US 20150317262A1

(19) **United States**

(12) **Patent Application Publication**

ADDA et al.

(10) **Pub. No.: US 2015/0317262 A1**

(43) **Pub. Date: Nov. 5, 2015**

(54) **EXECUTING A KERNEL DEVICE DRIVER AS A USER SPACE PROCESS**

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**, Armonk, NY (US)

(72) Inventors: **Michael ADDA**, Tel Aviv (IL); **Dan ALONI**, Rishon Le-Zion (IL); **Avner BRAVERMAN**, Tel Aviv (IL)

(73) Assignee: **INTERNATIONAL BUSINESS MACHINES CORPORATION**, Armonk, NY (US)

(21) Appl. No.: **14/800,396**

(22) Filed: **Jul. 15, 2015**

Related U.S. Application Data

(63) Continuation of application No. 14/457,203, filed on Aug. 12, 2014, now Pat. No. 9,092,356, which is a continuation of application No. 12/949,132, filed on Nov. 18, 2010, now Pat. No. 8,806,511.

Publication Classification

(51) **Int. Cl.**

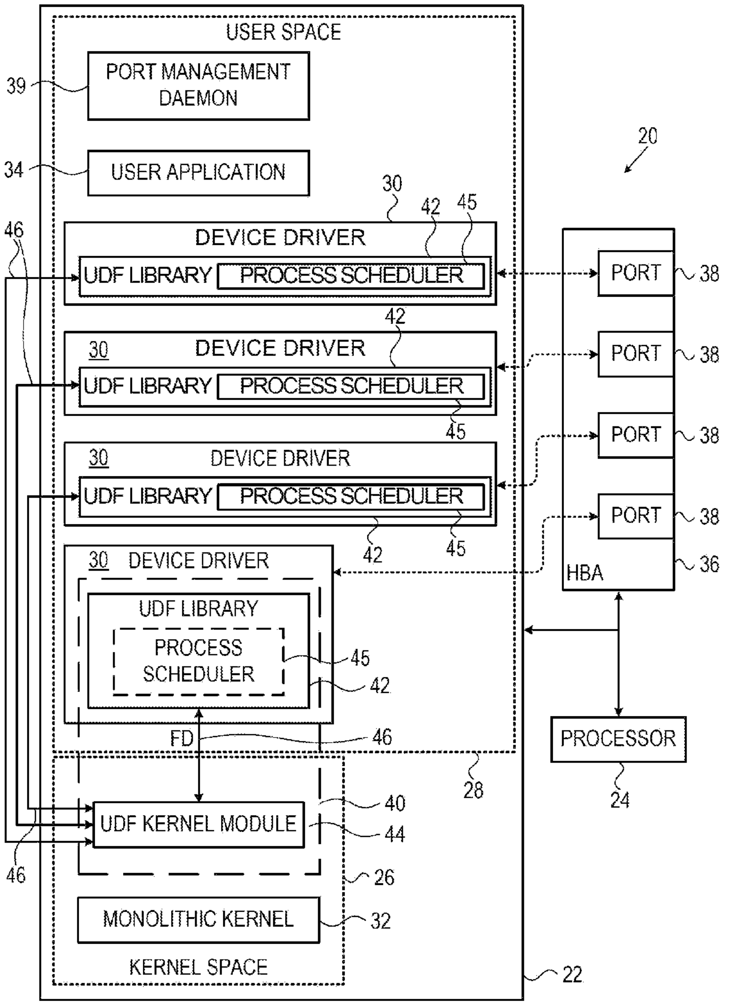
G06F 13/10	(2006.01)
G06F 9/54	(2006.01)
G06F 11/07	(2006.01)
G06F 9/48	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 13/102*** (2013.01); ***G06F 9/4843*** (2013.01); ***G06F 9/541*** (2013.01); ***G06F 9/545*** (2013.01); ***G06F 11/0793*** (2013.01)

(57) **ABSTRACT**

A method, including receiving, by a user space driver framework (UDF) library executing from a user space of a memory over a monolithic operating system kernel, a kernel application programming interface (API) call from a device driver executing from the user space. The UDF library then performs an operation corresponding to the kernel API call.



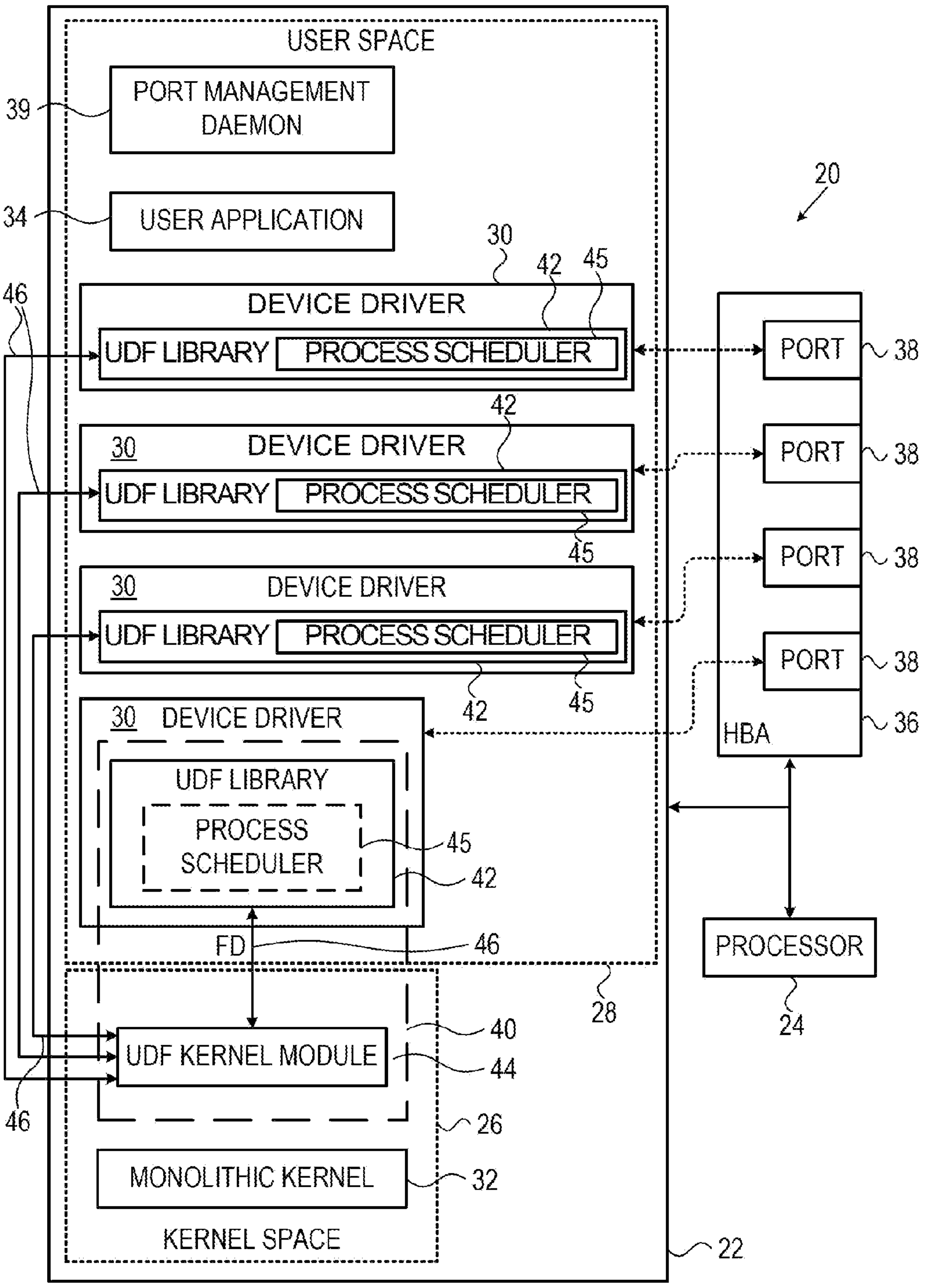

USER SPACE
PORT MANAGEMENT DAEMON
39
USER APPLICATION
34
20
30
42
45
46
DEVICE DRIVER
UDF LIBRARY
PROCESS SCHEDULER
PORT
38
42
30
DEVICE DRIVER
UDF LIBRARY
PROCESS SCHEDULER
PORT
38
45
30
DEVICE DRIVER
UDF LIBRARY
PROCESS SCHEDULER
PORT
38
42
45
PORT
38
30
DEVICE DRIVER
HBA
36
UDF LIBRARY
PROCESS SCHEDULER
45
42
FD
46
PROCESSOR
28
24
40
UDF KERNEL MODULE
44
46
26
MONOLITHIC KERNEL
32
KERNEL SPACE
22

FIG. 1

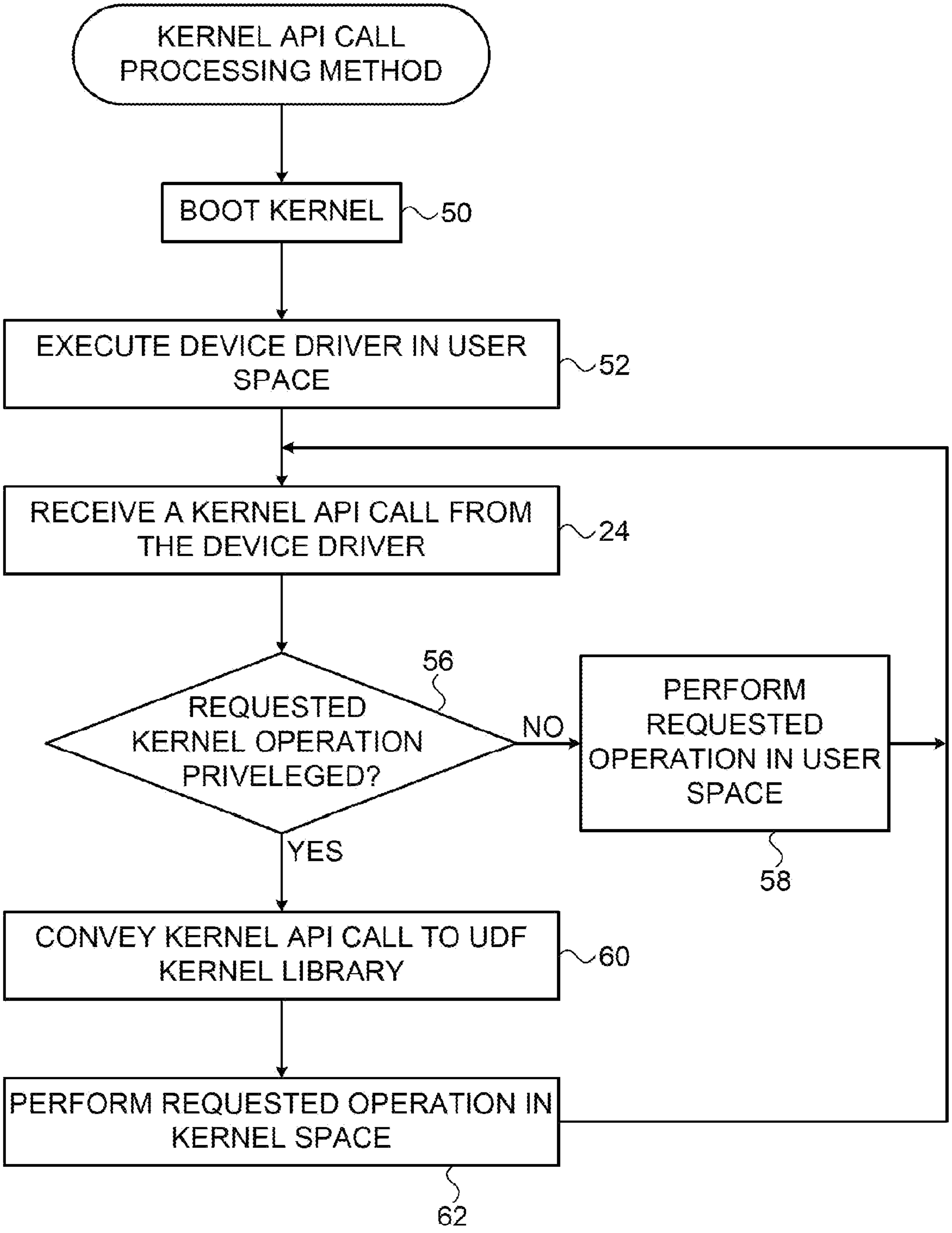

KERNEL API CALL PROCESSING METHOD
BOOT KERNEL
50
EXECUTE DEVICE DRIVER IN USER SPACE
52
RECEIVE A KERNEL API CALL FROM THE DEVICE DRIVER
24
REQUESTED KERNEL OPERATION PRIVELEGED?
56
NO
PERFORM REQUESTED OPERATION IN USER SPACE
58
YES
CONVEY KERNEL API CALL TO UDF KERNEL LIBRARY
60
PERFORM REQUESTED OPERATION IN KERNEL SPACE
62

FIG. 2

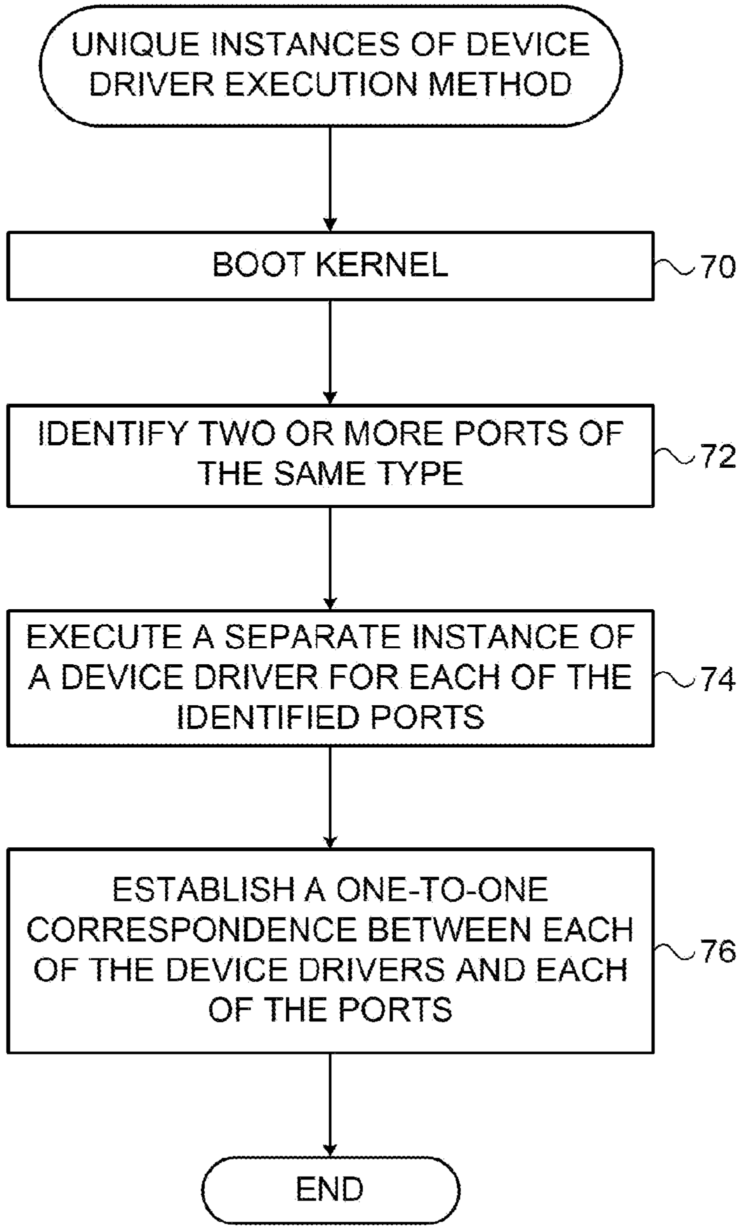

UNIQUE INSTANCES OF DEVICE DRIVER EXECUTION METHOD
BOOT KERNEL
70
IDENTIFY TWO OR MORE PORTS OF THE SAME TYPE
72
EXECUTE A SEPARATE INSTANCE OF A DEVICE DRIVER FOR EACH OF THE IDENTIFIED PORTS
74
ESTABLISH A ONE-TO-ONE CORRESPONDENCE BETWEEN EACH OF THE DEVICE DRIVERS AND EACH OF THE PORTS
76
END

FIG. 3

EXECUTING A KERNEL DEVICE DRIVER AS A USER SPACE PROCESS

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a Continuation of U.S. patent application Ser. No. 14/457,203, filed Aug. 12, 2014, which is a Continuation of U.S. patent application Ser. No. 12/949,132, filed Nov. 18, 2010.

FIELD OF THE INVENTION

[0002] The present invention relates generally to operating systems, and specifically to a software framework that enables kernel device drivers to execute as user-space processes.

BACKGROUND OF THE INVENTION

[0003] Operating systems are computer programs which manage the way software applications utilize the hardware of computer systems. A fundamental component of operating systems is the operating system kernel (also referred to herein as a "kernel"), which provides secure computer system hardware access to software applications executing on the computer system. Since accessing the hardware can be complex, kernels may implement a set of hardware abstractions to provide a clean and uniform interface to the underlying hardware. The abstractions provided by the kernel provide software developers easier access to the hardware when writing software applications.

[0004] Operating systems typically segregate virtual memory into kernel space and user space. User space is typically the virtual memory region for running user applications, while the kernel space is typically reserved for running the kernel and extensions to the kernel.

SUMMARY OF THE INVENTION

[0005] There are provided various embodiments for executing a kernel device driver as a user space process. In one embodiment, a method comprises, receiving, by a user space driver framework (UDF) library executing from a user space of a memory over an operating system kernel operated by a device driver executing from the user space, a kernel application programming interface (API) call from a device driver executing from the user space; determining that the operation comprises a privileged operation, wherein if the operation is non-privileged: performing, by the UDF library, an operation corresponding to the kernel API call, and detecting an interrupt and sending a notification of the interrupt via the file descriptor to the UDF library, wherein upon receiving the notification: scheduling by a scheduler an execution of an interrupt handling of the interrupt handling code of the device driver, wherein the interrupt handling code is a component of device driver configured to perform each one of: receiving a notification of an incoming message from a remote network node, and receiving a notification of a failure in firmware controlling a host bus adaptor (HBA).

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The disclosure is herein described, by way of example only, with reference to the accompanying drawings, wherein:

[0007] FIG. **1** is a block diagram of a computer system configured to execute device drivers as user mode processes, in accordance with an embodiment of the present invention;

[0008] FIG. **2** is a flow diagram that schematically illustrates a method of processing a kernel application programming interface (API) call from a device driver executing as a user space process, in accordance with an embodiment of the present invention; and

[0009] FIG. **3** is a flow diagram that schematically illustrates a method of executing a unique instance of a device driver for each port of the computer system, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS

Overview

[0010] Some operating systems such as Linux™, implement a monolithic kernel where the entire operating system executes from kernel space. In addition to the kernel, the operating system typically comprises kernel extensions and device drivers. A device driver is a software application that accepts a high-level command from a computer program (e.g., the kernel or a user application), and translates the high-level command to a series of low-level commands specific to a hardware device (e.g., a network interface controller).

[0011] During their execution, device drivers typically issue application programming interface (API) calls to the kernel. The API is an interface implemented in the kernel that enables the kernel to interact with other computer programs (e.g., device drivers and user applications). Computer programs issue API calls to gain access to and manage system resources.

[0012] Embodiments of the present invention provide methods and systems to enable device drivers to execute as user space processes in a monolithic kernel environment. More specifically, embodiments of the present invention enable device drivers executing from user space over a monolithic kernel to call kernel API functions. In some embodiments, an emulation layer executing over the monolithic kernel receives and processes an API call from a device driver executing from user space. The emulation layer replicates a subset of the kernel's API calls which are applicable to the device drivers. Since the emulation layer receives and processes kernel API calls, embodiments of the present invention enable existing device drivers to execute from user space with minimal modification.

[0013] If a device driver fails while executing from kernel space over a monolithic kernel, the failure of the device driver may cause the kernel to fail, thereby causing a system crash. Since embodiments of the present invention enable device drivers to execute from user space, kernel stability may be increased since a failure of a device driver (executing from user space) may only disable access to the device being controlled by the device driver.

[0014] In addition to executing device drivers from user space, embodiments of the present invention provide additional system stability by executing a separate instance (i.e., each instance executing as a separate process) of a device driver for each port of a same type in a computer system. For example, in a computer system comprising three network interface cards from a single vendor with four ports on each card, the kernel may execute twelve separate instances of an identical device driver, with each device driver dedicated to one of the twelve ports.

[0015] Since operating systems typically load and execute a single device driver for all devices of the same type, the device driver typically constitutes a single point of failure, i.e., a failure of the device driver disables access to all ports controlled by the device driver. By executing a separate instance of the device driver for each port, embodiments of the present invention increase system stability, since a failure of one of the device drivers only disables the single port controlled by the failed device driver.

System Description

[0016] FIG. **1** is a block diagram of a computer system **20** configured to execute device drivers as user mode processes, in accordance with an embodiment of the present invention. In the configuration shown in FIG. **1**, computer system **20** comprises a memory **22** coupled to a processor **24**. Memory **22** is divided into a kernel space **26** and a user space **28**.

[0017] Processor **24** executes device drivers **30** from user space **28**. Device drivers **30** accept high level commands from a monolithic kernel **32** and a user application **34**, and translate the high level commands to a series of low level commands for a host bus adapter (HBA) **36**, a hardware device which connects computer **20** to other network and storage devices (not shown). HBA **36** comprises hardware ports **38**, where each of the ports is controlled by a separate instance of device drivers **30**. A port manager daemon (i.e., a background process) **39**, typically executing from user space **28**, manages the relationships between device drivers **30** and ports **38** as described hereinbelow. While the configuration in FIG. **1** shows HBA **36** with hardware ports **38**, other system configurations can also be employed to implement embodiments of the present invention, and are thus considered to be within the spirit and scope of this invention.

[0018] An emulation layer **40** in memory **22** is configured to accept kernel API calls from device drivers **30**, and perform the requested kernel operation. The emulation layer comprises a user space driver framework (UDF) library **42** executing from the user space and a UDF kernel module **44** executing from the kernel space. In embodiments of the present invention, UDF library **42** is a component of device driver **30**, and is configured to implement a subset of application programming interface (API) calls for kernel **32** that are applicable for managing Peripheral Component Interconnect (PCI) devices, such as HBA **36**.

[0019] In the configuration shown in FIG. **1**, emulation layer **40** comprises UDF kernel module **44** and UDF library **42** of the device driver directly above the UDF kernel module. Since there are four device drivers **30**, there are actually four emulation layers **40**. For simplicity, only one layer **40** is shown in the figure. The combination of UDF kernel module **44** and the UDF library for each device driver **30** comprises a separate emulation layer **40**.

[0020] UDF library **42** is configured to implement API functions that can be run from user space **28**. Examples of API functions that can be performed by UDF library **42** from user space **28** include managing lists, timers and a process scheduler (PS) **45**. Lists typically store information such as message buffers to transfer to hardware devices such as HBA **36**, and timers can be used to detect a situation where the HBA (or another hardware device) does not respond to a command, and therefore needs to be reset.

[0021] Process scheduler **45** typically schedules threads and interrupt handling code within its associated device driver **30**, enabling the implementation of user-level threads. In computing, a thread is a component of a process in the sense that a single process (e.g., an instance of device driver **30** executing from user space **28**) may comprise multiple threads, where all threads within the single process share the same state, share the same memory space, and communicate with each other directly.

[0022] UDF kernel module **44** is configured to implement API calls that are typically performed from kernel space **26**, including mapping input output (I/O) memory addresses, allocating direct memory access (DMA) memory, and catching interrupts. Memory **22** comprises a file descriptor (FD) **46**, which is a software mechanism that enables software processes, in this case UDF library **42** and UDF kernel module **44**, to communicate with one another.

[0023] In computing, a privilege refers to a permission to perform a specific action. The monolithic kernel tasks performed by processor **24** are usually divided into privileged and non-privileged operations. Privileged operations typically have absolute control over critical system resources (e.g., memory and ports), as opposed to non-privileged operations which typically manage less critical system resources (e.g., timers and lists). In embodiments of the present invention, UDF kernel module **44** is configured to performed privileged operations, and UDF library **42** is configured to perform non-privileged operations.

[0024] When mapping I/O memory addresses (i.e., implementing memory mapped I/O), processor **24** assigns addresses in memory **22** to a device, such as HBA **36**. Kernel **32** and user application **34** can then access HBA **36** by reading from or writing to the assigned memory addresses. When allocating DMA memory, processor **24** assigns addresses in memory **22** that are then used to transfer data directly between memory **22** and a device (e.g., HBA **36**) without involving processor **24**, thereby reducing processor overhead.

[0025] An interrupt typically comprises a signal that causes processor **24** to temporarily suspend execution of a program (e.g., a process of kernel **32** or user application **34**). After detecting the interrupt, processor **24** may either resume executing the suspended program or start executing a different program (i.e., an application or a process). In general, there are hardware interrupts and software interrupts. A hardware interrupt occurs, for example, when an I/O operation is completed such as transferring data between HBA **36** and memory **22**. A software interrupt occurs, for example, when user application **34** terminates or requests certain services from kernel **32**.

[0026] In monolithic kernel environments, interrupts are typically handled from kernel space **26**. In some embodiments of the present invention, upon detecting an interrupt, UDF kernel library **44** conveys a notification, via file descriptor **46**, to UDF library **42** that there is an interrupt. Upon receiving the notification, scheduler **45** schedules execution of interrupt handling the device driver's interrupt handling code. The interrupt handling code is a component of device driver **30** configured to perform operations such as:

- **[0027]** Receiving a notification of an incoming message from a remote network node. The interrupt handling code is configured to start processing the notification upon receipt of the notification.

[0028] Receiving a notification of a failure in firmware controlling HBA **36**. The interrupt handling code is configured to reset HBA **36** upon receipt of the notification of failure.

[0029] Processor **24** typically comprises a general-purpose computer configured to carry out the functions described herein. Software operated by the processor may be downloaded to memory **22** in electronic form, over a network, for example, or it may be provided on non-transitory tangible media, such as optical, magnetic or electronic memory media. Alternatively, some or all of the functions of processor **24** may be carried out by dedicated or programmable digital hardware components, or by using a combination of hardware and software elements.

[0030] The present invention may be a system, a method, and/or a computer program product. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0031] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0032] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0033] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0034] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0035] These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0036] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

Processing Kernel API Calls from User Space

[0037] FIG. **2** is a flow diagram that schematically illustrates a method of processing a kernel API call from one of device drivers **30** executing as a process from user space **28**, in accordance with an embodiment of the present invention. In a first step **50**, processor **24** boots kernel **32**, and in an execute step **52**, processor **24** executes UDF library **42** and device drivers **30** from user space **28**, and UDF kernel module **44** from kernel space **26**.

[0038] In a receive step **54**, UDF library **42** receives a kernel API call from its associated device driver **30**. The device driver typically issues the API call after receiving a high level command from either kernel **32** or user application **34**. In a comparison step **56**, if the requested operation associated with the API call is not privileged, then in a first perform step **58**, UDF library **42** performs the requested operation from user space **28**, and the method continues with step **54**. Returning to step **56**, if the requested operation is privileged, then in a convey step **60**, UDF library **42** conveys the API call to UDF kernel module **44** via file descriptor **46**. In a second perform step **62**, UDF kernel module **44** performs the requested operation from kernel space **26**, conveys any necessary completion information back to UDF library **42** via file descriptor **46**, and the method continues with step **54**.

Device Driver Port Management

[0039] FIG. **3** is a flow diagram that schematically illustrates a method of executing unique instances of device drivers **30** for each port **38**, where each port **38** is a same type, in accordance with an embodiment of the present invention. In a first step **70**, processor **24** boots kernel **32**. While the computer system shown in FIG. **1** includes a monolithic kernel, processor **24** may boot a different type of kernel in step **70**, including, but not limited to a hybrid kernel or a microkernel. In an identification step **72**, port manager daemon **39** identifies two or more ports **38** of a same type. For example, ports **38** may be positioned on a single device, such as HBA **36**. Additionally or alternatively, ports **38** may be physically positioned on multiple devices.

[0040] In an execute step **74**, processor **24** executes, in memory **22**, a separate instance of an identical device driver **30** for each of identified ports **38**, where each instance is executed as a separate process from a unique address in memory **22**. For example, in the computer system shown in FIG. **1**, processor **24** executes four instances of the same HBA driver **30** for each HBA port **38**, and executes each of the HBA drivers as a separate process.

[0041] Finally, in an establish step **78**, processor **24** couples device drivers **30** to ports **38** and establishes a one-to-one correspondence between each of the device drivers and each of the ports, and the method terminates. As discussed supra, a failure of one of device drivers **30** only disables the port corresponding to the failed device driver. Typically, in the event of a failure of one of the device drivers, port manager daemon **39** detects and identifies the port associated with the failed device driver, re-launches the failed device driver as a new process, and couples the re-launched device driver to the identified port.

[0042] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function (s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0043] It will be appreciated that the embodiments described above are cited by way of example, and that the present invention is not limited to what has been particularly shown and described hereinabove. Rather, the scope of the present invention includes both combinations and subcombinations of the various features described hereinabove, as well as variations and modifications thereof which would occur to persons skilled in the art upon reading the foregoing description and which are not disclosed in the prior art.

1. A method, comprising:
receiving, by a user space driver framework (UDF) library executing from a user space of a memory over an operating system kernel operated by a device driver executing from the user space, a kernel application programming interface (API) call from a device driver executing from the user space;
determining that the operation comprises a privileged operation, wherein if the operation is non-privileged:
performing, by the UDF library, an operation corresponding to the kernel API call, and
detecting an interrupt and sending a notification of the interrupt via the file descriptor to the UDF library, wherein upon receiving the notification:
scheduling by a scheduler an execution of an interrupt handling of the interrupt handling code of the device driver, wherein the interrupt handling code is a component of device driver configured to perform each one of:
receiving a notification of an incoming message from a remote network node, and
receiving a notification of a failure in firmware controlling a host bus adaptor (HBA).

2. The method according to claim **1**, wherein the operation comprises a privileged operation.

3. The method according to claim **2**, wherein if the operation is privileged:
conveying, via a file descriptor, the API call to a UDF kernel module executing from a kernel space of the memory over the operating system kernel, and
performing, by a kernel space emulation module, the privileged operation from the kernel space while conveying completion information back to the UDF library.

4. The method according to claim **1**, wherein the non-privileged operation is selected from a group of operations comprising maintaining a list, maintaining a timer, and maintaining a process scheduler.

5. The method according to claim **3**, wherein the privileged operation is selected from a group of operations comprising catching an interrupt, allocating direct memory access (DMA) memory, and mapping input/output (I/O) memory addresses.

6. The method according to claim **1**, wherein the device driver comprises a Peripheral Component Interconnect (PCI) device driver.

7. A computer program product, the computer program product comprising:
a non-transitory computer readable storage medium having computer readable program code embodied therewith, the computer readable program code comprising:

computer readable program code configured to receive, by a user space driver framework (UDF) library executing from a user space of a memory over an operating system kernel operated by a device driver executing from the user space, a kernel application programming interface (API) call from a device driver executing from the user space; and

computer readable program code configured to, if the operation is non-privileged, perform an operation, by the UDF library, corresponding to the kernel API call, and an interrupt is detected and a notification of the interrupt is sent via the file descriptor to the UDF library, wherein upon receiving the notification:

a scheduler schedules an execution of an interrupt handling of the interrupt handling code of the device driver, wherein the interrupt handling code is a component of the device driver configured to perform each one of receiving a notification of an incoming message from a remote network node, and receiving a notification of a failure in firmware controlling a host bus adaptor (HBA).

8. The computer program product according to claim **7**, wherein the operation comprises a privileged operation.

9. The computer program product according to claim **8**, further including computer readable program code configured to determine that the operation comprises a privileged operation, wherein if the operation is privileged, the API call is conveyed, via a file descriptor, to a UDF kernel module executing from a kernel space of the memory over the operating system kernel, and the privileged operation is performed, by a kernel space emulation module, the privileged operation from the kernel space while conveying completion information back to the UDF library.

10. The computer program product according to claim **7**, wherein the computer readable program code is configured to select the non-privileged operation from a group of operations comprising maintaining a list, maintaining a timer, and maintaining a process scheduler.

11. The computer program product according to claim **9**, wherein the computer readable program code is configured to select the privileged operation from a group of operations comprising catching an interrupt, allocating direct memory access (DMA) memory, and mapping input/output (I/O) memory addresses.

12. The computer program product according to claim **7**, wherein the device driver comprises a Peripheral Component Interconnect (PCI) device driver.

* * * * *