisor

A hypervisor tries to provide an efficient and isolated view of a real machine. Code size is not a major concern. There has been some recent effort to move some services provided by hyperisor to user space, for eg. Driver domains or DOM 0 in Xen, which enables the reuse of native drivers.

Differences with respect to individual subsystems

- CPU : Hypervisors expose vCPU to guests, while microkernels abstract execution time as threads.

[1]Trusted computing base

- Memory : Hypervisors provide virtual MMU by virtualizing the page tables or by using shadow page tables. Microkernels virtualize memory by providing separate address spaces.
- I/O : The drivers for the device interfaces exported by hypervisor generally reside in the VMM itself, whereas in microkernels the device drivers exist as separate user space processes which communicates with rest of the system via IPCs. Xen's approach for I/O virtualization is similar to that microkernels'.
- Communication: Hypervisors provide a virtual network for inter-VM communication. Microkernels rely on IPC for communication between execution entities.

OKL4 microvisor provides hypervisor like interfaces while keeping the codebase to a bare minimum. It abstracts CPUs as vCPUs, MMU as vMMU. I/O abstraction consists of memory mapped virtual device registers and virtual interrupts. Communication is abstracted as vIRQs and channels (which are generic bidirectional FIFO buffers on which different communication mechanisms can be built, including TCP).

Performance evaluation

The author do not do a thorough analysis of OKL4 microvisor, instead refer to lmbench scores reported by one of their customers. The performance overhead reported by them is quite significant in some system calls like socket and null syscall. Few operations, like open/close showed a negative overhead, which authors attribute to changed patterns of cache conflict, and so these results should not be worried about. Performance overhead as reported by netperf was low ($< 5\%$).

3 Positive points

1. Code base is minimal, thus more secure as there is lesser chance of bugs.
2. Providing memory mapped virtual device registers may allow native drivers of guest vm to work.

4 Negative points

1. The paper was published in 2010, by that time there were already several hypervisor implementations for ARM, so a comparison with them could have been done.
2. The data for performance evaluation that the authors present in table 1 is borrowed from a source which they don't cite thus authenticity of stats cannot be established, moreover they could have easily obtained these metrics on their own.

3. Exposing hypervisor like interfaces using a microkernel might have required significant changes to microkernel code, which they should have mentioned, so as to throw some light on the way these primitives interact with the underlying microvisor kernel.

4. A macro benchmark of individual subsystems like vMMU, vCPU, etc. could have been done by the authors to better understand how well these subsystems perform when implemented on top of a microkernel.

5. Benchmarking application performance would have given a test of real world usability of such a system, which the authors fail to give.

6. Guaranteeing isolation when device registers are memory mapped is difficult as different guests might try to wrote different values to it, thus arbitrarily changing configuration of devices between context switches. Also such a mechanism will require intricate knowledge of device.

5 Future extensions

For the proposed hypervisor to work efficiently it must address challenges specific to the processor on which it is supposed to run and must also leverage unique features provided by it. For eg. ARM processors have a wide variety of configuration of hardware peripherals, and thus any such software must ensure that it is compatible with all of them. OKL4 microvisor, at the time of publishing of the paper only supported ARM, thus an evaluation in the context of ARM might help to better optimize the hypervisor.

Instead of providing memory mapped device registers to guests, driver domains, as in case of Xen could be used, by running an instance of Linux, in one of the VMs. This would significantly increase portability as Linux is supported on a wide variety of platforms.