200101552-8
14 MAR 2001

METHOD FOR REPLACING A DEVICE DRIVER DURING SYSTEM OPERATION

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates in general to computer operating systems and device drivers, and in particular, to method for replacing a device driver without disrupting the connection of a device to an operating system.

2. Description of the Related Art:

Computer systems use device drivers to provide an interface between an operating system and physical devices such as storage devices, network adapters, and display adapters. This interface is supplied to operating systems services and/or application programs. Traditionally, device drivers have been static modules that are loaded at initialization time and contain all of the program instructions required for sending commands to and exchanging data with the physical device.

More recently, dynamically loadable device driver interfaces have been implemented in operating systems such as Windows NT® and Windows 2000® that are manufactured by Microsoft® Corporation. Dynamically loadable device drivers can be loaded selectively based upon need to access the particular device supported by the driver. Dynamic device driver interfaces also allow the replacement of device drivers by shutting down the device driver, copying a new device driver in place of

the old device driver and then starting the new device driver. This procedure typically requires shutting down all layers of device driver connections (or driver .stack.) since in operating systems such as WINDOWS NT and WINDOWS 2000 the driver stack connections are built from the lowest level (physical layer) device driver to the highest level device driver. This presents a problem when an upgrade is desired that affects primary storage devices or devices that form part of the core processing components such as bus interface drivers or memory managers. If the device driver being replaced is a boot device driver or the storage driver for the device containing an operating system swap file, the operating system typically has to be shut down in order to replace the driver, since file device input output must be maintained on these devices for proper operation of the operating system.

Within an Internet server, computer systems that control critical systems, or other downtime intolerant computer application, there is a great need for a facility to upgrade device drivers without taking the system off-line or disrupting operations. Therefore, it is desirable to provide a method that allows for .hot-swapping. a device driver without shutting down the computer system. The hot-swapping capability is also useful in the development environment where the operating system typically must be re-booted after every device driver change when a critical file system device or core processing device driver is being developed.

SUMMARY OF THE INVENTION

The objectives of providing a method to swap a device driver controlling a device in a computer while maintaining operating systems device presence and

operating systems interface operation are achieved by a method including the steps of copying the new device driver to a location within the computer, indicating to a shell device driver that an update of the device driver has been requested, disconnecting the device driver from the device in response to the update indication, connecting the new device driver to the device.

The method may be further embodied in a computer program product adapted for use with a computer system having program instructions for carrying out the steps of the method.

The objectives above are also achieved in a computer system that includes file transfer means for copying a new device driver to the computer system, and a shell device driver having an update interface. The shell device driver also includes an operating system side entry point interface, and a device driver side entry point interface for selectively coupling a selected device driver. The shell device driver uncouples the original device driver and couples to the new device driver upon receiving an update request.

All objects, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a block diagram of a computer system in which a method of the present invention may be implemented;

Figure 2 is a block diagram depicting the relationship between device drivers and system components, in accordance with an embodiment of the present invention;

Figure 3 is a block diagram depicting device driver modules and connections in accordance with an embodiment of the present invention; and

Figure 4 is a flowchart depicting a method for replacing a device driver during system operation, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

With reference now to the figures and in particular with reference to Figure 1, there is depicted a block diagram of a computer system in which the present invention can be implemented. As shown, a local bus 12 couples a processor 10 for executing program instructions with a memory 11 for storing value data and program instructions. A cache memory and controller 13 are optionally supplied for storage of frequently accessed value data and program instructions. A bus interface 14 couples local bus 12 to a system bus 15. System bus 15 couples various peripheral components to bus interface 14, providing user interface and storage for the computer system. In the illustrative embodiment of Figure 1, several peripherals are coupled to system bus 15, including a video interface 21 for interfacing a user display, a network interface 20 for connecting the computer system to a network, and an Integrated Device Eletronics (IDE) adapter 16 and a Small Computer Systems Interface (SCSI) adapter 18 for coupling storage devices 17 and 19, respectively.

Device drivers for controlling critical systems components that an operating system relies on for basic services cannot be easily replaced at run-time in the prior art. For example, cache memory and controller 13 and bus interface 14 may have their own device drivers for managing different hardware. These device drivers are typically loaded at boot time and may not be replaced except by copying over the device driver and rebooting the computer. Memory 11 may have memory manager drivers that likewise must remain installed during operation of the computer system. SCSI adapter 18 and IDE adapter 16 may control boot devices and the device on which page

files are stored by the operating system's virtual memory manager. These devices must also be constantly connected to the typical operating system. Additionally, network interface 20 may need to be constantly connected in a network application such as a file server connected to the Internet. If a device driver for the network stack needs replacement, typically the server must be brought off-line to accomplish this result.

Referring now to Figure 2, a device driver connection in accordance with a preferred embodiment of the invention is depicted. A physical device 35 is coupled to a physical device adapter 34, which is interfaced to the operating system by means of input/output (I/O) or memory-mapped registers or other interface means appropriate to physical device 35. The operating system may include a hardware abstraction layer 33, or there may be a direct access from a device driver 30 to the hardware. Device driver 30 utilizes hardware abstraction layer 33 to connect to the hardware resources of physical device adapter 34. According to a preferred embodiment of the present invention, device driver 30 is split into two discrete device drivers: a shell device driver 31, which provides the system driver interfaces to connect to the operating systems services, and a physical layer device driver 32 to connect to the device. By using this structure, it is possible to dynamically connect a replacement for physical layer device driver 32, without rebooting the operating system and providing only momentary delay of device access, instead of disrupting the flow of data to and from physical device 35.

Referring now to Figure 3, the structure of device driver 30 is shown in detail. Entry points 46 correspond

to function pointers provided by device driver 30 to the operating system, which in turn correspond to services provided by each particular device, for example, read, write and positioning services for a storage device. Shell device driver 31 provides interfaces for these entry points, shielding physical layer device driver 32 from system calls that would otherwise render it impossible to disconnect physical layer device driver 32 from the system and still maintain active connection of the device.

To manage disconnection of physical layer device driver 32, shell device driver 31 provides a special update() entry point, that indicates to shell device driver 31 to attempt replacement of physical layer device driver 32. Device configuration and state information 45 represents the current state of physical layer device driver and the physical connection 47 (device state). In order to maintain proper operation when physical layer device driver 32 is replaced, the state of the replacement driver must be matched to that of the previously operational physical layer device driver 32.

Shell device driver 31 provides a get_state() function and a set_state() function for this purpose. Entry point management 42 is functionality within shell device driver 31, enabling it to track when entry points have been reflected into the entry points in physical layer device driver 32, so that disconnection of physical layer device driver 32 can be managed. Reference counts or tokens that can determine the number of threads that have actively entered each entry point of physical layer device driver 32 must be kept so that the disconnection of the original driver entry points and connection of the replacement driver entry points can be achieved. An entry point table in shell device driver 31 maintains the

current physical layer device driver 32 entry points as function pointers. Additionally, shell device driver 31 has request queues 43 to queue up requests received at entry points 46 while physical layer device driver 32 is being replaced. The entry point 46 calls will not be reflected to the original physical layer device driver's entry points, but will be held in abeyance for the replacement driver to handle, once initialization and state synchronization of the replacement driver is complete.

The device state within shell device driver 31 also contains pending operation information, and the update of physical layer device driver 32 can be delayed until certain operations that must be completed on the device are complete. This state information can also include information about outstanding operating system responses (callbacks) that have not occurred such as hardware interrupts or deferred procedure calls that are pending.

Referring now to Figure 4, a method for replacing a device driver during system operation in accordance with a preferred embodiment of the invention is depicted in flow diagram form. An application program (update utility program) or an operating system feature initiates the update process by copying the replacement device driver to a location in the system, which may be a different file name or the same filename in a different location (step 52). The original driver location and filename could be used for the replacement driver, but it is not preferred, as if an error occurs in the update process, such as device incompatibility with the replacement driver, the original driver would be overwritten causing potential problems on the next boot.

Next, the replacement driver is loaded (step 53) and checked for version errors or incompatibility with the shell driver or device (decision 54). The version error could be flagged by incompatible data structures, incompatible feature sets, attempting to .jump. an incremental update requirement, etc. If an error or incompatibility exists, the replacement driver is unloaded (step 64). If no problem exists, then the update proceeds. A start_update() request is sent from the replacement driver to shell driver 31, to determine whether the original physical layer device driver 32 can be disconnected from the physical device adapter, or whether the replacement driver must wait. Shell device driver 31 checks to see if pending transactions require completion (decision 56), and if so, the shell driver can stall (step 57) until the transactions are complete. (As an alternative, the update request may return a .busy. response.)

When all transactions requiring completion are complete (decision 56), the original device driver is called to disconnect from the physical device (step 59). The replacement driver then calls the get_state() function in shell device driver 31 to initialize itself with the state information maintained in shell device driver 31 (step 60) and the replacement driver is connected to the physical device (step 61). If an error occurs during connection to the physical device (for example, a hardware incompatibility that shell device driver 32 was not designed to detect), the replacement device driver disconnects itself from the physical device and calls cancel_update() in shell device driver 31. The shell device driver 31 calls the original driver's reconnect() function to reconnect the original driver to the physical devices (step 63) and the replacement device

driver is unloaded (step 64). If hardware initialization completes with no error (decision 62) the replacement driver calls the shell device driver's 31 finish_update() function with the replacement driver entry points. The shell device driver 31 then unlinks the entry points of the original driver (step 65), and the entry points of the replacement device driver are connected (step 66) and operation of the device is resumed. Finally, the original device driver is unloaded (step 67). From the time start_update() is called until finish_update() or cancel_update() is called, requests made at entry points to shell device driver 31 are queued on request queues 43.

There are several options available for the initial load of the replacement driver. Another novel feature of this driver architecture is that all or most of the storage for the physical layer device driver 32 can be maintained in the shell driver 31. This provides a benefit in that the memory does not have to be allocated and deallocated when physical layer device driver 32 unloads, and in the case of drivers such as the WINDOWS NT or WINDOWS 2000 SCSI minidriver, follows the design objective requiring that all memory allocation be made at system boot time. The shell driver 31 may allocate extra memory for use with an upgraded driver that might require more memory on installation, and the compatibility check in step 54 may be extended to provide a reduced feature set for the replacement driver on initial load. For example, an upgraded file system driver (FSD) that has a special compression capability requiring large calculation buffers might be loaded as an ordinary FSD during a dynamic update due to lack of buffer space, and at the next system boot will be able to obtain the memory

to utilize its full feature set. The error checking of step 54 may include detection of improper hardware version with respect to the replacement driver, incompatible hardware configuration info structures (physical layer driver state), a condition where a replacement driver has already been loaded without a system reboot (replace once restriction), skipping of version numbers, an attempt to load an older driver than the current version, or lack of sufficient allocable memory.

The stalling operations of decision 56 and step 57 may take alternative forms. For some device drivers it may be advantageous to design physical layer device driver 32 so that it may take over without completing pending transactions. This complicates driver design in that the replacement driver may have to take over interrupt completion for interrupts generated due to actions performed by the original driver and handle any deferred procedure calls (callbacks) that have been queued by the original driver. In other driver designs, it is simpler to allow the original driver to complete all pending tasks and handle all pending system responses while the shell driver .spins. waiting for the original driver to finish. Alternatives between these extremes exist, for example, requiring the original driver to handle all pending interrupts and DPCs, but not to complete queued commands that have not been initiated. The replacement driver can take over the command stack, but not have to handle mid-stream responses from prior issued commands.

Although this specification refers to physical layer device drivers and physical devices such as storage

devices, this should not be construed in a limiting sense. For example, a file system device (FSD), while not actually a physical device, has a device driver that typically must be continuously connected within an operating system. Therefore, the present invention would be very useful in providing dynamic upgrade capability for FSDs as well as other types of device drivers. In addition, for development of device drivers, the techniques of the present invention can improve testing and development facility in any device driver that cannot be easily unloaded and reloaded.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

CLAIMS

What is claimed is:

1. A method for replacing a device driver within a computer system during system operation, when said device driver interfaces at least one physical device with an operating system, said method comprising the steps of:

providing a shell device driver;

providing a replacement device driver;

copying said replacement device driver to a memory location within said computer system;

in response to said shell device driver receiving a request for an update of said device driver, disconnecting said device driver from said at least one device; and

connecting said replacement device driver to said at least one physical device after said disconnection.

2. The method of Claim 1, wherein said method further comprises the step of maintaining a current device state of said at least one physical device within said shell device driver so that control of said at least one physical device is passed from said device driver to said replacement device driver without disruption.

3. The method of Claim 1, wherein said method further comprises the step of maintaining command and response

queues in said shell device driver so that operation of said at least one physical device is not disrupted.

4. The method of Claim 1, wherein said method further comprises the steps of:

unlinking said shell device driver from entry points of said device driver; and

responsive to said unlinking step, linking said shell device driver to entry points of said replacement device driver.

5. The method of Claim 1, wherein said method further comprises the steps of:

responsive to a result of said connecting step, second determining whether or not said replacement driver should be used;

responsive to determining that said replacement device driver should be used, linking said shell device driver to entry points of said replacement device driver; and

responsive to determining that said replacement device driver should not be used, disconnecting said replacement device driver from said at least one physical device and reconnecting said device driver to said at least one physical device, such that prior operation is resumed.

6. The method of Claim 1, wherein said method further comprises the steps of:

responsive to said indicating step, second determining whether or not at least one callback is pending; and

responsive to determining that at least one callback is pending, stalling said steps of disconnecting and connecting until said at least one callback has been handled.

7. The method of Claim 1, wherein said method further comprises the steps of:

responsive to said indicating step, second determining whether or not at least one command requires completion; and

responsive to determining that said at least one command requires processing, processing said at least one command prior to performing said step of disconnecting and said step of connecting.

8. The method of Claim 1, wherein said method further comprises the steps of:

responsive to said indicating step, determining whether or not an entry point routine in said device driver has been entered by at least one thread; and

responsive to determining that said entry point is has been entered, stalling said steps of disconnecting and connecting until said entry point routine is exited by said at least one thread.

9. A computer program product for replacing a device driver within an computer system during system operation use with a data processing system, when said device

driver interfaces at least one physical device with an operating system, and wherein said computer program product comprises:

program code means for providing a shell device driver;

program code means for providing a replacement device driver;

program code means for copying said replacement device driver to a memory location within said computer system;

program code means for in response to said shell device driver receiving a request for an update of said device driver, disconnecting said device driver from said at least one device; and

program code means for connecting said replacement device driver to said at least one physical device after said disconnection.

10. The computer program product of Claim 9, further comprising program code means for maintaining a current device state of said at least one physical device in said shell device driver so that control of said at least one device is passed from said device driver to said replacement device driver without disruption.

11. The computer program product of Claim 9, further comprising program code means for maintaining command and response queues in said shell device driver so that

operation of said at least one physical device is not disrupted.

12. The computer program product of Claim 9, further comprising:

program code means for unlinking said shell device driver from entry points of said device driver; and

program code means for responsive to said unlinking step, linking said shell device driver to entry points of said replacement device driver.

13. The computer program product of Claim 9, further comprising:

program code means for responsive to a result of said connecting step, second determining whether or not said replacement device driver should be used;

program code means for responsive to determining that said second device driver should be used, linking said shell device driver to entry points of said replacement device driver; and

responsive to determining that said replacement device driver should not be used, disconnecting said replacement device driver from said at least one device and reconnecting said device driver to said at least one device, such that prior operation is resumed.

14. The computer program product of Claim 9, further comprising:

* C L M S *

program code means for responsive to said indicating step, second determining whether or not at least one callback is pending; and

program code means for responsive to determining that at least one callback is pending, stalling said steps of disconnecting and connecting until said at least one callback has been handled.

15. The computer program product of Claim 9, further comprising:

program code means for responsive to said indicating step, second determining whether or not at least one command requires completion; and

program code means for responsive to determining that said at least one command requires processing, processing said at least one command prior to performing said step of disconnecting and said step of connecting.

16. The computer program product of Claim 9, further comprising:

program code means for responsive to said indicating step, second determining whether or not an entry point routine in said device driver has been entered by at least one thread; and

program code means for responsive to determining that said entry point is has been entered, stalling said steps of disconnecting and connecting until said entry point routine is exited by said at least one thread.

17. A computer system including a facility for replacing a device driver within said computer system during system operation, when said device driver interfaces at least one physical device with an operating system, wherein said computer system comprises:

file transfer means for copying a replacement device driver to a memory said computer system; and

a shell device driver within said memory, said shell device driver having an update interface to receive an indication that an update of said device driver has been requested, an operating system side entry point interface, and a device driver side entry point interface for selectively coupling a selected device driver, and wherein said shell device driver uncouples said device driver and couples said replacement device driver in conformity with said indication.

18. The computer system of Claim 17, wherein said shell device driver further comprises means for maintaining a current device state of said at least one physical device so that control of said at least one physical device can be passed from said device driver to said replacement device driver without disruption.

19. The computer system of Claim 17, wherein said replacement device driver supplies an indication to said shell device driver of whether or not an error condition occurs on an initialization of said replacement device driver, and said shell device driver re-couples said device driver in conformity with said indication indicating an error condition.

* C L M S *

20. The computer system of Claim 17, wherein said shell device driver further comprises means for stalling said uncoupling of said device driver if an entry point within said entry point interface has been entered.