A Lightweight OS-Level Virtualization Architecture Based on Android

Bo-wen LIU, Nai-jie GU and De-he GU

School of Computer Science and Technology, University of Science and Technology of China,
Hefei 230027, China

Institute of Advanced Technology, University of Science and Technology of China,
Hefei 230027, China

Anhui Province Key Laboratory of Computing and Communication Software,
Hefei 230027, China

Keywords: Virtualization, Android, Namespace.

Abstract. All kinds of intelligent terminal equipment emerge in endlessly and grow explosively. Typical devices, such as smartphones and tablets, are becoming more and more popular, furthermore, intelligent equipment market share has more than that on PCs. And on mobile devices area, the demand for virtualization technology is gradually increasing. As an important branch of computer technology development, operating system level virtualization is becoming a hot spot of current research due to its low overhead and the advantages of lightweight. In this paper, based on the Android system on mobile devices, implement a prototype which can run multiple virtual system instances on only an Android OS by modifying the namespace mechanism in the Linux kernel to and expand into driver namespace mechanism. In addition, put forward a model called the active-inactive model guaranteeing the process if and only if in the active namespace to operate the device drivers, which was designed and implemented a new scheme of operating system level virtualization.

Introduction

Nowadays, the demand on mobile devices for virtualization technology is gradually increasing. The reasons are below[1]: 1) mobile device configuration is higher and higher, some luxury configurations are close to desktop equipment. 2) the usage scenarios of users on mobile terminal equipment are various. 3) the requirements of the multiple users. For example, an ordinary staff, sometimes need different independent use environment during the busy work and daily life. 4) security and privacy issues. Many users rely on multiple mobile devices to meet the needs of the work, personal geographical flexibility. The privacy of mobile devices including all software account and password, email password, bank financial software often force users to carry a variety of equipment to meet the requirements above.

Traditional virtualization technology virtualizing operating system by adding a layer that can manage hardware resources, and then provides a unified API to upper layer[2]. The hardware differences between different systems are blocked, the upper software can be run directly in the corresponding virtual platform, breaking the computer system traditional close relationship between hardware and software. At the same time, the model that a single host system with multiple virtual systems also makes full use of resources, bringing new vigor for the computer research.

Virtualization technics implements at different levels, mainly divided into the four fields below: hardware partitioning virtualization, full virtualization, para-virtualization and operating system level virtualization. Operating system level[3] virtualization is different from the other three, it did not have to create a virtual machine, but create an isolation environment to multiple applications by sharing the same operating system kernel. This technology is equivalent to the function provided by kernel, the operating system can provide each customer any virtual OS with an isolate environment, which is equivalent to a copy of the current operating system. For it do not need instruction translation or hardware management, the operating system level virtualization can reduce the performance.
overhead of virtualization, closely reach the original performance, reduce software redundancy, and maximize shared hardware resources\(^4\).

Mainstream technology researches are focus on PC virtualization products which adopt the Hypervisor architecture (shown in Figure 1). Through adding hypervisor-layer between the physical resource and the application software to be responsible for the management and isolate multiple virtual OS, each virtual OS has a relatively independent system resources. However, combining virtualization technology with mobile devices has become a new research area\(^5\), it can solve the following problems: 1) solve the switch the usage environment problems. 2) feed the multiple users demand. According to specific work and life environment users can customize the environment with special requirements which protects the security of the data and privacy issues. 3) reduce the cost. For mobile terminal manufacturers, multiple systems run on single chip makes the better competitiveness.

The remainder of this paper is organized as follows. Section 2 gives related work. Section 3 gives an overview of architecture and describes its design and implementation. Section 4 concludes the paper.

**Related Work**

Andrew and Avi Kivity designed KVM product, it adopted the hypervisor architecture. Focus on the hardware running on x86 architecture, KVM is implemented as a kernel module. After loading this module, Linux kernel becomes an KVM-aware system. At present it has been widely used in data center, improving the server efficiency.

Xen is also an open source virtual machine monitor(vmm) developed by Cambridge University, it can run more than 100 complete features of virtual operating system on host operating system. The architecture of Xen is shown in Figure 2.

This technology can already be used in NetBSD\(^6\), GNU/Linux, FreeBSD and Plan 9 so far. No instruction translation, modify the guest operating system, making it optimize the instructions and can perform in a virtual environment. Xen is adopted this method divided into two main parts:

1) Vmm, the classic hypervisor layer, is deployed between the hardware and virtual machines. Only import the core layer the Xen can begin to deploy the virtual machine.
2) Domain0. All virtual machines are named after "domainXXX" in order to increased gradually. Domain0 is most important and is one of the core. It has the very high privilege, as the first deployment of virtual operating system, which is responsible for some special tasks.

Xen has good performance, but it need to virtual complete hardware device, which is too difficult. This paper proposes a more lightweight virtualization solution based on the operating system level.

**Design & Implementation**

Android, Google open-source mobile operating system based on Linux platform, contains a complete operating system, user interface and mobile applications and meanwhile with touch screen, senior graphic display and Internet access function. The Android system architecture also adopts a layered architecture: application layer, application framework runtime layer, system library layer and the Linux kernel layer. According to the characteristics above, this paper adopts Android as the host.

**System Architecture**

Figure 3 provides an overview of the system architecture. The whole system firstly starts a minimum system initialization and each virtual Android system runs in user space. By using the Linux kernel lightweight operating system isolation mechanism, including the file system isolation (mnt namespace), process identifier isolation (pid namespace), as well as username isolation (uid namespace), it prevents conflicts and ensures the process running in a virtual Android system cannot see that in other system. Every virtual system has their own private virtual namespace which can concurrently use operating system resources in their own virtual system.

![System Architecture](image)

Figure 3. System architecture.

However, the basic operating system virtualization is not enough to run a complete Android user space environment. Virtualization mechanism is mainly used in server environment and the relatively small number of devices, mobile smartphone applications want to be able to interact directly with a large number of hardware devices. Operating system level virtualization is designed to support at least these devices list below (shown in Table 1), including hardware and Android environment unique hardware, such as the Framebuffer. The devices supported must meet these two requirements: (1) exclusive across the virtual Androids. (2) prevent interference between one and the other virtual environment.

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>Display output</td>
</tr>
<tr>
<td>Sensors</td>
<td>GPS, Accelerometer</td>
</tr>
<tr>
<td>LEDs</td>
<td>Indicator</td>
</tr>
<tr>
<td>Input</td>
<td>Touchscreen and input button</td>
</tr>
<tr>
<td>LCD</td>
<td>Backlight framework</td>
</tr>
</tbody>
</table>

**Active-inactive Model**

Linux provides a resource isolation mechanism called kernel namespace. It is a lightweight method of virtualization. The mechanism isolates the global system resources, makes the UTS hostname, IPC process communication, PID process number, and net network device resources belong to a particular
namespace. The process in different namespace has independent and isolated global system resources, different namespaces are transparent and not visible. Linux kernel provides six different types namespace (shown in Table 2), just specify the corresponding flag when call clone().

Table 2. Types of kernel namespace.

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC</td>
<td>CLONE_NEWIPC</td>
<td>System IPC, POSIX message queues (since Linux 2.6.19)</td>
</tr>
<tr>
<td>Network</td>
<td>CLONE_NEWNET</td>
<td>Network devices, stacks, ports, etc. (since Linux 2.6.24)</td>
</tr>
<tr>
<td>Mount</td>
<td>CLONE_NEWNS</td>
<td>Mount points (since Linux 2.6.19)</td>
</tr>
<tr>
<td>PID</td>
<td>CLONE_NEWPID</td>
<td>Process IDs (since Linux 2.6.24)</td>
</tr>
<tr>
<td>User</td>
<td>CLONE_NEWUSER</td>
<td>User and group IDs (since Linux 3.8)</td>
</tr>
<tr>
<td>UTS</td>
<td>CLONE_NEWUTS</td>
<td>Hostname and NIS domain (since Linux 2.6.19)</td>
</tr>
</tbody>
</table>

Traditional namespace mechanism provides the virtual Android operating system an isolate PID with MNT view. Furthermore, this paper extended it called driver namespace. In the realization of active-inactive model presented in this framework. The whole system consists of multiple inactive state virtual Androids and only one active state instant under the activated state. All other inactive virtual Androids run in kernel but does not display on screen content, and still can receive system events and perform the task. If and only if under the processes under active namespace operate the hardware device effectively. When a process needs to access the device driver resources, first of all, this model will judge whether the process of the virtual namespace is activated, if so, this process can directly access the device driver; Otherwise, the system needs to change the current active namespace into inactive state. And then, after the virtual namespace is set to enabled, the process is effective to the access the specific device drivers. This simple but powerful model makes system can effectively realize the multiplexing of hardware device drivers across multiple virtual Android environment.

The struct task_ struct contains the whole process information and as a member in it, nsproxy, pointed to the process’s all kinds of namespace. Specifically, add the struct driver_namespace (shown in Figure 4) in nsproxy structure structure. For a process, the driver_ ns members under nsproxy represent the driver’s namespace. The nsproxy is shared by tasks which share all namespaces. As soon as a single namespace is cloned or unshared, the nsproxy is copied.

Figure 4. System architecture.

Active represents the current namespace is whether active state or not; pid_ ns points to PID namespace because a driver namespace has a one-to-one relationship with a PID namespace; notifiers are a notifier_ block list, it can list every member in driver_ info structure members. Through leveraging the linux notifier chains mechanisms, it makes set active driver namespace state when switching active-inactive namespace registered driver callback functions. Every single driver_ info represents a single driver in the namespace, which is created when a process needs to access to driver in a namespace (if you haven't created it).

Working Principle

Based on the implementation of virtualization system level, focusing on the characteristic of diversity of Android devices, this paper also virtualizes some subsystems such as display, input events, indicator light and so on, which guarantees the normal operation of the basic Android system. Specific implementation process is as follows:

Step1: to associate drivers with specific data structure. As stated above, driver_info represents a single driver in the namespace, the member list can list the same device drivers in the different namespace structure called driver_global. Structure driver_global is a system global array and each
element represents a device driver which needs to be aware namespace. By setting the macro DRIVER _MAX maximum 32, the driver_global is associated with specific device drivers.

Step2: to register the device drivers in the namespace. Define the macro in every need-to-be-aware-namespace subsystem, which can generate device driver operate functions. These functions will generate A_driver_ns structure the bridge between driver and namespace framework. A_driver_ns members will provide A_create callback function and by calling callback function, data structure represented driver is registered in a namespace. The relations above among these data structure and functions are shown in Figure 5.

![Figure 5. Data structure design.](image)

Step3: to leverage active-inactive model (described in Section 3.2). This model guarantees that a process only under the namespace can access the device drivers effectively, meeting the isolation between different virtual systems.

**Conclusion**

We design and implement a solution for mobile devices virtualization. Compared with the traditional computers, mobile devices have specific implementation mechanism. By modifying the linux kernel space nsproxy structure and using active-inactive model, this paper provides this effective solution used to isolate and multiplexing device drivers. The future works are involved two major aspects: 1) support all Android subsystem virtualization.2) perfect the active-inactive model to support multiple active namespace running at the same time.

**References**


