(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 January 2007 (18.01.2007)

PCT

(10) International Publication Number
WO 2007/008880 A2

(51) International Patent Classification:
G06F 3/00 (2006.01)

(21) International Application Number:
PCT/US2006/026860

(22) International Filing Date: 10 July 2006 (10.07.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/177,079 8 July 2005 (08.07.2005) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventor: **BEN-ZVI, Nir**; One Microsoft Way, Redmond, Washington 98052-6399 (US).

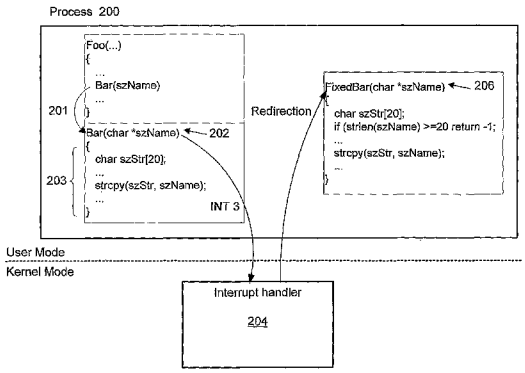
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ,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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) **Title:** CHANGING CODE EXECUTION PATH USING KERNEL MODE REDIRECTION



(57) **Abstract:** A mechanism for redirecting a code execution path in a running process. A one-byte interrupt instruction (e.g., INT 3) is inserted into the code path. The interrupt instruction passes control to a kernel handler, which after executing a replacement function, returns to continue executing the process. The replacement function resides in a memory space that is accessible to the kernel handler. The redirection mechanism may be applied without requiring a reboot of the computing device on which the running process is executing. In addition, the redirection mechanism may be applied without overwriting more than one byte in the original code.

WO 2007/008880 A2

CHANGING CODE EXECUTION PATH USING KERNEL MODE REDIRECTION

FIELD OF THE INVENTION

[0001] This invention relates in general to the field of computer software. More particularly, this invention relates to a method of updating a process running in memory.

BACKGROUND OF THE INVENTION

[0002] It is often desirable to change a code execution path in a running process without changing the original on-disk image of the executing modules or without requiring a restart of the computer. One way to accomplish this is via a "Hotpatching" mechanism. Hotpatching is in-memory patching mechanism that enables the installation of software updates without requiring users to restart their computers by automatically inserting code from a software update into a running process. This means that system files can be updated while they are in use.

[0003] For example, Hotpatching may bypass a vulnerable function in a running process by injecting a JMP instruction at the beginning of the vulnerable function. When the function is called, it jumps to a new function that is also loaded into the process space by the Hotpatching mechanism. The problem with this approach is that an injected JMP instruction may overwrite multiple instructions in a way that leads to unexpected behavior. In the Hotpatching case, if the beginning of the vulnerable function includes 3 assembly opcodes in the first 5 bytes (1 byte opcode, 2 bytes opcode, 2 bytes opcode), the JMP injection will replace all five bytes. If the processor is executing the first byte opcode, and the injection has changed the next two opcodes, unexpected processor behavior may result.

SUMMARY OF THE INVENTION

[0004] A mechanism for redirecting a code execution path in a running process. A one-byte interrupt instruction (e.g., INT 3) is inserted into the code path. The interrupt instruction passes control to a kernel handler, which after executing a replacement function, returns to continue executing the process. The replacement function resides in a memory space that is accessible to the kernel handler. The redirection mechanism may be applied without requiring a reboot of the computing device on which the running process is executing. In addition, the redirection mechanism may be applied without overwriting more than one byte in the original code.

[0005] Additional features and advantages of the invention will be made apparent from the following detailed description of illustrative embodiments that proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The foregoing summary, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0007] Fig. 1 is a block diagram showing an exemplary computing environment in which aspects of the invention may be implemented; and

[0008] Fig. 2 illustrates an exemplary process performed in accordance with the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0009] **Exemplary Computing Environment**

[0010] Fig. 1 illustrates an example of a suitable computing system environment 100 in which the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

[0011] The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0012] The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally,

program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network or other data transmission medium. In a distributed computing environment, program modules and other data may be located in both local and remote computer storage media including memory storage devices.

[0013] With reference to Fig. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, Peripheral Component Interconnect (PCI) bus (also known as Mezzanine bus), Peripheral Component Interconnect Express (PCI-Express), and Systems Management Bus (SMBus).

[0014] Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and non-volatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by computer 110. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal

that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[0015] The system memory 130 includes computer storage media in the form of volatile and/or non-volatile memory such as ROM 131 and RAM 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, Fig. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

[0016] The computer 110 may also include other removable/non-removable, volatile/non-volatile computer storage media. By way of example only, Fig. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, non-volatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, non-volatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, non-volatile optical disk 156, such as a CD-ROM or other optical media. Other removable/non-removable, volatile/non-volatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

[0017] The drives and their associated computer storage media, discussed above and illustrated in Fig. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In Fig. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a

minimum, they are different copies. A user may enter commands and information into the computer 110 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195.

[0018] The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in Fig. 1. The logical connections depicted include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[0019] When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, Fig. 1 illustrates remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0020] **Exemplary Embodiments**

[0021] The present invention is directed to a mechanism for redirecting a code execution path in a running process in memory that does not lead to unexpected behavior by

advantageously using a one-byte interrupt instruction (e.g., INT 3) that is inserted into the code path. The interrupt instruction passes control to a kernel handler, which after executing a replacement function, returns to continue executing the process.

[0022] With reference to Fig. 2, there is illustrated an exemplary process 200 running in memory. In accordance with the present invention, an execution path 201 (e.g., the beginning of a vulnerable function to be replaced) of the process 200 running in memory may be changed by overwriting an existing instruction 202 with a one byte interrupt instruction (e.g., INT 3), where the remainder of the original code 203 remains unaltered. INT 3 is typically used as a trap to a debugger to break out of execution in order for other code to be executed.

[0023] The interrupt instruction will cause a kernel handler 204 to be called. The kernel handler 204 for that interrupt includes a mechanism that will cause a return from the interrupt to continue into a new instruction 206 (e.g., the replacement function) instead of returning to the original function. The new instruction 206 is placed in a memory space known to the kernel handler 204. Because the interrupt instruction is a one byte instruction, the present invention advantageously provides a mechanism for code diversion that does not overwrite more than one byte in the original code 203.

[0024] While the present invention has been described in connection with the preferred embodiments of the various Figs., it is to be understood that other similar embodiments may be used or modifications and additions may be made to the described embodiment for performing the same function of the present invention without deviating therefrom. For example, one skilled in the art will recognize that the present invention as described in the present application may apply to any computing device or environment, whether wired or wireless, and may be applied to any number of such computing devices connected via a communications network, and interacting across the network. Furthermore, it should be emphasized that a variety of computer platforms, including handheld device operating systems and other application specific operating systems are contemplated, especially as the number of wireless networked devices continues to proliferate. Still further, the present invention may be implemented in or across a plurality of processing chips or devices, and storage may similarly be effected across a plurality of devices. Therefore, the present invention should not be limited to any single embodiment, but rather should be construed in breadth and scope in accordance with the appended claims.

WHAT IS CLAIMED IS:

1. A method of redirecting a code execution path in a running process, comprising:

injecting an instruction into said code execution path;

passing control to a kernel handler;

executing a replacement function called by said kernel handler; and

returning to said code execution path.

2. The method of claim 1, wherein said instruction is an interrupt.

3. The method of claim 2, wherein said interrupt is an INT 3 interrupt instruction and wherein said interrupt instruction is one-byte in length.

4. The method of claim 2, wherein said kernel handler includes a mechanism to cause a return from said interrupt instruction to continue into said replacement function.

5. The method of claim 1, further comprising inserting said instruction such that no more than one byte of the original code in said code path is overwritten.

6. The method of claim 1, further comprising loading said replacement function into a memory space accessible by said kernel function.

7. The method of claim 1, further comprising performing said method without requiring a reboot of a computing device on which said running process is executing.

8. A method of changing a code execution path by using an interrupt to replace an existing function, comprising:

injecting said interrupt into said existing function;

passing control to a kernel handler;

executing a replacement function called by said kernel handler; and

returning to said code execution path.

9. The method of claim 8, wherein said interrupt is an INT 3 interrupt instruction and wherein said interrupt instruction is one-byte in length.

10. The method of claim 9, wherein said kernel handler includes a mechanism to cause a return from said interrupt instruction to continue into said replacement function.

11. The method of claim 8, further comprising inserting said interrupt such that no more than one byte in the original code in said code path is overwritten.

12. The method of claim 8, further comprising loading said replacement function into a memory space accessible by said kernel function.

13. The method of claim 8, further comprising performing said method without requiring a reboot of a computing device on which a process executing said existing function said is running.

14. A computer readable medium having computer executable instructions thereon for redirecting a code execution path in a running process, said computer executable instructions performing a method, comprising:

injecting an instruction into said code execution path

passing control to a kernel handler;

executing a replacement function called by said kernel handler; and

returning to said code path.

15. The computer readable medium of claim 14, wherein said instruction is an interrupt.

16. The computer readable medium of claim 15, wherein said interrupt is an INT 3 interrupt instruction and wherein said interrupt instruction is one-byte in length.

17. The computer readable medium of claim 15, wherein said kernel handler includes a mechanism to cause a return from said interrupt instruction to continue into said replacement function.

18. The computer readable medium of claim 14, further comprising instructions for inserting said instruction such that no more than one byte in the original code in said code path is overwritten.

19. The computer readable medium of claim 14, further comprising instructions for loading said replacement function into a memory space accessible by said kernel function.

20. The computer readable medium of claim 14, further comprising instructions for performing said method without requiring a reboot of a computing device on which said running process is executing.

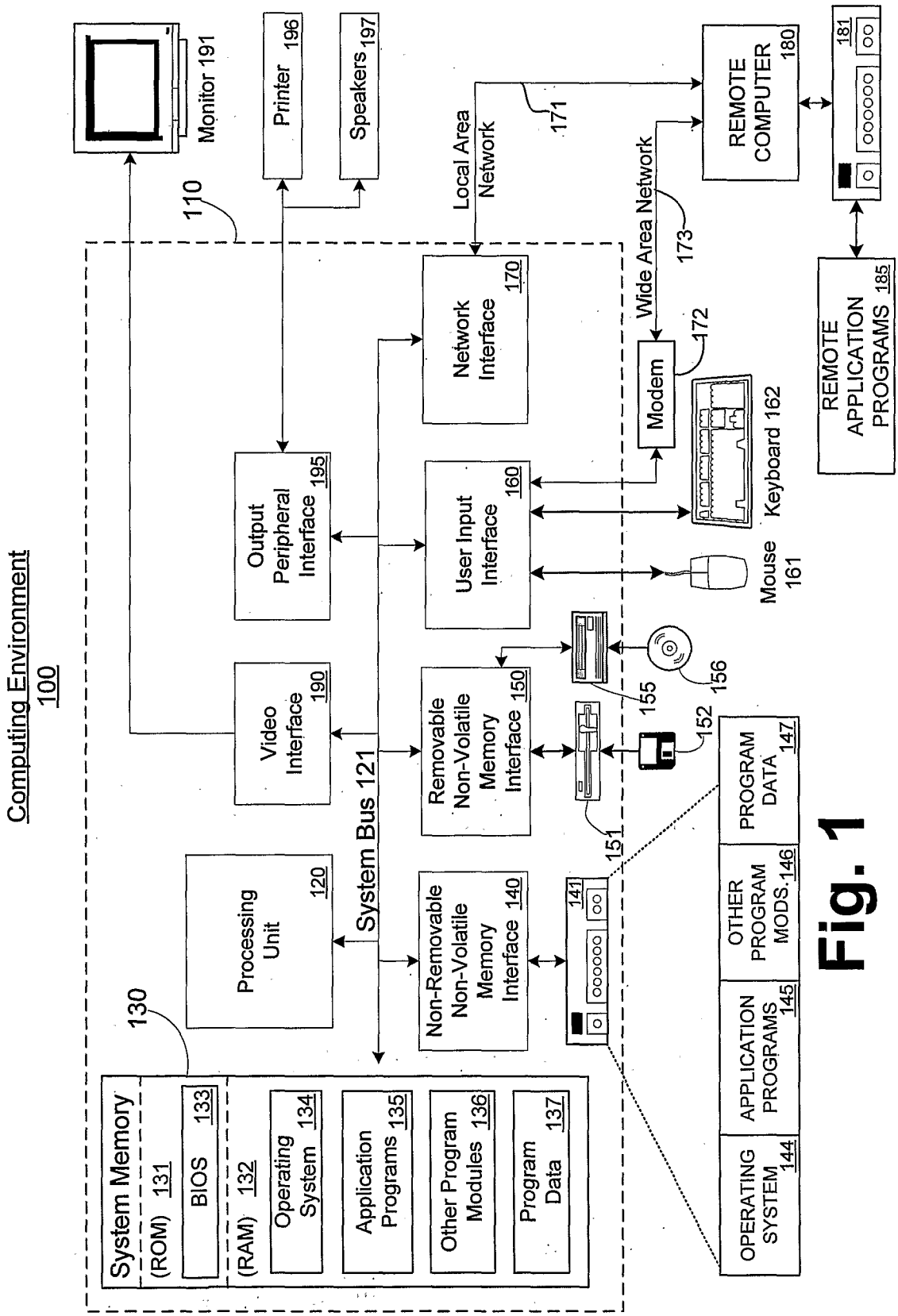
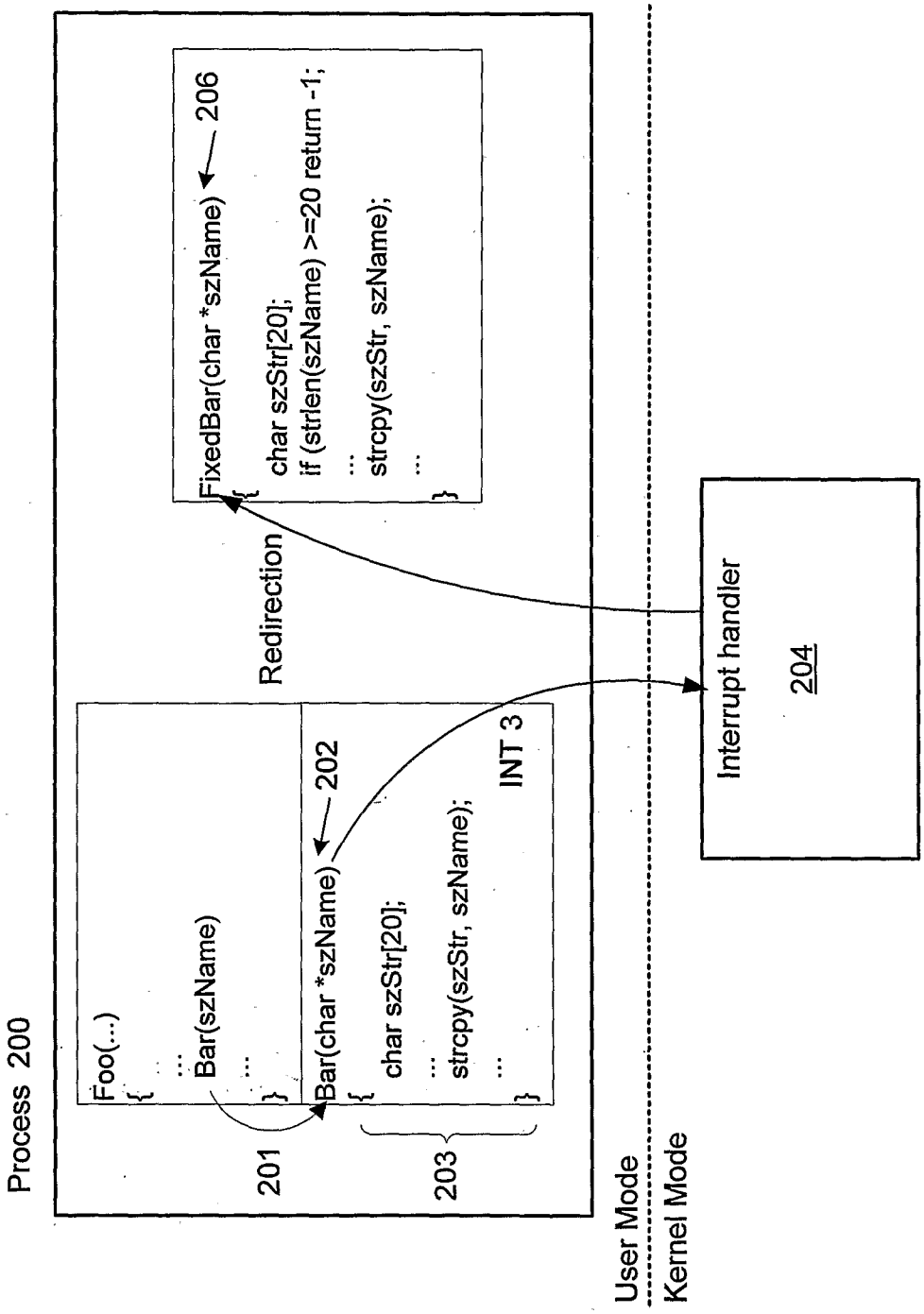

Computing Environment
100
110
Monitor 191
Printer 196
Speakers 197
Local Area Network
171
Wide Area Network
173
REMOTE COMPUTER
180
181
Network Interface 170
Modem
172
Output Peripheral Interface 195
User Input Interface 160
Keyboard 162
Mouse 161
REMOTE APPLICATION PROGRAMS 185
Video Interface 190
Removable Non-Volatile Memory Interface 150
155
156
152
151
Processing Unit 120
System Bus 121
Non-Removable Non-Volatile Memory Interface 140
141
130
System Memory
(ROM) 131
BIOS 133
(RAM) 132
Operating System 134
Application Programs 135
Other Program Modules 136
Program Data 137
OPERATING SYSTEM 144
APPLICATION PROGRAMS 145
OTHER PROGRAM MODS. 146
PROGRAM DATA 147

Fig. 1

Fig. 2


Process 200
Foo(...)
{
...
Bar(szName)
...
}
201
Bar(char *szName)
{
char szStr[20];
...
strcpy(szStr, szName);
...
}
202
203
INT 3
Redirection
FixedBar(char *szName)
{
char szStr[20];
if (strlen(szName) >=20 return -1;
...
strcpy(szStr, szName);
...
}
206
Interrupt handler
204
User Mode
Kernel Mode