(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 824 233 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
18.02.1998 Bulletin 1998/08

(51) Int Cl.[6]: **G06F 1/00**, G06F 1/30, G06F 11/00

(21) Application number: **97305891.0**

(22) Date of filing: **04.08.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV RO SI

(30) Priority: **07.08.1996 US 693458**

(71) Applicant: **Compaq Computer Corporation**
Houston Texas 77070 (US)

(72) Inventors:
- **Angelo, Michael F.**
  Houston, Texas 77068 (US)
- **Miller, Craig A.**
  Cedar Park, Texas 78613 (US)

(74) Representative: **Brunner, Michael John**
GILL JENNINGS & EVERY
Broadgate House
7 Eldon Street
London EC2M 7LH (GB)

(54) Method and apparatus for secure execution of software prior to a computer system being powered down or entering a low energy consumption mode

(57) A computer system that automatically and securely executes registered programs immediately prior to a transition to a reduced energy consumption state. A registrar table specifying registered programs and a secure modification detection value for each registered program are maintained in system management mode memory or other secure memory space in the computer system. A system management interrupt is generated following a request to remove power from the computer system or the occurrence of an event that triggers an energy saving mode. The system management interrupt handler routine then generates a current modification detection value for each registered program. The current modification detection values are compared with the secure modification detection values. Execution of a registered program is permitted if the values match. After all registered programs have been executed, the computer system automatically powers down or enters an energy saving mode. The computer system thereby allows secure and convenient execution of programs or commands that would typically interfere with normal computer use.

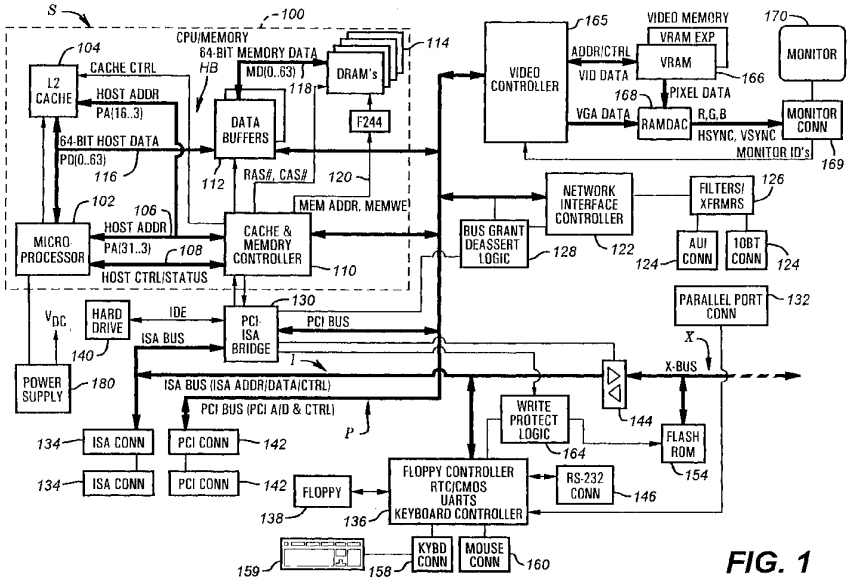

FIG. 1

EP 0 824 233 A2

Description

The present invention relates to computer system security.

The present invention relates to a method for securely executing registered software applications in a computer system that is either being powered down or entering an energy saving mode.

Computers are becoming increasingly important in many aspects of modern life, both in homes and in businesses. Huge amounts of money are invested by companies and individuals to purchase executable software. Even more money and time is spent developing the information contained in data files such as text documents and spreadsheets. Protecting these resources is therefore an important concern. Security-conscious users are requesting that security and integrity features be incorporated into their personal computers to protect access to critical files and to guarantee the trustworthiness of installed programs. Ideally, these security features should interfere with normal computer operation as little as possible.

Two main causes of software untrustworthiness are file corruption and viruses. File corruption usually follows a system failure occurring during a file transfer (i.e. the system is turned off while a file is being copied onto the hard disk, etc.) or similar occurrence.

Controlling the power-down of the computer system is therefore important, particularly in computers with advanced operating systems such as Windows 95™ and Windows NT™, available from Microsoft Corp. These operating systems require the user to shut down via specific software steps rather than by simply turning off the power switch. For example, in Windows 95™, the user should click a START button and select the SHUT DOWN item from the START menu. The selection of the SHUT DOWN item causes a dialog box to appear on the screen, giving the user the options of shutting down completely, restarting the PC, or exiting to the disk operating system (DOS).

In these advanced operating systems, the shut down procedure is needed because the numerous pieces of status information and configuration data contained in the Windows Registry file are not updated until the system has been properly shut down. Further, data stored in the disk cache may not be flushed to the disk unless the user properly exits Windows 95™ or Windows NT™. Network connections that are not properly severed can cause additional problems. Thus, the removal of power without following the proper shutdown procedure can corrupt the Windows Registry file and compromise the overall reliability ofthe computer during subsequent operations. It should be noted, however, that properly exiting these operating systems requires the user to take affirmative action via menu commands prior to toggling the on/offpower switch.

Another threat to software integrity is the problem of "malicious code", also referred to as computer viruses. While many computer viruses are relatively benign, computer viruses can be hostile, clandestine and created to target specific types of software or hardware. They can be introduced into a computer in as many ways as the computer can communicate externally, such as through the floppy drive, a network connection or a modem connection. Viruses are typically designed to replicate by secretly attaching copies of themselves to files or boot records so that the user is unaware of the intrusion. It is important to note that once a virus has attached itself to a host program, the program must be different and its integrity has been violated.

Once infected, any subsequent copies of the host file also contain the virus, thereby increasing the potential for destruction. The virus is then activated when the file is executed. Consequently, a virus attached to a data file may remain dormant because the data file is not executable.

One common commercial method of assessing the integrity of user software is to check for viruses by running a virus checking software program. Such programs rely on the characteristics of the known viruses to detect their presence. A new virus may not be detectable by the virus checking software. If a virus is present, the virus checking software itself is susceptible because it is loaded from the infected hard disk and must run in memory that could be infected. In addition, virus checking software can be inconvenient to execute. A thorough check of system resources can take several minutes, and the user is not able to run other applications during this time. Although virus checking software can be configured to execute automatically during system boot up, the user must again take affirmative action to execute or schedule a virus scan at other times.

Another method of assessing a file's integrity prior to executing involves computing an integrity assessment code for the file and verifying that the code matches a predetermined value. Checksums (a type of integrity assessment code) are adequate for detecting accidental modifications of data. However, they are an insecure defense against viruses. A well-designed virus aimed at bypassing normal security features can easily attach itself to a host program without resulting in a different checksum.

To address this problem, advanced modification detection codes (or MDC's) have been developed to specifically detect deliberate corruption of data, and are superior to simple checksums. The intent of MDC's is to make it computationally infeasible to modify data so as to preserve a specific modification detection code value. Modification detection codes are sometimes referred to by other names, including: "cryptographic checksums", "cryptographic hashes", "secure hash algorithms", and "message digests".

In some earlier systems, a secure hash value is calculated and stored for newly installed software. Thereafter, when the computer is turned on again, the stored hash value is compared to a newly calculated value. If

a discrepancy is found, the user is alerted. A main disadvantage with this method is that the integrity assessment codes must be stored on the hard disk, thus making the codes themselves susceptible to attack by malicious code. Reverse-engineering a modification detection code, while difficult, is not a mathematically intractable problem. Thus, software-only protective products can offer only limited insurance against the attack of malicious code, due mainly to architectural weakness present in most computer systems. A potential solution is to embed the modification detection code in a permanent read-only memory device, but this can make system reconfiguration quite difficult.

Some degree of protection from data loss is afforded by performing regular backups to a tape drive or similar storage medium. If a file becomes corrupted, an earlier, trusted version can be restored from a backup tape. Any changes made to the file after the backup was performed are lost. Like virus scanning and various other administrative procedures, performing backup operations usually preempts other uses of the computer. To circumvent this potential inconvenience, it is desirable to schedule backups during non-working hours or at times when the user is away from the machine. Scheduling and running the backups also require some sort of affirmative action to be taken by the user or system administrator.

A problem can arise if backups and other operations are scheduled to execute at times when it is unlikely that the computer system will be in use. Most modern computer systems incorporate "energy saving" or "hibernation" features. Techniques that are utilized to conserve energy include powering down disk drives, disabling monitors and reducing processor and system clock frequencies. These features are typically activated when the computer is not used for a predetermined period of time. Depending on its programming and hardware, a computer system may not acknowledge and execute a scheduled operation while the system is in an energy saving mode. Even if a scheduled operation is recognized, current computer architectures cannot ensure secure execution.

Briefly, the present invention provides a computer system having the capability to automatically and securely execute registered commands or applications immediately prior to the computer powering down or entering a low energy consumption mode.

Following a request to remove power from the computer system or enter a low power consumption mode, a system management interrupt (SMI) is generated. According to the invention, a variety of methods can be used to generate the SMI. In one embodiment, closure or toggling of the power supply on/off switch causes special interrupt circuitry to generate an interrupt service request that instructs the processor to jump to an interrupt service routine which results in a power down SMI being asserted. Alternatively, circuitry coupled to the power supply on/off switch Is configured to bypass the interrupt request and generate the power down SMI directly without the need for a standard interrupt. In yet another embodiment, toggling the power supply on/off switch initiates a software process that results in a power down SMI.

A computer system according to the present invention also allows automatic and secure execution of registered applications immediately prior to the computer system entering a low power consumption mode. Examples of such a low power consumption mode include "hibernation mode" and "energy saving mode". In this embodiment, an SMI is again generated in one of a number of ways. Special interrupt circuitry, a keyboard interrupt, activity timers or a software process can all be used to generate the SMI.

Regardless of the manner in which it is generated, the power down or hibernation mode SMI places the computer system in system management mode, causing an SMI handler routine to be executed. In turn, the SMI handler responds by executing all applications registered with the application registrar. Importantly, the registered applications are verified and executed in a secure manner. Before executing a registered application, the SMI handler first generates a current hash value for the program. The term "secure hash value" or "hash value" is used throughout the remainder of this specification to refer generally to a value generated by a modification detection code, the value being specific to a given software application. A "secure hash value" in the preferred embodiment is 160 bits of data (20 bytes) that is essentially a mathematical representation of a file. If any bits in the file are changed, a different hash value will result.

In general, a secure hash table (or other type of integrity assessment code) is provided that contains a secure hash value for each program that the user wants to execute prior to the power down or entry into hibernation mode. The hash table is stored in protected memory that can only be accessed when the computer system is in system management mode. After it has generated a current hash value for the registered application, the SMI handler checks this stored hash table for a secure entry for the application. If a hash value entry is found, it is compared with the newly-calculated hash value for the secured application. In the event the two values match, the integrity of the application is guaranteed and it is loaded into memory and executed. The process is repeated until all applications registered with the application registrar have been executed.

If the two values do not match, the user is alerted to the discrepancy and may be given the option to update or override the stored hash table entry by entering an administrative password. For security sensitive applications, the entire application or a portion of it is loaded into system management mode memory (hereinafter "SMM memory") prior to application.

In an alternate embodiment of the invention, a secured hash value for the table is maintained in SMM

memory, with the hash table itself is stored in normal memory. A current table hash value is generated for the hash table before a hash table entry is accessed. The current table hash value is then compared with the table hash value stored in SMM memory. If the values are equal, the integrity of the hash table is verified and the new hash value of the program to be executed can be safely compared with its original value. This embodiment ofthe invention is useful for overcoming problems associated with the limited size of SMM memory. Both of the aforementioned embodiments of the invention have the additional advantage of being operating system independent.

After all of the registered applications have been executed, the SMI handler transmits a shutdown command to a decoder over a system bus if the SMI was generated as a result of a power down request. Upon detecting that the computer system has issued a shutdown command, the decoder logic causes a SHUTDOWN input to the power supply to be asserted, thereby disabling power to the system. If the SMI was generated as a result of low power consumption mode being activated, the SMI handler transmits appropriate commands to hibernation logic that controls various system components.

The present invention has a wide variety of potential applications, including secure execution of virus detection and removal programs and backing up files prior to shutting down. These and other registered applications are executed securely and without need for intervention by the user.

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

Figure 1 is a schematic block diagram of a computer system incorporating system management mode capabilities in accordance with the present invention;

Figure 2 is a graphical representation of System Management Mode memory according to the present invention;

Figure 3 is a schematic block diagram of a power down circuitry associated with the power supply of the computer system of Figure 1;

Figure 4 is a block diagram of the power supply of the computer system of Figure 1;

Figure 5 is a schematic block diagram of hibernation circuitry according to the present invention;

Figure 6 is a flowchart illustration of a method according to the present invention for securely executing and verifying the integrity of software applications prior to the computer system being turned off or entering hibernation mode; and

Figure 7 is a flowchart illustration of a secure method according to the present invention for updating a stored hash table or stored hash value.

Referring first to Figure 1, a computer system S according to the present invention is shown. In the preferred embodiment, the system S incorporates two primary buses: a Peripheral Component Interconnect (PCI) bus P which includes an address/data portion and a control signal portion; and an Industry Standard Architecture (ISA) bus I which includes an address portion, a data portion, and a control signal portion. The PCI and ISA buses P and I form the architectural backbone of the computer system S.

A CPU/memory subsystem 100 is connected to the PCI bus P. The processor 102 is preferably the Pentium® processor from Intel Corporation, but could be an 80486 or any number of similar or next-generation processors. The processor 102 drives data, address, and control portions 116, 106, and 108 of a host bus HB. A level 2 (L2) or external cache memory 104 is connected to the host bus HB to provide additional caching capabilities that improve the overall performance of the computer system S. The L2 cache 104 may be permanently installed or may be removable if desired. A cache and memory controller 110 and a PCI-ISA bridge chip 130 are connected to the control and address portions 108 and 106 of the host bus HB. The cache and memory controller chip 110 is configured to control a series of data buffers 112. The data buffers 112 are preferably the 82433LX from Intel, and are coupled to and drive the host data bus 116 and a MD or memory data bus 118 that is connected to a memory array 114. A memory address and memory control signal bus is provided from the cache and memory controller 110.

The data buffers 112, cache and memory controller 110, and PCI-ISA bridge 130 are all connected to the PCI bus P. The PCI-ISA bridge 130 is used to convert signals between the PCI bus P and the ISA bus I. The PCI-ISA bridge 130 includes: the necessary address and data buffers, arbitration and bus master control logic for the PCI bus P, ISA arbitration circuitry, an ISA bus controller as conventionally used in ISA systems, an IDE (intelligent drive electronics) interface, and a DMA controller. A hard disk drive 140 is connected to the IDE interface of the PCI-ISA bridge 130. Tape drives, CD-ROM devices or other peripheral storage devices (not shown) can be similarly connected.

In the disclosed embodiment, the PCI-ISA bridge 130 also includes miscellaneous system logic. This miscellaneous system logic contains counters and activity timers as conventionally present in personal computer systems, an interrupt controller for both the PCI and ISA buses P and I, and power management logic. Additionally, the miscellaneous system logic may include circuitry for a security management system used for password verification and to allow access to protected resources.

The PCI-ISA bridge 130 also includes circuitry to generate a "soft" SMI (System Management Interrupt), as well as SMI and keyboard controller interface circuitry. The miscellaneous system logic is connected to the flash ROM 154 through write protection logic 164. Pref-

erably, the PCI-ISA bridge 130 is a single integrated circuit, but other combinations are possible.

A series of ISA slots 134 are connected to the ISA bus I to receive ISA adapter cards. A series of PCI slots 142 are similarly provided on the PCI bus P to receive PCI adapter cards.

A video controller 165 is also connected to the PCI bus P. Video memory 166 is used to store graphics data and is connected to the video graphics controller 165 and a digital/analog converter (RAMDAC) 168. The video graphics controller 165 controls the operation of the video memory 166, allowing data to be written and retrieved as required. A monitor connector 169 is connected to the RAMDAC 168 for connecting a monitor 170.

A network interface controller (NIC) 122 is also connected to the PCI bus P. Preferably, the controller 122 is a single integrated circuit that includes the capabilities necessary to act as a PCI bus master and slave, as well as circuitry required to act as an Ethernet interface. Attachment Unit Interface (AUI) and 10 base-T connectors 124 are provided in the system S, and are connected to the NIC 122 via filter and transformer circuitry 126. This circuitry forms a network or Ethernet connection for connecting the computer system S to a local area network (LAN).

A combination I/O chip 136 is connected to the ISA bus I. The combination I/O chip 136 preferably includes a real time clock two UARTS, a floppy disk controller for controlling a floppy disk drive 138, and various address decode logic and security logic to control access to the CMOS memory (not shown) and power-on password values. A control line is provided to the read and write protection logic 164 to further control access to the flash ROM 154. Serial port connectors 146 and parallel port connector 132 are also connected to the combination I/O chip 136.

An 8042 or keyboard controller is also included in the combination I/O chip 136. The keyboard controller is of conventional design and is connected in turn to a keyboard connector 158 and a mouse or pointing device connector 160. A keyboard 159 is connected to the computer system S through the keyboard connector 158.

A buffer 144 is connected to the ISA bus I to provide an additional X-bus X for various additional components of the computer system S. A flash ROM 154 receives its control, address and data signals from the X-bus X. Preferably, the flash ROM 154 contains the BIOS information for the computer system and can be reprogrammed to allow for revisions of the BIOS.

In the computer system S of Fig. 1, all electronic devices discussed above, including the processor 102, are powered by a regulated power supply 180. In the preferred embodiment, the regulated power supply (Figs. 3 and 4) has a power supply supervisory circuit 192 that provides shutdown capability via a SHUTDOWN input. The power supply 180 is shut-down via an SMI software/hardware process that is initiated by toggling the on/off switch 182 (Fig. 3). The power supply 180 receives an AC voltage supply via an AC plug 190 (Fig. 3).

An additional feature of the computer system S is a System Management Mode (SMM), as discussed at length immediately below. It is also noted that Figure 1 presents an exemplary embodiment of the computer system S and it is understood that numerous other effective embodiments could readily be developed as known to those skilled in the art.

Certain microprocessors, such as the Pentium® processor from Intel Corporation, have included a mode referred to as system management mode (SMM), which is entered upon receipt of a system management interrupt (SMI). Originally, SMIs were power management interrupts devised by Intel Corporation for portable systems. Portable computers often draw power from batteries which provide a limited amount of energy. To maximize battery life, an SMI is typically asserted to turn off or reduce the power to any system component that is not currently in use. Although originally meant for laptop computers, SMIs have become popular for desktop and other stationary models as well.

SMIs are asserted by either an SMI timer, by a system request, or by other means. An SMI is a non-maskable interrupt having almost the highest priority in the system. Only the reset signal R/S* and cache flush signal FLUSH*, which can be conceptualized as interrupts, have a higher priority than the SMI. When an SMI is asserted, a microprocessor maps a portion of memory referred to as the system management mode memory ("SMM memory") into the main memory space. The entire CPU state is then saved in the SMM memory (in the CPU register dump 210 of Fig. 2) in stack-like, last in/first out fashion. After the initial processor state is saved, the processor 102 begins executing an SMI handler routine, which is an interrupt service routine to perform specific system management tasks such as reducing power to specific devices or, as in the case of the present invention, providing security services. While the routine is executed, other interrupt requests are not serviced, and are ignored until the interrupt routine is completed or the microprocessor is reset. When the SMI handler completes its task, the processor state is retrieved from the SMM memory, and the main program continues. An SMI active signal referred to as the SMIACT* signal is provided by the processor to indicate operation in SMM.

As mentioned, following assertion of its SMI input (this is generally an active low signal), the processor 102 calls the SMI handler, which addresses an address space that is separate from ordinary main memory. Thereafter, all memory accesses refer only to SMM memory 200. Input/output ("I/O") accesses via instructions such as IN or OUT are still directed to the normal I/O address space, however. One advantageous side-effect of the hardwired separate address SMM area is that the routines stored in this space cannot be snooped by the cache, providing an additional layer of protection.

In a typical system management mode implemen-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.