9704718-7
IPOS

Disk-Based Software Image Use from a Read Only Memory Device

Background of the Invention

1. Field of the Invention

The present invention is directed to a system and method for enabling a computer system to use a read only memory device to store and execute software written for a disk-based environment. More particularly, the present invention relates to computer system and operating system processes and structures necessary to use disk-based software stored on a linear read only memory.

2. Background and Related Art

Semiconductor read only memory devices (ROM) have historically been used in computer systems to store invariant portions of executable code such as system initialization tasks and the initial program load of the operating system. The requirement that the ROM code be

invariant and the relative cost of ROM devices compared to other forms of media limited use of these ROM devices to small portions of code. The major parts of personal computer operating systems have traditionally been contained on removable diskettes or hard disk drives. Semi-conductor read only memory devices, such as memories manufactured by Texas Instruments and Advanced Micro Devices are computer chips that store data at linearly addressable locations. These memories are distinguished from compact disk read only memories (CDROM) that use optical technologies for storing read only code on a removable media having a particular physical organization.

Personal computer operating systems such as DOS, IBM OS/2, and Microsoft Windows were written for systems using diskette or hard disk storage devices for operating system storage. The loading and execution of these programs assumes that the operating system is stored on the disks based upon the geometry of those disks. Loading the operating system initially and accessing

operating system function in support of application programs relies upon the disk geometry of the removable or hard disk. The geometry includes defined disk sectors containing data. Operating system performance is enhanced by directly accessing disk sectors by location. The system therefore becomes dependent on the physical disk organization.

The reduction in size and cost of microprocessors and semiconductor memory have expanded the use of computers to provide embedded functions in machinery from automobiles to home appliances. Embedded controllers provide increasing amounts of function and often operate in a manner similar to a stand alone personal computer. Application programs can be more rapidly and efficiently developed for these embedded systems when the programmers can use standard personal computer or workstation tools during the development phase and then simply load those applications onto the read only device as part of the embedded system.

Historically embedded microprocessors used specialized operating environments or operating systems to support their function. The ability to use standard operating systems for development requires that standard operating system function be provided for the embedded system on the read only memory.

Special versions of standard disk-based operating systems have been provided for the embedded system market. For example, Microsoft offers a product entitled "ROM DOS" that is a version of their DOS operating system usable on a ROM device. IBM offers a version of the IBM Microkernel for use in embedded controllers. The creation of a ROM based version of an operating system, however, requires that the operating system be modified to consider the fact that it is executing from a ROM device instead of a disk image. The analysis and modification required for a complex operating system such as IBM OS/2 or Microsoft Windows 95 would consume several person years and be uneconomical.

THIS SHEET HAS BEEN REPLACED

The technical problem therefore exists to devise a system and method that will enable a standard disk-based operating system to be executed from a ROM image. The solution is required to handle the limitations imposed by operating system dependent on disk geometries.

Summary of the Invention

The present invention is directed to providing a system, method and program product for enabling a computer system to load and use a disk-based operating system from a ROM image. The present invention provides a means for creating the ROM image as well as using the ROM image.

The present invention provides a means for intercepting a computer system request for data from an operating system disk image and redirecting those requests to the ROM image.

The present invention is directed to a method for executing a disk based operating system stored on a ROM

This Sheet Has Been Replaced

THIS SHEET HAS BEEN REPLACED

device, the method operates on a computer system having a processor and memory. The method comprises the steps of generating a mapping of disk geometry to ROM device address for the disk based operating system; intercepting system requests for access to the disk-based operating system; transforming the disk based access request into a ROM access request; and accessing and returning the ROM data corresponding to the disk-based access request.

It is therefore an object of the present invention to provide a method for loading a disk-based operating system from a ROM device.

It is yet another object of the invention to enable an application program to use operating system services of a disk-based operating system stored as a ROM image.

It is yet another object of the invention to provide a method for creating and using a ROM image containing a disk-based operating system in an embedded environment.

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular description of a preferred embodiment of the invention, as illustrated in the accompanying drawing wherein like reference numbers represent like parts of the invention.

Brief Description of the Drawing

Fig. 1 is a diagram showing the processing flow of prior art computer systems during boot and load of the operating system.

Fig. 2 illustrates the flow of application requests for operating system function in real and protect mode computer systems.

Fig. 3 is an illustration of disk sector geometry in a typical hard disk.

This Sheet Has Been Replaced



Fig. 4 is an illustration of linear memory of a ROM device according to the preferred embodiment of the present invention.

Fig. 5 is a block diagram illustrating a hardware configuration according to the present invention.

Fig. 6 is a diagram illustrating memory relationships according to the present invention.

Fig. 7 is a flow chart illustrating the process for creating an executable ROM image of a disk-based operating system according to the present invention.

Fig. 8 is a flow chart illustrating the flow of control during IPL and operation of a computer system according to the present invention.

Fig. 9 is a process diagram illustrating the flow of power on, IPL and execution of a disk-based operating

system from a ROM image according to the present invention.

Detailed Description

Computer systems require programmed instructions to operate. Most modern computer systems employ some form of operating system to control the processing of application programs that accomplish the user function. The operating system programs manage computer resources such as memory, storage devices, and communication devices. The operating system is initially loaded when power is applied to the computer system. System hardware, however, cannot begin to load the operating system without programmed instructions to do so. Thus, the computer hardware is typically provided with a read only memory (ROM) containing the basic instructions necessary to start the computer and begin loading the operating system from a disk drive or diskette.

The preferred embodiment of the present invention works in conjunction with an Intel microprocessor and

computer system similar to an IBM Personal Computer. The preferred embodiment, however, is an embedded computer system that is designed to perform a limited set of application functions without general purpose interaction typical of a personal computer. The embedded computer system of the present invention will limit the ability of the user to change programs or applications and will in most cases not provide any writable storage for the system user.

When the power switch 100 of the computer system of the preferred embodiment (Figure 1) is turned, on the system performs initial hardware tests to verify the hardware configuration and begin loading the operating system. The power on self test (POST) function 102 is contained in semiconductor ROM device 104. The final step of the power on self test processing is to initiate the IPL 106 (initial program load) of the operating system. The system configuration typically causes the system to check first on removable media such as diskette and secondly on hard disks to find the operating system

code. Loading of the operating system 108 proceeds from the selected disk or diskette 110 until the operating system is fully initialized. After initialization the operating system performs the necessary supervisory functions 112 including the servicing of application program requests for operating system services 114.

Application access to operating system functions stored in a disk drive follow the flow illustrated in Figure 2. Application 114 makes the request of the operating system 112. In a real mode operating system such as DOS, the operating system generates an interrupt 13 202 requesting access to data on the hard disk. Interrupt 13 (INT13) is a system API that requests services from the basic input/output system (BIOS) responsible for interacting with the hardware.

The operating system int13 202 requests I/O from a specific address. The address RBA (relative block address) is translated into a specific disk sector address that is passed to the hard disk controller 206

which accesses the hard disk itself 208. In this sequence, the hard disk is either a permanently mounted disk or removable diskette device. When the operating system is running in the processor protect mode the operating system is able to directly request data from the hard disk controller 206. In both the real mode and protect mode operations, the I/O requests are dependent upon the disk geometry. For example, an int13 request 202 causes an I/O address RBA translation 204 providing a specific disk geometry based request (a sector request) to the hard disk controller 206. In protect mode, the sector request is made directly to the hard disk controller.

The layout of a hard disk is illustrated in Fig. 3. The hard disk is divided into a number of cylinders and sectors each containing data. The hard disk controller 206 accesses disk 302 by specifying a sector to be retrieved.

An operating system loaded on a read only memory (ROM) device is organized linearly in an address space running, for example, from address 0X to address nnnx. Access to data in the random access read only memory requires specification of a linear memory address rather of a hard disk sector.

The prior art solution to providing a ROM based operating system required that the operating system code be analyzed in every invocation of the Int13 API or, protect mode hard disk access be located and changed to access the linear memory of the ROM device. This effort can consume a significant amount of development resources limiting the use of operating systems to those situations that justify this high level of expense.

The preferred embodiment of the present invention enables the use of any disk based operating system in a ROM environment. The present invention introduces a device driver into the system environment that intercepts Int13 interrupts and hard disk controller accesses and

transforms those requests into addressable reads of the ROM device. The introduction of this device driver allows the operating system to run without changes even though it has been loaded into a ROM device.

The hardware environment of the present invention is shown generally in Figure 5. An embedded system typically has a processing unit 502 consisting of one or more central processing units (CPU) such as the Intel Pentium processor (Pentium is a trademark of the Intel Corporation) or the IBM PowerPC processor (PowerPC is a trademark of the IBM Corporation). In addition the system has random access memory 504 and ROM storage 506. An input/output controller 508 enables the system to communicate with external devices necessary for the operation of the system. These devices may include a keyboard and display or may be connected to specialized equipment for interacting with a supported environment.

The ROM device 506 may be a single ROM chip or multiple ROM chips depending upon the particular

structure of the application. Use of a single ROM would replace the standard system ROM with a larger ROM containing both system function and the embedded operating system. Another approach is to separate a system ROM from the operating system ROM and supply two different devices.

The present invention uses a system ability to "hook" interrupts and to cause execution of specified code whenever that interrupt is raised. The ability to hook and execute provided code is used in existing systems to implement device driver logic. The present invention introduces a device driver-like structure that hooks the operating system disk interrupt 13 and disk controller I/O accesses.

As discussed above, the operating system on a ROM device is organized linearly much like the memory of the computer system itself. Linear access to this memory requires that device driver specify an address of the section of memory containing the required code. Fig. 6

illustrates the typical memory layout of a computer system according to the present invention. Memory is shown extending from zero to 16M (16 megabytes of memory) extending from the bottom to the top of the figure. The address area between zero and C0000 is reserved for program execution. Part of this area of memory (640KB in the range 0000-A0000) is used by DOS or windows applications during program execution. The remainder of this area is used for VGA and monochrome video adapter memory.

The memory between C0000 and 1MG is reserved for device drivers. Video device drivers and drivers associated with adaptor cards et cetera are mapped into this area of the memory. This mapping enables the operating system to access data contained on the adapters by accessing addresses in this memory range. Address access over the system bus causes the return of data from the adapter or from the random access memory as appropriate.

The example in Figure 6 assumes that the computer system has a random access memory physical size of 8 megabytes. The preferred embodiment maps the ROM based operating system into the area beyond the system's physical memory. In this example, the operating system ROM will be mapped to memory locations beginning at 8 megabytes and extending to the 16 megabyte limit. Access to these memory addresses will be intercepted by the ROM and the appropriate operating system information returned.

The disk-based operating system is converted to a ROM image using the process described in Figure 7. First, a bootable disk is created containing the operating system.

The disk area occupied on the bootable disk is then defragmented to ensure that the code is compressed into a single block rather than being placed on the disk with spaces of unused disk area in-between. Typical disk writing algorithms place the data in available sectors

which may not be contiguous and may leave substantial unused space in the disk image.

The defragmented image 704 will be compressed in the sense that the files will follow linearly from the beginning of the disk without blank spaces. Next, a sector based copy of the disk including the boot record is created 706. This sector based copy is a single linear image created from the disk but containing information on the relative sectors. The image is then loaded onto the ROM device along with the Int 13 hook device. The ROM device is then available for use on an embedded system.

One additional step is required to enable use of the ROM image on the computer system. The power on self-test ends by initiating the IPL of the operating system. In Intel-based systems this occurs using an interrupt 19 (int19). The initial boot ROM must be changed, according to the present invention, so that the final step of the

boot sequence enables the loading of the Int 13 device driver prior to the operating system load.

In Figure 8 the power on self test is performed 802. The system next hooks interrupt 19 so that the IPL sequence recognizes the need to load the Int13 device driver. During IPL the Int13 and the hard disk controller trap device drivers are loaded and the operating system IPLed. The mapping of disk RBA and sector information to linear ROM addresses can be hand coded or can be generated by the device driver when it detects the ROM image. The operating system then runs 808 in a standard manner. The overall process is shown in Fig. 9 where the prior art boot sequence and the modified steps of the present invention are shown. ROM 104' contains the modified system power on self test and IPL initiation sequence. This is modified slightly to hook the Int 19. ROM 902 contains the image of the operating system and the Int 13 device driver code.

The hook routine that hooks the Int 13 for real time mode and traps the I/O request in protect mode finds the location of the OS ROM and reads the associated file system structure from the ROM. The system uses the physical disk geometry information gathered from the file system in the OS ROM and creates a table which correlates a physical read request to a specific RBA (the memory location containing that data). The system uses this table to access the ROM device to satisfy operating system code requests referenced by disk geometry.

Physical disk geometry will differ by operating system and file system. The present invention is adaptable for use with any operating system and file system and also can be used on processors that contain difference basic I/O system structures. The different geometry and BIOS processes are reflected in modified device driver code and disk request intercept processing.

The preferred embodiment of the present invention locates the device driver in the ROM scan area 602 of the

system. This area is scanned at boot time by systems that use an IBM personal computer compatible architecture. Use of the present invention with differing architectures would require placement of the intercepting device driver code in an appropriate location for that system.

It will be understood from the foregoing description that various modifications and changes may be made in the preferred embodiment of the present invention without departing from its true spirit. It is intended that this description is for purposes of illustration only and should not be construed in a limiting sense. The scope of this invention should be limited only by the language of the following claims.

THIS SHEET HAS BEEN REPLACED



Claim(s)

1. A method for executing a disk based operating system stored on a ROM device, the method operating on a computer system having a processor and memory, the method comprising the steps of

generating a mapping of disk geometry to ROM device address for the disk based operating system;

intercepting system requests for access to the disk-based operating system;

transforming the disk based access request into a ROM access request; and

accessing and returning the ROM data corresponding to the disk-based access request.

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED



2. The method of claim 1, wherein the step of generating a mapping comprises the steps of:

generating a bootable disk image of said disk-based operating system;

copying said bootable disk image to said ROM device;

generating data indicating the corresponding ROM address location for each bootable disk image sector based on said bootable disk image geometry.

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED



3. The method of claim 1, wherein the step of intercepting comprises:

intercepting an interrupt request for disk access to a disk sector; and

intercepting a request to a disk controller for access to a disk sector.

4. The method of claim 1, wherein the step of intercepting comprises:

loading a device driver responsive to disk access requests;

passing control to said device driver whenever a disk access interrupt or disk access command is detected.

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

5. The method of claim 4, wherein the disk access interrupt intercepted is an INT13 on an Intel processor.

6. A system for executing a disk based operating system contained in a read only memory (ROM) device, said ROM device being addressable using linear memory addresses, the system comprising:

means for intercepting operating system disk access requests;

means for transforming said disk access requests into ROM device addressable accesses;

means for accessing said ROM device using said transformed request.

7. The system of claim 6, further comprising:

This Sheet Has Been Replaced

THIS SHEET HAS BEEN REPLACED

* C L M S *

means for returning accessed information to a requesting process.

8. The system of claim 6, wherein said means for intercepting is a device driver.

9. The system of claim 8, wherein said means for intercepting intercepts INT13 interrupts.

10. The system of claim 8, wherein said means for intercepting intercepts protect mode disk sector access requests to a hard disk controller.

11. The system of claim 6, further comprising:

means for transforming a disk sector request into a linear memory addressable request.

12. A computer program product having a computer readable medium having computer program logic recorded

This Sheet Has Been Replaced

THIS SHEET HAS BEEN REPLACED



thereon for running a disk-based operating system from a ROM device, said computer program product comprising:

computer program product means having computer readable means for intercepting operating system disk access requests;

computer program product means having computer readable means for transforming said disk access requests into ROM device addressable accesses;

computer program product means having computer readable means for accessing said ROM device using said transformed request.

13. The program product of claim 12, further comprising:

computer program product means having computer readable means for returning accessed information to a requesting process.

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

14. The program product of claim 12, wherein said computer program product means for intercepting is a device driver program product.

15. The program product of claim 14, wherein said program product means for intercepting intercepts INT13 interrupts.

16. The program product of claim 14, wherein said program product means for intercepting intercepts protect mode disk sector access requests to a hard disk controller.

17. The program product of claim 12, further comprising:

computer program product means having computer readable means for transforming a disk sector request into a linear memory addressable request.

This Sheet Has Been Replaced