Cells: A Virtual Mobile Smartphone Architecture (Review)

Jeremy Andrus, Christopher Dall, Alexander Van't Hof,
Oren Laadan, and Jason Nieh

Conference : SOSP 2011

Reviewed by: Sahil Singh

Summary

The authors present a virtualization framework which uses Linux containers to provide different isolated virtualized environments on a mobile phone (ARM based) which they call Virtual Phones or VPs. They combile VOIP with the cellular network in a novel way to provide each VP a different phone number. The USP of the paper lies in providing GPU acceleration to all VP (one at a time), and per device access policy (no access/shared/exclusive). They do several optimizations to reduce the memory and power footprint of the device, when virtulized.

Detailed Notes

System assumption

Hardware: ARM processor Software: Linux based OS, support of namespaces in kernel

Main contributions

Cells can run multple VP simultaneosly. There is a per-device access control (access modes: no-access/shared/exclusive) which allows to fine-tune security and enhance sharing of devices. Access to devices is arbitrated using device namespaces, which is a technique introduced by this paper. The solution presented here stands in contrast to the apporach taken by VMware [1], which allows only one guest VM to run, and doesn't provide graphics acceleration to it. This solution is also more generic than vNative [2], which allocates all devices to FVM exclusively, and keeps the BVP in a reactive state (i.e. they can react to events but have no code executing while they are in background). The unique approach used by cells which combines cellular network telephony and VOIP to give each

[1] Paper- The VMware mobile virtualization platform:Is that a hypervisor in your pocket?
[2] Paper- A virtualisation solution for BYOD with Dynamic Platform context switch

VP a uniques phone number and birectional communication capability on it, which stands out from previous approaches. Using Linux containers instead of booting different OSes leads to low overhead virtualisation.

System Architecture

The running VPs are categorised into FVP (foreground virtual phones) and BVP (background virtual phones). All VPs are different linux containers running on the same kernel, and having different namespaces. These VPs are created on a PC and then downloaded to a mobile phone. An FVP, as the name suggests is a VP that currently occupies the screen. All user interactions are handled by FVP and only an FVP is allowed to render content on screen. The content (graphics) rendered by BVPs are redirected to a background buffer. All the virtual phones can be in execution simultaneosly, however only the FVP is allowed to control configuration which can affect the state of whole system, such as power management (putting device to sleep), WiFi and cellular data configuration. FVP can access all H/W (excluding the ones on which it has no access rigths), BVP can only use device for which FVP allows shared access. System may switch a BVP to foreground on an event such as a call or sms for a BVP, otherwise to switch to a VP user has to select it from a list of running BVPs.

Security

The processes running in different VPs are isolated from one another with the help of namespaces. The namespaces are also used to isolate filesystem mount points. VPs can han have per device access rights (no-access/shared/exclusive), however all devices may not support all three modes of access, for example frame buffer cannot be used in shared mode. To prevent direct access to kernel, creation of device nodes is prohibited, moreover the configuration of a VP can be password protected. This may be helpful in creating a secure, and isolated VP for work/business applications.

Optimizations/Scalability improvements

Cells comes with a number of optimizations which make it more suitable for running on a resource constrained device.

- Unioning filesystem: This allows sharing of data between VPs and reduces disk space required.
- KSM: Cells tries to merge common pages between VPs using kernel same page merging. This happens only when a new VP boots, as searching and merging pages requires processing. When the rate at which merging is done reduces below a threashold it is stopped. Also according to the paper there isn't much gain in trying to merge application pages, as the heap memory is the main consumer of resouces while an application is running which doesn't has much common data.
- Low memory killer: cells leverages android low memory killer to kill background, inactive processes consuming large amounts of RAM. These

mainly contain the processes android uses for system optimization such as speding up application start time, and doesn't affect functionality. Critical system processes are not affected.

Virtualizaing devices

Cells uses a number of kernel level and user level techniques to virtualize devices.

Kelnel level device virtualizaion

Kernel level mechanisms include ways to implement device namespaces. Device namespaces is a method to provide isolation and hardware multiplexing between VPs. Device namespaces help in creating per VP data structures for each device. Each VP uses a unique device namespace to access the device. There are three methods to implement device namespaces.

(i) Device driver wrapper- Device driver wrappers passes through all requests from the FVP to the actual driver. For other VPs the wrapper makes changes to a VP specific data structure, which is maintained for each device. The frame buffer is virtualized by cells in this way. The FVP writes to screen memory, while the BVPs write a a buffer maintained in RAM.

(ii) Modifying device subsystem to be aware of device namespaces- for eg. the input subsystem in linux is responsible for sending input events to processes listening for it. If the subsystem is made aware of the device namespaces it will only send the events to a resitricted set of processes.

(iii) Modifying Device driver itself to be aware of device namespaces.

User level device virtualization

To virtualize access to cellular network, Cells uses a userspace proxy. Cellular networks come with a proprietary software stack, and only a user-space library is exposed, hence a kernel space driver or a driver wrapper wouldn't be of any use. The userspace proxy exposes a similar interface in VPs as exposed by the proprietary library. If a VP wants to use cellular network, let's say to place a call, the userspace proxy of Cells recieves the request and forwards it to cellular stack (BVPs aren't allowed to place call). Similarly for an incoming call the proxy decides which VP is the call meant for and then forwards accordingly.
Similarly to manage WiFi configuration Cells uses a userspace proxy. Only FVP is allowed to configure WiFi.

Graphics

Cells provides full graphics acceleration to all the VPs, however only the FVP is allowed to access screen memory and therfore only it can render graphics on screen, rest of the VPs render to a memory buffer. A driver wrapper arbitrates access to FB (Frame Buffer). FVP is allowed to write to it, while for other VPs the wrapper makes the read/write by the GPU happen to different buffer. The contents of this buffer is used to render the screen of a BVP when a switch is made to it, at the same time the FB of the previous FVP is backed up to a

different buffer. The following are the steps carried out while making a switch to a different VP.

(i) screen memory remapping- processes map part of frame buffer to their memory to access it directly, thus when a switch is made the page table entries of the processes mapping the FB have to be changed. Also the page table entries of the processes of the BVP to which a switch is being made are updated to point to the FB.

(ii) screen memory deep copy - copy the memory of FB to a buffer allocated for VP that is being switched out and copy the corresponding buffer of the BVP being switched to, to FB.

(iii) H/W state synchronization : Save FB state (configurations, etc.) of the outgoing VP and restore state of incoming VP.

(iv) GPU coordination : notify GPU of the changes made to FB.

Telephony

Only FVP is allowed to make calls. When the first VP initializes the SIM card, the communication is recorded, and when other VPs try to initialize SIM card the recoreded responses are replayed whithout actually reinitializing the SIM card.

Using VOIP server each VP is alloted a different number. When a call is made to any of these numbers the VOIP server appends a digid to the Caller-ID of the incoming call, and forwards it to the actual phone number of the mobile (the number provided by cellular network). This digit is read by cell to determine the VP for which the call is destined. Cells removes this digit and forwards it to the correct VP.
Similarly when a call is made by a VP (only FVP can make call). Cells intercepts the reuqest, calls the VOIP server using cellular network, then then forwards the call with the number corresponding to the VP, by first sending to VOIP server the number that has to be dialled using DTMF tones.

Networking

Network resources are isolated by using namespaces. WiFi and cellular data are not exposed directly in VPs, instead virtual ethernet is exposed which is routed through WiFi/cellular data using NAT. Only an FVP is allowed to change configurations of WiFi and cellular data.

Power Managememnt

The power management policy of FVP is applied to device, i.e. only FVP is allowed to change these policies, and hence only they can put the phone to sleep.

Experimental evaluation

Setup

Nexus 1 and Nexus S running Android 2.3.4 (Latest version of Android at the time) were used. On each non-Virtualized android was compared with Cells running 1VP, 2VP, 3VP, and 5VP. CPU, graphics, disk I/O, network and web browser performance were tested using Linpack, Neocore, Quadrant I/O, wget (wget a 400 MB file), and Sun Spider respectively.
Memory footprint, power consumption for different number of VPs was also noted.

Results

Neocore showed the similar performance for virtualized FVP to non-virtualized Android. This is because Cells allows full graphics virtualization for all VPs.
I/O was slower on nexus 1 when the number of VPs were more (4 and 5), while such a slowdown wasn't obeserved on Nexus s, since the latter uses internal flash memory, which is faster than external SD card used by the former.
CPU and web browser performances were close to stock. All benchmarks, except Neocore showed a degradation of performance when a load was run on one of the BVPs. Quadrant was most affected because of contention of SD card.
The memory footprint linearly increased as number of VPs increased, however this was still low because of KSM.
Power consumtion on Nexus S wasn't affected noticeably due to virtualization, Nexus 1 was slightly affected. This may be because of Nexus S being a newer H/W, with better power management capabilities.

Positive points

1. Virtualization overhead was low.
2. VPs had access to all devices, as far as their access policies allowed.
3. VPs were provided with different numbers, in a way truly creating virtual phones.

Negative points

1. Nexus devices come with superior hardware, a performance evaluation on an inferior hardware of a cheaper mobile phone would have tested applicability of Cells for all kinds of smartphones.
2. Since linux containers are used for Virtualizaing, different OSes cannot run, but the authors say that such a requirement may not be feasible either, due to licensing restrictions. For example running iOS on non-apple H/W will not be allowed by Apple.