International Journal of u- and e- Service, Science and Technology
Vol.9, No. 9 (2016), pp.419-430
http://dx.doi.org/10.14257/ijunesst.2016.9.9.38

Transplant LXC Framework for Mobile Virtualization

Xin Li[1], Hee-Kyung Moon[2], Zhan-Fang Zhao[3] and Sung-Kook Han[4]

Department of Computer Engineering,Wonkwang University
460 Iksandae-ro, Iksan, Jeonbuk, Republic of Korea
[1]leexin0723@gmail.com, [2]ybnjcw@wku.ac.kr, [3]zzfsjz@gmail.com,
[4]skhan@wku.ac.kr

Abstract

This paper proposes an Android mobile virtualization scheme based on the Linux Container technology. The scheme transplants the LXC tool into Android system. An OS that support LXC is built on SD card as the host environment of virtualization management, and the container is built on the client machine.By configuring the system root file, network and equipment in the container, make modified Android system can run in this environment. So one or more Android systems can run, and share a Linux kernel, which complete the virtualization of the operating system level. The results of the experiment show that the scheme has great advantages in balance between efficiency and isolation.

Keywords: *Mobile Virtualization; Operating System Level Virtualization; Container technology; Cgroup System; NameSpace*

1. Introduction

At present, the personal computer market is almost saturated, so an important feature of this era is the rise of wearable mobile devices. In addition, with the popularity of BYOD (Your Own Device Bring) in recent years, many employees are keen to run two or more operating systems in the device to distinguish the needs between daily life and work, which need use virtualization technology to realize the isolation between enterprise system and personal system. The virtualization technology can provide the safety of system, cut down the cost of hardware, and is the cornerstone of the cloud computing development. Based on these, this paper realizes the transplantation of LXC.

2. LXC Technology

With full name of Linux Container, LXC is an OS(operation system)-level virtualization technology. Simply put, it is a technology that isolates the progress and resources without providing instruction interpretation mechanism and other complexity such as overall virtualization, which is similar to *NameSpace* in C++. The container can effectively allocate management resources for single OS into isolated groups, so that it can better balance conflicting resource use demands among isolated groups.

The LXC program provides a tool set for user space in Linux system, the resource management framework *Cgroup*(Control group) provided by the Linux kernel is called with the LXC kit to conduct resource management and restriction of container, and the *NameSpace* mechanism is used to conduct isolation within the container. So it is necessary to realize two key kernel characteristics of Linux container: *Cgroup* and *NameSpace* mechanism. The LXC program can bind a specific CPU and memory node for the container, assign specific proportion of CPU time and IO time, limit the size of memory that can be used (both memory and swap space), provide device access control, and provide independent *NameSpace* (network, PID, IPC, MNT, UTS)[1].

ISSN: 2005-4246 IJUNESST

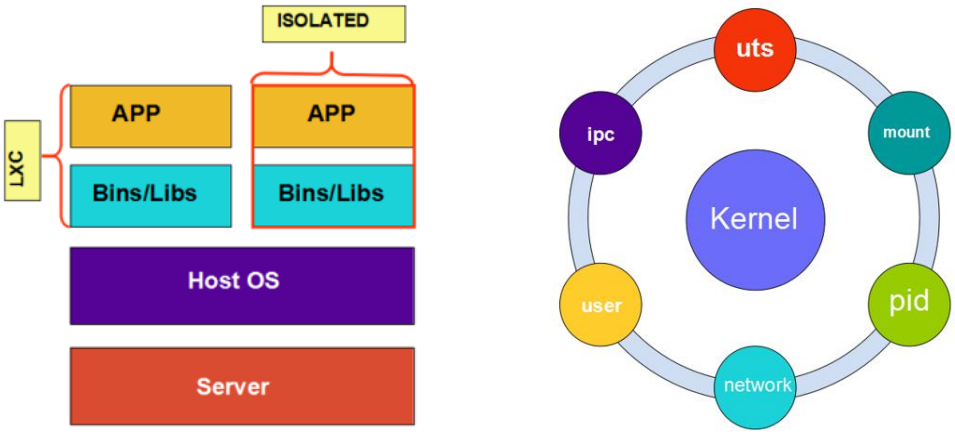

Figure 1. LXC Architecture and Six Isolation Mechanism that be Provided by Kernel

Comparing with the traditional virtualization technology, the advantages of LXC include:

- Using the same kernel with the host machine, the performance loss is small;
- Instruction level simulation is not required;
- Don't need Just-in-time compilation;
- Containers in the local of CPU core can run instructions, and do not need any special explanation mechanism;
- The complexity in para-virtualization and system call is avoided.
- The lightweight isolation provides isolation and sharing mechanism, which can realize resource sharing between the container and the host machine.

Linux Container provides a mechanism which allows multiple isolated container servers to run concurrently on a single controllable host node. Linux Container is a bit like *chroot*, which provides a virtual environment with its own process and network space, but is different from the virtual machine because it is resource virtualization of OS level [2].

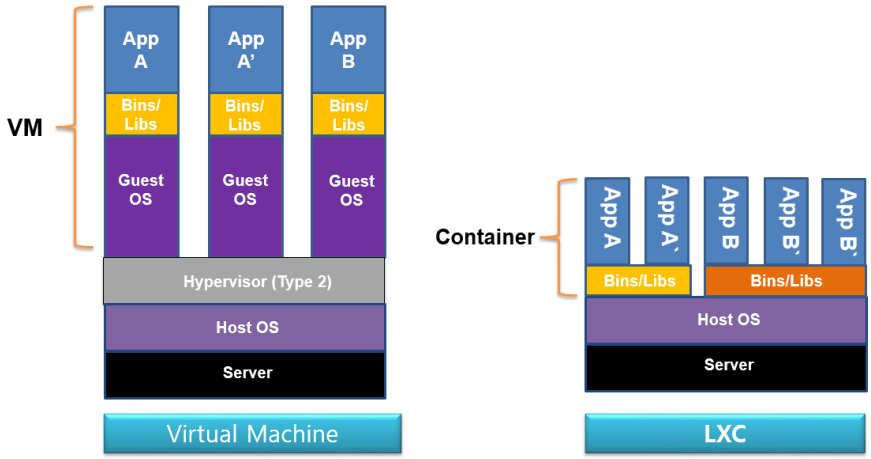

Figure 2. Comparison Between Common Virtualization and LXC Virtualization

3. Overall System Architecture

The overall architecture for Android mobile virtualization is as the following:

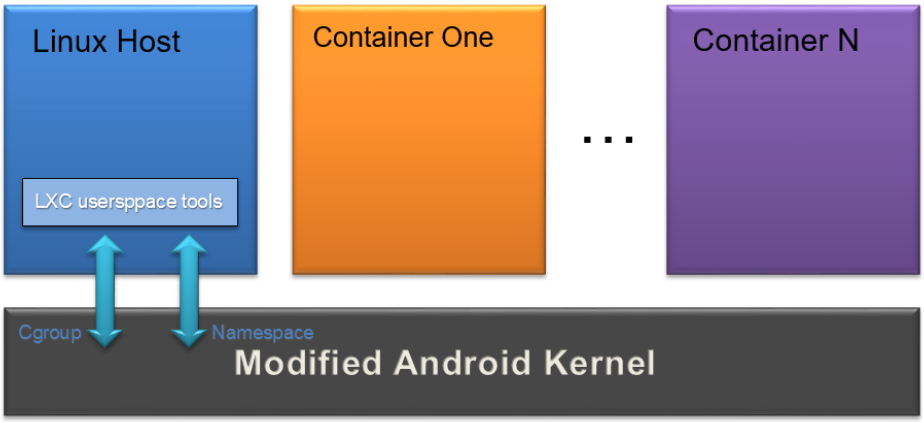

Figure 3. Overall System Architecture

The hardware resources on Android terminal are maintained the same. A modified Android kernel is ran on the hardware layer, and this kernel need support the *Cgroup* system and *NameSpace* mechanism. What has replaced the original ecological operation system is a system which supports using LXC's Linux system as the host system for virtualization management.

LXC interacts with the kernel through the *Cgroup* system and *NameSpace* mechanism, and by dividing progress groups, it can run in independent *NameSpace* to form container. LXC provides a group of tools in the user space, so that the host system can create and management the operation in container through explicit command, which can also monitor the progress and operation within container at any time. In order to run Android operating system in Linux container, the root file system in Android system must makes corresponding transplantation and modification because *NameSpace* has instantly created independent file system view for each container[3-4].

4. Adaptive Android Kernel

On the market at present, there is almost no kernel that supports the *Cgroup* system and *NameSpace* mechanism required by LXC, so we need to conduct related modification and configuration of the Android kernel to make it supports the *Cgroup* system and *NameSpace* mechanism. The Android kernel version adopted in this paper is 2.6.35, the experiment terminal for simulation operation is Nexus, the *busybox* is also configured correctly, and the CPU is Qualcomm series, so that the source code can be directly obtained from the Android tree.

$ git clone https://Android.googlesouree.com/kernel/msm.git

Extract the branch kernel source code by the checkout command of git tool.

4.1. The Configuration of Config File

Before compiling the new kernel, we must reconfigure the kernel options to make the *Cgroup* system, the *Namespace* mechanism and the related functions properly compiled into the kernel. But, the default kernel options may include a large number of driver which test machine hardware does not require, or lose some particular driver. So we can make the existing *config* file on the terminal as the native configuration file, and modify it.

Through the following command to make PC to connect with the testing machine and extract the information of configuration file:

$adb shell // Connect the testing machine by ADB protocol

$cat /proc/config.gz > /sdcard/config.gz // Extract the config file of the machine

$gunzip /sdcard/config.gz // decompress files

Copy the *config* file to kernel source directory of MSM on the PC. Start the kernel function configuration with the following command, and the architecture parameter is set to *arm*.

$make menuconfig --arch=arm

To make the *Cgroup* system and the *Namespace* mechanism run right, we must properly configure the following features:

```
CONFIG_GROUP_SCHED=y
CONFIG_FAIR_GROUP_SCHED=y
CONFIG_RT_GROUP_SCHED=y
CONFIG_CGROUP_SCHED=y
CONFIG_CGROUPS=y
CONFIG_CGROUP_NS=y
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CPUSETS=y
CONFIG_PROC_PID_CPUSET=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_RESOURCE_COUNTERS=y
CONFIG_CGROUP_MEM_RES_CTLR=y
CONFIG_CGROUP_MEM_RES_CTLR_SWAP=y
CONFIG_MM_OWNER=y
CONFIG_NAMESPACES=y
CONFIG_UTS_NS=y
CONFIG_IPC_NS=y
CONFIG_USER_NS=y
CONFIG_PID_NS=y
CONFIG_NET_NS=y
CONFIG_NET_CLS_CGROUP=y
CONFIG_SECURITY_FILE_CAPABILITIES=y
CONFIG_DEVPTS_MULTIPLE_INSTANCES=y
```

Figure 4. Kernel Configuration Options

Among these parameters, ROUP_SHED represents the switch variable for gang scheduling, the configuration options name ending with NAMESPACES and NS represents the switch variable for the *Namespace* mechanism and several *Namespace* supported by currently kernel.

5. Modify the Kernel Source Code

The native Android kernel source code lacks some critical calls, or some unpredictable bugs will occur when Android OS is running in container, so it is necessary to do corresponding modification on the kernel source code. In the following, we list several important places that require source code modification.

5.1. Increase the Call of Setns System

When the *NameSpace* mechanism is added to the kernel, *setns* has not appeared in the kernel source code, and the role of this system call is to add progress into an existing *NameSpace*.

5.2 Add the Proc Entrance and Operation Interface of NameSpace

NameSpace mechanism has not been shown when it is initialized in kernel, by adding the *NameSpace* entrance, we can easily query *NameSpace* instance existed in current system. Various *NameSpace* sub-modules are used to realize related operation, including get add reference, put reduced reference count and installing *NameSpace* into specified *nsproxy NameSpace* proxy instances.

5.3. Modification During the Implementation Process of the Fork Call

The *copy_process* function is called to copy the progress information during the fork, which include memory information of the parent process and *NameSpace*, *etc.*,during adaptation to this kernel version, it is not suitable to judge the CLONE flag NEWPID when a leading process of a process group is cloned. The numbers member in PID structure has a corresponding actual process ID in *NameSpace*, if the actual process ID value is 1, and the process is an *init* process which will receive orphaned process in *NameSpace*. Meanwhile, the *unshare* system in *fork.c* file misses multiple judgments for *NameSpace* when it calls the kernel version [5].

5.4. Modification of Kernel Startup Parameters and the Generation of Boot Partition

- Kernel start-up command line CMDLINE

The scheme in this paper need start the Linux system on Android testing machine as the OS-level virtualization management system, namely, the host system. The native Android OS is installed in the ROM of mobile phone, and its file system format is *yaffs*, which is not suitable to be installed in the Linux system. In order to save memory space and prevent polluting the native Android OS, this scheme starts host system from outside, and install the host system on the SD card. Therefore, the kernel need start the OS from the SD card, and the start-up root file path must be added as the path start-up parameter of SD card device during starting the kernel. The ext2 partition device path of the external memory SD card on the testing machine is */dev/mmcblk0p2*, when starting CMD, the root path is */dev/mmcblk0p2*, *init* process path is */sbin/init* . The identification and mounting of SD card requires certain time, so the system waits for root file system mount complete when the kernel start-up, which requires adding parameter *rootwait*. The partition format of the Linux host system is ext2, so it adds the parameter *notinitrd*, does not need the assistance of *initrd*. Corresponding start-up CMD as following[6]:

```
"no_console_suspend=1 \
 wire.search_count=5 \
 root=/dev/mmcblk0p2   rw rootfs=ext2 \
 init=/sbin/init rootwait   noinitrd"
```

The scheme in this paper does not involve the *ramdisk* and base, these two parts in the native *boot.img* can be kept, and we only need to replace the kernel *kerne* and start-up parameter *cmdline*. Using *fastboot* tool provided by the SDK can write the new *boot.img* into the MTD partition. Start the phone to the *fastboot* mode, configure corresponding *fastboot* protocol permission of PC terminal, write the kernel part into the *boot.img*, and download it to the start-up partition, the commands as shown below:

$fastboot -c 'CMDLINE' boot zImage

zImage is the new kernel file after modification, which is mentioned above, and *cmdline* is the start-up parameter that can start from the SD card[7][8][9].

6. Build the Host Environment

The virtualization scheme adopted in this paper uses a standard Linux system as the host system for virtualization management, so that the LXC kit can directly operate in the host system without any modification. Therefore, this paper chooses the *Debian*(*Debian* is an operating system and a distribution of Free Software, *Debian* supports Linux officially) system or CentOS system based on *armel* architecture as the host system. *Debian* is a free operating system (OS) for your computer. An operating system is the set of basic programs and utilities that make your computer run. Generally, *Debian* is suitable for the server operating system, with excellent stability. The *Debian* system, as long as the application level does not appear logical flaws, basically impregnable, is a system that does not need to be always restarted. The basic core of entire *Debian* system is very small. It not only is stable, but also occupies a very small hard disk and memory space. The CentOS system is a little bigger in comparison with *Debian* system. 128M VPS can run more smoothly *Debian* system than CentOS system, so this paper uses *Debian* system as the host operating system.

6.1. Download and Install the *Debian* Host System on the SD Card

The downloading and installation of *Debian* system mainly depends on the *debootstrap* tool. The *debootstrap* tool is an official *Debian* tool to install the *Debian* Base system, which provides the function to install the *Debian* Base system into a subdirectory of a system that has been installed. The *debootstrap* does not require to install CD, only requires to connect the repository of *Debian* by internet.

Partition SD card with *gparted* or similar software, and ensure the first partition is in *vfat* format, so that the testing machine can recognize it. Another partition is in ext2 format, which is used to install the Linux file system. Read the SD card from the PC terminal, mount the ext2 partition into any directory, and install the smallest *Debian* system into this partition by *debootstrap* tool[10][11]:

#debootstrap --foreign --arch-armel --variant=minbase/

squeeze <EXT2_DIR> http://ftp.Debian.org/Debian

In this step, some necessary tools and applications will be downloaded, which will be installed in the SD card, and we will install and configure them in Step II.

6.2. Configure the Host System

Until now, the smallest *Debian* system has been successfully installed on the SD card. We need do some necessary configuration in order to enable it to assume the task of the host system of operating system level virtualization, which include starting scripting configuration and others.

6.3. Start Scripting Configuration

When the new kernel becomes effective and starts the *Debian* system, if network is not connected and application software is not installed, in addition, no serial lines can be connected to the phone, so that the PC terminal will finally lose its connection to mobile phone, and we cannot operate the mobile phone. Therefore, after the system is loaded, we should start the *adbd* program to make the PC terminal to control the mobile phone by the ADB protocol. On the common system start-up level, the system will execute commands

in the */etc./rc.local* script. Add the following command before the "*exit 0*" command in the script file:

/sbin/adbd &

Then, it will execute *adbd* after starting the system. The precondition is that the *adbd* in native system must be copied to corresponding directory.

It is very important to configure the "*/etc./fstab*" file. If the configuration of this file is wrong, the kernel will be unable to mount the root file system, which makes following start-up process can't be executed. For the Linux system, the device path of SD card is */dev/mmcblk0p2*, which refers to the second partition of MMC device. In the *fstab* file, in addition to mounting common virtual file systems such as *proc* and *sysfs*, the *Cgroup* file system must also be mounted.

none /Cgroup Cgroup defaults 0 0

Of course, in order to conduct data interaction with the system partition and *userdata* partition which are in native mobile phone system, corresponding *mtd* device can also be mounted[12].

6.3.1. Network Configuration: Compared with the traditional PC, mobile phone uses wireless network card to access internet. If we want to support the Wifi network in the system, we need to start the wireless network card driver module. New kernel will have to use the new drive module because modules must be matched to the kernel. The testing machine in this paper is Nexus One, and the wireless chips is the series of Broad-com 4329.

6.4. Transplant LXC and Build the Android Container

As mentioned above, the *Debian* system is used as the host system of virtualization management to conduct environment configuration of mobile terminal, including the modification of start-up script, the mounting order setting of file system, *etc.*, In this way, after writing into the modified start-up partition image *boot.img*, it will restart and enter pure Linux system. In the host system, the LXC user kit is used to create and configure the container environment, so that it can run the Android OS as a virtual machine.

6.4.1.The LXC Tool Set of Users: LXC provides a software packages for user space, which can create and manage containers. Meanwhile, it is also very easy to install the software packages. Download the source code from the LXC project official website, and install directly it after configurations[13].

<lxc_DIR># ./configure

<lxc_DIR># make

<lxc_DIR># make install

6.4.2. The Building of Virtual Bridge: It must perform network conjunction by the way of the establishment of the bridge because the OS in a container can't use hardware network device of host system directly to connect to the network. But container environment is different from the common virtual machine environment, so we use the new virtual device "*veth*" of LXC, also known as Virtual Ethernet, which can play a role in transfer. After configuration, we can view the information in host system by *ifconfig* command.

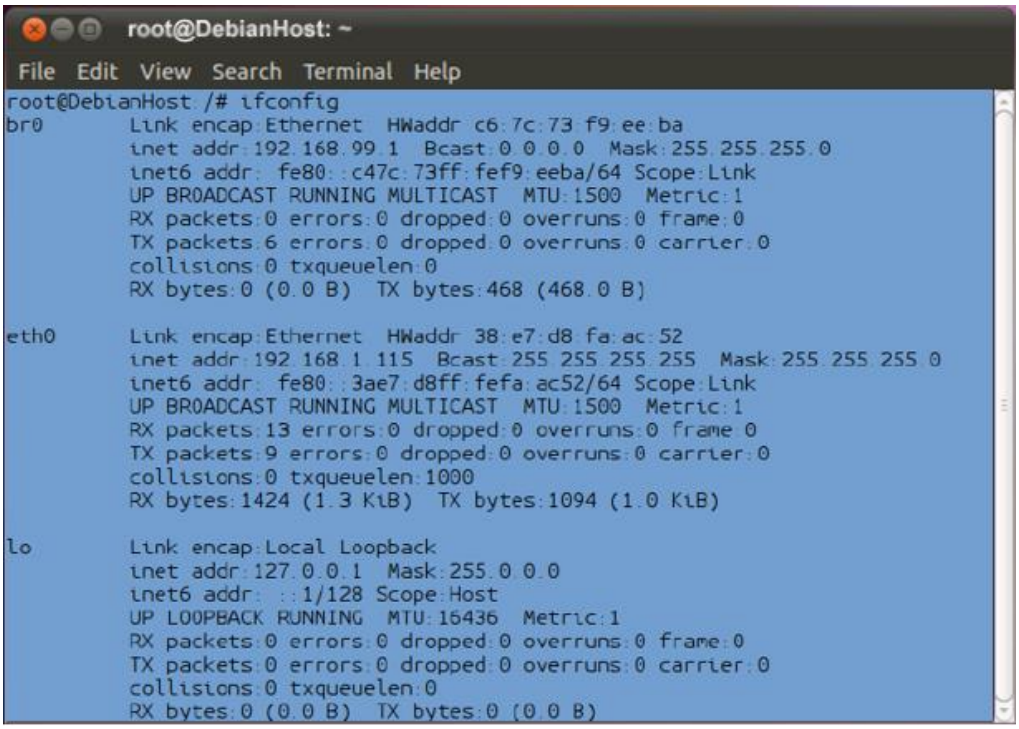

Figure 5. Network Configuration Diagram of Host System

6.4.3. Create the Android Root File System: The native Android root file system cannot be completely copied into the container, and due to restriction of operation environment in the container, some adjustment must be made before the Android system runs in the Linux Container. Therefore, we need to generate related nodes in the */dev* directory, and through modification of the source code of Zygote and starting script *Init.re*, we finally create the configuration file to complete the final work [13].

6.5. Running and Result Analysis of Android Container

In this paper, the testing machine is Nexus One, the kernel version is Modified 2.6.35. The system is *Debian* (armel) Host system on the SD card, which is mentioned in the previous section. After starting the machine, the testing machine enters the *Debian* system, which is connected with the PC terminal by the *adb* protocol. We can check the *Cgroup* system and *NameSpace* function start-up situation in current kernel through the *lxc-checkconfig* tool.


```
root@DebianHost:/# lxc-checkconfig
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup namespace: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled
```

Figure 6. Special Function Check in Kernel

Enter the root file system directory storing the Android container, can see the configuration file for the container:

```
root@DebianHost:/home/Android_LXC_2# ls
config  logfile  rootfs
root@DebianHost:/home/Android_LXC_2# cat config
lxc.utsname = android_testTwo
lxc.tty = 4
lxc.rootfs = /home/Android_LXC_2/rootfs

# network
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
lxc.network.ipv4 = 192.168.99.203 0.0.0.0
lxc.network.name = eth0
lxc.network.veth.pair = vethvm2

lxc.cgroup.devices.deny = a # deny all first

lxc.mount.entry=none /home/Android_LXC_2/rootfs/proc proc defaults 0 0
lxc.mount.entry=none /home/Android_LXC_2/rootfs/sys sysfs defaults 0 0
lxc.mount.entry=/lib /home/Android_LXC_2/rootfs/lib none ro,bind 0 0
lxc.mount.entry=/usr/share/locale /home/Android_LXC_2/rootfs/usr/share/locale none rw,bind 0 0
```

Figure 7. Container Configuration Diagram

```
# consoles
lxc.cgroup.devices.allow = c 5:* rwm
lxc.cgroup.devices.allow = c 4:* rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
# /dev/pts/* - pts namespaces
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
# nexus specific
lxc.cgroup.devices.allow = c 1:7 rwm # dev/full
lxc.cgroup.devices.allow = c 1:11 rwm # dev/kmsg
lxc.cgroup.devices.allow = c 7:* rwm #
# cpu and memory
#lxc.cgroup.cpuset.cpus = 0
#lxc.cgroup.cpu.shares = 1024
#lxc.cgroup.memory.limit_in_bytes = 512M
#lxc.cgroup.memory.memsw.limit_in_bytes = 512M

# 10:* devices
lxc.cgroup.devices.allow = c 10:0 rwm # dev/pmem
lxc.cgroup.devices.allow = c 10:1 rwm # dev/pmem_adsp
lxc.cgroup.devices.allow = c 10:2 rwm # dev/pmem_camera
lxc.cgroup.devices.allow = c 10:223 rwm # dev/uinput

lxc.cgroup.devices.allow = c 10:30 rwm
lxc.cgroup.devices.allow = c 10:31 rwm
lxc.cgroup.devices.allow = c 10:32 rwm
lxc.cgroup.devices.allow = c 10:33 rwm
lxc.cgroup.devices.allow = c 10:34 rwm
lxc.cgroup.devices.allow = c 10:35 rwm
lxc.cgroup.devices.allow = c 10:36 rwm
lxc.cgroup.devices.allow = c 10:37 rwm
lxc.cgroup.devices.allow = c 10:39 rwm

lxc.cgroup.devices.allow = c 10:40 rwm # /dev/keychord
lxc.cgroup.devices.allow = c 10:41 rwm
lxc.cgroup.devices.allow = c 10:42 rwm
lxc.cgroup.devices.allow = c 10:43 rwm
lxc.cgroup.devices.allow = c 10:44 rwm
lxc.cgroup.devices.allow = c 10:45 rwm
lxc.cgroup.devices.allow = c 10:46 rwm
lxc.cgroup.devices.allow = c 10:47 rwm
lxc.cgroup.devices.allow = c 10:48 rwm
lxc.cgroup.devices.allow = c 10:49 rwm
```

Figure 8. Container Configuration Diagram

The *config* file is used as a container configuration file. Start the container through *lxc-start* command, and run the Android system. After starting the container, the system operation interface will enter the Android system.

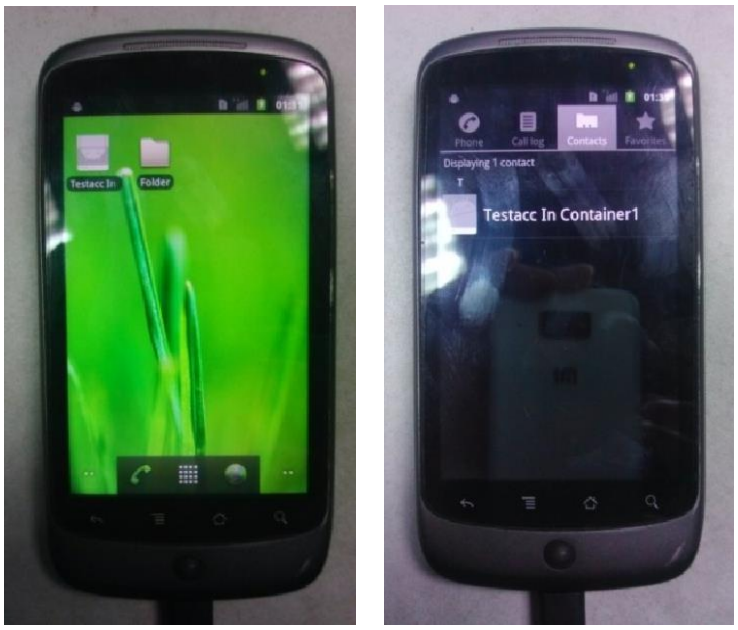

Figure 9. Android Container Start-Up Completion Interface

View host system by *pstree* command, the following diagram is the situation of Android system process after starting container by LVC-START command:

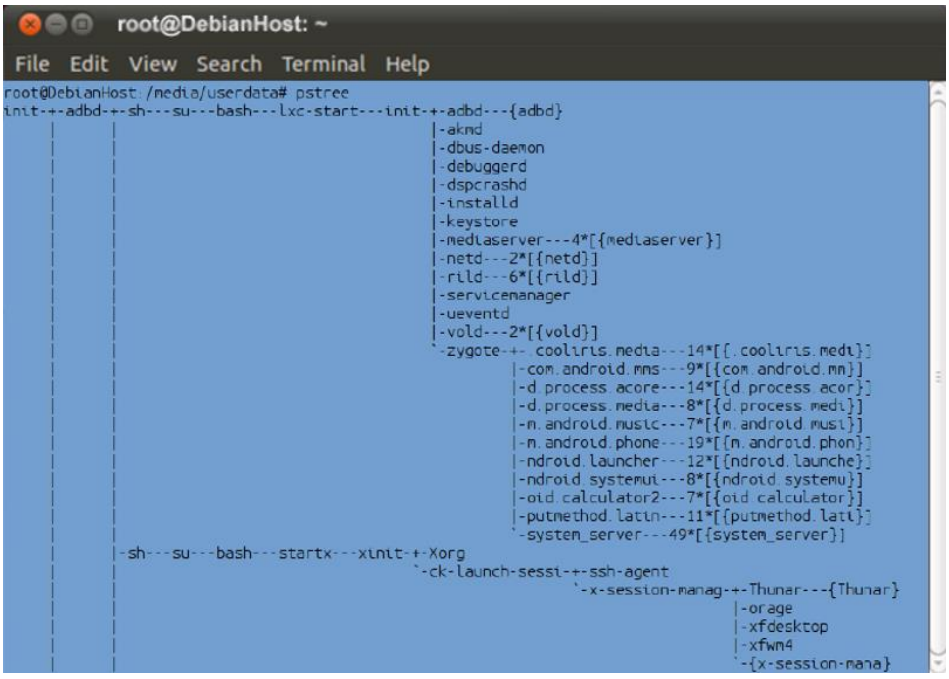

Figure 10. Host System Derivative Graph

7. Conclusion

At present, cloud computing is a *hotspot* in the computer field, while virtualization technology is the basis of cloud computing. This paper tries to look for a balance point between efficiency and isolation of virtualization technology, and the OS-level virtualization technology of Linux Container is adopted. Based on analysis of how Linux Container realizes progress resource restriction and data isolation, we provide specific scheme to realize mobile virtualization on Android intelligent terminal.

References

[1] Wikipedia, https://en.wikipedia.org/wiki/LXC.
[2] Linux Containers, https:/linuxcontainers.org.
[3] G. Heiser, "The role of virtualization in embedded systems", Proceedings of the 1st workshop on Isolation and integration in embedded systems, Sydney, Australia, **(2008)** April 11-16.
[4] H. Härtig and M. Roitzsch, "Ten years of research on L4-based real-time systems", Proceedings of the Eighth Real-Time Linux Workshop, Dresden, Germany, **(2006)**.
[5] Sun Peng, "Research and Application of Smartphone Mobile Office based on Virtual Technology", Fudan University, Shanghai, China, **(2011)**.
[6] OKL4 Microvisor, http://www.ok-labs.com/products/okl4-microvisor.
[7] Y. D. Ke, "Android Kernel Analysis", Publishing House of Electronics Industry, Beijing, **(2011)**.
[8] L. Shi, D. Q. Zou and H. Jin, "Xen Virtualization Technology", The Publisher of Huazhong University of Science and Technology, Wuhan, **(2009)**.
[9] Q. Jiang, "A Dynamic Migration Method Based On Linux Container", Information Technology and Informatization, no. 2, **(2015)**, pp. 66.
[10] N. Matthew and R. Stones, "Beginning Linux Programming", Posts & Telecom press, Beijing, **(2010)**.
[11] Debian System, https://www.Debian.org/releases/stable/installmanual.
[12] G. Wu, J. Li and Y. X. Lai, "Research of JBS Model Based on the Virtual Technology of Operating System Container", Network Security Technology & Application, no. 4, **(2010)**, pp. 39-41.
[13] Q. Li, "Research of USB Network Device Driver Based on Linux", Huazhong University of Science and Technology. Wuhan, China, **(2010)**.

Authors

Xin Li, received his B.S. degree in Communication Engineering from Henan Normal University in 2012, in China, and he is currently studying a M.S. degree at dept. of computer engineering at WonKwang University from 2014, in South Korea. His main research interests include Ontology, virtualization technology, Linux container and Embedded.

Hee-Kyung Moon, received her B.S. degree in computer engineering from WonKwang University in 1993, and M.S. degrees in dept. of computer engineering from WonKwang University in 1996, and he is currently studying a Ph.D. degree at dept. of computer engineering at WonKwang University from 2014, in South Korea. Her main research interests include Ontology, Linked Data, Semantic Web, knowledge-based system and smart service.

Zhan-Fang Zhao, Received her B.S. degree in surveying engineering from Hebei Institute of Technology in 2000, and M.S. degree in geodetection and information technology from China University of Mining & Technology (Beijing) in 2005, and she is currently studying Ph.D. degree at dept. of computer engineering at WonKwang University, in South Korea. She is an associate professor of college of information and engineering in Hebei GEO University, in China. Her major research areas are ontology, Semantic Web and Machine Learning.

Sung-Kook Han, received his B.S. and M.S. and Ph.D. degree in electronic engineering from Inha University in 1978, 1981 and 1988, respectively, in South Korea. Currently, he is a professor at dept. of computer engineering in WonKwang University. He was a visiting researcher including UPENN and University of Innsbruck. His major research areas are ontology, Semantic Web, knowledge-based system and smart service.