User Isolation in Multi-user Multi-touch Devices Using OS-Level Virtualization

Minkyeong Lee[1], Minho Lee[1], Inhyeok Kim[1], and Young Ik Eom[2(✉)]

[1] College of Information and Communication Engineering, Sungkyunkwan University, Suwon 440-746, Korea
{krong96,minhozx,kkojiband}@skku.edu
[2] College of Software, Sungkyunkwan University, Suwon 440-746, Korea
yieom@skku.edu

Abstract. Providing multi-user isolation is one of the most important issues in computing environments with multi-touch displays because multi-touch devices such as tabletops allow multiple users to interact simultaneously. However, existing window manager, which is commonly used to control the placement and appearance of windows, never provides isolation among the users. In this paper, we present a method that provides isolation in multi-touch multi-user computing systems using OS-level virtualization and show the effectiveness of the method with serveral experimental results. Our experimental results show that OS-level virtualization sucessfully provides both multi-user interaction and isolation in multi-touch devices.

Keywords: Multi-user environment · Multi-touch device · User interaction · User isolation · OS-level virtualization · Docker

1 Introduction

As multi-user computing systems with multi-touch devices become increasingly available, users can interact more easily in those environments [1]. In those multi-user environments, user isolation is important to prevent negative interferences among the users. However, it is difficult to provide multi-user isolation in such computing systems, because they were designed only for a single user. To support multi-user functionalities, there are two major considerations: (1) user interaction and (2) user isolation.

First, user interaction is necessary to share data of each user. Generally, user spaces are logically partitioned in multi-user systems, and to support user interaction in such systems, window manager should be used [2], which is a software layer between kernel and application, and manages window placement on the screen and provides control of common actions (e.g., click, drag and drop). For user interaction, the window manager exploits Inter-Process Communication (IPC) which has lower overhead than Remote Procedure Call (RPC). However, this makes it more difficult to achieve multi-user isolation. Second, isolation of user space should be guaranteed to exclude negative interferences among the tasks of users. To do this, previous works used full virtualization or para-virtualization on mobile devices, desktops, and servers [3]. However, these technologies are inappropriate for multi-user multi-touch devices, because they use multiple

Y. Zhang et al. (Eds.): CloudComp 2015, LNICST 167, pp. 30–38, 2016.
DOI: 10.1007/978-3-319-38904-2_4

guest kernels for multiple execution environments. In these technologies, window manager is difficult to provide efficient interactions via UNIX domain sockets. In contrast, OS-level virtualization can efficiently support user interactions because all the separate user spaces share only a single kernel. Moreover, it facilitates isolation by partitioning user space with containers.

In this paper, we verify that OS-level virtualization is suitable for user isolation in multi-user multi-touch systems. For evaluation, we performed experiments using Docker, which is a framework based on OS-level virtualization [4]. In case of Docker, it has some advantages: (1) there is no modification of kernel and applications, and (2) it reduces waste of space by managing file system images via AuFS, which provides a layered stack of file systems. Experimental results show that Docker has just 0.5 % performance degradation, compared to native system. Furthermore, the overhead of Docker is less than 1 % in the experiments that measure frame rates of User Interface (UI) applications and costs of client-server communication and UNIX domain socket communication.

The remainder of this paper is organized as follows. Section 2 discusses related work, including window manager and virtualization technologies. Section 3 describes the reason why it is necessary to use OS-level virtualization in multi-user multi-touch systems. Section 4 presents experimental results. Finally, Sect. 5 concludes this paper and describes a future direction.

2 Related Work

In this section, we describe window manager and discuss typical virtualization technologies.

2.1 Window Manager

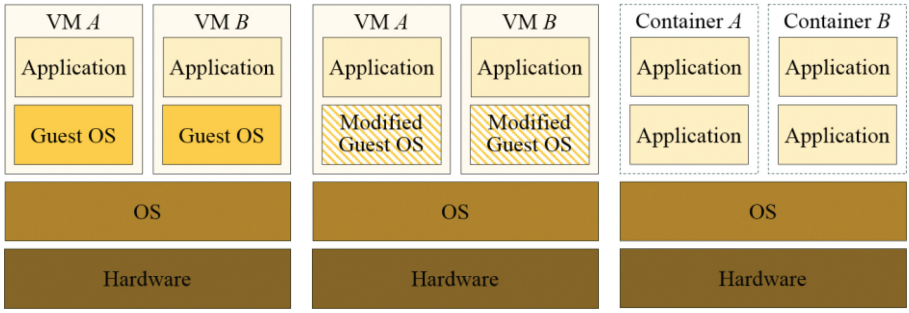
Window manager is a software package that manages windows on a screen and processes user's inputs on appropriate windows [5]. In addition, it performs not only framing windows but also controlling user's actions [6].

X window system is one of the standard window systems. It is based on client-server model and handles shaping and staking of windows, which are separate spaces on a screen. In this system, applications are connected to the server called a compositor, which handles rendering of changed windows. Even though X is commonly used for UNIX-like systems, it has a problem; X has become increasingly heavy, while it used for decades [7]. In other words, it includes unnecessary functions and is difficult to apply new technologies. In order to replace this system, many researchers have developed a novel lightweight window system, Wayland, by removing unnecessary remote-assisted functions and using open source API [8]. It uses the reference compositor called Weston [9], which has the same role of the server in the X window system.

2.2 Virtualization Technologies

To run multiple OSs on a single hardware device, virtualization technology is commonly used. In virtualized system, since the guest OS in each virtual machine does not have permission to directly access hardware resources, it cannot execute privileged instructions. For this reason, hypervisor handles requests of the guest OS to mediate between guest OS and hardware resources. In accordance with its control mechanism, virtualization technologies can be classified into two cases: (1) platform-level virtualization (full virtualization and para-virtualization) and (2) OS-level virtualization.

Full virtualization. Exploits either Binary Translation (BT) or hardware assistance. BT is a software-based approach. When guest OS tries to access hardware resources with privileged instructions, hypervisor translates the instructions and allow guest OS to access the hardware resources, as shown in Fig. 1(a). In case of hardware assistance, hardware such as Intel VT-x and AMD-v supports the process of translation [10].



(a) Platform-level virtualization (full virtualization) (b) Platform-level virtualization (para-virtualization) (c) OS-level virtualization

Fig. 1. Architectures of virtualization technologies

Para-virtualization. Allows modified guest OS to directly access hardware resources [11]. Modified guest OS translates privileged instruction into *hypercall*, a software *trap* from guest to hypervisor, by inserting control codes in the guest OS, as shown in Fig. 1(b). By doing so, para-virtualization reduces interference of hypervisor. So, it outperforms full virtualization in terms of I/O performance.

OS-level virtualization. Packages several guests into containers. Guest does not have its own kernel unlike full virtualization or para-virtualization. In other words, all the guests share a single kernel of the host, as shown in Fig. 1(c). Since OS-level virtualization does not have interference of hypervisor, its performance is similar to non-virtualized system. Moreover, it consumes fewer resources compared to full virtualization or para-virtualization, because it has a single kernel [12].

2.3 Case Study

Hwang et al. proposed a prototype of platform-level virtualization for Xen on ARM CPU architecture [13]. It provides mobile phone security for high trusted computing capability by using Xen hypervisor. In addition, it supports secure execution by classifying secure guests and non-secure guests in user mode. However, it is unsuitable for user isolation in multi-user multi-touch devices, because para-virtualization has communication overhead, where each guest should communicate with other guests via RPC. Linux VServer was proposed for isolation of servers by using OS-level virtualization technology [14]. In this system, user space is separated into Virtual Private Server (VPS) by the container. VPS is almost identical to a real server and is able to be regarded as a guest on the systems of full virtualization or para-virtualization. Soltesz et al. verified the effectiveness of VServer, compared to Xen [15]. They showed that VServer provides more efficient user isolation, in that VPS has no additional software layer such as guest kernel.

3 Why OS-Level Virtualization is Necessary

As aforementioned in Sect. 2, virtualization technology is commonly used for user isolation. However, both full virtualization and para-virtualization are inappropriate to provide user interactions. In virtualized system, each guest has its own kernel, and thus it is difficult to share data among the guests, although it is not impossible. Even if RPC can be used for interactions such as sharing data, it has higher overhead than IPC, due to a long communication path [16]. On the other hand, OS-level virtualization can provide interactions among the guests by using IPC. This is because it just separates user space into containers and makes guests share a single kernel. Thus, it can share file descriptors and exploit shared memory. In other words, it can support both user interactions and user isolation. For this reason, OS-level virtualization is suitable for multi-user multi-touch systems such as tabletops.

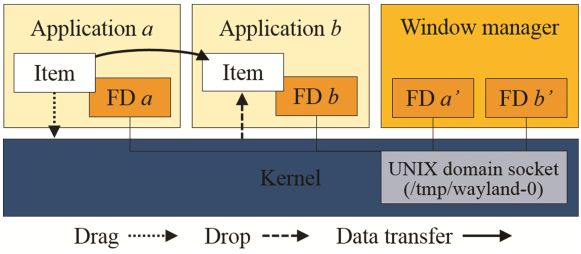


Fig. 2. Process of drag and drop

Now, let us explain the advantage of OS-level virtualization by describing the process of drag and drop. As shown in Fig. 2, when application *a* gets drag event, the application registers *data source* of the item at window manager. Then application *b* receives the item by using the *data source* from window manager and processes drop

event. These two applications can communicate with each other on a single kernel. In this way, window manager exploits UNIX domain socket and file descriptors. Figure 3 depicts that it is possible for applications to communicate on OS-level virtualization framework, because all the applications share a single kernel, even though they belong to different containers that allow them to be isolated logically. In contrast to OS-level virtualization, full virtualization does not allow applications in different guests to communicate in the way of OS-level virtualization, as shown in Fig. 4. The reason is that it has multiple kernels, and *data source* of the item in application *c* cannot be transmitted from guest *c* to guest *d* via UNIX domain socket. In case of para-virtualization, it also has a number of kernels. For this reason, it is difficult to apply both full virtualization and para-virtualization in multi-user multi-touch devices.

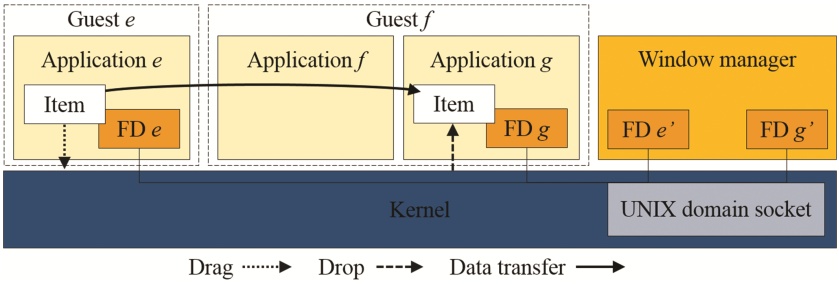


Fig. 3. Process of drag and drop in OS-level virtualization

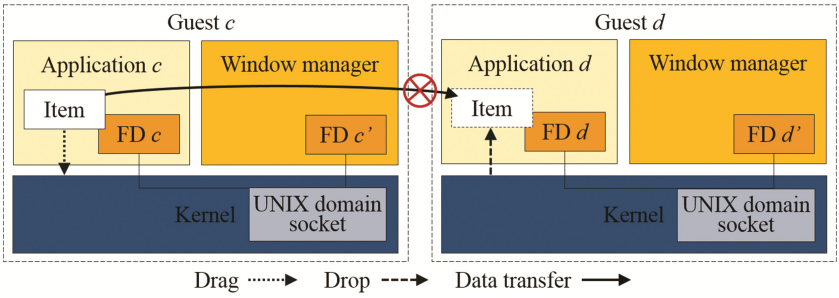


Fig. 4. Process of drag and drop in full virtualization

4 Evaluation

4.1 Experimental Setup

We performed our experiments on a system equipped with Intel Core i5-3570, 4 GB of RAM, and 1 TB Samsung HDD. We set two types of system configuration. The first system configuration is Ubuntu 14.04 LTS with Linux kernel 3.16.0. In this environment, we used two virtualization technologies: (1) Docker and (2) KVM with QEMU and hardware assistance of Intel-VTx. To verify the effectiveness of Docker, we

measured the performance with a series of common benchmarks including Sysbench, ApacheBench, and IOzone. The second system configuration is Ubuntu 15.04 with Linux kernel 3.19.0. We installed Wayland 1.7.0 with Weston 1.7.0 on the system and compared Docker with native system, in terms of frame rate and costs that is incurred by client-server communication and UNIX domain socket communication.

4.2 Experimental Results

CPU performance. We first evaluated CPU performance of each virtualization technology by using Sysbench, which is a multi-threaded benchmark tool [17]. We performed calculation of n prime numbers. As shown in Fig. 5, the performance of Docker is similar to that of native system. It indicates that Docker does not have CPU overhead for the calculation. In contrast to Docker, KVM has the performance degradation by up to 11.68 %, compared to the native system. This is because it has interference of hypervisor and should pass additional layers, such as guest kernels, to access hardware resources.

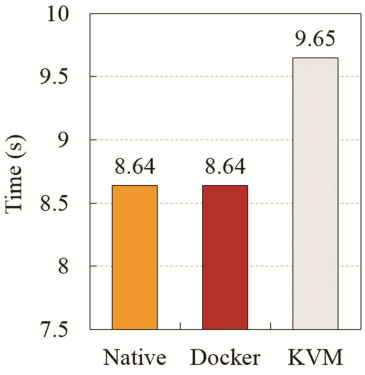


Fig. 5. Comparison of CPU performance

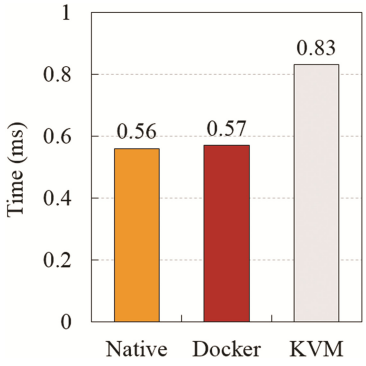


Fig. 6. Comparison of network performance

Network performance. To evaluate network performance of each virtualization technology, we created a server on host and then sent 100 requests to the server with 10 concurrency levels by using the ApacheBench [18], which is a tool for benchmarking Apache Hypertext Transfer Protocol (HTTP) server. We measured round-trip time, which takes for a request to travel from an application to the server and back to the application. As shown in Fig. 6, the performance of Docker is approximately equal to that of native system with only 0.89 % performance degradation. On the other hand, KVM has higher round-trip time by up to 47.32 %. The major reason is that it performs additional processes such as a request transmission between guest and host.

Performance of file operations. We performed experiments related to the operation of directory by using fileop, a file operation benchmark in IOzone [19]. The fileop performs file operations such as access, chdir, stat, open, and close. In this experiment, Docker has the worst performance as shown in Fig. 7. On average, its elapsed time is

1.46x higher than that of native system. This is because Docker has the overhead of namespace isolation and *cgroups*. However, since UI application used in multi-user multi-touch system consists mainly of CPU-bound and network-bound tasks, it is a negligible overhead.

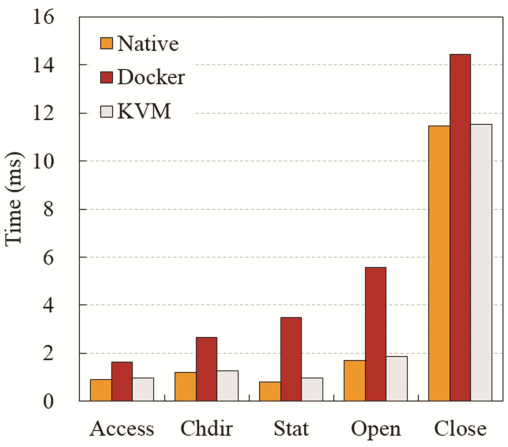


Fig. 7. Comparison of performance of file operations

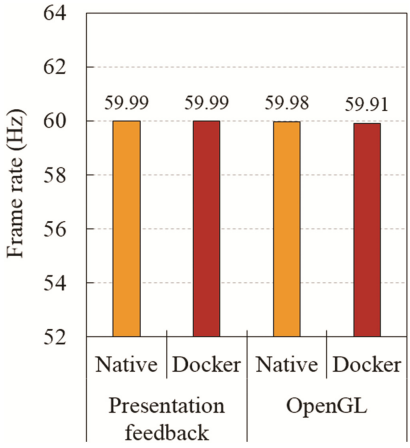


Fig. 8. Comparison of frame rate of UI application

Frame rate of UI applications. We first measured frame rate of an animation using Presentation feedback, which delivers information of display synchronization to the application [20]. As shown in Fig. 8, both native system and Docker have about 60 Hz frame rate. In case of frame rate of the animation using OpenGL, Docker and native system have 59.90 Hz and 59.98 Hz frame rate, respectively. It indicates that Docker does not incur performance degradation for running UI applications, although user space is partitioned with the containers.

Client-server communication. We measured round-trip time, taken for a dummy message to travel back and forth between window manager and application in a container. For comparison, we used *Weston-simple-shm*, which is one of the UI applications in Weston. As shown in Fig. 9, Docker has higher round-trip time by up to 2.55 %, compared to the native system. It implies that there is little overhead of communication between window manager and an isolated application.

UNIX domain socket communication. To evaluate communication costs between two applications, where one is in a container and the other is not, we set up request size as 10 bytes and then sent 1,000,000 requests by using UNIX domain socket in Wayland (/tmp/wayland-0). As shown in Fig. 10, total execution time of Docker has delays of only 0.01 s. In other words, applications inside and outside containers can communicate with each other by using UNIX domain socket without overhead.

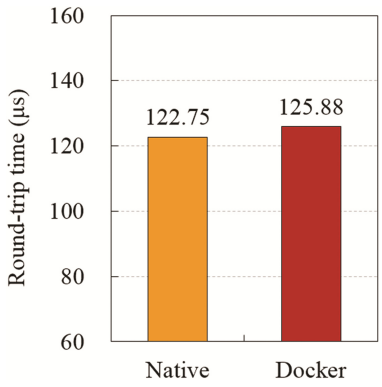


Fig. 9. Comparison of communication costs between window manager and application

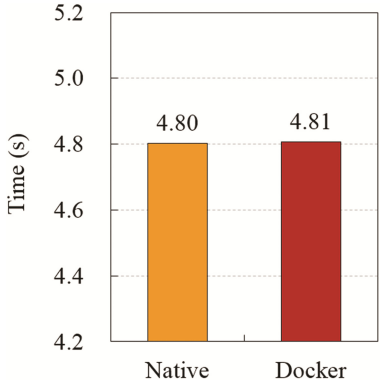


Fig. 10. Comparison of communication costs between applications

5 Conclusion

In multi-user multi-touch systems, user interaction and user isolation should be carefully considered for collaboration of users and tasks and prevention of interferences between tasks. However, since full virtualization and para-virtualization have communication overhead between kernel and applications, OS-level virtualization can be more appropriate candidate for multi-user environment. Moreover, it guarantees user isolation by partitioning the user space into containers.

In this paper, we verified the effectiveness of Docker, which is based on OS-level virtualization, in terms of system performance and window manager operations. To compare Docker with native system and KVM, we performed experiments by using some common benchmarks, such as Sysbench, ApacheBench, and IOzone. Experimental results show that Docker has little performance degradation, despite overheads of namespace isolation and *cgroup*. Moreover, the performance of Docker is nearly equal to that of native system in the experiments that measure frame rates of UI applications and costs of client-server communication and UNIX domain socket communication. Therefore, we demonstrated that Docker is greatly appropriate for multi-user multi-touch systems, when we consider system performance, user interaction, and user isolation.

In future work, we will perform additional evaluation including Xen, which make use of para-virtualization, and then compare it with native system, Docker, and KVM. In addition, we will evaluate the performance of Docker by applying it to a practical device for multiple users and multiple applications.

Acknowledgments. This work was supported by ICT R&D program of MSIP/IITP. [R0126-15-1065, (SW StarLab) Development of UX Platform Software for Supporting Concurrent Multi-users on Large Displays]. Young Ik Eom is the corresponding author of this paper.

References

1. Gartner Identifies the Top 10 Strategic Technology Trends for 2015. http://www.gartner.com/newsroom/id/2867917
2. Hamdan, N., Voelker, S., Borchers, J.: Conceptual framework for surface manager on interactive tabletops. In: CHI 2013 Extended Abstracts on Human Factors in Computing Systems, pp. 1527–1532. ACM, USA (2013)
3. Reshetova, E., Karhunen, J., Nyman, T., Asokan, N.: Security of OS-level virtualization technologies. In: Bernsmed, K., Fischer-Hübner, S. (eds.) NordSec 2014. LNCS, vol. 8788, pp. 77–93. Springer, Heidelberg (2014)
4. What is Docker? https://www.Docker.com/whatisDocker/
5. Scheifler, R., Gettys, J.: The X Window System. ACM Trans. Graph. **5**(2), 79–109 (1986)
6. Stern, D., Herbrich, R., Graepel, T.: Matchbox: large scale online bayesian recommendations. In: 18th International Conference on World Wide Web, pp. 111–120. ACM, USA (2009)
7. Wayland: Motivation. http://wayland.freedesktop.org/docs/html/ch01.html#sect-Motivation
8. Wayland. http://wayland.free-desktop.org/
9. Wayland: the Compositing Manager as the Display Server. http://wayland.freedesktop.org/docs/html/ch01.html#sect-Compositing-manager-display-server
10. Walters, J., Chaudhary, V., Cha, M., Guercio, S., Gallo, S.: A comparison of virtualization technologies for Hpc. In: 22nd International Conference on Advanced Information Networking and Applications, pp. 861–868. IEEE, USA (2008)
11. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: 19th ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, USA (2003)
12. Yu, Y.: Os-level Virtualization and Its Applications. ProQuest, USA (2007)
13. Hwang, J., Suh, S., Heo, S., Park, C., Ryu, J., Park, S., Kim, C.: Xen on arm: system virtualization using xen hypervisor for arm-based secure mobile phones. In: 5th Consumer Communications and Network Conference, pp. 257–287. IEEE, USA (2008)
14. Linux Vserver. http://linux-vserver.org/Overview
15. Soltesz, S., Potzl, H., Fiuczynski, M., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, pp. 275–287. ACM, USA (2007)
16. Kim, K., Kim, C., Jung, S., Shin, H., Kim, J.: Inter-domain socket communications supporting high performance and full binary compatibility on xen. In: 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 11–20. ACM, USA (2008)
17. Sysbench. http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf
18. Apache Http Server Benchmarking Tool. http://httpd.apache.org/docs/2.2/en/programs/ab.html
19. Iozone Filesystem Benchmark. http://www.iozone.org/
20. Weston 1.7.0 Presentation Extension. http://lists.freedesktop.org/archives/wayland-devel/2015-February/019977.html