Notes for the Development of CertiKOS Prototype

October 30, 2011

Contents

1 About CertiKOS

CertiKOS (Certified Kit Operating System) may refer to our project: applying new advances in certified software [6] to the development of a novel OS kernel. For more information about the whole project, please check the project tech report [7]. In this document, we mostly use CertiKOS to refer to the prototype of the OS kernel.

For the long term, our CertiKOS is supposed to offer features as follows: safe and application-specific extensibility, provable security properties with information flow control, and accountability and recovery from hardware or application failures. The implementation will have different stages to provide these features. At current stage, CertiKOS is trying to provide some demonstration features (Section 2). This document will be updated later with the progress of implementation.

2 CertiKOS at Current Stage

2.1 Overview

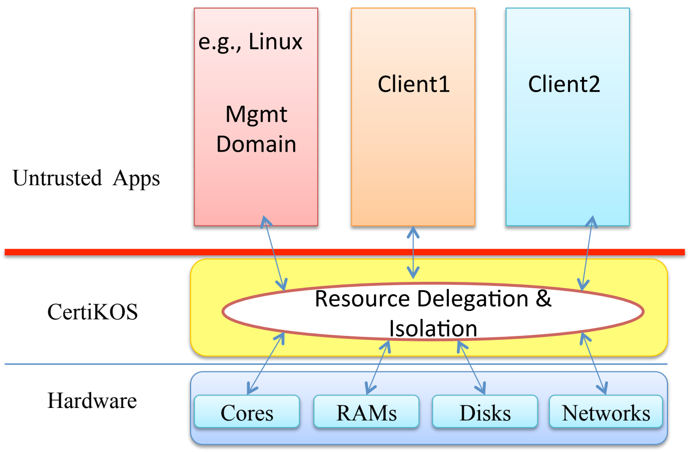

Figure 1: Overview of CertiKOS at current stage

The overview of current CertiKOS is shown in Figure 1. It is trying to provide demonstration features as follows:

- Abstract resources for cloud computing, including CPU cores, RAMs, Disk, and Networks
- Separate the resource management from usage, move resource management facilities up to the untrusted application layer

- Support legacy OS as the management software
- Provide isolated execution environment to protect the execution of mission-critical (or security sensitive) operations, according to the request from the application layer processes.

2.2 Typical Usages

We use several usage cases to specifically explain these above features.

2.2.1 Resource Management

Manage software runs at the untrusted application layer and it provides interfaces for administrator to manage the allocation of resources to each application. As shown in Figure 2, manage software instructs CertiKOS to allocate or revoke resources for applications. Each application will be authorized to access its own resources, including memory space, cpu cores, disk and network.

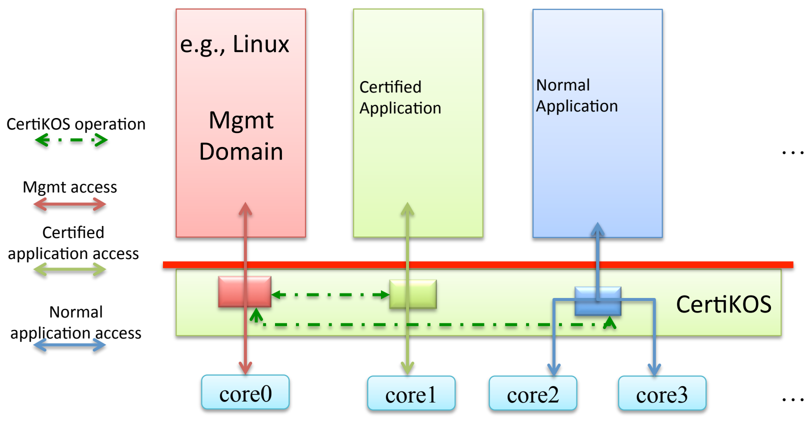

Figure 2: Resource Management

2.2.2 Isolation

Protect applications from each other, including management software. The management software and other applications are equally treated as untrusted applications. As shown in Figure 3, CertiKOS separately allocates memory spaces for them and they can only access their own space. The management software can manage the ownership of these resources, but it can not access these resources. Applications are protected from each other and accessing among each other are prohibited.

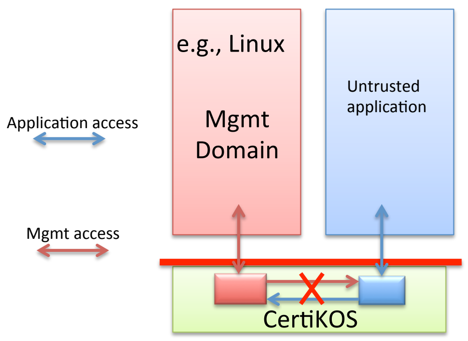

Figure 3: Isolation among untrusted applications

2.2.3 Execute Certified Applications

As shown in Figure 4, certified applications can run on top of CertiKOS as normal applications and it will be protected from the management software and other applications.

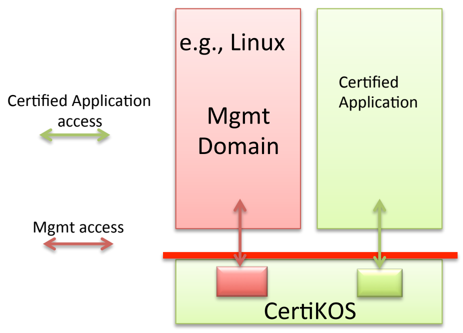

Figure 4: Certified Application on CertiKOS

2.2.4 Protect Security Operations in Applications

Many applications involve security related operations, in which sensitive information might be revealed, such as private key, credit card number, social security number, etc. These applications may run atop of untrusted OS, and these security information can not be protected from compromised OS.

With Hardware-based Virtualization support, CertiKOS can provides isolated execution environment for these security related operations. As shown in Figure 5, The legacy OS and applications can run on top of CertiKOS. When the application tries to execute security sensitive operations, CertiKOS will setup a separated space for the security code block to execute. After finishing execution in the isolated space, CertiKOS will return the result to the application and it will continue execution in the legacy OS.

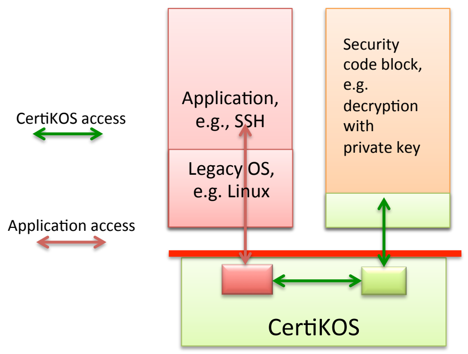

Figure 5: Isolated Execution for Security Operations of Applications

3 CertiKOS Implementation

3.1 CertiKOS Framework

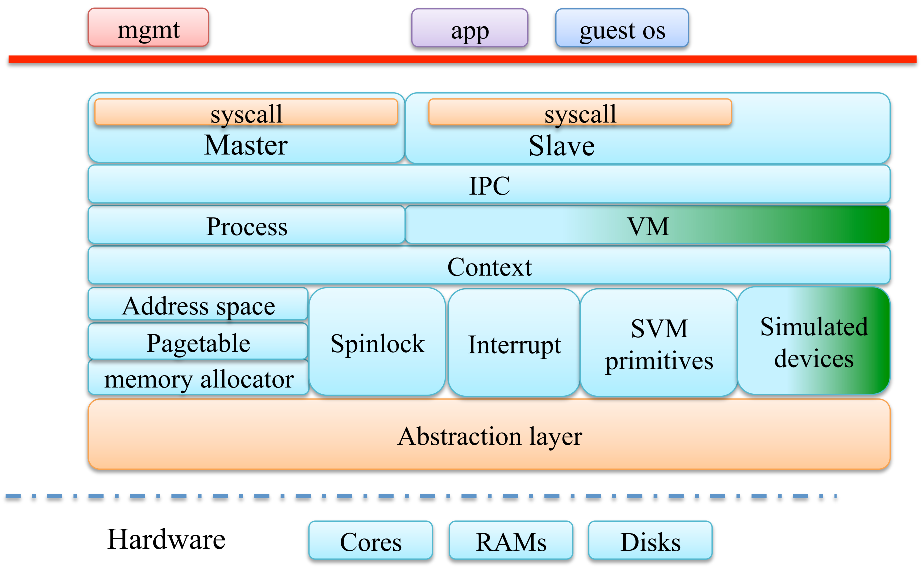

Figure 6: CertiKOS Framework

Current version of CertiKOS is organized as in Figure 6. Upper layers depend on(invoke) lower layers components. Ported from PIOS [1], CertiKOS can now support the management of normal processes. To support legacy OS as the management software, CertiKOS will leverage the Hardware-based Virtualization. The current implementation work is mostly focusing on SVM based virtualization support for legacy OS. These components with mixed color of Green and Blue denotes these undergoing implementation.

3.2 Memory Management

3.2.1 address mode for host

3.2.2 memory space for guest os

[TOFILL]

3.3 CPU Core Management

CPU core scheduling. [TOFILL]

3.4 Hardware-based Virtualization

AMD SVM provides the processor instruction support to enhance the implementation of Virtual Machine Monitor. These key features for virtualization support in CertiKOS include: Nested Paging, Interception. For more specific info, please check specification [3, 2].

3.4.1 Virtual Machine Control

VMCB (Virtual Machine Control Block) is the most important data structure in SVM. It contains all information for the guest to run. The VMM controls the execution, resumption and interception of the guest os by setting the VMCB accordingly. For more details, please refer to the SVM specification.

The basic execution flow of AMD SVM based VMM is shown as in Figure 7.When a VM is scheduled to run, its VMCB address will be transferred to VMRUN instruction. Then the VMRUN will continue the execution of the VM, until exceptions or interception happen in the VM. VM exit Handling will specifically deal with the exit events according to its exit code. After the exit handling, the VM will be ready for scheduling again.

3.4.2 Nested Paging

With Nested Paging in SVM, VMM does not have to maintain a shadow page table for the guest OS.

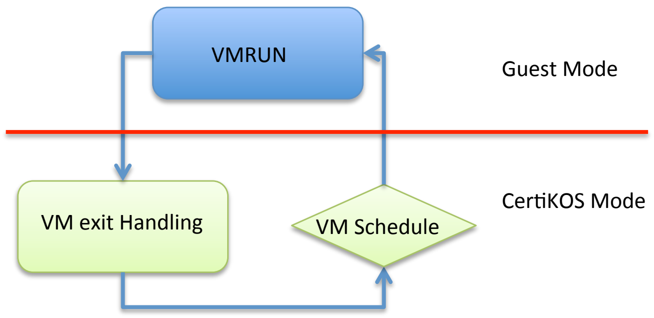

Figure 7: Execution Flow of VMs

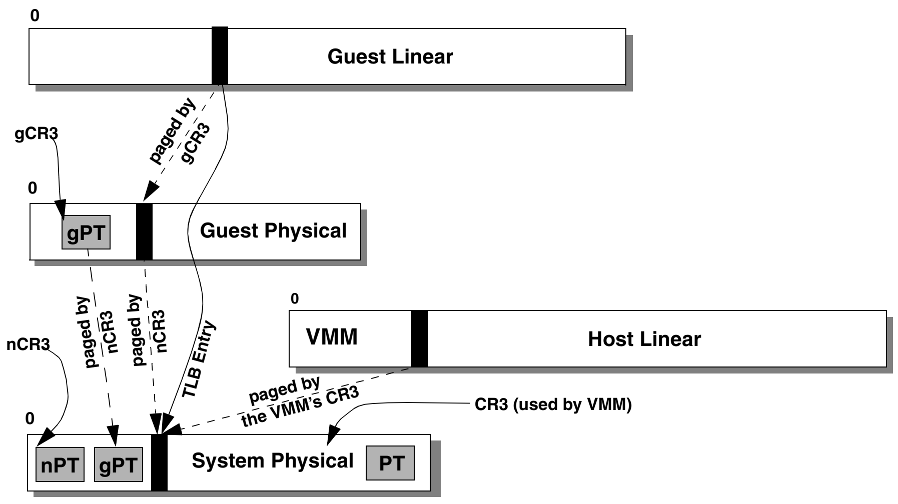

Figure 8: Address Translating with Nested Paging

With nested paging enabled, two levels of address translation are applied; Shown as in Figure 8.

- Both guest and host levels have their own copy of CR3, referred to as gCR3 and nCR3, respectively.
- Guest page tables (gPT) map guest linear addresses to guest physical addresses. The guest page tables are in guest physical memory, and are pointed to by gCR3.
- Nested page tables (nPT) map guest physical addresses to system physical addresses. The nested page tables are in system physical memory, and are pointed to by nCR3.

- The most-recently used translations from guest linear to system physical address are cached in the TLB and used on subsequent guest accesses.

Usage of Nested Paging First create a layered page tables according to the addressing mode. Then set the page table according to the memory space setting for the guest OS. For example, mapping some physical memory region for the guest os, including some parts of the lower 1MB memory space. Then configure the VMCB.

3.4.3 Interception

By setting the control data structure in VMCB, the running guest with VMRUN can be intercepted when its execution satisfies the interception condition. CertiKOS uses interception to handle events of guest OS, including IO handling, Interrupt handling. Following interceptions are currently concerned in our implementation:

- IO
- Interrupts
- Debug
- Page Fault
- Other exceptions

3.5 Device Simulation for Supporting Legacy OS

To support the co-exist of CertiKOS and up layer legacy OS, it requires virtualization for critical devices. Previous implementation of CertiKOS simply exposes all hardwares to the legacy OS and it will change the settings of some critical devices, like PIC chips. This results the failure after switching back from legacy OS to CertiKOS.

We are now working at two directions: Configure and test the legacy OS to identify the minimum set of hardware devices, and enable the booting of ttyLinux with SVM on physical platform; Implement the simulation of these necessary devices one by one.

According to our current observation, following devices need to be specifically concerned: DMA, PIC, PCI, APIC, IOAPIC.

3.5.1 Test CertiKOS on Physical Platform

[TODO] by Haozhong

3.5.2 IO Port Interception for Devices Simulation

Some devices are accessed via IO ports. Their simulations can be implemented with I/O interceptions.

List of critical devices By enabling the interception of all IO accessing, we noticed that following IO ports are accessed by the guest domain when it boots ttyLinux with Grub:

- DMA: 0x80
- PIC: 0xa0 0x20 0x21
- 0xed
- CGA: 0x3d4 0x3d5
- PCI: 0xcf8 0xcfc
- 0x1020

By Intercepting IO operations on these ports, we can implement the simulation of these devices while handling vm exit events.

3.5.3 IO interception handling for device simulation

For simplicity, we currently introduce IO handling in the CertiKOS kernel, as in Figure 9. When a the guest OS is going to access a simulated device, CertiKOS intercepts the operation. It either sets the simulated data structure with these setting from guest OS, or returns the configuration from these simulated data structure to the guest OS.

For the long term, we will move IO handling up to application layer, as in Figure 10

3.5.4 PIC

We first try to support guest domain with only PIC interrupt mode. The PIC interrupt mode is shown as in Figure 11. From the point view of guest OS, it can only see and access these virtualized 8259 pic chips. So we need to modify the multiprocessor configuration presented to the guest OS. In recent years, these latest hardware platforms have already employed the ACPI specification for providing MP configuration [4].

For old version of MP configuration, MP Floating Pointer Structure is employed to provide MP information to the guest OS. MP Floating Pointer Structure specification is given in [5].

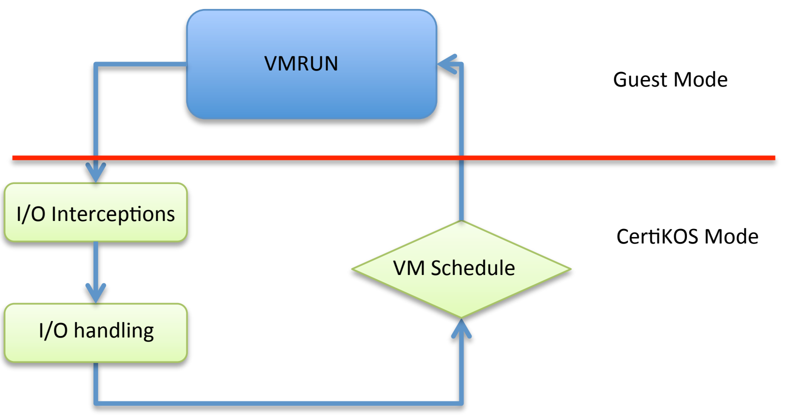

Figure 9: IO handling for VM at kernel mode

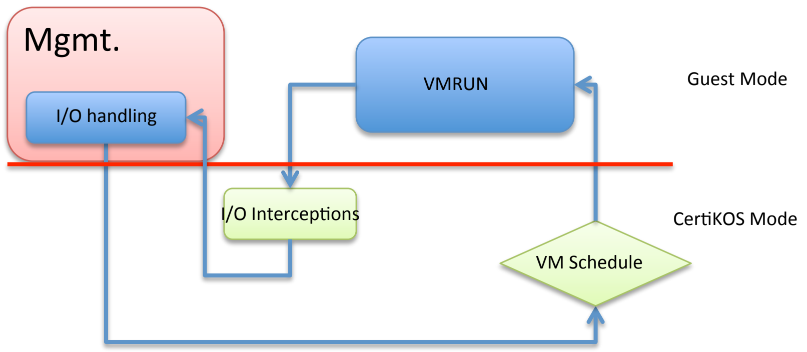

Figure 10: IO handling for VM at guest mode

Fake the hardware configuration for Guest OS First copy the hardware configuration from the lower memory region. Then modify the new copy of configuration to be of single processor with only PIC. Last, modify the Nested Page Table for the Guest OS to include this modified copy of configuration as the booted hardware configuration source.

We have already finished the code for PIC simulation. It will work with interrupt handling (Section 3.5.5) to serve the guest OS. To support guest OS with multi physical cores, I think we need to later implement the simulation of APIC and IOAPIC.

3.5.5 Interrupt Handling

As CertiKOS and Guest OS share some of these physical devices, like keyboard, it requires the CertiKOS to handle some external interrupts for the Guest OS to correctly

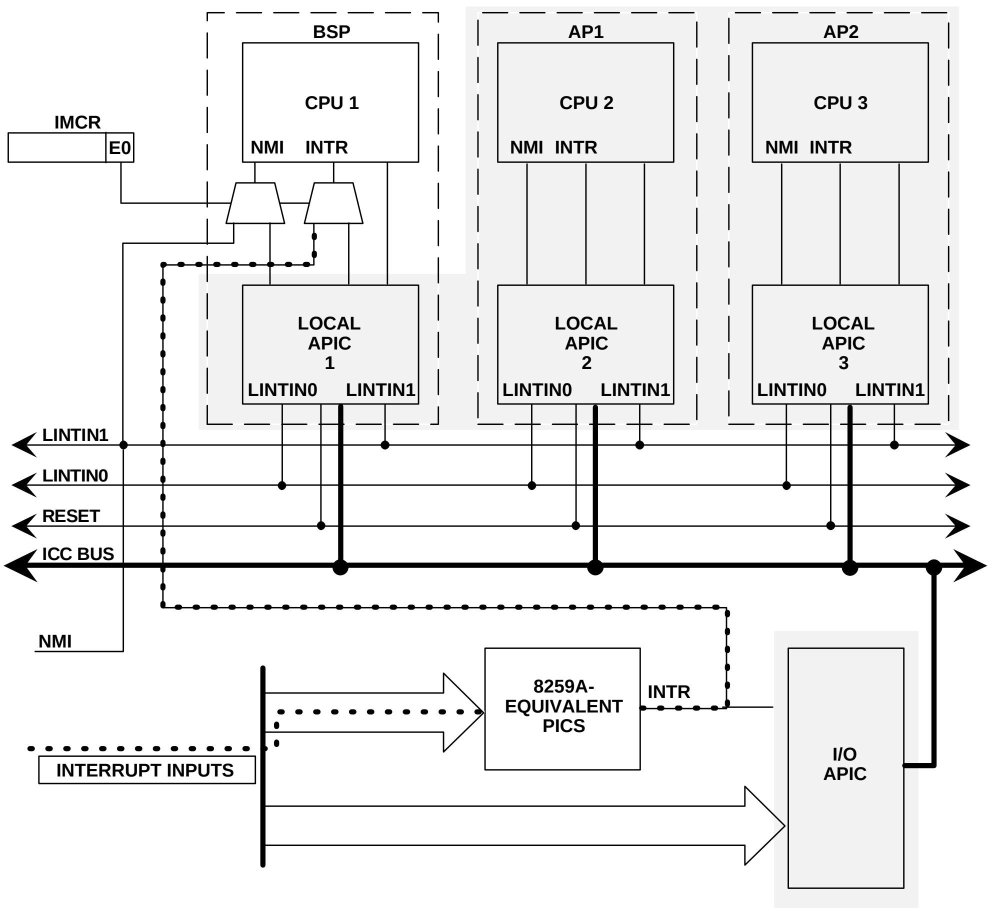

Figure 11: PIC interrupt mode

handle interruption events.

As in Figure 12, CertiKOS intercepts the interrupt events for handling these external interrupts for guest OS. First, CertiKOS will set the interception setting of the guest domain (Step 1 in Figure 12). Once the guest has these specified interrupts, the CertiKOS will intercept the event and return the correct IRQ handler for the guest OS (Step 2 in Figure 12). Then the guest can continue its execution for handling the interrupt.

Now we are implementing and testing the code of interrupt handling for guest OS.

3.5.6 PCI

For simplicity, we are now modifying the MP configuration for the guest OS and make guest think that there is no PCI devices. However, in future, if CertiKOS needs to share PCI devices between the host and guest, we have to implement the simulation

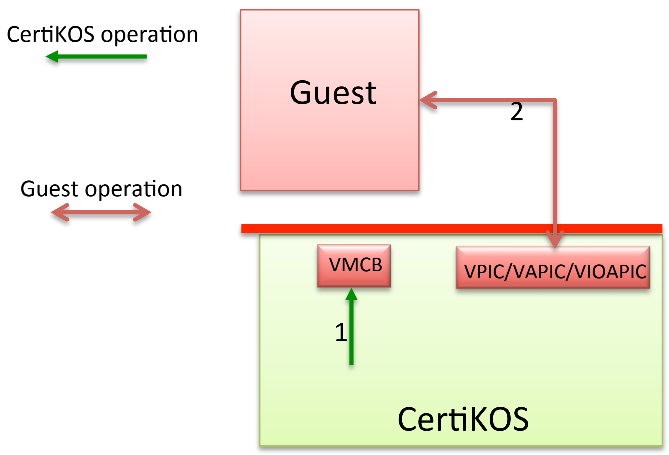

Figure 12: Interrupt handling for guest OS

of PCI bus and schedule the device accessing.

We are now testing the codes and setting for PCI simulation for the Guest.

3.6 Ongoing Tasks

We are now mostly working on finish following tasks :

- boot ttyLinux as guest OS with CertiKOS on Physical Platform
- boot ttyLinux with simulated PIC interrupt mode in CertiKOS
- intercept critical operation in ttyLinux

4 Related Work

4.1 Interrupt simulation in QEMU and KVM

kvm_arch_pre_run

References

[1] Parallel instructional operating system. http://zoo.cs.yale.edu/classes/cs422/pios.

[2] AMD. AMD64 virtualization codenamed "Pacifica" technology — secure virtual machine architecture reference manual. Technical Report Publication Number 33047, Revision 3.01, AMD, May 2005.

[3] AMD. AMD64 Architecture Programmers Manual Volume 2: System Programming. Technical Report Publication No. 24593 , Revision 3.17, AMD, June 2010.

[4] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. *Advanced Configuration and Power Interface Specification 4.0a*, April 2010.

[5] Intel. *MultiProcessor Specification*, 1997.

[6] Zhong Shao. Certified software. *Commun. ACM*, 53:56–66, December 2010.

[7] Zhong Shao and Bryan Ford. Advanced development of certified os kernels. Technical report.