

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
5 April 2012 (05.04.2012)

(10) International Publication Number
WO 2012/044558 A2

- (51) **International Patent Classification:**
G06F 13/14 (2006.01) **G06F 9/06** (2006.01)
G06F 9/22 (2006.01)
- (21) **International Application Number:**
PCT/US2011/053130
- (22) **International Filing Date:**
23 September 2011 (23.09.2011)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**

61/389,117	1 October 2010 (01.10.2010)	US
61/507,209	13 July 2011 (13.07.2011)	US
61/507,206	13 July 2011 (13.07.2011)	US
61/507,203	13 July 2011 (13.07.2011)	US
61/507,201	13 July 2011 (13.07.2011)	US
61/507,199	13 July 2011 (13.07.2011)	US
13/217,099	24 August 2011 (24.08.2011)	US

Hill, Ontario L4S 0A2 (CA). **CHINCISAN, Octavian** [CA/CA]; 57 Romance Drive, Richmond Hill, Ontario L4S 2R7 (CA). **HANCILA, Cristian** [CA/CA]; 44 Langevin Crescent, Scarborough, Ontario M1C 2B6 (CA). **RUSSELLO, Anthony** [CA/CA]; 246 Jacqueline Boulevard, Hamilton, Ontario L9B 2T6 (CA).

(74) **Agent:** **CROUCH, Robert, G.**; Marsh Fischmann & Breyfogle LLP, 8055 E. Tufts Avenue, Suite 450, Denver, CO 80237 (US).

(71) **Applicant (for all designated States except US):** **IMERJ, LLC** [US/US]; 305 Interlocken Parkway, Broomfield, CO 80021 (US).

(72) **Inventors; and**

(75) **Inventors/Applicants (for US only):** **BENEDEK, Laszlo, Csaba** [CA/CA]; 37 Old Orchard Crescent, Richmond

(81) **Designated States (unless otherwise indicated, for every kind of national protection available):** AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States (unless otherwise indicated, for every kind of regional protection available):** ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,

[Continued on next page]

(54) **Title:** CROSS-ENVIRONMENT COMMUNICATION FRAMEWORK

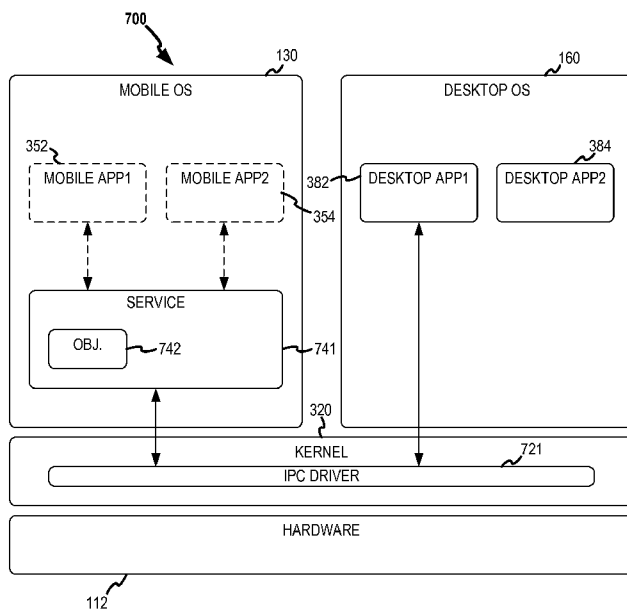


FIG. 7

(57) **Abstract:** A mobile computing device with a mobile operating system and desktop operating system running concurrently and independently on a shared kernel without virtualization. The mobile operating system provides a user experience for the mobile computing device that suits the mobile environment. The desktop operating system provides a full desktop user experience when the mobile computing device is docked to a secondary terminal environment. Applications of the desktop operating system communicate with applications and services of the mobile operating system through a cross-environment communication framework. The cross-environment communication framework may include interfaces to remotable objects allowing processes in the mobile operating system and processes in the desktop operating system to share memory in a thread-safe manner. The mobile computing device may be a smart-phone running the Android mobile operating system and a full desktop Linux distribution on a modified Android kernel.

WO 2012/044558 A2

EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU,
LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK,
SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ,
GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished
upon receipt of that report (Rule 48.2(g))*

CROSS-ENVIRONMENT COMMUNICATION FRAMEWORK

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This Application is a non-provisional of and claims the benefit of the filing date of
5 U.S. Provisional Application Nos. 61/389,117, filed Oct. 1, 2010, entitled “Multi-
Operating System Portable Docking Device”; 61/507,199, filed July 13, 2011, entitled
“Dockable Mobile Software Architecture”; 61/507,201, filed July 13, 2011, entitled
“Cross-Environment Communication Framework”; 61/507,203, filed July 13, 2011,
entitled “Multi-Operating System”; 61/507,206, filed July 13, 2011, entitled “Auto-
10 Configuration of a Docked System in a Multi-OS Environment”; and 61/507,209, filed
July 13, 2011, entitled “Auto-Waking of a Suspended Secondary OS in a Dockable
System,” wherein the entire contents of the foregoing priority applications are
incorporated herein by reference for all purposes.

BACKGROUND

15 [0002] **1. Field**

[0003] This Application relates generally to the field of mobile computing environments,
and more particularly to supporting multiple user environments through the use of
multiple operating systems in a single mobile computing device.

[0004] **2. Relevant Background**

20 [0005] Mobile computing devices are becoming ubiquitous in today’s society. For
example, as of the end of 2008, 90 percent of Americans had a mobile wireless device. At
the same time, the capabilities of mobile devices are advancing rapidly, including
smartphones that integrate advanced computing capabilities with mobile telephony
capabilities. Mobile providers have launched hundreds of new smartphones in the last
25 three years based upon several different platforms (e.g., Apple iPhone, Android,
BlackBerry, Palm, and Windows Mobile). In the U.S., smartphone penetration reached
almost 23% by the middle of 2010, and over 35% in some age-groups. In Europe, the
smartphone market grew by 41% from 2009 to 2010, with over 60 million smartphone
subscribers as of July 2010 in the five largest European countries alone.

[0006] While smartphones are gaining in popularity and computing capability, they provide a limited user experience. Specifically, they typically have an operating system that is modified for mobile device hardware and a restricted set of applications that are available for the modified operating system. For example, many smartphones run

5 Google's Android operating system. Android runs only applications that are specifically developed to run within a Java-based virtual machine runtime environment. In addition, while Android is based on a modified Linux kernel, it uses different standard C libraries, system managers, and services than Linux. Accordingly, applications written for Linux do not run on Android without modification or porting. Similarly, Apple's iPhone uses the

10 iOS mobile operating system. Again, while iOS is derived from Mac OS X, applications developed for OS X do not run on iOS. Therefore, while many applications are available for mobile operating systems such as Android and iOS, many other common applications for desktop operating systems such as Linux and Mac OS X are not available on the mobile platforms.

15 [0007] Accordingly, smartphones are typically suited for a limited set of user experiences and provide applications designed primarily for the mobile environment. In particular, smartphones do not provide a suitable desktop user experience, nor do they run most common desktop applications. As a result, many users carry and use multiple computing devices including a smartphone, laptop, and/or tablet computer. In this instance, each

20 device has its own CPU, memory, file storage, and operating system.

[0008] Connectivity and file sharing between smartphones and other computing devices involves linking one device (e.g., smartphone, running a mobile OS) to a second, wholly disparate device (e.g., notebook, desktop, or tablet running a desktop OS), through a wireless or wired connection. Information is shared across devices by synchronizing data

25 between applications running separately on each device. This process, typically called "synching," is cumbersome and generally requires active management by the user.

SUMMARY

[0009] Embodiments of the present invention are directed to providing the mobile computing experience of a smartphone and the appropriate user experience of a secondary

30 terminal environment in a single mobile computing device. A secondary terminal environment may be some combination of visual rendering devices (e.g., monitor or display), input devices (e.g., mouse, touch pad, touch-screen, keyboard, etc.), and other

computing peripherals (e.g., HDD, optical disc drive, memory stick, camera, printer, etc.) connected to the computing device by a wired (e.g., USB, Firewire, Thunderbolt, etc.) or wireless (e.g., Bluetooth, WiFi, etc.) connection. In embodiments, a mobile operating system associated with the user experience of the mobile environment and a desktop
5 operating system associated with the user experience of the secondary terminal environment are run concurrently and independently on a shared kernel.

[0010] According to one aspect consistent with various embodiments, an application of a desktop operating system communicates with a service of a mobile operating system by requesting to bind to a remotable interface of the service of the mobile operating system,
10 receiving a remotable object from the service of the mobile operating system, the remotable object instantiated by the service of the mobile operating system in response to the request by the desktop operating system, sending a call to the remotable object, the call to the remotable object including a first parcel object, and receiving, by the application of the desktop operating system, a second parcel object from the service of the
15 mobile operating system.

[0011] According to other aspects consistent with various embodiments, the mobile operating system and the desktop operating system run concurrently on a shared kernel. An inter-process communication driver of the shared kernel may marshal the first parcel object to the process space of the service of the mobile operating system. The first parcel
20 object may include an action to be performed by the service of the mobile operating system. The mobile operating system may return the second parcel object after performing the action of the first parcel object. The first parcel object may include a reference to a shared memory space and/or a file descriptor. An inter-process communication library that includes base classes for the remotable object may be
25 compiled against user libraries of the desktop operating system. The inter-process communication library may be a binder library.

[0012] According to other aspects consistent with various embodiments, a first application of a mobile operating system and a second application of a desktop operating system communicate by receiving, by a remote communication service of the mobile
30 operating system, a request by the first application to bind to the remote communication service, instantiating, in response to the request by the first application, a remotable object, passing the remotable object from the remote communication service to the first

application, receiving, by the remote communication service of the mobile operating system, a request by the second application to bind to the remote communication service through an inter-process communication channel, and passing the remotable object from the remote communication service to the second application.

5 [0013] According to other aspects consistent with various embodiments, the mobile operating system and the desktop operating system may run concurrently on a shared kernel. The communication between the first application and the second application may further include receiving, by the remote communication service of the mobile operating system, a call to the remotable object from the first application, the call including a first
10 parcel object. The call may be a synchronous or asynchronous call. The remote communication service of the mobile operating system may return a second parcel object in response to receiving the call to the remotable object. The first parcel object may include a reference to a shared memory space.

[0014] According to other aspects consistent with various embodiments, a computing
15 device includes a computer readable medium storing instructions for a physical processor, the instructions, when executed, causing the processor to perform steps comprising receiving, by a service of a mobile operating system, a request to bind to the service by an application of a desktop operating system, instantiating, by the service of the mobile operating system, a remotable object in response to the request to bind to the service by
20 the application of the desktop operating system, sending, by the service of the mobile operating system, the remotable object through an inter-process communication channel to the application of the desktop operating system, receiving, by the service of the mobile operating system, a call to the remotable object by the application of the desktop operating system, the call to the remotable object including an action to be performed by
25 the service, and returning, from the service of the mobile operating system, a reference to the remotable object in response to receiving the call to the remotable object.

[0015] According to other aspects consistent with various embodiments, the instructions may further cause the processor to perform the steps of running the mobile operating system in a first execution environment on a shared kernel, and concurrently running the
30 desktop operating system in a second execution environment on the shared kernel. The action may include invoking a remotable method of the service. The service may be a bound service.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] Embodiments of the present invention are illustrated in referenced figures of the drawings, in which like numbers refer to like elements throughout the description of the figures.

5 [0017] FIG. 1 illustrates a computing environment that provides multiple user computing experiences, according to various embodiments.

[0018] FIG. 2 illustrates an exemplary system architecture for a mobile computing device, according to various embodiments.

10 [0019] FIG. 3 illustrates an operating system architecture for a computing environment, according to various embodiments.

[0020] FIG. 4 illustrates one embodiment of a mobile operating system for a mobile computing device, according to various embodiments.

[0021] FIG. 5 illustrates aspects of an operating system architecture for a computing environment, according to various embodiments.

15 [0022] FIG. 6 illustrates a desktop operating system that may be employed in an operating system architecture, according to various embodiments

[0023] FIG. 7 illustrates aspects of a cross-environment communications framework, according to various embodiments.

20 [0024] FIG. 8 illustrates a communication flow employing aspects of the cross-environment communications framework, according to various embodiments.

[0025] FIG. 9 illustrates an alternative communication flow employing aspects of the cross-environment communications framework, according to various embodiments.

[0026] FIG. 10 illustrates an alternative communication flow employing aspects of the cross-environment communications framework, according to various embodiments.

25 [0027] FIG. 11 illustrates an alternative communication flow employing aspects of the cross-environment communications framework, according to various embodiments.

DETAILED DESCRIPTION

- [0028] Traditionally, handheld mobile telephony devices (i.e., “handsets”) were developed independently and served a separate and distinct purpose from that of personal computers (“PCs”) such as desktops and laptops. Handheld mobile telephony devices were focused primarily on communication while PCs were focused on computing tasks such as creating and editing documents, text-based communication (e.g., email, etc.), multimedia, and web browsing. However, mobile telephony devices are including ever-increasing computing ability and users increasingly desire convergence of communication and computing capabilities into multi-use mobile devices.
- [0029] For example, mobile telephony devices called “smartphones” that include computing capabilities are increasing in popularity. Many of these smartphones include a mobile operating system (“OS”) running on a mobile processor. While mobile processors and mobile OSs have increased the capabilities of these devices, smartphones have not tended to replace PC environments such as desktop or notebook computers at least because of the limited user experience provided. In particular, for some tasks such as typing or editing documents, a full-size keyboard and large display are easier to use than the user interface components typically found on a smartphone. For example, smartphones typically use a small thumb-style QWERTY keyboard, touch-screen display, click-wheel, and/or scroll-wheel as user interface components. Selecting menu options or items typically involves either using a touch-screen display, or using the click-wheel or scroll-wheel to navigate menus and select items. This interface is suited to the small display screens and limited menu options typically found in smartphones, but not suited to controlling more traditional programs with a larger number of menu options, larger screen area, and user interface built around a pointing device such as a traditional mouse.
- [0030] Embodiments of the invention are directed to an innovative cross-environment communications framework (“CECF”) that allows for fast communications between applications and services of multiple operating systems running independently and concurrently on the same shared kernel. For example, embodiments allow fast communication between applications and services of an Android mobile OS running on a shared kernel and applications and services of a Linux desktop OS running on the same shared kernel. Embodiments include an extension to the Android “Binder” inter-process communication (“IPC”) mechanism that allows for app-to-app communications within

the Android OS. For example, Android OS applications within can register with the remote communication service to be able to communicate (e.g., synchronize data, settings, etc.) in a synchronous and/or asynchronous way. Other embodiments include base classes of the Binder mechanism for a Linux desktop OS that allow communication
5 between applications of the Linux desktop OS and Android OS applications and services. For example, Android applications running on the Android OS can share data, settings, and preferences with Linux applications running on the Linux desktop OS.

[0031] FIG. 1 illustrates a computing environment 100 that provides multiple user computing experiences, according to various embodiments. Computing environment 100
10 includes mobile computing device 110. Mobile computing device 110 includes mobile computing hardware and software components. Hardware components of mobile computing device 110 include mobile processor 114, display 116, I/O device(s) 118, and/or port 120. Software components of mobile computing device 110 include a first OS 130 and a second OS 160. In one embodiment, first OS 130 is a mobile OS and second
15 OS 160 is a desktop OS. When mobile computing device 110 is operated as a stand-alone mobile device, mobile OS 130 presents a typical mobile computing user experience through display 116 and I/O device(s) 118. The mobile computing experience provided by mobile OS 130 typically includes mobile telephony capabilities and a graphical user interface (“GUI”) suited to the mobile environment including display 116 and I/O
20 device(s) 118. For example, display 116 may be a touch-screen display and application programs (i.e., “Apps”) running on mobile OS 130 may be controlled through the GUI of mobile OS 130 on touch-screen display 116.

[0032] Mobile computing device 110 may be connected to I/O devices 144, 146 and/or 148 through port 120. I/O devices 144, 146, and/or 148 may make up a secondary
25 terminal environment 140. In some instances, secondary terminal environment 140 may be more suited to desktop OS 160 than mobile OS 130. For example, secondary terminal environment 140 may include a keyboard 144, pointing device 146, and a display device 148. In these instances, desktop OS 160 can be associated with secondary terminal environment 140 to provide the full capabilities of a notebook or desktop computer
30 environment through secondary terminal environment 140. In other instances, secondary terminal environment 140 may be more suited for mobile OS 130 than desktop OS 160. For example, secondary terminal environment 140 may include a touch-screen display. In

these instances, mobile OS 130 may be associated with secondary terminal environment 140.

[0033] In FIG. 1, port 120 is shown as connected to port 142 of secondary terminal environment 140 through interface 120. However, port 120 may include separate
5 connections to each I/O device 144, 146 and 148 through interface 122. Interface 122 may be any suitable wired or wireless interface or combination of wired and wireless interfaces for connecting devices such as keyboards, monitors, pointing devices, etc. For example, interface 122 may be a combination of a display interface (e.g., VGA, DVI, HDMI, etc.) and a device communications interface (e.g., USB, Bluetooth, Firewire,
10 other serial communications interface, etc.). Alternatively, interface 122 may be a single communications interface that supports both video and device communications signals (e.g., Thunderbolt, etc.).

[0034] Traditionally, the hardware and software development paths for the handset environment and the PC environment have been completely independent because of
15 different use models and competing constraints on product design. PCs, including desktop and laptop computers, are designed to be flexible and powerful. Specifically, PC hardware architecture is typically based around a general purpose PC processor connected to memory, graphics, and external components through various general purpose interfaces on a motherboard. For example, a personal computer motherboard may include a
20 processor connected through a logic chipset to a graphics processor, system memory (e.g., RAM), and various other components through communication interfaces (PCI, USB, ISA, IDE, etc.). Some more highly integrated PC processors include an interface to a graphics processor (e.g., AGP, etc.) and/or interface to system memory (e.g., SDR, DDR, DDR2, DDR3, DRDRAM, etc.) on the processor.

[0035] PC processors are optimized for high processor clock speed and computationally intensive tasks. The personal computer market is presently dominated by processors based on the x86 CPU architecture. Current x86-based PC processors for desktop computers have multiple 64-bit central processing units (“CPUs”) (or “cores”) with clock speeds exceeding 2.5 GHz and power consumption approaching 100 Watts. Current x86-
30 based PC processors for laptop computers typically run at clock speeds up to 2.0 GHz and have power consumption in the range of 15-45 Watts. Because of the large power consumption of these processors, desktop and laptop computers may require cooling

devices such as fans or heat-sinks to remove waste heat from the processor. In addition, the battery life of laptop computers using x86-based PC processors is typically less than four hours.

[0036] In contrast, mobile processors for handsets are optimized for low power
5 consumption and a high level of integration to reduce the overall size of the handset. For example, mobile processors for handsets such as smartphones typically run at clock speeds lower than 2.0 GHz, have power consumption of less than 1 Watt, and integrate functions common to the mobile environment such as graphics controllers, communications interfaces, and camera controllers. The most common mobile processor
10 architectures are reduced instruction set computing (“RISC”) processor architectures. Specifically, the “ARM” mobile processor architecture is currently the predominant architecture for mobile processors designed for smartphones and other ultra-portable and low power computing devices. Some PC processor manufacturers also refer to PC processors designed for use in laptop computers as “mobile processors.” However, as
15 used herein, the term “mobile processor” refers to a processor suited for use in a handset or smartphone, typically consuming less than 1 Watt and integrating mobile functionality.

[0037] Personal computers and handsets also typically have different system resources, input/output (“I/O”) devices, and peripherals. For example, desktop and laptop computers typically have much larger amounts of system memory and storage capacity than
20 handsets. While a typical laptop computer may have more than 2 GB of RAM and a hard-drive with a capacity of more than 250 GB, handsets typically have less than 512 MB of RAM and a solid-state drive with a capacity of less than 32 GB. User interface components of personal computers typically include a display screen larger than 9 inches diagonally, a full keyboard, and pointing device(s) for user input. In contrast, handsets
25 typically include a display screen smaller than 7 inches diagonally and user interface components such as a thumb-style QWERTY keyboard, touch-screen display, click-wheel, and/or scroll-wheel. Peripherals found on personal computers typically include an optical disk drive (e.g., CD, DVD, DVD-RW, etc.), expansion ports (e.g., PCMCIA, SCSI, Express Card, etc.), video output port (e.g., VGA, DVI, HDMI, etc.), and generic
30 device ports (e.g., USB, etc.). In contrast, handsets typically do not have optical disk drives or expansion ports. However, other devices are typically integrated into handsets including wireless communications interface(s) (e.g., GSM, CDMA, LTE, EDGE, WiFi,

WiMax, etc.), GPS chipset, accelerometer, camera(s), and/or solid-state memory port (SD, Memory Stick, etc.).

[0038] Software for personal computers and handsets has also traditionally been developed independently. For example, personal computers including desktops and laptops typically run different operating systems than mobile devices. An operating system is software that manages computer hardware and resources and provides common services for execution of applications software on the computer hardware. Operating systems are generally described as having various abstraction layers, where each layer interfaces with the layer below through an interface.

10 [0039] Generally, the kernel of an operating system refers to the core OS layer that manages the computing devices resources such as the CPU(s) (CPU scheduling), memory, and I/O (including peripheral and file system access). A kernel will usually provide features for low-level scheduling of processes (dispatching), inter-process communication, process synchronization, context switching, manipulation of process control blocks, interrupt handling, process creation and destruction, and process suspension and resumption. The OS kernel may or may not include device drivers. Other layers of the OS interface with the kernel through system calls or an application programming interface (“API”) layer.

20 [0040] Generally, other OS layers include the libraries layer, application framework layer, and application layer. The libraries layer typically includes system libraries and other user libraries. The application framework layer includes services, managers, and runtime environments. The application layer includes user applications, which may run within a runtime environment of the application framework layer. A user interacts with the OS through the OS GUI. The GUI presents menus, buttons, and controls that the user selects to control and use applications running on the OS. Commonly, the term “desktop environment” is used to refer to a style of GUI through which the user interfaces with the OS using icons, windows, toolbars, folders, and/or desktop widgets, and is not limited to a desktop OS. For example, a mobile OS could have a desktop environment, referring to the look and feel of the mobile OS GUI.

30 [0041] Operating systems for personal computers (desktop OSs) were designed for multi-tasking, larger screen areas, and to provide a flexible environment for application

developers. As used herein, the term desktop OS refers to an operating system designed for use with a personal computer environment, for example a desktop or laptop use environment. Examples of desktop OSs include various distributions of Linux, Mac OS X, and Windows 7, among many others.

- 5 [0042] Operating systems for mobile devices (mobile OSs) were developed for the smaller screen area, lower processing power, smaller memory, and smaller disk space typically found on handsets and smartphones. Example mobile OSs include Android, Apple's iOS (for the iPhone and iPad), Microsoft's Windows Mobile (superseded by Windows Phone 7), Nokia's Symbian, and Palm's Palm OS (superseded by HP webOS).
- 10 As used herein, the term mobile OS refers to an operating system designed for use with a mobile environment including running on a low-power processor with reduced system resources compared to the PC environment (i.e., desktop or laptop computer system).

- [0043] Mobile operating systems generally have a particular application development environment that is used to create application programs (i.e., "apps") that run on the mobile OS. The application development environment both facilitates application
- 15 development by providing common tools and APIs for accessing system resources and services, and limits what applications are allowed to do such that the mobile device is able to continue to provide other required functions. For example, incoming phone calls and texts may interrupt a running application to notify the user of the incoming call or
- 20 text.

- [0044] The most widely adopted mobile OS is Google's Android. While Android is based on Linux, it includes modifications to the kernel and other OS layers for the mobile environment and mobile processors. In particular, while the Linux kernel is designed for the x86 CPU architecture, the Android kernel is modified for ARM-based mobile
- 25 processors. Android device drivers are also particularly tailored for devices typically present in a mobile hardware architecture including touch-screens, mobile connectivity (GSM/EDGE, CDMA, Wi-Fi, etc.), battery management, GPS, accelerometers, and camera modules, among other devices.

- [0045] In Android, applications run within the Dalvik virtual machine on an object-oriented application framework designed specifically for the memory and processor speed
- 30 constraints of mobile hardware architectures. Applications are developed for the Dalvik

virtual machine through the Android SDK. In addition, Android does not have a native X Window System nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing GNU/Linux applications or libraries to Android.

- [0046] Apple's iOS operating system (run on the iPhone) and Microsoft's Windows Phone 7 are similarly modified for the mobile environment and mobile hardware architecture. For example, while iOS is derived from the Mac OS X desktop OS, common Mac OS X applications do not run natively on iOS. Specifically, applications are developed for iOS through an SDK to run within the "Cocoa Touch" runtime environment of iOS, which provides basic application infrastructure and support for key iOS features such as touch-based input, push notifications, and system services. Therefore, applications written for Mac OS X do not run on iOS without porting them through the iOS SDK. In addition, it may be difficult to port Mac OS X applications to iOS because of differences between user libraries and application framework layers of the two OSs, and differences in system resources of the mobile and desktop hardware.
- [0047] Because of the differences in processing requirements, system resources, and application development, applications developed for desktop OSs typically do not run on mobile OSs. Additionally, desktop applications may not be easily ported to mobile OSs because they are optimized for a larger screen area, more processing speed, more system memory, different libraries, and commonly a different GUI. As a result, users typically use separate computing devices for each user environment, including a smartphone, tablet computer, laptop computer, and/or desktop computer. In this instance, each device has its own CPU, memory, file storage, and OS.

[0048] Connectivity and file sharing between smartphones and other devices involves linking one device (e.g., smartphone, running a mobile OS) to a second, wholly disparate device (e.g., notebook, desktop, or tablet running a desktop OS), through a wireless or wired connection. Information is shared across devices by synchronizing data between applications running separately on each device. This process, typically called "synching," is cumbersome and generally requires active management by the user.

[0049] Recently, some attempts have been made to provide a more complete user experience with a single mobile computing device. For example, a smartphone may be connected to an external monitor and input devices such as a full keyboard to provide a

more desktop-like user experience, with the mobile OS graphical user interface extended to the larger screen and accepting input from the input devices. However, because the external monitor and input devices are only an extension of the smartphone's operating system and user interface, the capabilities of the docked environment are limited by the smartphone's mobile OS. For example, many software applications available on desktop OSs are not available or have limited functionality on mobile OSs. Accordingly, these devices do not present a full desktop user experience when connected to an external environment.

[0050] Referring still to FIG. 1, computing environment 100 provides multiple user computing experiences without the above limitations. Specifically, because mobile computing device 110 includes multiple OSs, where each OS is suited to a particular computing environment, mobile computing device 110 may be adapted with external devices to provide a broad range of user experiences with a single mobile computing device. For example, a user may have a mobile computing device 110 and a secondary terminal environment 140 that provides the user experience of a laptop when connected to mobile computing device 110. In this instance, desktop OS 160 of the mobile computing device is associated with the secondary terminal environment 140 when the secondary terminal environment is connected to mobile computing device 110. To the user, the full capabilities of desktop OS 160 are available through secondary terminal environment 140.

[0051] FIG. 2 illustrates an exemplary hardware system architecture for mobile computing device 110, according to various embodiments. Mobile computing device hardware 112 includes mobile processor 114 that includes one or more CPU cores 204 and external display interface 220. Generally, mobile computing device hardware 112 also includes I/O device(s) 118, memory 206, storage device(s) 208, touch-screen display controller 210 connected to touch-screen display 116, power management IC 214 connected to battery 216, cellular modem 218, communication device(s) 222, and/or other device(s) 224 that are connected to processor 114 through various communication signals and interfaces. I/O device(s) 118 generally includes buttons and other user interface components that may be employed in mobile computing device 110. For example, I/O device(s) 118 may include a set of buttons, (e.g., back, menu, home, search, etc.), off-screen gesture area, click-wheel, scroll-wheel, QWERTY keyboard, etc. Other device(s)

224 may include, for example, GPS devices, LAN connectivity, microphones, speakers, cameras, accelerometers, and/or MS/MMC/SD/SDIO card interfaces. External display interface 220 may be any suitable display interface (e.g., VGA, DVI, HDMI, etc.).

[0052] Processor 114 may be an ARM-based mobile processor. In embodiments, mobile processor 114 is a mobile ARM-based processor such as Texas Instruments OMAP3430, Marvell PXA320, Freescale iMX51, or Qualcomm QSD8650/8250. However, mobile processor 114 may be another suitable ARM-based mobile processor or processor based on other processor architectures such as, for example, x86-based processor architectures or other RISC-based processor architectures.

[0053] While FIG. 2 illustrates one exemplary hardware implementation 112 for mobile computing device 110, other architectures are contemplated as within the scope of the invention. For example, various components illustrated in FIG. 2 as external to mobile processor 114 may be integrated into mobile processor 114. Optionally, external display interface 220, shown in FIG. 2 as integrated into mobile processor 114, may be external to mobile processor 114. Additionally, other computer architectures employing a system bus, discrete graphics processor, and/or other architectural variations are suitable for employing aspects of the present invention.

[0054] FIG. 3 illustrates OS architecture 300 that may be employed to run mobile OS 130 and desktop OS 160 concurrently on mobile computing device 110, according to various embodiments. As illustrated in FIG. 3, mobile OS 130 and desktop OS 160 are independent operating systems. Specifically, mobile OS 130 and desktop OS 160 may have independent and incompatible user libraries and/or framework layers. Functions and instructions for OS architecture 300 may be stored as computer program code on a tangible computer readable medium of mobile computing device 110. For example, instructions for OS architecture 300 may be stored in storage device(s) 208 of mobile computing device hardware 112.

[0055] As illustrated in FIG. 3, mobile OS 130 has libraries layer 330, application framework layer 340, and application layer 350. In mobile OS 130, applications 352 and 354 run in application layer 350 supported by application framework layer 340 of mobile OS 130. Application framework layer 340 includes manager(s) 342 and service(s) 344 that are used by applications running on mobile OS 130. For example, application

framework layer 340 may include a window manager, activity manager, package manager, resource manager, telephony manager, gesture controller, and/or other managers and services for the mobile environment. Application framework layer 340 may include a mobile application runtime environment that executes applications developed for mobile OS 130. The mobile application runtime environment may be optimized for mobile computing resources such as lower processing power or limited memory space. The mobile application runtime environment may rely on the kernel for process isolation, memory management, and threading support. Libraries layer 330 includes user libraries 332 that implement common functions such as I/O and string manipulation (“standard C libraries”), graphics libraries, database libraries, communication libraries, and/or other libraries.

[0056] As illustrated in FIG. 3, desktop OS 160 has libraries layer 360, framework layer 370, and application layer 380. In desktop OS 160, applications 382 and 384 run in application layer 380 supported by application framework layer 370 of desktop OS 160. Application framework layer 370 includes manager(s) 372 and service(s) 374 that are used by applications running on desktop OS 160. For example, application framework layer 370 may include a window manager, activity manager, package manager, resource manager, and/or other managers and services common to a desktop environment. Libraries layer 360 may include user libraries 362 that implement common functions such as I/O and string manipulation (“standard C libraries”), graphics libraries, database libraries, communication libraries, and/or other libraries.

[0057] In various embodiments of the present disclosure, desktop OS 160 runs in a separate execution environment from mobile OS 130. For example, mobile OS 130 may run in a root execution environment and desktop OS 160 may run in a secondary execution environment established under the root execution environment. Processes and applications running on mobile OS 130 access user libraries 332, manager(s) 342 and service(s) 344 in the root execution environment. Processes and applications running on desktop OS 160 access user libraries 362, manager(s) 372 and service(s) 374 in the secondary execution environment.

[0058] Generally applications developed for mobile OS 130 do not run directly on desktop OS 160, and applications developed for desktop OS 160 do not run directly on mobile OS 130. For example, application 382 running in application layer 380 of desktop

OS 160 may be incompatible with mobile OS 130, meaning that application 382 could not run on mobile OS 130. Specifically, application 382 may use manager(s) 372 and service(s) 374 of application framework layer 370 of desktop OS 160 that are either not available or not compatible with manager(s) 342 and service(s) 344 in application
5 framework layer 340 in mobile OS 130. In addition, application 382 may attempt to access user libraries 362 that exist in libraries layer 360 of desktop OS 160 but are either not available or not compatible with user libraries 332 available in libraries layer 330 of mobile OS 130.

[0059] In OS architecture 300, mobile OS 130 and desktop OS 160 run concurrently on
10 shared kernel 320. This means that mobile OS 130 and desktop OS 160 are running on shared kernel 320 at the same time. Specifically, mobile OS 130 and desktop OS 160 both interface to shared kernel 320 through the same kernel interface 322, for example, by making system calls to shared kernel 320. Shared kernel 320 manages task scheduling for processes of both mobile OS 130 and desktop OS 160. In this regard, mobile OS 130 and
15 desktop OS 160 are running independently and concurrently on shared kernel 320. In addition, shared kernel 320 runs directly on mobile processor 114 of mobile computing device hardware 112, as illustrated by hardware interface 312. Specifically, shared kernel 320 directly manages the computing resources of mobile computing device hardware 112 such as CPU scheduling, memory access, and I/O. In this regard, hardware resources are
20 not virtualized, meaning that mobile OS 130 and desktop OS 160 make system calls through kernel interface 322 without virtualized memory or I/O access.

[0060] There are several known techniques for providing multiple OSs on the same computing device. However, none of these techniques provide multiple different OSs running concurrently and independently on a shared kernel. More particularly, none of
25 these techniques provide a solution for a mobile OS and a desktop OS running on a shared kernel.

[0061] In one technique, known as dual-boot, multiple OSs are loaded on the computing device one at a time. For example, at boot time, a user may select one OS from multiple available OSs to be run on the device, where each OS has its own kernel, libraries,
30 framework, and applications. The system then boots up into that operating system and the other OS(s) are not running (i.e., no processes of the other OS(s) are loaded concurrently

with the running OS). Therefore, this technique does not run multiple OSs on a shared kernel, nor does this technique run multiple OSs concurrently.

[0062] Another technique for running multiple OSs on the same device is to use a Virtual Machine Manager (“VMM”), or “Hypervisor.” A VMM or Hypervisor runs directly on
5 the hardware and separates the individual kernels of each OS from the hardware, controlling which computer hardware resources are available to each OS at any given time. A Hypervisor effectively creates multiple virtual machines from one device, such that each OS sees a separate virtual machine. Therefore, multiple OSs running on the same device through Hypervisor and VMM are not running on a shared kernel. The
10 Hypervisor adds system overhead due to each OS having to access system resources through virtualization in the Hypervisor. Additionally, because the Hypervisor must allocate CPU and other computing resources, each OS may not be able to effectively schedule processes and tasks.

[0063] Yet another technique for running multiple OSs on the same device is to use a
15 hosted virtual machine. In this technique, each OS has its own kernel, with the kernel of the guest OS running on a virtual machine in the host OS. The virtual machine may be a virtualized hardware platform different than the physical hardware platform. The virtual machine in the host OS may be implemented in the kernel of the host OS. In this instance, the kernel of the host OS acts as a hypervisor through which the kernel of the guest OS
20 accesses the processor and hardware resources. Regardless of where the virtual machine is implemented in this technique, the host OS and the guest OS have separate kernels. Therefore, hosted virtual machines do not have multiple OSs running on a shared kernel. System performance using this technique may be reduced due to virtualization of hardware resources for the guest OS.

[0064] Another form of virtualization is operating system level virtualization. In this
25 technique, multiple isolated user-space instances may be created on the kernel of an operating system, which look like separate OS instances from the point of view of users of each user-space instance. In this technique, the host OS and guest OS(s) must be the same OS. Accordingly, this technique does not provide a solution for a mobile OS and
30 desktop OS running independently and concurrently on a shared kernel. Further, similarly to a hosted virtual machine, this technique uses disk space and memory virtualization for

the guest OS(s). Accordingly, this technique does not provide direct access to memory and system resources for each concurrent OS.

[0065] These techniques of running multiple OSs have limitations with regard to running both operating systems concurrently and independently. For example, virtualization
5 involves setting up a distinct address space for the guest OS and simulating I/O to the guest OS. Therefore, access to hardware including system memory has higher overhead for the guest OS using virtualization. Additionally, techniques using Hypervisors result in lack of certainty in process control of each OS. Specifically, the Hypervisor manages the amount of CPU time allocated to each OS, and each OS then allocates CPU time for
10 processes within the OS, without knowledge of what is occurring in the other OS. In this regard, high priority processes within one OS may not be given the required CPU time to complete their tasks because the OS is sharing CPU time through the Hypervisor, which cannot account for the relative priorities of processes running within each OS. Because processing power may be limited in mobile processor architectures relative to desktop
15 processor architectures, techniques that depend on virtualization, including hypervisors, and operating system level virtualization, may not offer optimal performance for a desktop OS running concurrently with a mobile OS on a mobile processor.

[0066] In one embodiment consistent with OS architecture 300, an Android mobile OS and a full Linux OS run independently and concurrently on a modified Android kernel. In
20 this embodiment, the Android OS may be a modified Android distribution while the Linux OS (“Hydroid”) may be a modified Debian Linux desktop OS. FIGS. 4-6 illustrate Android mobile OS 430, Android kernel 520, and Hydroid OS 660 that may be employed in OS architecture 300 in more detail, according to various embodiments.

[0067] As illustrated in FIG. 4, Android OS 430 includes a set of C/C++ libraries in
25 libraries layer 432 that are accessed through application framework layer 440. Libraries layer 432 includes the “bionic” system C library 439 that was developed specifically for Android to be smaller and faster than the “glibc” Linux C-library. Libraries layer 432 also includes inter-process communication (“IPC”) library 436, which includes the base classes for the “Binder” IPC mechanism of the Android OS. Binder was developed
30 specifically for Android to allow communication between processes and services. Other libraries shown in libraries layer 432 in FIG. 4 include media libraries 435 that support recording and playback of media formats, surface manager 434 that managers access to

the display subsystem and composites graphic layers from multiple applications, 2D and 3D graphics engines 438, and lightweight relational database engine 437. Other libraries that may be included in libraries layer 432 but are not pictured in FIG. 4 include bitmap and vector font rendering libraries, utilities libraries, browser tools (i.e., WebKit, etc.),
5 and/or secure communication libraries (i.e., SSL, etc.).

[0068] Application framework layer 440 of Android OS 430 provides a development platform that allows developers to use components of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, etc. Framework layer 440 also allows applications to publish their capabilities and
10 make use of the published capabilities of other applications. Components of application framework layer 440 of Android mobile OS 430 include activity manager 441, resource manager 442, window manager 443, dock manager 444, hardware and system services 445, desktop monitor service 446, multi-display manager 447, and remote communication service 448. Other components that may be included in framework layer 440 of Android
15 mobile OS 430 include a view system, telephony manager, package manager, location manager, and/or notification manager, among other managers and services.

[0069] Applications running on Android OS 430 run within the Dalvik virtual machine 431 in the Android runtime environment 433 on top of the Android object-oriented application framework. Dalvik virtual machine 431 is a register-based virtual machine,
20 and runs a compact executable format that is designed to reduce memory usage and processing requirements. Applications running on Android OS 430 include home screen 451, email application 452, phone application 453, browser application 454, and/or other application(s) (“App(s)”) 455.

[0070] For these reasons, applications written for Android do not generally run on
25 Hydroid OS 660 and applications written for standard Linux distributions do not generally run on Android OS 430. In this regard, applications for Android OS 430 and Hydroid OS 660 are not bytecode compatible, meaning compiled and executable programs for one do not run on the other.

[0071] FIG. 5 illustrates modified Android kernel 520 in more detail, according to
30 various embodiments. Modified Android kernel 520 includes touch-screen display driver 521, camera driver(s) 522, Bluetooth driver(s) 523, shared memory allocator 524, IPC

driver(s) 525, USB driver(s) 526, WiFi driver(s) 527, I/O device driver(s) 528, and/or power management module 530. I/O device driver(s) 528 includes device drivers for external I/O devices, including devices that may be connected to mobile computing device 110 through port 120. Modified Android kernel 520 may include other drivers and
5 functional blocks including a low memory killer, kernel debugger, logging capability, and/or other hardware device drivers.

[0072] FIG. 6 illustrates Hydroid OS 660 in more detail, according to various embodiments. Hydroid is a full Linux OS that is capable of running almost any application developed for standard Linux distributions. In particular, libraries layer 662 of
10 Hydroid OS 660 includes Linux libraries that support networking, graphics processing, database management, and other common program functions. For example, user libraries 662 may include the standard Linux C library (glibc) 664, Linux graphics libraries 662 (e.g., GTK, etc.), Linux utilities libraries 661, Linux database libraries, and/or other Linux user libraries. Applications run on Hydroid within an X-Windows Linux graphical
15 environment using X-Server 674, window manager 673, and/or desktop environment 672. Illustrated applications include word processor 681, email application 682, spreadsheet application 683, browser 684, and other application(s) 685.

[0073] In one embodiment, Hydroid OS 660 includes components of a cross-environment communication framework that facilitates communication with Android OS 430 through
20 shared kernel 520. These components include IPC library 663 that includes the base classes for the Binder IPC mechanism of the Android OS and remote communications service 671.

[0074] In one embodiment, Hydroid OS 660 is run within a chrooted (created with the
25 'chroot' command) secondary execution environment created within the Android root environment. Processes and applications within Hydroid OS 660 are run within the secondary execution environment such that the apparent root directory seen by these processes and applications is the root directory of the secondary execution environment. In this way, Hydroid OS 660 can run programs written for standard Linux distributions without modification because Linux user libraries 662 are available to processes running
30 on Hydroid OS 660 in the chrooted secondary execution environment.

[0075] In preemptive multi-tasking operating systems, direct communication between independent processes is typically prevented for security and reliability reasons. Instead, a form of IPC must be used to send information across process boundaries (i.e., between process address spaces). Objects sent from one process to another process must be
5 marshaled across the process boundaries. Marshaling is the process of packaging and unpackaging data and parameters so an IPC communication can take place. Some forms of IPC can be used to pass information between processes on different computers on a network. One method of IPC is to allow a process in one address space to cause a subroutine or procedure to execute in another address space. This method of IPC is called
10 a remote procedure call (“RPC”).

[0076] Some methods of IPC are platform independent. For example, Internet sockets or network sockets are a mechanism of bidirectional IPC across an Internet Protocol (“IP”) based computer network. Internet sockets can be used to deliver data packets to application processes or threads based on local or remote IP addresses and port numbers.
15 The operating systems of each local or remote machine maintain mappings of each socket to the appropriate application process or thread.

[0077] Referring back to FIG. 3, OS architecture 300 includes OS 130 and desktop OS 160 running concurrently on shared kernel 320. As illustrated in FIG. 1, mobile computing device 110 can be docked with secondary terminal environment 140.
20 Secondary terminal environment 140 may have a user experience profile that is associated with desktop OS 160. In this instance, mobile OS 130 provides a mobile computing experience through touch-screen display 116 and I/O devices 118 of mobile computing device hardware 112. At the same time, desktop OS 160 provides a desktop computing experience through secondary terminal environment 140.

[0078] Accordingly, the user is able to use and interact with mobile applications (for example, applications 352 and 354) running on mobile OS 130 through the mobile computing experience provided on the user interface components of mobile computing device 110 while using and interacting with desktop applications (for example, applications 382 and 384) running on desktop OS 160 through secondary terminal
25 environment 140. In this instance, applications of mobile OS 130 are independent from applications of desktop OS 160. In this regard, settings, user preferences, user data, and application states of applications of mobile OS 130 are separate and independent from
30

settings, user preferences, user data, and application states of applications of desktop OS 160. For example, application 352 of mobile OS 130 and application 382 of desktop OS 160 may be spreadsheet applications. When the user opens a spreadsheet file with application 352 and begins editing data, the spreadsheet file and associated data are
5 locked by application 352. Application 382 is not able to access the edited spreadsheet data as that data is maintained in the process space of application 352.

[0079] Embodiments of the invention include a cross-environment communications framework that extends IPC between applications and services of a mobile OS running on a shared kernel and applications and services of a desktop OS running on the same shared
10 kernel. The cross-environment communications framework supports communication between applications of the desktop OS and applications and services of the mobile OS. The cross-environment communications framework also supports communication between mobile OS applications and other mobile OS applications.

[0080] FIG. 7 illustrates one embodiment of a cross-environment communications
15 framework. In cross-environment communications framework 700, mobile application 352, service 741, and other mobile application(s) and service(s) 354 run on mobile OS 130 while desktop applications 382 and 384 run on desktop OS 160. Each application and service includes one or more processes, with each process having its own process space.

[0081] Services are components that perform background operations (i.e., not directly
20 involving user interaction with the application) for applications and/or assist in interactions between applications and hardware. While various operating systems may refer to these components by other names, a service, as used herein, is intended to be construed broadly as a component including one or more processes that implements functionality not directly involving user interaction that may be used by applications, the
25 operating system, and/or other services.

[0082] As illustrated in FIG. 7, desktop application 382, running on desktop OS 160, communicates with service 741 running on mobile OS 130 through an inter-process communication channel that includes IPC driver 721 in kernel 320. Initially, desktop application 382 sends a call through IPC driver 721 to bind to service 741. Service 741
30 creates an instance of a remotable object 742, which defines the IPC interface through which desktop application 382 communicates with service 741. Service 741 returns

remotable object 742 to desktop application 382 through IPC driver 721. Subsequently, desktop application 382 can make calls to remotable object 742 to communicate with service 741 through IPC driver 721.

[0083] In one form of communication of cross-environment communication framework
5 700, desktop application 382 may call remotable methods of service 741 by making calls to remotable object 742. A call to remotable object 742 may include zero or more parameters passed to a remotable method of service 741 and/or zero or more parameters to be returned from the remotable method of service 741. Parameters are parceled by the IPC interface to the remotable object and marshaled across process boundaries by IPC
10 driver 721. In this regard, desktop application 382 may pass parameters in a call to remotable object 742 in the form of a parcel object. IPC driver 721 marshals the parcel object to the process space of service 741. Service 741 unparcels the parcel object and performs the remotable method in its process space using the unparceled parameters and returns another parcel object to desktop application 382. The returned parcel object may
15 include a reference to the remotable object.

[0084] Calls to the remotable object may be synchronous or asynchronous. For example, if desktop application 382 makes a synchronous call to the remotable object, the calling thread in desktop application 382 blocks and waits for the return from the remotable object before continuing. Alternatively, if desktop application 382 makes an
20 asynchronous call to the remotable object, the calling thread in desktop application 382 continues and does not wait for the call to the remotable object to return.

[0085] Remotable object 742 allows desktop application 382 and service 741 to both access a shared memory space in a thread-safe manner. For example, a reference to the shared memory space may be included in parcel objects passed by calls to remotable
25 object 742. Because remotable object 742 is reference counted across process boundaries, application 382 and service 741 can share memory across process boundaries in a thread-safe manner. In various instances, the reference to a shared memory space may be a file descriptor or file handle.

[0086] In embodiments, service 741 may be a core service of the mobile OS such as a
30 telephony service, camera service, location service. Service 741 may also be a custom

service that performs operations related to an application of mobile OS 130 or desktop OS 160.

[0087] In one embodiment, service 741 may be a service that enables app-to-app communication. For example, service 741 may be a remote communications service that implements aspects of the cross-environment communications framework to provide app-to-app communication between applications running on mobile OS 130. The remote communication service may also enable communication between desktop applications running on desktop OS 160 and mobile applications running on mobile OS 130. In one embodiment, remote communication service 448 of Android OS 430 facilitates app-to-app communication between applications running on Android OS 430 and/or Hybrid OS 660.

[0088] For example, mobile application 352 may bind to service 741. Service 741 then instantiates remotable object 742 and returns remotable object 742 to mobile application 352. Mobile application 354 may also bind to service 741 and service 741 may return remotable object 742 to mobile application 354. With mobile applications 352 and 354 may communicate through calls to remotable object 742. For example, mobile application 352 may pass parcel objects using calls to remotable object 742. The parcel objects may include data and/or references to shared memory. Mobile application may access the shared memory by also making calls to remotable object 742. In the same manner, desktop application 382 may also bind to service 741 and communicate with mobile applications 352 and 354 using calls to remotable object 742.

[0089] In one embodiment, Android mobile OS 430 and a Linux desktop OS 660 run concurrently on shared kernel 520. The Android OS includes a lightweight remote procedure call mechanism called "Binder." The Binder mechanism uses shared memory to achieve efficient, secure IPC. The Binder mechanism includes Binder IPC driver 525 in the Android kernel 520 and user class libraries including Binder IPC library 436 that contain the base classes for Binder objects. In Android OS 430, Binder IPC library 436 is compiled against Bionic C library 439. A Binder reference is a descriptor maintained by Binder driver 525 to an interface of a remotable Binder object. A process can call the remotable Binder object through the interface. The Android Binder mechanism provides reference counting and manages mapping of object references across processes. However, the Android Binder mechanism only provides communication between Android

applications and globally registered Android services. Therefore, Android applications may not communicate directly with each other through the Binder mechanism.

[0090] The IPC mechanisms available in the Linux OS are pipes, named pipes, system V message queues, system V semaphores, system V shared memory segments, and
5 networking sockets. These IPC mechanisms are not directly compatible with the Binder IPC mechanism in the Android OS or the Binder kernel driver used for the Android Binder IPC mechanism. Therefore, Linux applications running cannot use these standard Linux IPC mechanisms to communicate with applications and/or service(s) running on an Android OS.

10 [0091] In one embodiment, the cross-environment communications framework extends the Binder mechanism to allow applications of Linux desktop OS 660 to access services of Android OS 430. In one embodiment, the cross-environment communication framework includes a remote communication service that allows app-to-app
communication within Android OS 430. In other embodiments, the remote
15 communication services allows Linux applications running on Linux desktop OS 660 to communicate with Android applications running on Android OS 430.

[0092] In embodiments, the cross-environment communications framework also supports use of the user datagram protocol (“UDP”) for cross-environment communication. UDP is part of the suite of protocols of the Internet Protocol (“IP”). UDP is an asynchronous
20 data transmission mechanism without an implicit handshaking dialogue. Accordingly, UDP does not provide a reliable transmission protocol, not does it guarantee data ordering or data integrity during transmission. However, because UDP has minimal overhead, it may be a preferable transmission mechanism for time-sensitive information in a real-time system. For example, where dropping packets is preferable to delaying transmission, UDP
25 may be a preferred transmission mechanism.

[0093] FIG. 8 illustrates communication flow 800 employing aspects of the cross-environment communications framework, according to various embodiments. In communication flow 800, Android application 852, running on Android OS 430, registers with remote communication service 448 by calling a method of the Android OS context
30 as illustrated by process step 802. In response, remote communication service 448 returns an interface to a remotable object 804 (e.g., Binder object) at step 806. At step 808,

Android application 854 registers with remote communication service 448. Remote communication service 448 returns an interface to the same remotable object 804 to Android application 854 at step 810.

5 [0094] Android application 852, running on Android OS 430, can communicate with Android application 854 using remotable object 804 by making calls to the remotable object. Calls to the remotable object may include remote method invocations and/or parceled data. The parceled data may include various data types including integers, strings, arrays, and/or file descriptors, among other possible data types. At step 812 of communication flow 800, Android application 852 invokes remote method FN1 of remote communication service 448. Kernel IPC driver 525 marshals the remote call across the process boundaries and remote communication service 448 receives the call to the remotable object at step 814. Remote communication service 448 then unparcels data of the remote call and runs the remote method in its process space using the data at step 816. Remote communication service 448 returns another parcel object through the interface to the remotable object at steps 818 and 820.

10 [0095] At step 822, Android application 854 makes a call to remotable object 804. For example, Android application 854 may invoke a remote method (FN2) of remote communication service 448. Application 854 may pass a parcel object to the remote method that includes parameters for the remote method. Kernel IPC driver 525 marshals the parcel object to the process space of remote communication service 448 and remote communication service 448 receives the parcel object at step 824. Remote communication service 448 then unparcels the data and runs the remote method in its process space using the data at step 826. Remote communication service 448 returns another parcel object through the interface to the remotable object at step 828. Kernel IPC driver 525 marshals the parcel object across the process boundaries and Android application 854 receives the parcel object at step 830.

25 [0096] In one embodiment, remote communication service 448 may be a bound service of Android OS 430. In this embodiment, the bound service is created by Android OS 430 when an client binds to the service through the Context class of Android OS 430. The bound service runs as long as clients (applications and/or services that bind to the service) are bound to the service and is destroyed when the last client unbinds from the service.

The bound service may implement a shared memory interface allowing clients to share user preferences, settings, and/or user data.

[0097] Communications flow 800 provides thread-safe cross-process communication for sharing data between Android applications 852 and 854 running on Android OS 430.

- 5 Android applications 852 and 854 may share data in a shared memory space in a thread-safe manner by passing references to the shared data using remotable object 804. For example, Android applications 852 and 854 may pass a parcel object that includes a reference to a shared memory space through remotable object 804.

[0098] Communications flow 800 illustrates synchronous communication between

- 10 Android applications 852 and 854 and remote communication service 448. However, communication between Android applications 852 and 354 and remote communication service 448 may be asynchronous, meaning that Android applications 852 and 854 will return immediately from calls to the remotable object and not wait for a result from the remote method invocation.

- 15 [0099] FIG. 9 illustrates communication flow 900 employing aspects of the cross-environment communications framework, according to various embodiments. In communication flow 900, application 682 running on Hydroid OS 660 communicates with service 941 of Android OS 430.

[0100] Service 941 exposes a Binder interface to clients and at step 902, Hydroid

- 20 application 982 registers with service 941 by binding to service 941 using the exposed interface through Binder kernel driver 525. Service 941 returns in interface to a remotable object 904, which Hydroid application 982 receives at step 908. At step 910, Hydroid application 382 makes a call to remotable object 904. For example, Hydroid application 382 may invoke a remotable method (FN1) of service 941. The remote method call may
25 include a parcel object that includes parameters for the remote method call. Binder driver 525 marshals the parcel object to the process space of service 941 and service 941 receives the data at step 912. Service 941 unparcels the data and runs the method in its process space using the unparceled data at step 914. Service 941 returns a parcel object that includes return values of the method through binder driver 525 at steps 916 and 918.
30 Using communication flow 900, Hydroid application 982, running on Hydroid OS 660, can access methods and data of service 941 of Android OS 430.

[0101] Communications flow 900 provides thread-safe cross-process communication for sharing data between processes of Android OS 430 and applications of Hydroid OS 660. For example, Hydroid application 982 may access shared data of services of Android OS 430 through shared memory by passing references to the shared memory using remotable
5 object 904.

[0102] Communication flow 900 may be used to allow applications running on Hydroid OS 660 to access various services of Android OS 430. For example, service 941 of Android OS 430 may be a core service of Android OS 430 such as a telephony service, camera service, location service, etc. Alternatively, service 941 may be a custom service
10 such as remote communication service 448.

[0103] FIG. 10 illustrates communication flow 1000 employing the cross-environment communications framework, according to various embodiments. Initially, remote communication service 448 exposes a binder interface to clients. At step 1002, Android application 852 of Android OS 430 registers with remote communication service 448
15 (i.e., binds to remote communication service 448). Remote communication service 448 instantiates a remotable object 1004 and returns remotable object 1004 to Android application 852 at step 1006.

[0104] At step 1008, Hydroid application 982 of Hydroid OS 660 registers with remote communication service 448 by binding to remote communication service 448 using the exposed interface through binder kernel driver 525. Remote communication service 448
20 returns remotable object 1004, which Hydroid application 982 receives at step 1014.

[0105] At step 1016, Android application 852 makes a call to remotable object 1004. For example, Android application 852 may invoke a remote method (FN1) of remote communication service 448. Android application 852 may pass a parcel object to the
25 remote method that includes parameters for the remote method. The parcel object may include a reference to a shared memory space. Binder driver 525 marshals the parceled data across the process boundaries and remote communication service 448 receives the data at step 1018. Remote communication service 448 unparcels the data and runs the method in its process space using the unparceled data at step 1020 and returns the
30 remotable object through binder driver 525 at steps 1022 and 1024.

[0106] At step 1026, Hydroid application 982 calls a remote method (FN2) of remote communication service 448 through the remotable object 1004. The remote method call may include data that is parceled into remotable object 1004. Binder driver 525 marshals the parceled data across the process boundaries and remote communication service 448
5 receives the data at step 1028. Remote communication service 448 then unparcels the data and runs the method in its process space using the data at step 1030 and returns remotable object 1004 through binder driver 525 at steps 1032 and 1034.

[0107] Communications flow 1000 provides thread-safe cross-process communication for sharing data between applications of Android OS 430 and applications of Hydroid OS
10 660. For example, Hydroid application 982 of Hydroid OS 660 and Android application 852 of Android OS 430 may share data through shared memory by passing references to the shared memory using remotable object 1004. In communications flow 1000, Android application 852 and Hydroid application 982 block on return of the remote method invocation. Accordingly, communications flow 1000 provides synchronous
15 communication between Android application 852 and Hydroid application 982. However, remote method invocations from Android application 352 and Hydroid application 982 may be asynchronous, meaning that applications 852 and 982 will return immediately from calls to remotable object 1004 and not wait for a result to be returned.

[0108] In one embodiment of communications flow 1000, remote communication service
20 448 is a bound service of Android OS 430. In this embodiment, the bound service is created by Android OS 430 when either an Android application or a Hydroid application binds to the service. The service may implement a shared memory interface allowing Android applications and Hydroid applications to share user preferences, settings, and/or user data.

[0109] In one example consistent with communications flow 1000, Android application
25 852 and Hydroid application 982 are spreadsheet programs. User data is stored in shared memory accessed through a remotable object. In this example, Android spreadsheet application 852 and Hydroid spreadsheet application 982 both may manipulate the user data by passing a parcel object that includes a reference to the shared memory through
30 calls to remotable object 1004. In this way, Android spreadsheet application 852 and Hydroid spreadsheet application 982 can manipulate user data of a spreadsheet stored in shared memory in a thread-safe manner.

[0110] Accordingly, communication flow 1000 allows applications running on Hydroid OS 660 to communicate with applications running on Android OS 430 using the Android Binder mechanism. In this way, Android application 852 running on Android OS 430 and Hydroid application 982 running on Hydroid OS 660 can share settings, user preferences, and/or user data fast and efficiently.

[0111] FIG. 11 illustrates communication flow 1100 that employs aspects of the cross-environment communications framework, according to various embodiments. As illustrated in FIG. 11, remote communication service 448 of Android OS 430 includes UDP communication module 1146 and remote communication service 671 of Hydroid OS 660 includes UDP communication module 1176. In communication flow 1100, Hydroid application 982 of Hydroid OS 660 communicates with Android application 852 of Android OS 430 through UDP modules 1146 of Android OS 430 and 1176 of Hydroid OS 660.

[0112] In one example, Android application 852 binds to a socket of socket driver(s) 1121 in shared kernel 520 through UDP module 1146. Hydroid application 982 of Hydroid OS 660 also binds to the socket of socket driver(s) 1121 through UDP module 1176. Android application 852 then sends UDP packets to the socket and Hydroid application 982 receives the UDP packets through UDP module 1176 of Hydroid OS 660.

[0113] As described above, UDP does not include handshaking for providing reliability or implicit data ordering. Accordingly, communication flow 1100 may be suitable for communications between applications of Android OS 430 and Hydroid OS 660 where data reliability is not a primary concern. For example, communication flow 1100 may be used for a Linux streaming audio client to receive a streaming audio signal from an Android streaming audio application or service. In this instance, it may be more important for the Linux streaming audio client to keep receiving packets through communication flow 1100 rather than wait for handshaking or resending of any dropped packets.

[0114] As described above, in one embodiment an Android mobile OS and a Linux desktop OS (“Hydroid”) run concurrently on the same shared kernel of a single mobile computing device. The Android mobile OS provides a mobile computing experience through mobile computing hardware and the Linux desktop OS provides a desktop computing experience through a secondary terminal environment having a user

experience profile associated with the Linux OS. However, other OS combinations are contemplated as within various embodiments of the invention. For example, various aspects of the invention may be used to run Windows Mobile and Windows 7 on a shared kernel or sharing common kernel-mode processes. As another example, iOS and Mac OS X running on a shared kernel is also within the scope of various embodiments. Furthermore, aspects of the invention may be used advantageously by combinations of embedded OSs and desktop or mobile OSs running on a shared kernel.

[0115] The foregoing description has been presented for purposes of illustration and description. Furthermore, the description is not intended to limit embodiments of the invention to the form disclosed herein. While a number of exemplary aspects and embodiments have been discussed above, those of skill in the art will recognize certain variations, modifications, permutations, additions, and sub-combinations thereof.

[0116] The various operations of methods described above may be performed by any suitable means capable of performing the corresponding functions. The means may include various hardware and/or software component(s) and/or module(s), including, but not limited to a circuit, an application specific integrated circuit (ASIC), or processor.

[0117] The various illustrative logical blocks, modules, and circuits described may be implemented or performed with a general purpose processor, a digital signal processor (DSP), an ASIC, a field programmable gate array signal (FPGA), or other programmable logic device (PLD), discrete gate, or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein. A general purpose processor may be a microprocessor, but in the alternative, the processor may be any commercially available processor, controller, microcontroller, or state machine. A processor may also be implemented as a combination of computing devices, e.g., a combination of a DSP and a microprocessor, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other such configuration.

[0118] The steps of a method or algorithm described in connection with the present disclosure, may be embodied directly in hardware, in a software module executed by a processor, or in a combination of the two. A software module may reside in any form of tangible storage medium. Some examples of storage media that may be used include random access memory (RAM), read only memory (ROM), flash memory, EPROM

memory, EEPROM memory, registers, a hard disk, a removable disk, a CD-ROM and so forth. A storage medium may be coupled to a processor such that the processor can read information from, and write information to, the storage medium. In the alternative, the storage medium may be integral to the processor. A software module may be a single
5 instruction, or many instructions, and may be distributed over several different code segments, among different programs, and across multiple storage media.

[0119] The methods disclosed herein comprise one or more actions for achieving the described method. The method and/or actions may be interchanged with one another without departing from the scope of the claims. In other words, unless a specific order of
10 actions is specified, the order and/or use of specific actions may be modified without departing from the scope of the claims.

[0120] The functions described may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the functions may be stored as one or more instructions on a tangible computer-readable medium. A storage medium may be
15 any available tangible medium that can be accessed by a computer. By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM, or other optical disk storage, magnetic disk storage, or other magnetic storage devices, or any other tangible medium that can be used to carry or store desired program code in the form of instructions or data structures and that can be accessed by a computer.
20 Disk and disc, as used herein, include compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk, and Blu-ray® disc where disks usually reproduce data magnetically, while discs reproduce data optically with lasers.

[0121] Thus, a computer program product may perform operations presented herein. For example, such a computer program product may be a computer readable tangible medium
25 having instructions tangibly stored (and/or encoded) thereon, the instructions being executable by one or more processors to perform the operations described herein. The computer program product may include packaging material.

[0122] Software or instructions may also be transmitted over a transmission medium. For example, software may be transmitted from a website, server, or other remote source
30 using a transmission medium such as a coaxial cable, fiber optic cable, twisted pair,

digital subscriber line (DSL), or wireless technology such as infrared, radio, or microwave.

[0123] Further, modules and/or other appropriate means for performing the methods and techniques described herein can be downloaded and/or otherwise obtained by a user terminal and/or base station as applicable. For example, such a device can be coupled to a server to facilitate the transfer of means for performing the methods described herein. Alternatively, various methods described herein can be provided via storage means (e.g., RAM, ROM, a physical storage medium such as a CD or floppy disk, etc.), such that a user terminal and/or base station can obtain the various methods upon coupling or providing the storage means to the device. Moreover, any other suitable technique for providing the methods and techniques described herein to a device can be utilized.

[0124] Other examples and implementations are within the scope and spirit of the disclosure and appended claims. For example, due to the nature of software, functions described above can be implemented using software executed by a processor, hardware, firmware, hardwiring, or combinations of any of these. Features implementing functions may also be physically located at various positions, including being distributed such that portions of functions are implemented at different physical locations. Also, as used herein, including in the claims, “or” as used in a list of items prefaced by “at least one of” indicates a disjunctive list such that, for example, a list of “at least one of A, B, or C” means A or B or C or AB or AC or BC or ABC (i.e., A and B and C). Further, the term “exemplary” does not mean that the described example is preferred or better than other examples.

[0125] Various changes, substitutions, and alterations to the techniques described herein can be made without departing from the technology of the teachings as defined by the appended claims. Moreover, the scope of the disclosure and claims is not limited to the particular aspects of the process, machine, manufacture, composition of matter, means, methods, and actions described above. Processes, machines, manufacture, compositions of matter, means, methods, or actions, presently existing or later to be developed, that perform substantially the same function or achieve substantially the same result as the corresponding aspects described herein may be utilized. Accordingly, the appended claims include within their scope such processes, machines, manufacture, compositions of matter, means, methods, or actions.

CLAIMS

What Is Claimed Is:

1. A method for communicating between a service of a mobile operating system and an application of a desktop operating system over an inter-process communication channel, the method comprising:

requesting, by the application of the desktop operating system, to bind to a remotable interface of the service of the mobile operating system;

receiving, by the application of the desktop operating system, a remotable object from the service of the mobile operating system, the remotable object instantiated by the service of the mobile operating system in response to the request by the desktop operating system;

sending, by the application of the desktop operating system, a call to the remotable object; and

receiving, by the application of the desktop operating system, a return from the call to the remotable object from the service of the mobile operating system.

2. The method of claim 1, wherein the mobile operating system and the desktop operating system run concurrently on a shared kernel.

3. The method of claim 2, wherein an inter-process communication driver of the shared kernel marshals the first parcel object to the process space of the service of the mobile operating system.

4. The method of claim 3, wherein the desktop operating system includes an inter-process communication library of the mobile operating system compiled against user libraries of the desktop operating system, the inter-process communication library comprising base classes for the remotable object.

5. The method of claim 1, wherein the call to the remotable object includes a first parcel object.

6. The method of claim 1, wherein the call to the remotable object includes an action to be performed by the service of the mobile operating system.

7. The method of claim 6, wherein the mobile operating system returns a second parcel object after performing the action of the call to the remotable object.

8. The method of claim 1, wherein the call to the remotable object includes a reference to a shared memory space.

9. The method of claim 1, wherein the call to the remotable object includes a file descriptor.

10. A method for communicating between a first application of a mobile operating system and a second application of a desktop operating system, the method comprising:
receiving, by a remote communication service of the mobile operating system, a request by the first application to bind to the remote communication service;
instantiating, in response to the request by the first application, a remotable object;
passing the remotable object from the remote communication service to the first application;
receiving, by the remote communication service of the mobile operating system, a request by the second application to bind to the remote communication service through an inter-process communication channel; and
passing the remotable object from the remote communication service to the second application.

11. The method of claim 10, wherein the mobile operating system and the desktop operating system run concurrently on a shared kernel.

12. The method of claim 10, further comprising the steps of:
receiving, by the remote communication service of the mobile operating system, a call to the remotable object from the first application, the call including a first parcel object.

13. The method of claim 12, wherein the call to the remotable object comprises a synchronous call.

14. The method of claim 12, wherein the call to the remotable object comprises an asynchronous call.

15. The method of claim 12, further comprising the steps of:
returning, by the remote communication service of the mobile operating system, a second parcel object in response to receiving the call to the remotable object.

16. The method of claim 12, wherein the first parcel object includes a reference to a shared memory space.

17. A mobile computing device including a computer readable medium storing instructions for a physical processor, the instructions, when executed, causing the processor to perform steps comprising: receiving, by a service of a mobile operating system, a request to bind to the service by an application of a desktop operating system;

instantiating, by the service of the mobile operating system, a remotable object in response to the request to bind to the service by the application of the desktop operating system;

sending, by the service of the mobile operating system, the remotable object through an inter-process communication channel to the application of the desktop operating system;

receiving, by the service of the mobile operating system, a call to the remotable object by the application of the desktop operating system, the call to the remotable object including an action to be performed by the service; and

returning, from the service of the mobile operating system, a reference to the remotable object in response to receiving the call to the remotable object.

18. The mobile computing device of claim 17, wherein the instructions, when executed, further cause the processor to perform steps comprising:

running the mobile operating system in a first execution environment on a shared kernel;
and

concurrently running the desktop operating system in a second execution environment on the shared kernel;

19. The mobile computing device of claim 17, wherein the action includes invoking a remotable method of the service.

20. The mobile computing device of claim 17, wherein the service is a bound service.

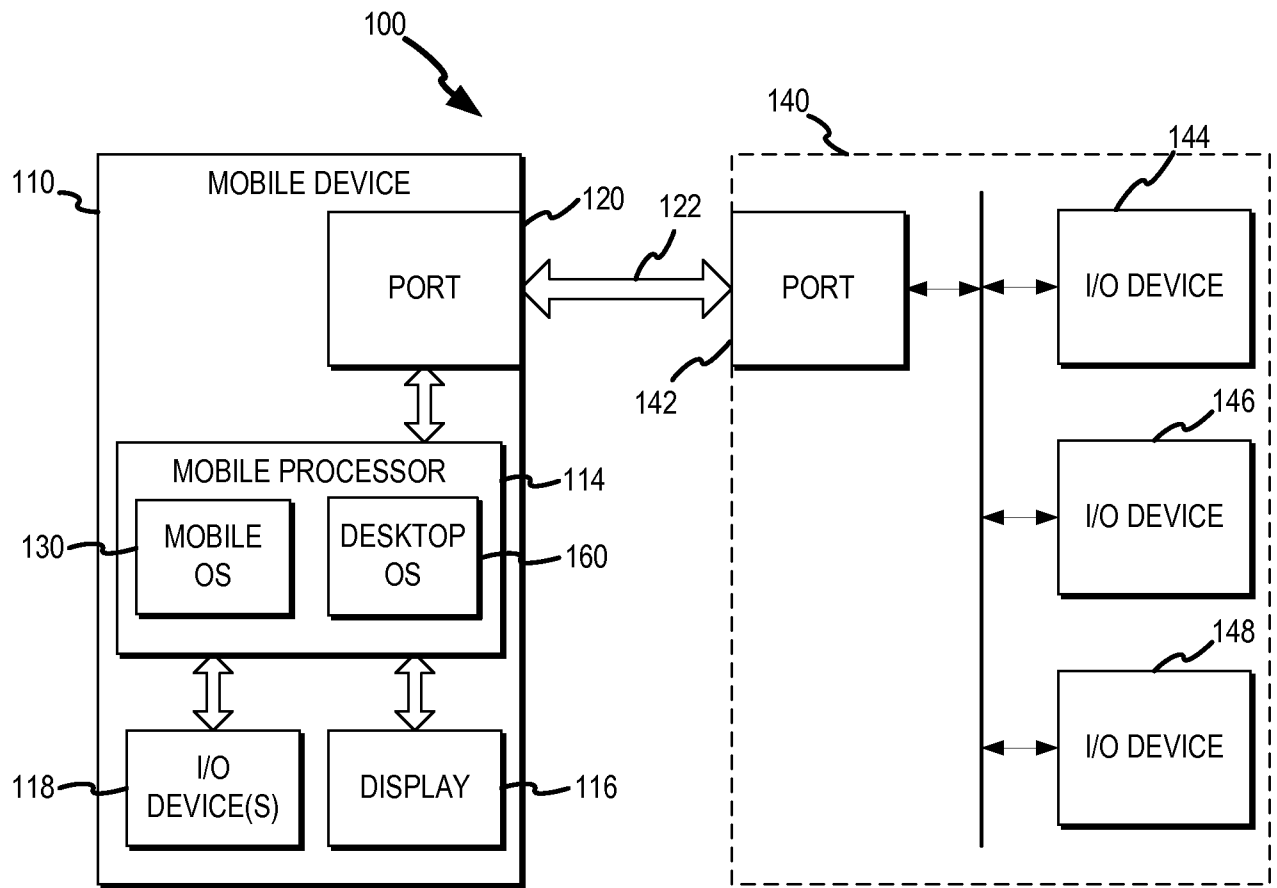


FIG. 1

2/11

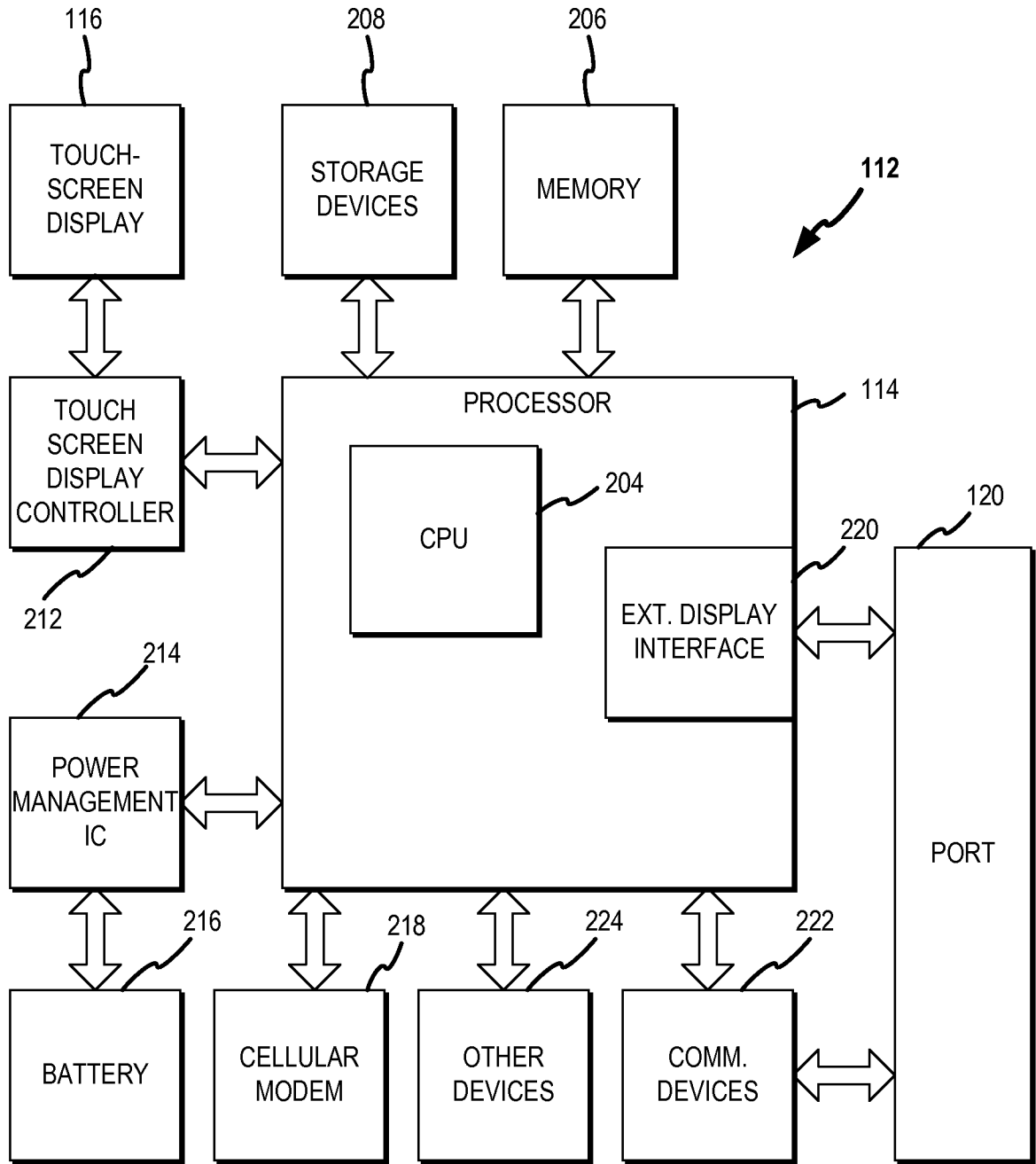


FIG. 2

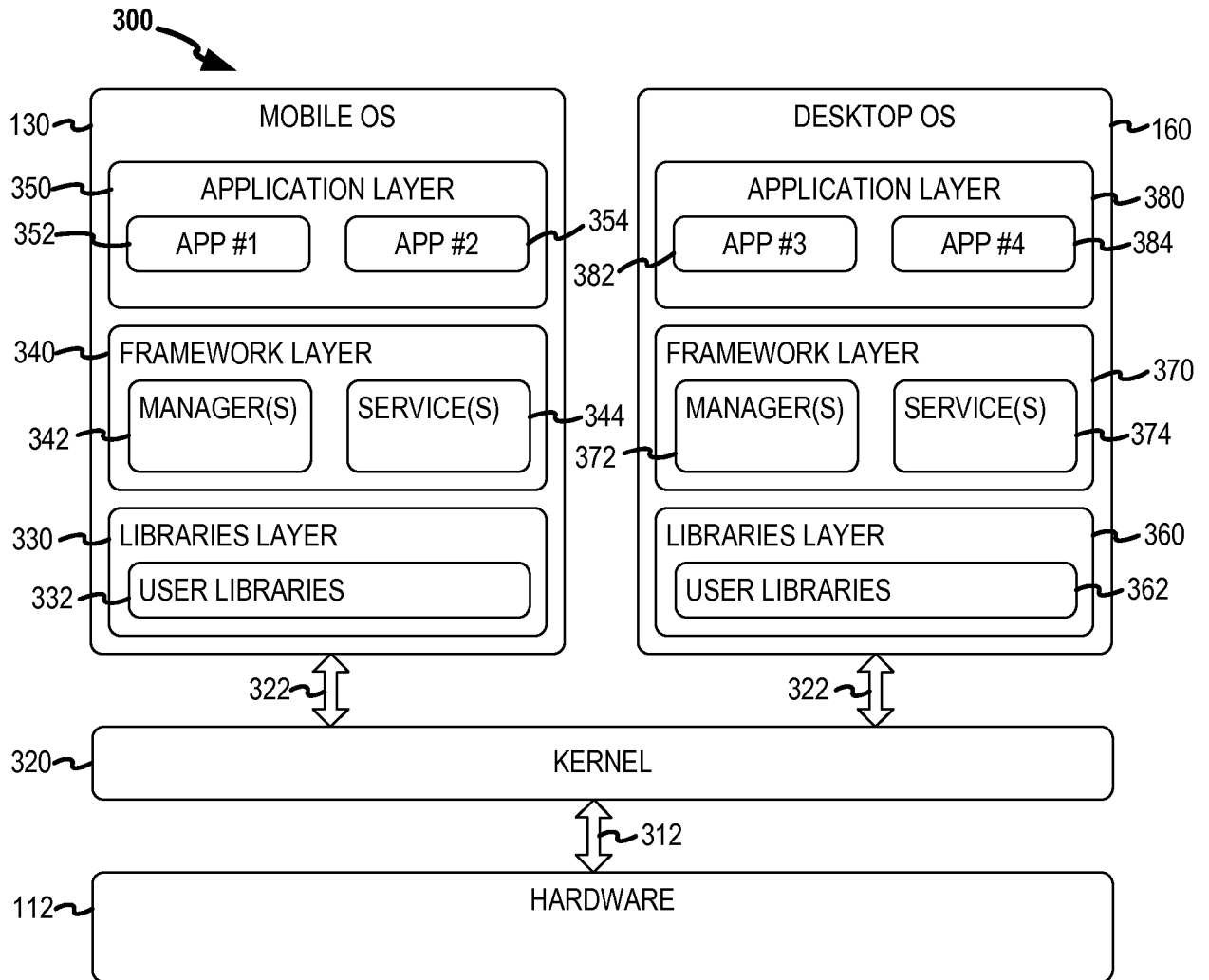


FIG. 3

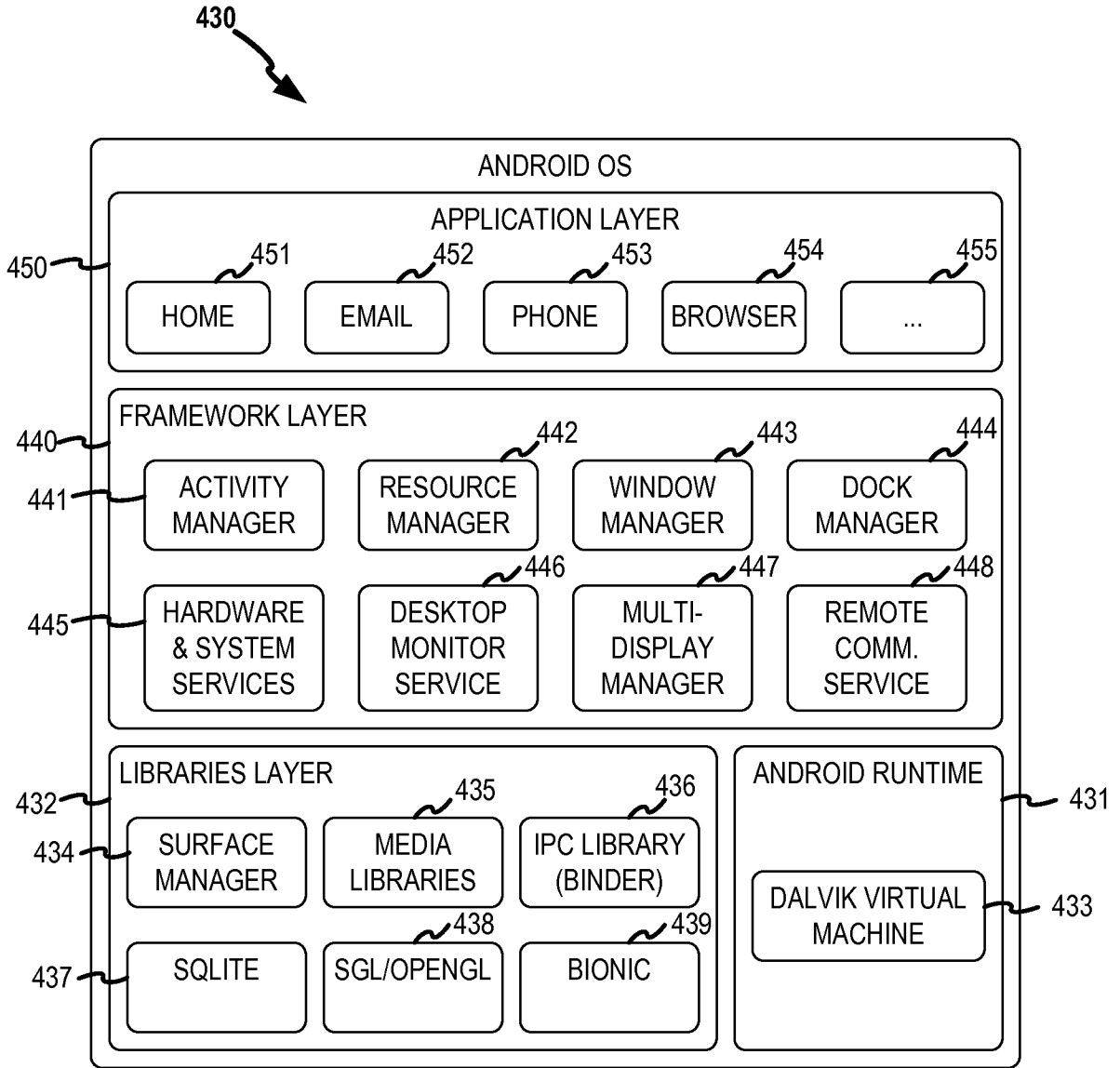


FIG. 4

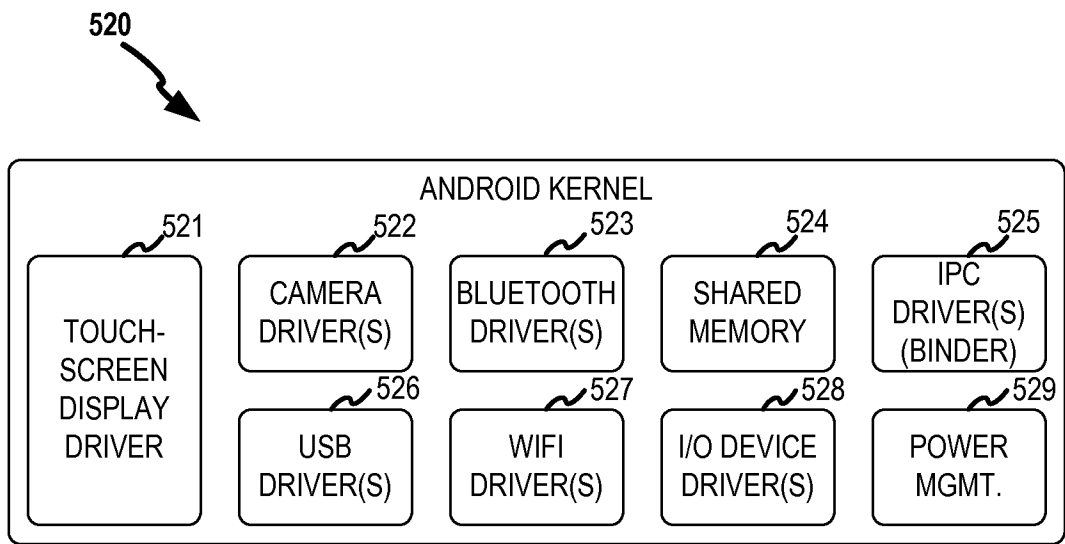


FIG. 5

6/11

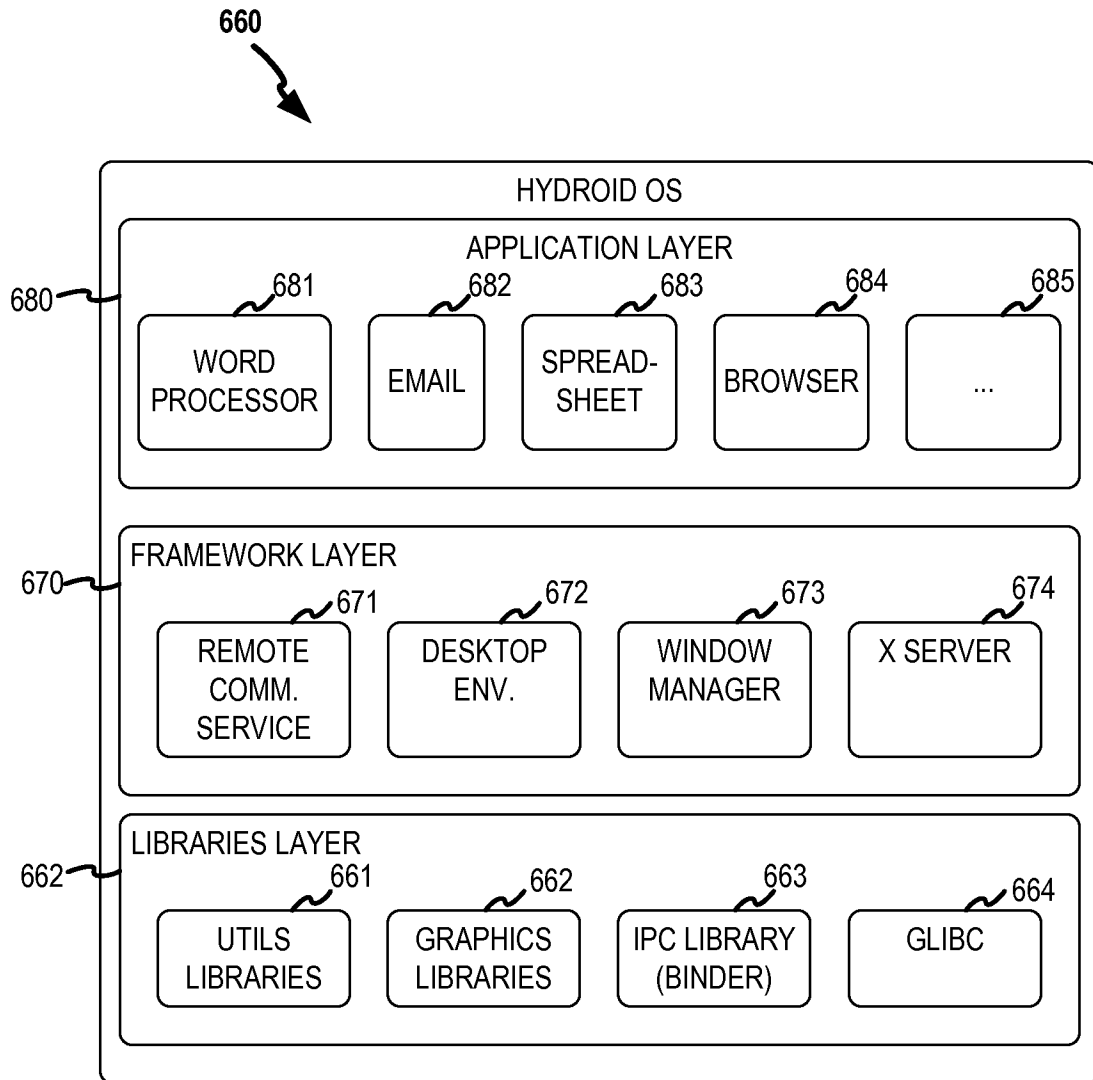


FIG. 6

7/11

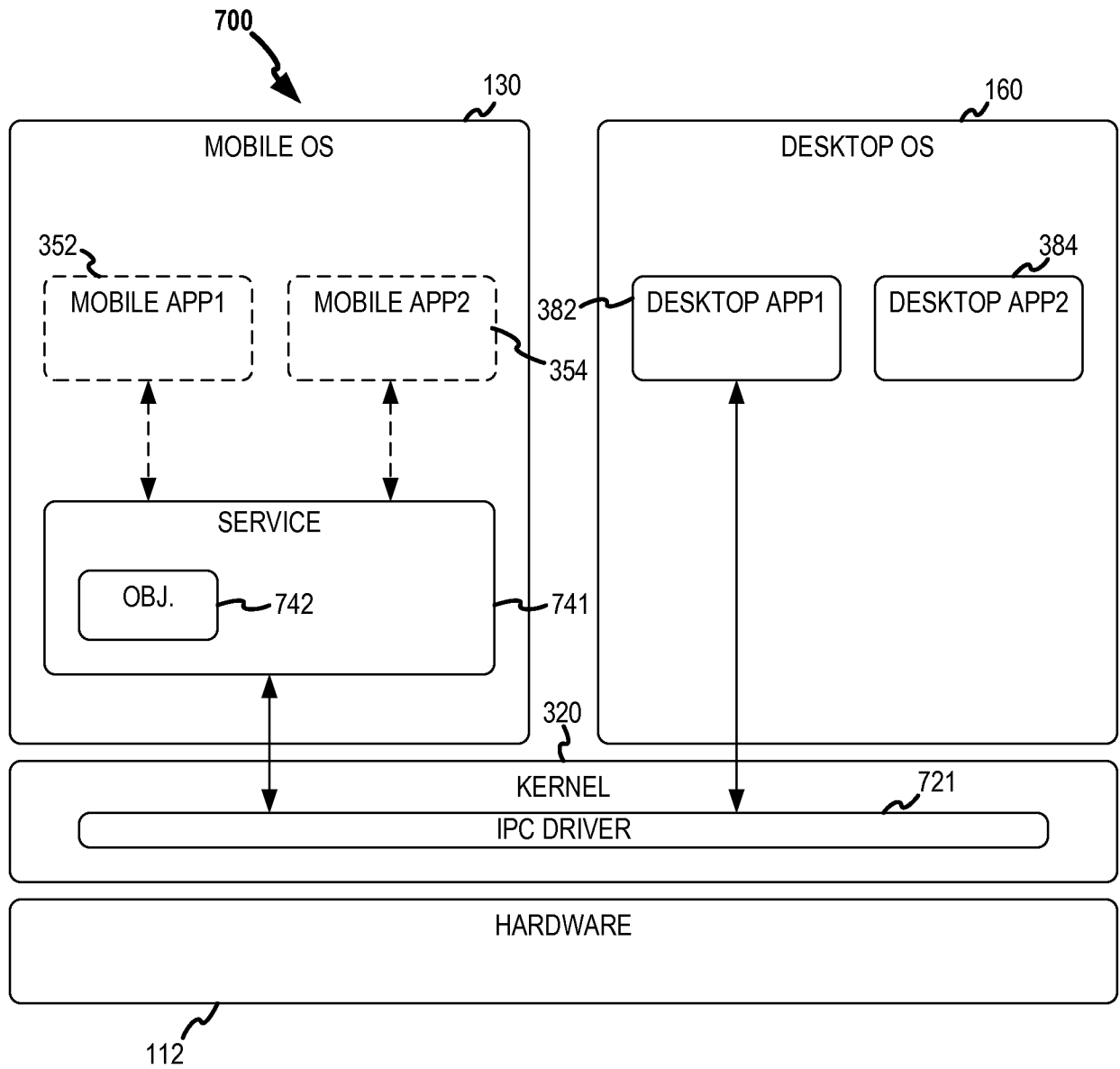


FIG. 7

8/11

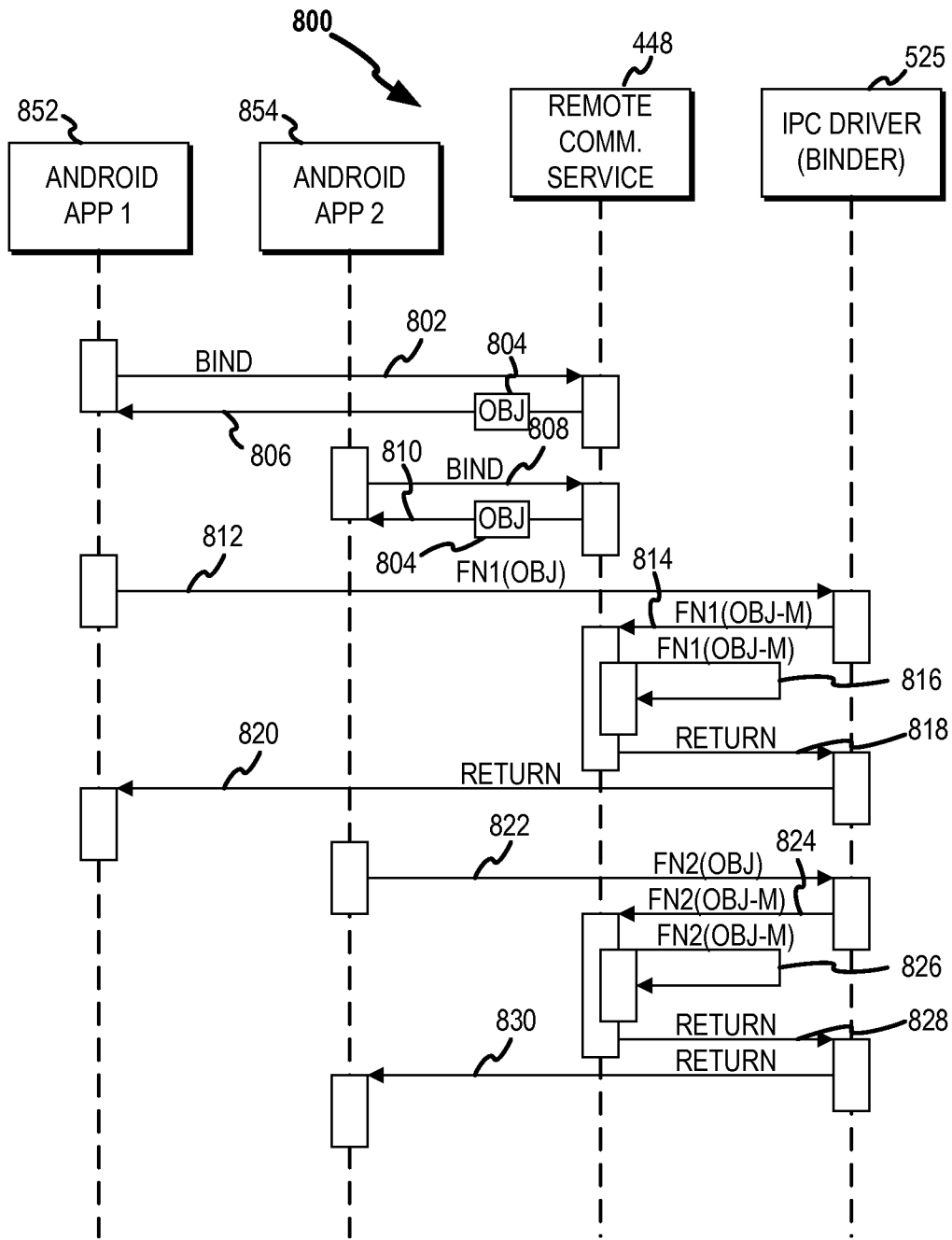


FIG. 8

9/11

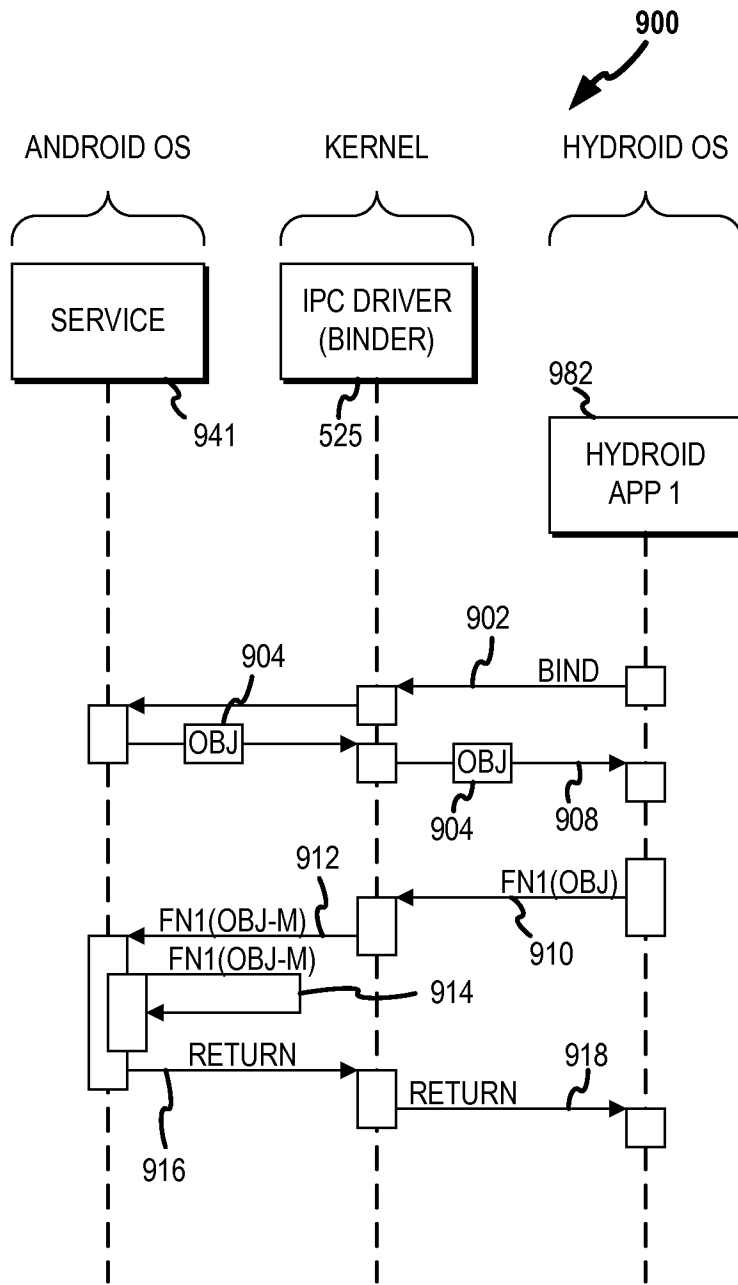


FIG. 9

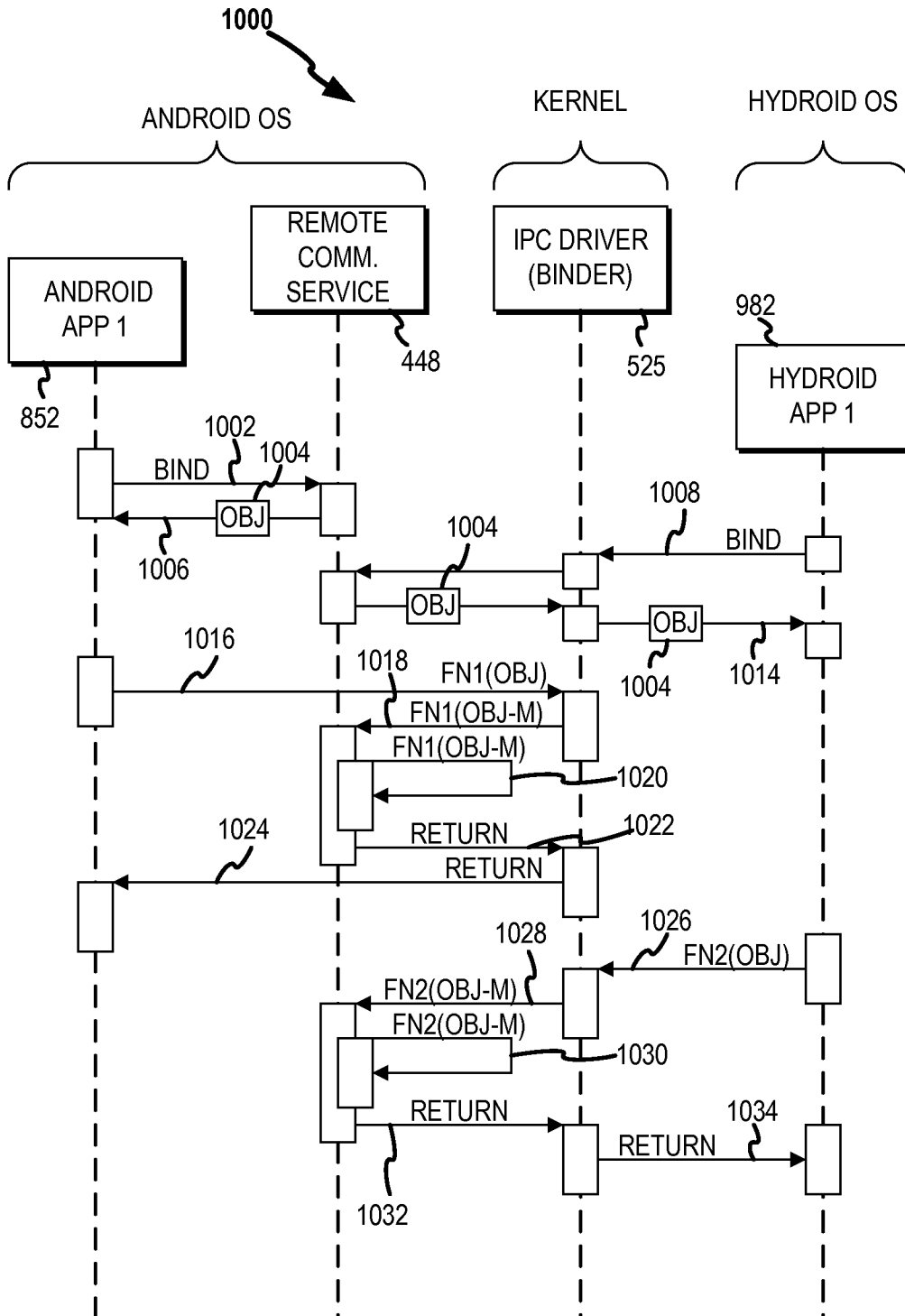


FIG. 10

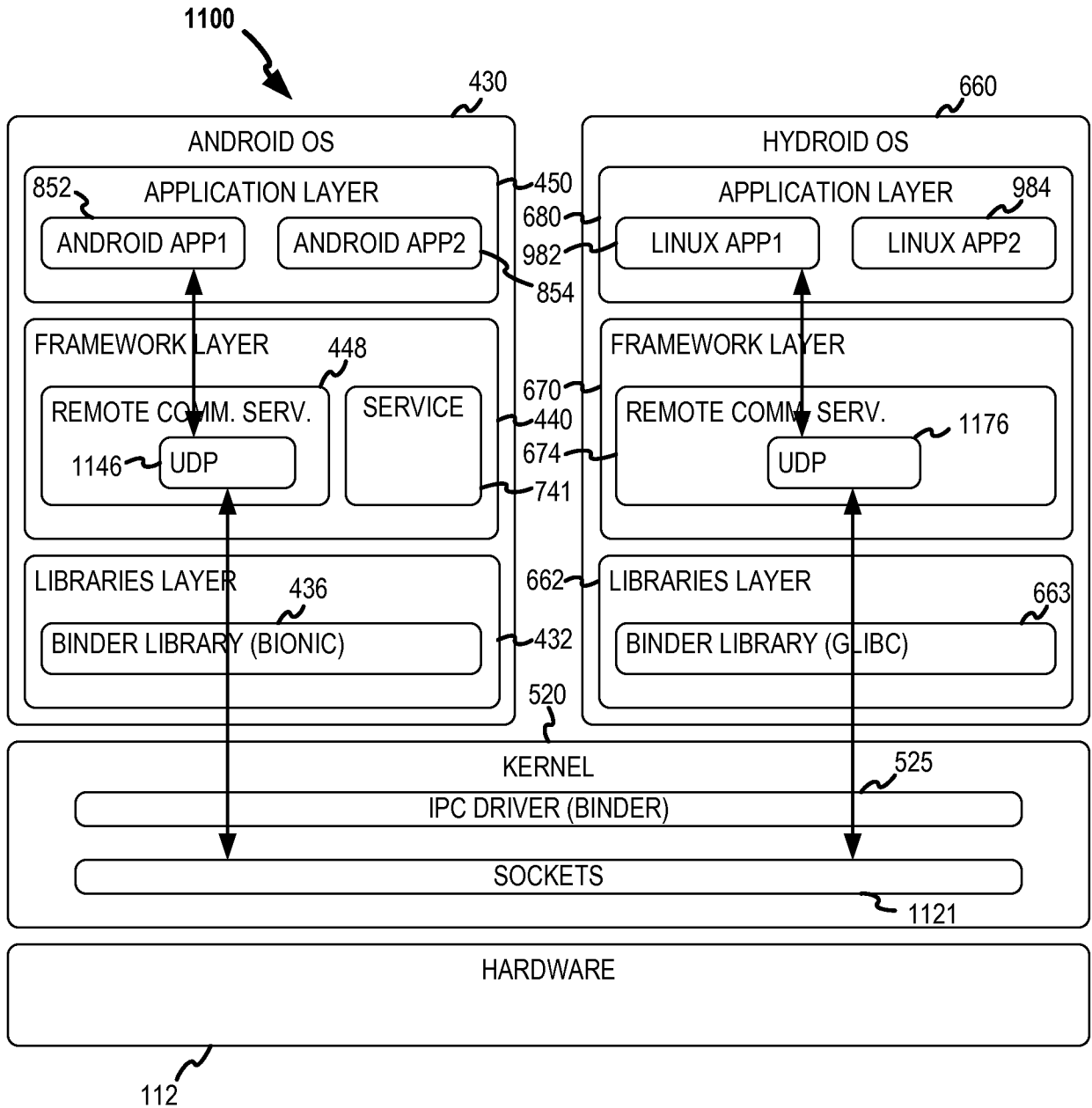


FIG. 11