PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/00, H04K 1/00, G06F 9/00, 15/00 — A1

(11) International Publication Number: **WO 96/41449**

(43) International Publication Date: 19 December 1996 (19.12.96)

(21) International Application Number: PCT/US96/09916

(22) International Filing Date: 7 June 1996 (07.06.96)

(30) Priority Data:
08/488,195 7 June 1995 (07.06.95) US

(71) Applicant: DIGITAL RIVER, INC. [US/US]; 5198 West 76th Street, Edina, MN 55439 (US).

(72) Inventor: RONNING, Joel, A.; 4212 Alden Drive, Edina, MN 55416 (US).

(74) Agent: BRUESS, Steven, C.; Merchant, Gould, Smith, Edell, Welter & Schmidt, 3100 Norwest Center, 90 South Seventh Street, Minneapolis, MN 55402 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: TRY-BEFORE-YOU-BUY SOFTWARE DISTRIBUTION AND MARKETING SYSTEM

(57) Abstract

A computer-based system is provided for demonstrating software programs to a potential purchaser and for gathering marketing information related to the demonstration of the programs. The system enables the software programs for execution upon selection by a user, and allows the user to subsequently operate or sample the selected software program. The system maintains the selected software program in a locked state to prevent unauthorized duplication of the selected software program, and selectively disables the sampling, such as when the user completes the sampling or if the system detects that the user is attempting to copy the sampled application. A code is generated that identifies one or more particular software programs and contains information relating to sampling of the particular software programs by the user, such as which applications were sampled and how many times they were sampled.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

TRY BEFORE YOU BUY SOFTWARE DISTRIBUTION AND MARKETING SYSTEM

FIELD OF THE INVENTION

The present invention relates to a system and method for gathering data related to usage of software programs sampled by a potential purchaser or other user of the programs.

BACKGROUND OF THE INVENTION

Customers or potential purchasers of software programs often desire to test the programs before determining whether or not to purchase them. This may occur because written literature does not adequately provide the customer with a feel for the functionality of the software program when in operation. The written literature simply describes the program, and customers often want to actually work with the software program in order to determine if they want to buy the program.

Some software vendors provide demonstration programs of their software programs for sale. These demonstration programs typically are not the fully operating version of the software program. The demonstration program usually contains only some of the functionality of the full software program in order to provide the customer with a feel for the functionality of the program. However, since the demonstration version normally is not a fully operating version of the program, it does not provide a customer with a complete picture of the program's functionality. Therefore, demonstration versions are limited in how they may assist a customer in deciding whether to purchase a particular software program.

Recently, some software vendors are providing fully operating versions of software programs which a customer may sample. These software programs are intended to be securely stored on a particular storage medium. Therefore, when a customer samples a fully

operating version of a program, the program being sampled is securely maintained by another program which seeks to prevent the customer from obtaining a free copy of the sampled program.

These software vendors typically distribute the programs to be sampled on some type of transportable storage medium. For example, a software vendor may provide a floppy diskette on which is contained the program to be sampled. The diskette may be locked through changes to the physical storage medium such that the customer may not obtain a free copy of the program. Other software vendors provide a CD-ROM which stores a large number of programs which may be sampled. Such a CD-ROM typically contains a program which manages and in effect supervises the sampling of the programs such that a customer may not obtain copies of the programs without first purchasing them and obtaining particular unlocking codes.

These distribution systems for sampling fully operating versions of programs, however, are generally limited in that they do not provide for on-line distribution. These systems typically distribute the programs on some type of transportable storage medium, such as a CD-ROM or hard disk drive sold with a computer. This type of distribution can be more burdensome to the computer user, because an on-line distribution channel is more easily accessible, since the computer users need simply "dial up" a network or bulletin board.

These systems for providing the sampling of fully operating versions of programs are also limited in the security provided to the programs being sampled. For example, they typically prevent unauthorized copying of programs by dividing the program or removing portions of the program such that a secure "key" is required to reassemble the program and thus operate it. These methods, however, require that one tamper with the program, which can make the program unstable. In

addition, these methods typically do not work with all applications. For example, some applications use a checksum program to prevent damage from computer viruses. These distribution methods change the format of the program, which can result in the checksum virus check incorrectly returning an error or message that the program is corrupted.

SUMMARY OF THE INVENTION

One aspect of the is a system and method for demonstrating software programs to a potential purchaser of the programs and for gathering marketing information related to the demonstration of the programs. The system receives a plurality of software programs to be demonstrated and maintains each of the software programs in a locked state in order to prevent unauthorized duplication of the software programs.

Any of the software programs are enabled for execution upon selection by a user, and the user is allowed to subsequently operate or sample the selected software program. The system maintains the selected software program in the locked state during the sampling in order to prevent unauthorized duplication of the selected software program. The sampling of the selected software program is selectively disabled, such as when the user completes the sampling or if the system detects that the user is attempting to copy the sampled application.

In addition, a code is generated that identifies a particular software program in the plurality of software programs and contains information relating to sampling of the particular software program by the user, such as which applications were sampled and how many times they were sampled.

Another aspect of the invention is an on-line system and method for demonstrating software programs to a potential purchaser of the programs. The system receives from an on-line system a software program to be

demonstrated, and maintains the software program in a locked state in order to prevent unauthorized duplication of the software program.

The software program is enabled for execution upon selection by a user, and the user is allowed to subsequently operate or sample the software program. The system maintains the software program in the locked state during the sampling in order to prevent unauthorized duplication of the software program. The sampling of the software program is selectively disabled, such as when the user completes the sampling or if the system detects that the user is attempting to copy the sampled application.

A further aspect of the invention is a self-launching system associated with a software program, and method for implementing a self-launching system, for demonstrating the software program to a potential purchaser of the program. The self-launching system is attached to a software program. The system maintains the software program in a locked state in order to prevent unauthorized duplication of the software program.

The self-launching system also includes the ability to launch itself when a user selects the software program. Upon launching itself, the system enables the software program for execution upon selection by the user, and allows the user to subsequently sample the software program. The system maintains the software program in the locked state during the sampling in order to prevent unauthorized duplication of the software program. The sampling of the software program is selectively disabled, such as when the user completes the sampling or if the system detects that the user is attempting to copy the sampled application.

An additional aspect of the invention is a self-launching system associated with a software program or other digital information, and method for

implementing such a system, for distributing the software program or other digital information to a potential purchaser of the program. The self-launching system is attached to a software program or other digital information. The system maintains the software program or other digital information in a locked state in order to prevent unauthorized copying of the software program or other digital information.

The self-launching system also includes the ability to launch itself when a user selects the software program or other digital information. Upon launching itself, the system unlocks the software program or other digital information in response to a request to purchase the software program or other digital information.

Still another aspect of the invention is a system and method for storing a code within an operating system of a computer in order to identify whether the computer has executed a particular software program. The system receives an indication that the computer has executed the software program, and searches a non-volatile memory in which the operating system for the computer is stored in order to locate spare memory locations. The system writes a code to at least one of the spare memory locations. The code provides an indication that the computer has executed the software program. The system associates the code with the software program in order to provide the indication that the computer has previously executed the program.

An even further aspect of the invention is a system and method for preventing unauthorized duplication of a particular software program among a plurality of active software programs executed on a computer. The system receives an indication that the computer is executing the particular software program, and then monitors operation of the computer to determine which of the plurality of the active software programs is being currently executed. When the system determines

through the monitoring that the particular software program is not the currently executed software program, it disables execution of the particular software program.

An additional aspect of the invention is a computer-based system for automatic sales of software programs. The system accesses a software program within a computer database and maintains the software program in a locked state in order to prevent unauthorized duplication of the software program. In response to a request to purchase the software program, the system unlocks a copy of the software program and distributes the unlocked copy. The system records how many copies of the software program have been distributed in response to the purchase requests.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of the conceptual operation of a software or digital information distribution system.

FIG. 2 is a block diagram of hardware components for implementing a software or digital information distribution system.

FIG. 3 is a user interface for a software or digital information distribution system illustrating examples of programs available to be sampled and/or purchased.

FIG. 4A is a diagram of software program files used by a software or digital information distribution system.

FIGS. 4B and 4C are diagrams of a package file system for use in maintaining software programs or digital information in a locked state.

FIG. 5 is a diagram of background processes used by a software or digital information distribution system.

FIG. 6 is a flow chart of a preferred active process detection routine.

FIG. 7 is a flow chart of a preferred background processing setup routine.

FIG. 8 is a flow chart of a preferred check status of sample count files routine.

FIG. 9 is a flow chart of a preferred "do open" routine.

FIG. 10 is a flow chart of a preferred "do prime" routine.

FIG. 11 is a flow chart of a preferred image driver routine.

FIG. 12 is a flow chart of a preferred install active process detection code routine.

FIG. 13 is a flow chart of a preferred install image driver routine.

FIG. 14 is a flow chart of a preferred open driver routine.

FIG. 15 is a flow chart of a preferred sample count manager routine.

FIG. 16A is a flow chart of a preferred sector encryption/decryption routine.

FIG. 16B is a flow chart of a preferred varying positional key encryption/decryption routine used with the routine of FIG. 16A.

FIG. 17 is a flow chart of a preferred setup sample counter routine.

FIG. 18 is a flow chart of a preferred watchdog task routine.

FIG. 19 is a flow chart of a preferred purchase routine for allowing users to purchase sampled software programs.

FIG. 20 is a flow chart of a preferred process for a self-launching and on-line self-launching software or digital information distribution system.

FIG. 21 is a flow chart of a preferred process for generating a serial number which contains identification of marketing information for sampled software programs.

FIG. 22 is a flow chart of a preferred process for automatic sale of software programs.

DETAILED DESCRIPTION

FIG. 1 is a conceptual diagram of the operation of a software or digital information distribution system. A user or customer 10 may sample a software program 12, which typically involves working with a fully operating version of the software program 12. The system preferably maintains a secure interface or protective envelope 14 around the software program 12 such that the user 10 may not obtain a copy of the software program without first purchasing it. The system preferably maintains the sampled program in a locked state during the sampling of the program in order to prevent unauthorized duplication of the program, for example, during the sampling. The system selectively disables the sampling of the selected software program, such as when the user completes the sampling or tampers with the program during the sampling. The use of a secure interface or protective envelope 14 provides the advantage of not tampering with the program for security purposes. This type of scheme for maintaining a program in a locked state is described below. Other methods for maintaining software programs in a locked state to prevent unauthorized copying of the program are possible, such as is disclosed in U.S. Patent Nos. 4,658,093; 4,740,890; 4,999,806; and 5,341,429.

If the user 10 purchases the software program 12, then the system copies an unlocked copy of the software program 16 to a storage medium where the user may access and continue to use the program. "Software program," "program," and "application," are used interchangeably herein.

The present invention can also distribute digital information (12), as indicated in FIG. 1. The present invention views the distributed entity as "bits," whether it is a software program which includes

functionality for controlling the operation of a computer, or simply other types of information in digital form. Therefore, the principles of the present invention used for distribution of information apply to digital information generally, software programs being a type of digital information. Digital information includes information capable of being represented in digital form. Digital information thus includes software programs and also includes intellectual property such as data representing creative or artist expression. Other examples of digital information include: textual works such as books and articles; music; video; music video; fonts; graphics; and clip art.

In distributing digital information, a user may have the option to sample the information. For example, when distributing locked movies in digital form, a user may be allowed to view the first few minutes of the movie. For distributing music, a user may be allowed to sample a representative portion of the music. If the digital information is a software program, the user may be allowed to sample the program, as mentioned above.

The digital information may also be distributed without sample capability. For example, a publisher may distribute a CD-ROM having several locked "white papers" or technical works. The system can, for example, only display titles or abstracts to a user for purposes of allowing the user to decide which if any textual works to purchase. When a user requests to purchase one or more of the papers, the system unlocks the paper and distributes it to the user. This distribution scheme thus provides the advantage of allowing one to purchase individual textual works in a collection without having purchase the entire compilation.

The system, therefore, maintains the digital information in a locked state by using the secure

interface or protective envelope 14. When a user requests to purchase the information and provides any required information such as a credit card number, the system distributes an unlocked copy (16) of the digital information to the user.

FIG. 2 is a diagram of typical hardware components for implementing a software or digital information distribution system. The system includes a computer 18 which is a typical digital computer such as a personal computer. The computer 18 includes a microprocessor 20 for executing software programs. The microprocessor 20 is interfaced with a read only memory 22 and random access memory 24. The computer 18 is interfaced with an input device 26 for entering commands or information into the computer 18. The input device 26 may be implemented with, for example, a keyboard, touch screen, light pen, "mouse" or other cursor control device such as a trackball, or other device for entering information or commands into the computer 18. The computer 18 is typically interfaced with a printer 28, which may be implemented with a typical computer printer for generating a hard copy of the information stored within the computer 18. The computer 18 is also typically interfaced with a display device 30, which may be implemented with a typical color or monochrome computer monitor. A preferred hardware platform for implementing the present invention is a Macintosh computer, developed and sold by Apple Computer, Inc.

The computer 18 is also preferably interfaced to devices from which it may receive software programs to be sampled optionally along with a software program for managing the sampling of additional software programs. These external devices may include a digital audiotape drive 32, disk drive 34, hard disk drive 36, CD-ROM drive 38, or an on-line system 42 interfaced through a modem 40. The on-line system 42 may be interfaced to existing networks or on-line services such as, for example, the Internet, America OnLine®, and

Prodigy®. Computer users may download information from networks such as the Internet.

FIG. 3 is a user interface of a software or digital information distribution system operating within the computer 18 by the microprocessor 20. User interface 50 displays in window 44 a plurality of programs which a user may sample and/or purchase. Alternatively, window 44 can display an identification of digital information available for purchase. In order to sample a particular program, a user would select one of the plurality of programs in window 44 and then select icon 46. In addition, the system allows a user to select other information identified in window 52, and that information is displayed in window 51, such as: program information; system requirements; information about "getting started" with a particular application; publisher information; licensing agreement; and packaging photos.

If the user desires to purchase one of the programs, a user would select the purchase icon 48 and optionally the install icon 47 in order to have the program automatically installed. When a user samples a particular program, the system preferably displays the number of samples remaining at location 49 and optionally additional information such as the version of the sampled program. As explained below, the system can thus control how many samples are available to the user. FIG. 3 is one example of user interfaces for a software or digital information distribution system. Other user interfaces or ways of allowing a user to interact with the system are possible for such a system.

FIG. 4A is a diagram of files used by a software or digital information distribution system. An example of a software or digital information distribution system is also referred to as a "distribution application" in the present specification. The system typically uses an image driver 56 and invisible files 60. FIG. 4A also illustrates how a

software or digital information distribution system may be implemented in a separate "stand-alone" version and a self-launching version. In one embodiment, the system 58 is a separate software program which interfaces an encrypted package 62 containing a usage file 64 and a program or digital information 66 to be distributed and/or sampled. The usage file 64 typically contains a separate redundant copy of how the program or digital information is to be used; for example, how many samples are available. This is in addition to the invisible files, which store encrypted copies of how many samples are available, as explained below. Accordingly, the usage file 64 provides another level of protection in the event that a user tampers with or otherwise alters the sample count information in the invisible files. The software program or digital information is preferably encrypted, as explained below, in order to prevent a user from obtaining a "free" copy of the information.

In another embodiment, a self-launching system implementing a software or digital information distribution system 72 is attached to a usage file 70 and program or digital information 74 to be distributed and/or sampled. In this embodiment, distribution application is attached to each program or information to be distributed and/or sampled and its operation is thus essentially invisible to the user. Therefore, instead of requiring a user to install a software or digital information distribution system and then sample other software programs, a user may simply download each program to be sampled and/or purchased. Each of those packages contains a distribution application for controlling the sampling or distribution of software programs or digital information along with such programs or information.

The distribution application 72 is typically attached to the encrypted package by joining the distribution application stored in a resource fork and

the encrypted package stored in the data fork into one file 71. The operation of the resource fork for storing structured data and the data fork for storing random access data in a Macintosh computer system is well known. Therefore, the file 71 includes the distribution application 72 "wrapped around" the encrypted package 68 which includes an encrypted application or digital information 74 along with the encrypted usage file 70. The distribution application 72 also contains the invisible files. The package 68 is typically encrypted in the same manner as package 62.

As mentioned above, applications used by a software or digital information distribution system are maintained in a locked state. This locked state is used to prevent unauthorized copying of the program both while it is stored and when a user samples it. The applications to be sampled are typically in a form only usable by a software or digital information distribution system in order to ensure security. Encryption as part of a locked state is typically accomplished by exclusive-ORing ("XORing") each byte of the application with a positional variant, as is explained with reference to FIGS. 16A and 16B. In addition, a package file system is also typically used as a part of a locked state to ensure security of the applications in a locked state.

This package file system is illustrated in FIGS. 4B and 4C. As shown in FIG. 4B, a package file in a Macintosh computer system works much like a random access memory (RAM) disk except that it is in non-volatile memory. An image file 77 which is the desired size of a "virtual volume" created by a software or digital information distribution system is allocated on a hard drive 75 or other non-volatile storage medium. Locked applications or digital information are stored within this partition. In a self-launching distribution application, the distribution application and image file

appear as one file 67. A distribution application file 73 is also stored on the storage medium 75.

As shown in FIG. 4C, the image file 77, or in the self-launching case the distribution application/image file 67, is internally given the same structure as a floppy disk. A distribution application 69 then informs the operating system of the corresponding computer that a "floppy disk" 79 is actually mounted. The operating system of a computer system 81 sends read/write requests 85 to the virtual volume 79. Instead of writing to a physical media, the driver of the computer's operating system writes to the virtual volume 79. The image driver 83 of the distribution application 69 performs reading and writing to the image files 77 and 67. This technique is also known as a "soft partition," because the hard drive on which the image file is located has been effectively partitioned via software. Accordingly, a distribution application 69 can reserve this virtual volume in order to control reading and writing to this partition (the physical media) and thus prevent unauthorized copying of applications or digital information in this partition.

FIG. 5 is a diagram of background processes used by a software or digital information distribution system implemented within the computer 18 and executed by the microprocessor 20. The background processes preferably include an active process detection 76, image driver 78, sample count manager 80, sector encryption/decryption 82, and watchdog task 84, all of which are described below.

FIG. 6 is a flow chart of an active process detection routine. On an exemplary embodiment on a Macintosh computer, this routine is implemented by a patch to the system trap called SystemTask. This trap is called consistently by all applications many times a second. What makes this work is that the Finder intercepts calls to SystemTask, and only passes on the call of the application that is the "active" or current

"front-most" process. Therefore, when a patch to SystemTask is called, the system assumes that the front application called it. The system then checks the low level variable AppName to retrieve the name of the current front-most application.

In a windows or multi-tasking environment, several applications can be represented by windows which may overlap. A currently-active application is typically represented in a window which appears in front of the other windows, or is otherwise highlighted or altered to indicate that it is the currently-active program if, for example, the displayed windows are not overlapping.

The routine begins when the system task was called (86). The system retrieves the front-most application name (88) and determines if the sample program is in front (90). The sample program is the software program being sampled by the user. The system determines if this is the top-most or front-most program displayed on a user's monitor. If it is the application in front, the system determines if sample time limit has expired (92); otherwise, the system determines if a distribution application is in front (94). A time limit provides for additional security by limiting how long a user may sample a particular application. If the time limit has expired, the system sets a flag to enable the image driver (100) and calls the original system task (108). Otherwise, if the sample time limit has not expired, the system terminates the sample application (98).

If as determined at step 94 the sample application is in front, the system sets a flag to enable the image driver (96); otherwise, a flag is set to disable the image driver (102). Next, the system determines if the sample application file is still open (104). If it is not, the system closes and unmounts the virtual volume (106) and calls the original SystemTask (108).

The routine of monitoring which application is "in front" thus provides a security measure by preventing the unauthorized copying of an application while it is being sampled. This system for preventing unauthorized duplication of a particular software program among a plurality of active software programs executed on a computer typically includes the following features. It receives an indication that the computer is executing the particular software program, and monitors operation of the computer to determine which of the plurality of the active software programs is being currently executed. Execution of the particular software program is disabled when the monitoring determines that the particular software program is not the currently-active or top-most software program.

FIG. 7 is a flow chart of a background processing setup routine. In this routine, the system decrypts and installs a background processing code (110); initializes the global variables (112); installs a watchdog task (114) (see FIG. 18); installs active process detection code (116) (see FIG. 12); sets up a sample counter (118) (see FIG. 17); sets up interprocess communication vectors (120); and installs an image driver (122) (see FIG. 13).

FIG. 8 is a flow chart of a check status of sample count files routine. The system preferably uses sample count files stored within the computer 18 as invisible files. These sample count files are preferably each identical and maintain the sample count. Each file typically contains an identification of each application and the number of allowed samples for each of the applications. If the user attempts to tamper with a particular sample count file in order to obtain more samples, the system detects that tampering by comparing the tampered file with the other sample count files. In addition, the invisible files are preferably encrypted for additional security, such as using an XOR

operation with a key and a bit shift of the sample counts.

In this routine, the system determines at steps 124, 128, and 132 if the sample count files one, two, and three, respectively, have been created. If the sample count files have not been created, the system creates the corresponding sample count files at steps 126, 130 and 134. While the system uses three sample count files, more or fewer files may be used.

FIG. 9 is a flow chart of a "do open" routine. The system first retrieves the address of "OK to Read" flag from a background code via interprocess communication (136). Next, the system retrieves the address of a sector decryption routine from the background code via interprocess communication (138) and sets standard open flags (140).

FIG. 10 is a flow chart of a "do prime" routine. The system first determines if this is read call (142). This test makes the driver read only. If this is not a read call, the system returns an error (154). Otherwise, the system proceeds with the routine and retrieves a value of a flag in the background process (144). It is next determined if the flag value has been updated within the last two seconds (146), for example. A flag is stored in the background process that is constantly updated by the watchdog task. If the flag has not been updated in the last two seconds, for example, it means that either a valid application is not "in front" (the sample application or distribution application) as explained above, or that a user is using a debugger to examine the code which implements the system. If the flag value has been updated in the last two seconds, the system reads requested sectors from "virtual" volume (150) and returns no error (152).

The system decrypts the sectors while reading them. The encryption/decryption of sector is explained with reference to FIGS. 16A and 16B. If the sectors of the application are compressed, the system also

decompresses the sectors while reading them. An example of an asymmetrical compression/decompression algorithm, which produces a relatively short decompression time in comparison to compression time, is Apple Computer, Inc.'s Cinepak compression scheme at a lossless level. If the flag value has not been updated, the system determines if this is a directory or volume information block (148). If it is, the system executes step 150 and returns no error (152). Otherwise, the system returns an error (154).

FIG. 11 is a flow chart of an image driver routine. The image driver preferably includes the following functions: open call (156); prime call (160); control call (164); status call (168); and close call (172). If any of these functions have been called, then the system performs the corresponding routine: do open (158); do prime (162); do control (166); do status (170); do close (174). The do prime call (162) is illustrated in FIG. 10. The other calls (158, 166, 170, 174) are well known in a Macintosh computer system.

FIG. 12 is a flow chart of an install active process detection code routine. This routine involves performing a patch system task trap (176), which is illustrated in FIG. 6.

FIG. 13 is a flow chart of an install image driver routine. In this routine, the system decrypts a driver code from a disk into memory within the computer 18 (178). The image driver is encrypted for security purposes so that a user cannot view the driver. The encryption/decryption of the driver is typically accomplished using the technique explained with reference to FIGS. 16A and 16B. The system locks the driver code into memory (180). The system then opens the driver (182).

FIG. 14 is a flow chart of an open driver routine. In this routine, the system sets up an interprocess communication vectors (184), which instructs the system where to locate global data.

FIG. 15 is a flow chart of a sample count manager routine. This routine is executed when the user requests a sample (186). The system checks to determine if samples are available (188) by checking the sample count files within the database. If samples are available, the system mounts the virtual volume (192). If the use is on-line, then the system downloads the software packages containing encrypted programs to be sampled and usage file and mounts the virtual volume (190). The packages are typically compressed for transmission using, for example, Apple Computer, Inc.'s Cinepak compression scheme at a lossless level, and are transmitted using TC/IP protocol.

The system then determines if the loaded image matches the database image (196) for security purposes. If the image does not match, the database data is rectified to that of the image (198) and the virtual volume is closed and unmounted (194) in order to maintain the application in a locked state. Otherwise, the system checks the redundant sample count on the virtual volume to determine if samples are available and in particular if a sample count files have been tampered with (200). If no samples are available, the virtual volume is closed and unmounted (220). Otherwise, the system decrements the sample count and launches the application (204) so that the user may operate the application to be sampled.

FIG. 16A is a flow chart of a sector encryption/decryption routine. This routine performs encryption of the distributed digital information for security purposes in order to prevent unauthorized duplication of the information. In this routine, the system determines if encryption or decryption is required (206). The system then performs the appropriate decryption (208) or encryption (210) function. FIG. 16B is a flow chart of a preferred varying positional key encryption/decryption routine used with the routine of FIG. 16A. The routine in FIG.

16B performs the actual encryption/decryption of data and is an example of how to encrypt/decrypt the encrypted packages 62 and 68 (see FIG. 4A) which contain the distributed software programs or digital information and usage files. Other encryption schemes are possible. The significance of the encryption scheme is in providing protection of the distributed information so that one may not obtain an unauthorized copy of the information without considerable time, effort, and processing capability.

The encryption/decryption routine of FIG. 16B uses a varying key based on byte position, also referred to as a positional variant. The system determines if sector encryption/decryption is required (201). If it is, the system decrypts the first 512 byte block (203) and then executes loops, as determined by steps 205, 207, 213, and 215, in order to encrypt/decrypt each byte in a series a 512 byte blocks. The encryption/ decryption of each byte involves first at step 209 a permutation to determine a key with the key = log(position MODULO 512) X $23FEC392, and then at step 211 applying the key to the byte with an XOR operation.

FIG. 17 is a flow chart of a setup sample counter routine. The system checks the boot block flags (212) and checks status of sample count files (214). This involves writing predetermined codes to spare boot blocks of the computer in order to mark the database to identify the execution of a software or digital information distribution system. Accordingly, these codes written to the boot blocks provide an indication that a distribution application has been run before on this particular computer.

This system for storing a code within an operating system of a computer in order to identify whether the computer has executed a particular software program typically includes the following features. It receives an indication that the computer has executed the software program, and searches a non-volatile memory

in which the operating system for the computer is stored in order to locate spare memory locations within the non-volatile memory. A code is written to at least one of the spare memory locations, and the code provides an indication that the computer has executed the software program. The code is associates with the software program to provide the indication.

In the routine shown in FIG. 17, the system determines if the sample counter system has been set up before on this particular computer (216). If it has been set up before, the system determines if all three sample count files are new (218) and if they are, it executes the following steps: fill new files with the value "one" at each location (200); write file creation dates to boot blocks (224); and write file check sums to boot blocks (226). When the sample count files are initialized the first time the system is executed, the value "one" is written to the files in order to signal that the files are secure and the system can, therefore, write sample count values to the files. Otherwise, the system fills new files with the value "minus one" at each location within the boot blocks (222). The "minus one" value indicates to the system that the corresponding sample count file has been tampered with or is otherwise corrupted or not secure. Accordingly, the system checks the other sample count files in order to verify there security. If all sample count files have a value of "minus one," this condition indicates that all sample count files have been corrupted and the user is not allowed no more samples. If less than all sample count files have a value of "minus one," then the system can reconstruct the corrupted files using a value in the secure or non-corrupted sample count file. The system also preferably verifies the sample count information in the invisible files against the information in the usage file.

FIG. 18 is a flow chart of a watchdog task routine. This is a VBL task, meaning it runs every time

there is a vertical blanking interrupt. For most monitors, that is approximately 72 times per second. The routine has three functions. It performs a check to determine if the sample application has timed out, and sets a flag accordingly. It tests to determine if it may allow the driver read, and updates a flag with the current time to be checked by the driver. It also encrypts and decrypts sample counts that are being written/to and read/from the sample count files. It performs this encryption in multiple phases, simulating an asynchronous process. What this does is make it very difficult for a user to determine where the encryption/decryption is being performed.

In the routine, the system checks for an application time out (228). It then determines if it is "OK" for the driver to read (230). If it is, it sets a flag to a current time (234). Otherwise, it sets a flag to zero (232). The system then determines if a sample count requires encryption (236). If it requires encryption, the system performs a phase of sample count encryption (238). Otherwise, the system determines if a sample count reads decryption (240), and if so, the system performs a phase of sample count decryption (242).

This phase encryption/decryption scheme is a subset of the sample count manage routine and works as follows. At various times, the distribution application needs to determine how many samples of a particular application remain. At these times, the distribution application reads an encrypted string from the invisible files. It then takes this information, transfers the information into global memory, signals to the watchdog task that a count needs decryption, and then places itself into a seemingly endless loop. The watchdog task, having been signaled to begin, decrypts the count in three phases to ensure that the full algorithm is never directly viewed by a user. On each pass, the watchdog task performs an XOR and bit rotation and then

increments the phase count. The next time the watchdog task executes, if there is still more work to be done, it executes another encryption phase. If not, it signals the application that encryption is now finished. The distribution application then exits the infinite loop and memory contains the correct sample count.

FIG. 19 is a flow chart of a purchase routine for allowing users to purchase sampled software programs. This routine is executed when a user requests to purchase a program or digital information (243). The system can unlock software programs or digital information in response to a request to purchase the software program or digital information. The system can optionally verify purchase information, such as a credit card number, before executing an unlocking process.

The system typically checks if the application has previously been purchased (244). If it has been purchased previously, the system checks to determine if this purchase request is an archive install (248). If it is an archive install, the system installs the application (254). For a regular purchase, the system generates or retrieves a serial number or key code (246) used for calculating a password to unlock the program. This serial number or key code is typically provided in a purchase dialog or window when the user selects the purchase key 48 (see FIG. 3). The purchase window also includes an area for a user to enter the password. The key code and corresponding password, as explained below, are preferably dynamically generated when the user opens the purchase window such that a new key code and password are generated each time the user opens the purchase window. In addition, the password preferably only exists in memory, and only as long as the user has a purchase dialog or window open. These features provide additional security by dynamically changing the key code and password.

If the use is on-line, the system preferably automatically registers the application with a vendor

and then unlocks the application or digital information (250) through the on-line connection with the distribution center.

Otherwise, the system can manually register the application with the vendor and provide a user with the password for unlocking the application or digital information (252). The manual registration typically occurs with the user calling up a distribution center and providing them with the serial number or key code as provided in a purchase dialog and possibly other information such as a credit card number. The distribution center in response provides the user with a password used for unlocking the application, and the user may then manually enter the password in the purchase window. Accordingly, steps 250 and 252 also involve generating the password from the serial number or key code. The serial number or key code provided by the user is processed using an identical decoding function, explained below, as on the user's machine, generating the same password that is stored in memory on the user's machine. The entered password is compared with the one stored in memory. If they match, the purchase is completed.

Accordingly, if the correct password is entered, either manually or automatically, the system proceeds to install the application or digital information (254). If the wrong password was entered, such as in the manual unlocking, then the system "returns" and does not unlock the application or digital information.

The installation typically occurs by decrypting a copy of the application or digital information and copying the unlocked application or digital information to a hard drive or other storage medium on the user's computer. The decryption is typically accomplished by copying the application or digital information via a pipeline from its current location in memory to a new (non-reserved) portion of

the user's hard disk drive or other storage medium which is outside of the partition reserved by a software or digital information distribution system. While the application or digital information is copied to the non-reserved portion, it is typically decrypted by using the technique explained with reference to FIGS. 16A and 16B. Accordingly, after this unlocking routine is complete, a locked copy of the application or digital information remains within the reserved partition and an unlocked copy of the application or digital information resides on the user's hard disk drive or some other storage medium. The unlocking and installation may occur simultaneously. In addition, while software programs typically require installation to run, other types of digital information may require only unlocking.

After the application is purchased, a user preferably has full use of the program or digital information. A vendor or distribution center may mail manuals or any other documentation for the purchased program to the user who purchased the program. Alternatively, the manuals and documentation may be distributed with the program and maintained in a locked state with the program. When a user purchases the program, the manuals and documentation may then also be unlocked so that the user can view them electronically or produce a hard copy using a computer printer.

FIG. 20 is a flow chart of a process for a self-launching and on-line self-launching software or digital information distribution system. A self-launching system has the advantage of not requiring a separate browser for distribution and/or sampling of applications or digital information. The operation of the system is thus essentially invisible to the user, since the system preferably "appears" to the user as an application or digital information and launches itself when a user selects the application or digital information.

According, a self-launching system for demonstrating applications typically includes the following features. The system is attached to a software program such as in one file as described above. The system maintains the software program in a locked state in order to prevent unauthorized duplication of the software program, such as with encryption and a package file system described above. When a user selects the software program, the system launches itself and can enable the software program for execution by the user and allow the user to subsequently sample the software program. The system maintains the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program, and selectively disables the sampling of the software program.

A self-launching system for distributing applications or digital information typically includes the following features. The system is attached to an application or digital information such as in one file as described above. The system maintains the application or digital information in a locked state in order to prevent unauthorized duplication, such as with encryption and a package file system described above. When a user selects the application or digital information, the system launches itself and can unlock the application or digital information in response to a purchase request.

Self-launching and on-line self-launching software or digital information distribution systems preferably use the processing described above in addition to the steps shown in FIG. 20. In order to sample an application, a user in a self-launching system typically selects an application (256). This may occur by, for example, "double clicking" on an icon displayed on a display device and corresponding to the application. In the self-launching system, therefore, the applications typically appear to the user as

executable programs even though they are locked and may only be sampled by the user or purchased upon providing the required information. After the user selects the application to sample, the code for the distribution application executes (258). If the use is on-line, the software package described above is first downloaded (257) before executing. The software package is typically one file (see FIG. 4A) which includes a distribution application and an encrypted package and is thus transmitted for on-line use as one file. The encryption is typically accomplished using the technique described with reference to FIGS. 16A and 16B. The encryption protects the software programs or digital information during transmission. This file is typically compression for transmission using, for example, Apple Computer, Inc.'s Cinepak compression scheme at a lossless level, and is transmitted using TC/IP protocol.

The system then, as described above, checks to determine if all samples have been used (260). If samples are remaining, the system code mounts a data fork of file as the virtual volume (262) and decrements the usage count and the files in the virtual volume (264). The system then proceeds with the processing described in the other flow charts provided in the present specification.

If the self-launching system involves distribution of digital information without allowing sampling, then the system typically checks to determine if the information has been purchased (259). If is has been purchased, then the system executes step 262 and bypasses step 264, since samples are not available. If the information has not been purchased, then the system typically checks to determine if the user wants to purchase the information (261). If the system receives a purchase request, then it executes a purchase routine 263 (see FIG. 19).

FIG. 21 is a flow chart of a process for generating a serial number which contains identification

of marketing information for software programs. The system retrieves the raw data from a sample count files (266), which includes an identification of how many times a user sampled each application. The system then calculates marketing statistics and formats such information into a series of bytes (268). This step may involve, for example, the following: determining the total number of samples used (270); determining how many times each application was sampled (272); determining a most frequently sampled application (274); determining a category of a most frequently sampled program (276); determining which version of particular applications a user has sampled (277); or other statistics to be determined, for example, by a distributor (278). Other statistics may include, for example, statistics related to time duration of the samples, such as an average time duration of sampling, which may be obtained using a computer's internal clock and timing each sample. Examples of categories of software programs include, but are not limited to: business software, games, financial management programs, and educational programs. A vendor or distributor may also create their own categories and electronically associate programs with those categories.

Gathering these statistics provides for many advantages and value in distributing programs to be sampled. For example, it allows vendors to identify programs which are not popular and replace them with programs which have a higher sales rate. It also allows vendors to identify the most popular programs and include more programs for sale in the same categories. Identifying a category of a user's most frequently sampled program also allows vendors to market additional similar products to that particular user and thus increase the likelihood that the user will purchase more software from the vendor. For example, when the user calls the vendor to purchase a program, the vendor can quickly identify the most frequently sampled program category by decoding the code (serial number) of the

purchased program. The vendor could then immediately offer any additional programs for sale which are in the same category and perhaps "on sale" or subject to a discount.

Given the information provided by the sample count files, the information for the marketing statistics can be determined with simple calculations. The sample count files contain an identification of each application and the number of samples remaining for the corresponding application. Therefore, by knowing how many samples were originally available, the system can determine the number of samples used for each application by samples remaining from the original number of allowed samples. The system can add up the number of samples used to determine a total number of samples used. Other statistics can be calculated in a similar manner using the information in the sample count files and possibly other information such as categories of sampled applications.

A distributor may determine that other statistics are desired and the system would then execute additional steps as part of step 268, as mentioned above. The system determines if more statistics are required (279). The system continues to execute steps for determining statistics until such processing is complete. As the system gathers and calculates the marketing statistics, it typically concatenates the resulting bytes, resulting in a series of bytes with byte representing a statistic. The system electronically associates each byte position with a statistic so that, by knowing a particular byte position, the system can decode the byte and produce the resulting statistic. The system then converts the series of bytes determine that step 268 into an ASCII serial number (280). Table 1 provides an example of a file for associating byte positions with statistics.

Table 1

byte position	statistic
1	total number of samples
2	number of application #1 samples
3	number of application #2 samples
4	most frequently sampled category
...	...
N	other statistic

Appendix A provides an example of a source code listing in C programming language for converting the series of bytes determined in the processing shown in FIG. 21, and optionally other information, into a key code and password for use in unlocking the application or digital information. The code shown in Appendix A generates both a key code and password using the marketing information (series of bytes described above) and the current date and time. The code shown in Appendix A can thus also be used to decode the key code and extract the series of bytes containing the marketing information. The key code is displayed to the user and in the purchase window and is what the user provides to a distribution center to obtain the password. This password is then used to unlock the application or digital information, as described above.

FIG. 22 is a flow chart of a process for automatic sale of software programs. One or more software programs, each in a locked state such as with the techniques described above, are stored in a computer database or available on-line. When one wants to purchase one or more of the programs, the system in response to the purchase request unlocks a copy of the program and maintains a record of how many copies were sold. An example of a use for this system is where a

company routinely purchases additional copies of software programs such as when new employees are hired.

Accordingly, this system for automatic sales of software programs typically includes the following features. It accesses a software program within a computer database and maintains the software program in a locked state in order to prevent unauthorized duplication of the software program. In response to a request to purchase the software program, the system unlocks a copy of the software program and distributes the unlocked copy. The system also records how many copies of the software program have been distributed in response to the requests to purchase the software program.

As shown in FIG. 22, the system typically displays an indication of locked applications available for purchase by a user (292). If the use is on-line, the system downloads encrypted software packages containing programs and usage file (293). The encryption is typically accomplished using the technique described with reference to FIGS. 16A and 16B. The packages are typically compressed for transmission using, for example, Apple Computer, Inc.'s Cinepak compression scheme at a lossless level, and are transmitted using TC/IP protocol. If a user requests to sample a particular application (294), the system executes sample routines (296). The sample routines may be, for example, the routines described in the present specification. If the system receives a request to purchase an application (298), it preferably performs the following steps. The application is registered with a vendor and unlocked (300). A serial number is then optionally assigned to the purchased application (302). The application is distributed and installed (304). Then the system updates a sales record and issues an invoice (306) in order to record the application purchased and how many copies have been purchased. The

step 300 may be implemented as described in the other routines in the present specification.

While the present invention has been described in connection with a preferred embodiment thereof, it will be understood that many modifications will be readily apparent to those skilled in the art, and this application is intended to cover any adaptations or variations thereof. It is manifestly intended that this invention be limited only by the claims and equivalents thereof.

APPENDIX A

```
void CDolphinApp::DoPurchase()
{
    CSampleApp *theApp;
    const short okItem = 1;
    const short cancelItem = 2;

    unsigned long temp1 = qd.randSeed;
    unsigned long temp3;

    GetDateTime(&temp3);
    temp3 = temp3 & 0x00000065;
    temp3 = temp3 >> 2;

    unsigned long temp0 = *((long*)0x0146);
    temp3 = temp3 & 0x2F;

    // ROL.L temp3,temp0
    long a = temp0 << temp3;
    long b = temp0 >> (32 - temp3);
    temp0 = a | b;

    GetDateTime(&temp3);
    temp1 = temp3 ^ temp1;

    a = temp3 << 7;
    b = temp3 >> (32 - 7);
    temp0 = a | b;
    temp3 = temp0 ^ temp3;

    gd.randSeed = temp0;
    temp3 = temp3 & 0x0000000E;

    long keyCode = 0;
    for (short r = 0; r < temp3; r++)
        keyCode = Random();

    CStr255 usageCode;

    {
        unsigned long installedDate = gUsageInfo.GetInstalledDate
        ("Dolphin Prefere
        installedDate -= 0xA81B3480;
        installedDate = installedDate / 0x00093A80;

        CStr255 instWkStr;
        CodeNumber(installedDate,2,instWkStr);

        long numOfProgsSampled = gUsageInfo.GetTotalAppsSampled ();
        CStr255 numProgsSampledStr;
        CodeNumber(numOfProgsSampled,2,numProgsSampledStr);

        TopSampleA top5;
        gUsageInfo.GetTop5SampledApps (top5);
```

```
            CStr255 num1AppStr;
            CStr255 num2AppStr;
            CodeNumber(top5[0],2,num1AppStr);
            CodeNumber(top5[1],2,num2AppStr);

            long totalSamples = gUsageInfo.GetTotalNumOfSamples();
            CStr255 totalSamplesStr;
            CodeNumber(totalSamples,2,totalSamplesStr);

            usageCode = instWkStr +
                  numProgsSampledStr +
                  num1AppStr +
                  num2AppStr +
                  totalSamplesStr;
            }

            CStr255 programNumStr;
            CodeNumber (fSelectedAppID, 3, programNumStr);
            CStr255 keyCodeStr;
            CodeNumber (keyCode, 4, keyCodeStr);

            CStr255 targetPassword = deecode (keyCodeStr);

            keyCodeStr = programNumStr + CStr255("-") + keyCodeStr;

            if ((itemHit == okItem) &&
            (IUEqualString(password,purchaseBackDoor) == 0))
               theApp->SetToPurchased();
               CRect upperLeftRect(0,0,150,60);
               InvalRect(upperLeftRect);
               InstallApp ();
            }
            if ((itemHit ==okItem) &&
            (IUEqualString(password,unPurchaseBackDoor) == 0
               theApp->SetToUnPurchased();
               CRect upperLeftRect(0,0,150,60);
               InvalRect(upperLeftRect);
            }

        } else {
            SysBeep(1);
            SysBeep(1);
        }
    }

    void CodeNumber(long number, short digits, CStr255& theString)
    {
        long temp2;

        theString[0] = digits;

        for (short index = digits; index > 0; index--) {
            temp2 = number;
            temp2 = temp2 & 0x0000001F;
            temp2 += 65;
            if (temp2 >= 'Z')
               temp2 -= 41;
```

```
            theString[index] = temp2;
            number = number >> 5;
        }
        if (number > 0)
            theString[1] = 42;
    }

    CStr255 deecode(const CStr255& input)
    }
        char output[7];
        char stg[2];
        int msg_number;
        int b1, B1, xx, len, retcode;
        long   code, bin;

        len = input.Length ();

        code = 0;

        for (xx = 1; xx < 5; xx++)
            {
               B1 = 0;
               sprintf(stg, "%d", input[xx] );
               b1=atoi( stg );
               if (b1 > 64 && b1 < 91)
                  B1 = b1 - 65;
               if (b1 > 49 && b1 < 56)
                  B1 = b1 - 24;

               switch (xx)
               {
                     case 1:
                                 code+= B1 * 32768;
                                 break;
                     case 2:
                                 code+= B1 * 1024;
                                 break;
                     case 3:
                                 code+= B1 * 32;
                                 break;
                     case 4:
                                 code+= B1;
                                 break;
                     default:
                                 break;
               }
            }

    code = (code ^ 43605) * 1523;

    for (xx = 0; xx < 6; xx++)
       {
             switch (xx)
             {
                   case 0:
                                 bin = 33554432;
                                 break;
```

```
                    case 1:
                                bin = 1048576;
                                break;
                    case 2:
                                bin = 32768;
                                break;
                    case 3:
                                bin - 1024;
                                break;
                    case 4:
                                bin = 32;
                                break;
                    case 5:
                                bin = 1;
                                break;
                    default:
                                break;
               {

               if (code >= bin)
                  {
                     B1 = code / bin;
                     code-=B1 * bin;
                     if (B1 + 65 <= 90)
                        {
                                b1 = B1 + 65;
                        }
                     else
                        {
                                b1 = B1 + 24;
                        }
               }
               else
               {
                     b1 = 65;
               }

               sprintf(&output[xx], "%c", b1);
               output[6] = '\0';
          }
       return(CStr255(output));

    }

    #pragma segment Main
    long intcode(const CStr255& input)
    {
          char  stg[2];
          int msg_number;
          long b1, B1, xx, len, retcode, code;

          len = input.Length();

          code = 0;

          for (xx = 0; xx < 4; xx++) {
             B1 = 0;
```

```
            sprintf(stg, "%d", input[xx] );
            b1=atoi( stg );
            if(b1 > 64 && b1 < 91)
               B1 = b1 - 65;
            if (b1 > 49 && b1 < 56)
               B1 = b1 - 24;

            switch (xx)
            {
               case 0:
                                 code+= B1 * 32768;
                                 break;
               case 1:
                                 code+= B1 * 1024;
                                 break;
               case 2:
                                 code+= B1 * 32;
                                 break;
               case 3:
                                 code += B1;
                                 break;
               default:
                                 break;
            }
         }
      return(code)) ;
   }
```

WHAT IS CLAIMED IS:

1. A system for demonstrating software programs to a potential purchaser of the programs and for gathering marketing information related to the demonstration of the programs, comprising:

receive means for receiving a plurality of software programs to be demonstrated;

means for maintaining each of the software programs in a locked state in order to prevent unauthorized duplication of the software programs;

sample means for enabling any of the software programs for execution upon selection by a user, for allowing the user to subsequently sample the selected software program, for maintaining the selected software program in the locked state during the sampling of the selected software program in order to prevent unauthorized duplication of the selected software program, and for selectively disabling the sampling of the selected software program; and

monitoring means for generating a code that identifies a particular software program in the plurality of software programs and contains information relating to sampling of the particular software program by the user.

2. The system of claim 1 wherein the monitoring means comprises means for including in the code information identifying which of the software programs the user sampled.

3. The system of claim 2 wherein the monitoring means comprises means for including in the code information identifying how many times the user sampled each of the software programs.

4. The system of claim 1 wherein the monitoring means comprises means for including in the code information

identifying a category of the software program most frequently sampled by the user.

5. The system of claim 1 wherein the sample means further comprises means for preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

6. The system of claim 1 wherein the sample means further comprises means for detecting if the software program is being copied during the sampling of the software program and for disabling the software program in response to the detecting.

7. The system of claim 1 wherein the sample means comprises means for limiting how many times the software program can be sampled and for displaying an indication of a number of samples remaining.

8. The system of claim 1 wherein the disable means comprises means for disabling the software program if the user has sampled the software program for a predetermined amount of time.

9. The system of claim 1 wherein:

the receive means comprises means for receiving the software program in an encrypted state; and

the sample means comprises means for decrypting the encrypted software program.

10. The system of claim 1 wherein the sample means comprises means for displaying an icon which identifies the software program.

11. The system of claim 1 wherein the receive means comprises means for receiving the software program from a non-volatile storage medium.

12. The system of claim 1, further comprising means for unlocking the software program in response to a request to purchase the software program.

13. A system for demonstrating software programs to a potential purchaser of the programs and for gathering marketing information related to the demonstration of the programs, comprising:

receive means for receiving a plurality of software programs to be demonstrated;

means for maintaining each of the software programs in a locked state in order to prevent unauthorized duplication of the software programs;

sample means for enabling any of the software programs for execution upon selection by a user, for allowing the user to subsequently sample the selected software program, for maintaining the selected software program in the locked state during the sampling of the selected software program in order to prevent unauthorized duplication of the selected software program, and for selectively disabling the sampling of the selected software program; and

monitoring means for generating a code for use in unlocking the software programs and for including in the code an identification of which of the software programs were sampled by the user and how many times each of the software programs were sampled by the user.

14. A computerized method for demonstrating software programs to a potential purchaser of the programs and for gathering marketing information related to the demonstration of the programs, comprising the steps executed by a computer of:

receiving a plurality of software programs to be demonstrated;

maintaining each of the software programs in a locked state in order to prevent unauthorized duplication of the software programs;

enabling any of the software programs for execution upon selection by a user, allowing the user to subsequently sample the selected software program, maintaining the selected software program in the locked state during the sampling of the selected software program in order to prevent unauthorized duplication of the selected software program, and selectively disabling the sampling of the selected software program; and

generating a code that identifies a particular software program in the plurality of software programs and contains information relating to sampling of the particular software program by the user.

15. The method of claim 14 wherein the monitoring step comprises the step of including in the code information identifying which of the software programs the user sampled.

16. The method of claim 15 wherein the monitoring step comprises the step of including in the code information identifying how many times the user sampled each of the software programs.

17. The method of claim 14 wherein the monitoring step comprises the step of including in the code information identifying a category of the software program most frequently sampled by the user.

18. The method of claim 14 wherein the enabling step further comprises the step executed by the computer of preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

19. The method of claim 14 wherein the disabling step further comprises the steps executed by the computer of detecting if the software program is being copied during

the sampling of the software program and disabling the software program in response to the detecting.

20. The method of claim 14 wherein the enabling step comprises the steps of limiting how many times the software program can be sampled and displaying an indication of a number of samples remaining.

21. The method of claim 14 wherein the disabling step comprises the step of disabling the software program if the user has sampled the software program for a predetermined amount of time.

22. The method of claim 14 wherein:

the receiving step comprises the step of receiving the software program in an encrypted state; and

the enabling step comprises the step of decrypting the encrypted software program.

23. The method of claim 14 wherein the enabling step comprises the step of displaying an icon which identifies the software program.

24. The method of claim 14 wherein the receiving step comprises the step of receiving the software program from a non-volatile storage medium.

25. The method of claim 14, further comprising the step of unlocking the software program in response to a request to purchase the software program.

26. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to demonstrate software programs to a potential purchaser of the programs and gather marketing information related to the demonstration of the

programs, the computer readable program code means in the computer program product comprising:

receive means for causing the computer to receive a plurality of software programs to be demonstrated;

means for causing the computer to maintain each of the software programs in a locked state in order to prevent unauthorized duplication of the software programs;

sample means for causing the computer to enable any of the software programs for execution upon selection by a user, allow the user to subsequently sample the selected software program, maintain the selected software program in the locked state during the sampling of the selected software program in order to prevent unauthorized duplication of the selected software program, and selectively disable the sampling of the selected software program; and

monitoring means for causing the computer to generate a code that identifies a particular software program in the plurality of software programs and contains information relating to sampling of the particular software program by the user.

27. An on-line system for demonstrating software programs to a potential purchaser of the programs, comprising:

receive means for receiving from an on-line system a software program to be demonstrated;

means for maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program; and

sample means for enabling the software program for execution upon selection by a user, for allowing the user to subsequently sample the software program, for maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software

program, and for selectively disabling the sampling of the software program.

28. The system of claim 27 wherein the sample means further comprises means for preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

29. The system of claim 27 wherein the sample means further comprises means for detecting if the software program is being copied during the sampling of the software program and for disabling the software program in response to the detecting.

30. The system of claim 27 wherein the sample means comprises means for limiting how many times the software program can be sampled and for displaying an indication of a number of samples remaining.

31. The system of claim 27 wherein the disable means comprises means for disabling the software program if the user has sampled the software program for a predetermined amount of time.

32. The system of claim 27 wherein:

the receive means comprises means for receiving the software program in an encrypted state; and

the sample means comprises means for decrypting the encrypted software program.

33. The system of claim 27, further comprising means for generating a code identifying the software program.

34. The system of claim 27 wherein the sample means comprises means for displaying an icon which identifies the software program.

35. A computerized on-line method for demonstrating software programs to a potential purchaser of the programs, comprising the steps executed by a computer of:

receiving from an on-line system a software program to be demonstrated;

maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program; and

enabling the software program for execution upon selection by a user, allowing the user to subsequently sample the software program, maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program, and selectively disabling the sampling of the software program.

36. The method of claim 35 wherein the enabling step further comprises the step executed by the computer of preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

37. The method of claim 35 wherein the disabling step further comprises the steps executed by the computer of detecting if the software program is being copied during the sampling of the software program and disabling the software program in response to the detecting.

38. The method of claim 35 wherein the enabling step comprises the steps of limiting how many times the software program can be sampled and displaying an indication of a number of samples remaining.

39. The method of claim 35 wherein the disabling step comprises the step of disabling the software program if the user has sampled the software program for a predetermined amount of time.

40. The method of claim 35 wherein:

the receiving step comprises means the step of receiving the software program in an encrypted state; and

the enabling step comprises the step of decrypting the encrypted software program.

41. The method of claim 35, further comprising the step executed by the computer of generating a code identifying software program.

42. The method of claim 35 wherein the enabling step comprises the step of displaying an icon which identifies the software program.

43. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to demonstrate on-line software programs to a potential purchaser of the programs, the computer readable program code means in the computer program product comprising:

receive means for causing the computer to receive from an on-line system a software program to be demonstrated;

means for causing the computer to maintain the software program in a locked state in order to prevent unauthorized duplication of the software program; and

sample means for causing the computer to enable the software program for execution upon selection by a user, allow the user to subsequently sample the software program, maintain the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program, and selectively disable the sampling of the software program.

44. A self-launching system associated with a software program for demonstrating the software program to a potential purchaser of the program, comprising:

means for attaching the self-launching system to a software program;

means for maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program; and

activation means for launching the self-launching system when a user selects the software program, the activation means comprising: sample means for enabling the software program for execution upon selection by the user, for allowing the user to subsequently sample the software program, for maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program, and for selectively disabling the sampling of the software program.

45. The system of claim 44 wherein the sample means further comprises means for preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

46. The system of claim 44 wherein the sample means further comprises means for detecting if the software program is being copied during the sampling of the software program and for disabling the software program in response to the detecting.

47. The system of claim 44 wherein the sample means comprises means for limiting how many times the software program can be sampled and for displaying an indication of a number of samples remaining.

48. The system of claim 44 wherein the disable means comprises means for disabling the software program if

the user has sampled the software program for a predetermined amount of time.

49. The system of claim 44, further comprising means for generating a code identifying the software program.

50. The system of claim 44 wherein the sample means comprises means for displaying an icon which identifies the software program.

51. The system of claim 44, further comprising means for receiving the software program and the attached self-launching system from an on-line system.

52. A computerized method using a self-launching system associated with a software program for demonstrating the software program to a potential purchaser of the program, comprising the steps executed by a computer of:

attaching the self-launching system to a software program;

using the self-launching system to maintain the software program in a locked state in order to prevent unauthorized duplication of the software program; and

launching the self-launching system when a user selects the software program, comprising the steps of:

enabling the software program for execution upon selection by the user;

allowing the user to subsequently sample the software program;

maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program; and

selectively disabling the sampling of the software program.

53. The method of claim 52 wherein the enabling step further comprises the step executed by the computer of

preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

54. The method of claim 52 wherein the disabling step further comprises the steps executed by the computer of detecting if the software program is being copied during the sampling of the software program and disabling the software program in response to the detecting.

55. The method of claim 52 wherein the enabling step comprises the steps of limiting how many times the software program can be sampled and displaying an indication of a number of samples remaining.

56. The method of claim 52 wherein the disabling step comprises the step of disabling the software program if the user has sampled the software program for a predetermined amount of time.

57. The method of claim 52, further comprising the step executed by the computer of generating a code identifying the software program.

58. The method of claim 52 wherein the enabling step comprises the step of displaying an icon which identifies the software program.

59. The method of claim 52, further comprising the step of receiving the software program and the attached self-launching system from an on-line system.

60. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to execute a self-launching system associated with a software program for demonstrating the software program to a potential purchaser of the program, the

computer readable program code means in the computer program product comprising:

means for attaching the self-launching system to a software program;

means for causing the computer to maintain the software program in a locked state in order to prevent unauthorized duplication of the software program; and

activation means for causing the computer to launch the self-launching system when a user selects the software program, the activation means comprising: sample means for causing the computer to enable the software program for execution upon selection by the user, allow the user to subsequently sample the software program, maintain the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the software program, and selectively disable the sampling of the software program.

61. A self-launching system associated with a software program for distributing the software program to a potential purchaser of the program, comprising:

means for attaching the self-launching system to a software program;

means for maintaining the software program in a locked state in order to prevent unauthorized copying of the software program; and

activation means for launching the self-launching system when a user selects the software program, the activation means comprising means for unlocking the software program in response to a request to purchase the software program.

62. The system of claim 61, further comprising means for generating a code identifying the software program.

63. The system of claim 61, further comprising means for displaying an icon which identifies the software program.

64. The system of claim 61, further comprising means for receiving the software program and the attached self-launching system from an on-line system.

65. A method for using a self-launching system associated with a software program for distributing the software program to a potential purchaser of the program, comprising the steps executed by a computer of:

attaching the self-launching system to a software program;

maintaining the software program in a locked state in order to prevent unauthorized copying of the software program; and

launching the self-launching system when a user selects the software program, comprising the step of unlocking the software program in response to a request to purchase the software program.

66. The method of claim 65 wherein the launching step further comprises the step of generating a code identifying the software program.

67. The method of claim 65 wherein the launching step further comprises the step of displaying an icon which identifies the software program.

68. The method of claim 65, further comprising the step executed by the computer of receiving the software program and the attached self-launching system from an on-line system.

69. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a

computer to execute a self-launching system associated with a software program for distributing the software program to a potential purchaser of the program, the computer readable program code means in the computer program product comprising:

means for attaching the self-launching system to a software program;

means for causing the computer to maintain the software program in a locked state in order to prevent unauthorized copying of the software program; and

activation means for causing the computer to launch the self-launching system when a user selects the software program, the activation means comprising means for causing the computer to unlock the software program in response to a request to purchase the software program.

70. A self-launching system associated with digital information for distributing the digital information to a potential purchaser of the program, comprising:

means for attaching the self-launching system to digital information;

means for maintaining the digital information in a locked state in order to prevent unauthorized copying of the digital information; and

activation means for launching the self-launching system when a user selects the digital information, the activation means comprising means for unlocking the digital information in response to a request to purchase the digital information.

71. The system of claim 70, further comprising means for generating a code identifying the digital information.

72. The system of claim 70, further comprising means for displaying an icon which identifies the digital information.

73. The system of claim 70, further comprising means for receiving the digital information and the attached self-launching system from an on-line system.

74. A method for using a self-launching system associated with a digital information for distributing the digital information to a potential purchaser of the program, comprising the steps executed by a computer of:

attaching the self-launching system to a digital information;

maintaining the digital information in a locked state in order to prevent unauthorized copying of the digital information; and

launching the self-launching system when a user selects the digital information, comprising the step of unlocking the digital information in response to a request to purchase the digital information.

75. The method of claim 74 wherein the launching step further comprises the step of generating a code identifying the digital information.

76. The method of claim 74 wherein the launching step further comprises the step of displaying an icon which identifies the digital information.

77. The method of claim 74, further comprising the step executed by the computer of receiving the digital information and the attached self-launching system from an on-line system.

78. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to execute a self-launching system associated with a digital information for distributing the digital information to a potential purchaser of the program, the

computer readable program code means in the computer program product comprising:

means for attaching the self-launching system to a digital information;

means for causing the computer to maintain the digital information in a locked state in order to prevent unauthorized copying of the digital information; and

activation means for causing the computer to launch the self-launching system when a user selects the digital information, the activation means comprising means for causing the computer to unlock the digital information in response to a request to purchase the digital information.

79. A system for storing a code within an operating system of a computer in order to identify whether the computer has executed a particular software program, comprising:

receive means for receiving an indication that the computer has executed the software program;

search means for searching a non-volatile memory in which the operating system for the computer is stored in order to locate spare memory locations within the non-volatile memory;

write means for writing a code to at least one of the spare memory locations, the code providing an indication that the computer has executed the software program; and

means for electronically associating the code with the software program.

80. The system of claim 79 wherein:

the search means comprises means for searching the memory for spare boot block locations; and

the write means comprises means for writing the code to the spare boot block locations.

81. The system of claim 79, further comprising:

means for searching the memory in order to locate the code associated with the particular software program; and

means for providing an indication that the computer has previously executed the software program, if the code is found.

82. The system of claim 79 wherein the write means comprises means for including with the code information related to use of the software program on the computer.

83. The system of claim 79 wherein the write means comprises means for writing a predetermined number of copies of the code to the spare memory locations.

84. The system of claim 79, further comprising:

means for searching the spare memory locations for the predetermined number of copies of the code; and

means for writing copies of the code to the spare memory locations in order to recreate the predetermined number of copies of the code, if the predetermined number of copies of the code were not found.

85. A computerized method for storing a code within an operating system of a computer in order to identify whether the computer has executed a particular software program, comprising the steps executed by a computer of:

receiving an indication that the computer has executed the software program;

searching a non-volatile memory in which the operating system for the computer is stored in order to locate spare memory locations within the non-volatile memory;

writing a code to at least one of the spare memory locations, the code providing an indication that the computer has executed the software program; and

associating the code with the software program.

86. The method of claim 85 wherein:

the searching step comprises the step of searching the memory for spare boot block locations; and

the writing step comprises the step of writing the code to the spare boot block locations.

87. The method of claim 85, further comprising the steps executed by the computer of:

searching the memory in order to locate the code associated with the particular software program; and

providing an indication that the computer has previously executed the software program, if the code is found.

88. The method of claim 85 wherein the writing step comprises the step of including with the code information related to use of the software program on the computer.

89. The method of claim 85 wherein the writing step comprises the step of writing a predetermined number of copies of the code to the spare memory locations.

90. The method of claim 89, further comprising the steps executed by the computer of:

searching the spare memory locations for the predetermined number of copies of the code; and

writing copies of the code to the spare memory locations in order to recreate the predetermined number of copies of the code, if the predetermined number of copies of the code were not found.

91. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to store a code within an operating system of the computer in order to identify whether the computer

has executed a particular software program, the computer readable program code means in the computer program product comprising:

receive means for causing the computer to receive an indication that the computer has executed the software program;

search means for causing the computer to search a non-volatile memory in which the operating system for the computer is stored in order to locate spare memory locations within the non-volatile memory;

write means for causing the computer to write a code to at least one of the spare memory locations, the code providing an indication that the computer has executed the software program; and

means for causing the computer to electronically associate the code with the software program.

92. A system for preventing unauthorized duplication of a particular software program among a plurality of active software programs executed on a computer, comprising:

receive means for receiving an indication that the computer is executing the particular software program;

monitoring means for monitoring operation of the computer to determine which of the plurality of the active software programs is being currently executed; and

disable means for disabling execution of the particular software program when the monitoring means determines that the particular software program is not the currently executed software program.

93. The system of claim 92 wherein the monitoring means comprises means for monitoring the operation of the computer within a multi-tasking environment which includes a distinct visual indicator for each of the active software programs.

94. The system of claim 92 wherein the monitoring means further comprises means for identifying a front-most indicator among the plurality of active software programs.

95. The system of claim 92 wherein the disable means comprises means for disabling an image driver corresponding to the particular software program.

96. A computerized method for preventing unauthorized duplication of a particular software program among a plurality of active software programs executed on a computer, comprising the steps executed by a computer of:

receiving an indication that the computer is executing the particular software program;

monitoring operation of the computer to determine which of the plurality of the active software programs is being currently executed; and

disabling execution of the particular software program when the monitoring means determines that the particular software program is not the currently executed software program.

97. The method of claim 96 wherein the monitoring step comprises the step of monitoring the operation of the computer within a multi-tasking environment which includes a distinct visual indicator for each of the active software programs.

98. The method of claim 97 wherein the monitoring step further comprises the step executed by the computer of identifying a front-most indicator among the plurality of active software programs.

99. The method of claim 96 wherein the disabling step comprises the step of disabling an image driver corresponding to the particular software program.

100. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to prevent unauthorized duplication of a particular software program among a plurality of active software programs executed on the computer, the computer readable program code means in the computer program product comprising:

receive means for causing the computer to receive an indication that the computer is executing the particular software program;

monitoring means for causing the computer to monitor operation of the computer to determine which of the plurality of the active software programs is being currently executed; and

disable means for causing the computer to disable execution of the particular software program when the monitoring means determines that the particular software program is not the currently executed software program.

101. A computer-based system for automatic sales of software programs, comprising:

means for accessing a software program within a computer database and for maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program;

receive means for receiving a request to purchase the software program;

unlocking means for unlocking a copy of the software program in response to the request to purchase the software program;

means for distributing the unlocked copy; and

recording means for recording how many copies of the software program have been distributed in response to the requests to purchase the software program.

102. The system of claim 101 wherein the unlocking means comprises means for assigning serial numbers to the purchased copies of the software program.

103. The system of claim 101 wherein the recording means comprises means for automatically issuing an invoice for the purchased copies of the software program.

104. The system of claim 101 wherein the means for accessing comprises means for receiving the software program from an on-line system.

105. The system of claim 101, further comprising means for allowing a potential purchaser to sample the software program prior to receiving the request to purchase the software program.

106. The system of claim 105 wherein the means for allowing a potential purchaser to sample the software program comprises: sample means for enabling the software program for execution upon selection by a user, for allowing the user to subsequently sample the software program, for maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the selected software program, and for disabling the sampling of the selected software program.

107. The system of claim 106 wherein the sample means further comprises means for preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

108. The system of claim 106 wherein the sample means further comprises means for detecting if the software program is being copied during the sampling of the software program and for disabling the software program in response to the detecting..

109. The system of claim 106 wherein the sample means comprises means for limiting how many times the software program can be sampled and for displaying an indication of a number of samples remaining.

110. The system of claim 106 wherein the sample means comprises means for disabling the software program if the user has sampled the software program for a predetermined amount of time.

111. The system of claim 106 wherein:

the means for accessing comprises means for receiving the software program in an encrypted state; and

the sample means comprises means for decrypting the encrypted software program.

112. The system of claim 106 wherein the sample means comprises means for displaying an icon which identifies the software program.

113. A computerized method for automatic sales of software programs, comprising the steps executed by a computer of:

accessing a software program within a computer database and maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program;

receiving a request to purchase the software program;

unlocking a copy of the software program in response to the request to purchase the software program;

distributing the unlocked copy; and

recording how many copies of the software program have been distributed in response to the requests to purchase the software program.

114. The method of claim 113 wherein the unlocking step comprises the step of assigning serial numbers to the purchased copies of the software program.

115. The method of claim 113 wherein the recording step comprises the step of automatically issuing an invoice for the purchased copies of the software program.

116. The method of claim 113 wherein the means for accessing comprises means for receiving the software program from an on-line system.

117. The method of claim 113, further comprising the step executed by the computer of allowing a potential purchaser to sample the software program prior to receiving the request to purchase the software program.

118. The method of claim 117 wherein the step of allowing a potential purchaser to sample the software program comprises the steps of:

enabling the software program for execution upon selection by a user;

allowing the user to subsequently sample the software program;

maintaining the software program in the locked state during the sampling of the software program in order to prevent unauthorized duplication of the selected software program; and

disabling the sampling of the selected software program.

119. The method of claim 118 wherein the enabling step further comprises the step of preventing the enabling of the software program when the user has already sampled the software program a predetermined number of times.

120. The method of claim 118 wherein the disabling step further comprises the steps executed by the computer of

detecting if the software program is being copied during the sampling of the software program and disabling the software program in response to the detecting.

121. The method of claim 118 wherein the enabling step comprises the steps of limiting how many times the software program can be sampled and displaying an indication of a number of samples remaining.

122. The method of claim 118 wherein the disabling step comprises the step of disabling the software program if the user has sampled the software program for a predetermined amount of time.

123. The method of claim 118 wherein:

the accessing step comprises the step of receiving the software program in an encrypted state; and

the enabling step comprises the step of decrypting the encrypted software program.

124. The method of claim 118 wherein the enabling step comprises the step of displaying an icon which identifies the software program.

125. A computer program product, comprising:

a computer usable medium having computer readable program code means embodied therein for causing a computer to execute a system for automatic sales of software programs, the computer readable program code means in the computer program product comprising:

means for accessing a software program within a computer database and for maintaining the software program in a locked state in order to prevent unauthorized duplication of the software program;

receive means for causing the computer to receive a request to purchase the software program;

unlocking means for causing the computer to unlock a copy of the software program in response to the request to purchase the software program;

means for causing the computer to distribute the unlocked copy; and

recording means for causing the computer to record how many copies of the software program have been distributed in response to the requests to purchase the software program.

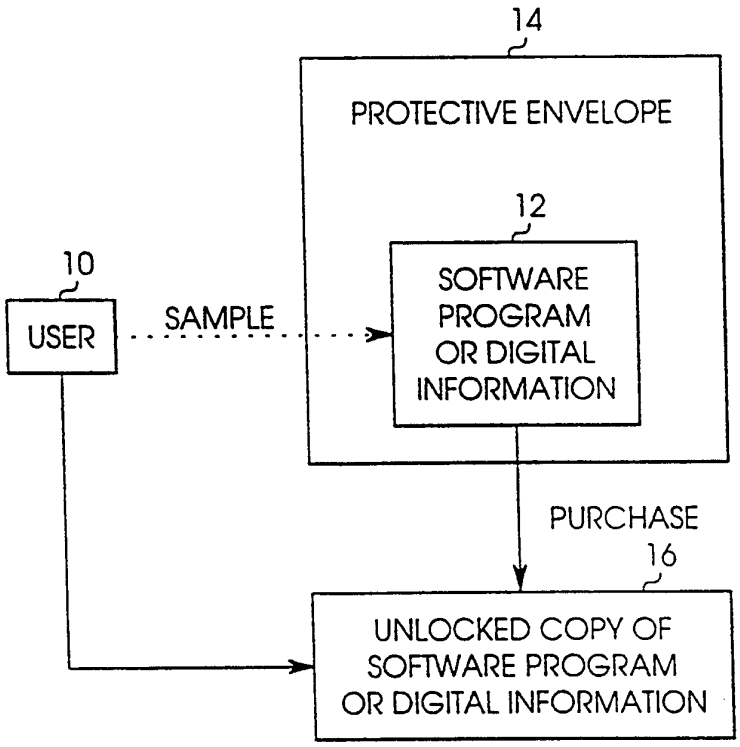

14
PROTECTIVE ENVELOPE
12
SOFTWARE PROGRAM OR DIGITAL INFORMATION
10
USER
SAMPLE
PURCHASE
16
UNLOCKED COPY OF SOFTWARE PROGRAM OR DIGITAL INFORMATION

Fig. 1

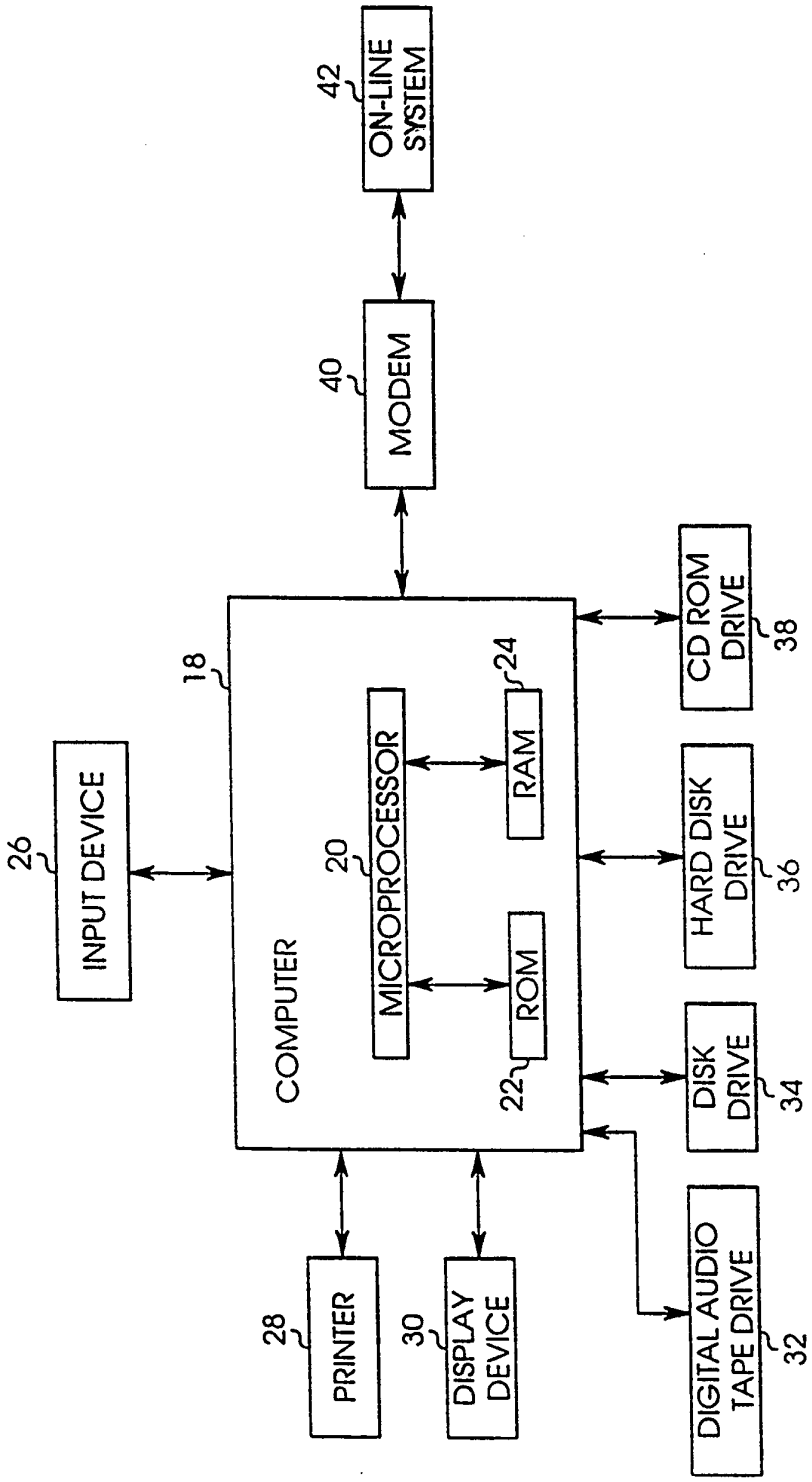

42
ON-LINE SYSTEM
40
MODEM
18
COMPUTER
20
MICROPROCESSOR
24
RAM
22
ROM
26
INPUT DEVICE
CD ROM DRIVE
38
HARD DISK DRIVE
36
DISK DRIVE
34
DIGITAL AUDIO TAPE DRIVE
32
28
PRINTER
30
DISPLAY DEVICE

Fig. 2

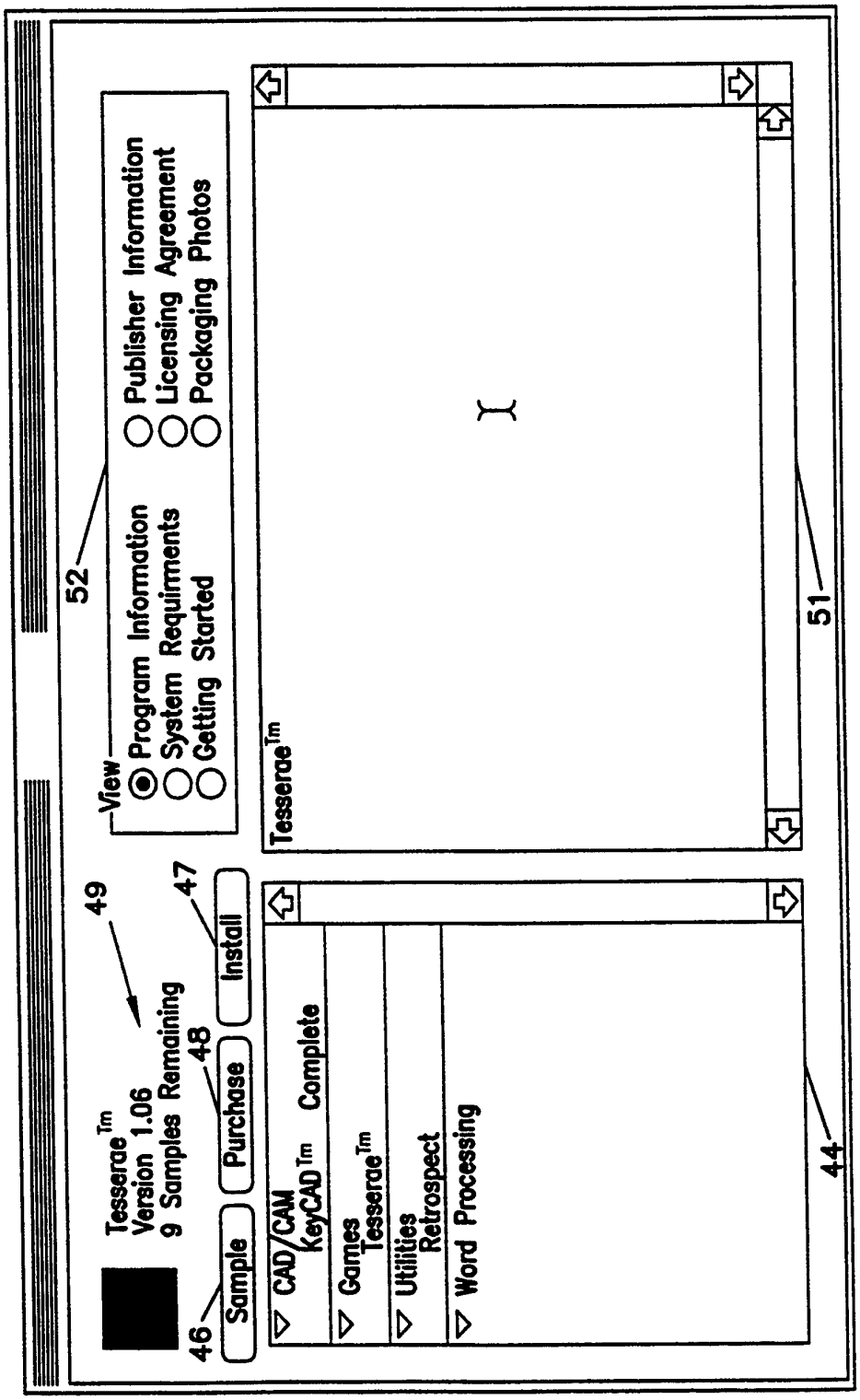

Tesserae Tm
Version 1.06
9 Samples Remaining
Sample
Purchase
Install
View
Program Information
System Requirments
Getting Started
Publisher Information
Licensing Agreement
Packaging Photos
Tesserae Tm
CAD/CAM
KeyCAD Tm
Complete
Games
Tesserae Tm
Utilities
Retrospect
Word Processing
46
47
48
49
51
52
44

FIG. 3

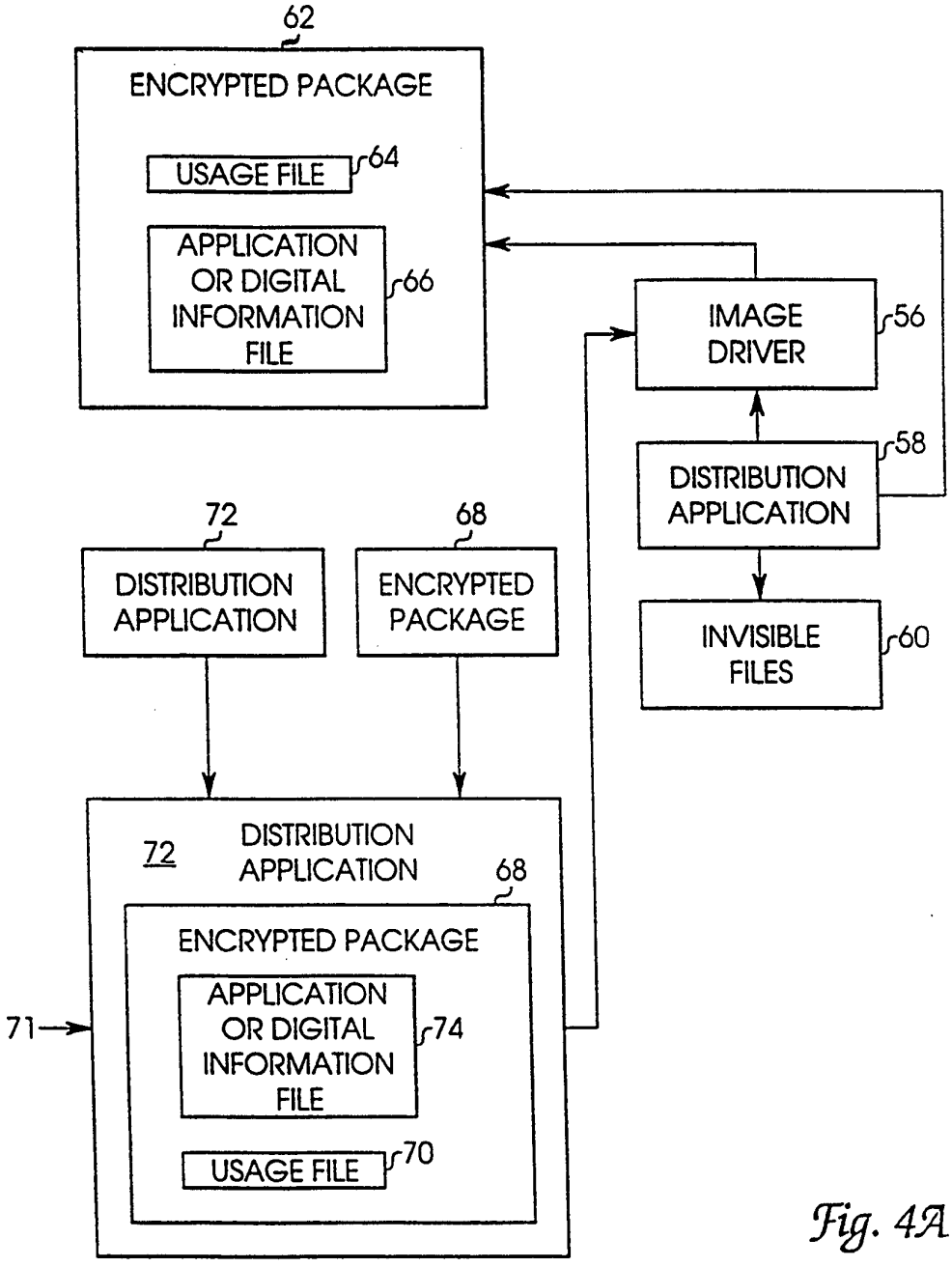

62
ENCRYPTED PACKAGE
USAGE FILE
64
APPLICATION OR DIGITAL INFORMATION FILE
66
IMAGE DRIVER
56
DISTRIBUTION APPLICATION
58
INVISIBLE FILES
60
72
DISTRIBUTION APPLICATION
68
ENCRYPTED PACKAGE
71
72
DISTRIBUTION APPLICATION
68
ENCRYPTED PACKAGE
APPLICATION OR DIGITAL INFORMATION FILE
74
USAGE FILE
70

Fig. 4A

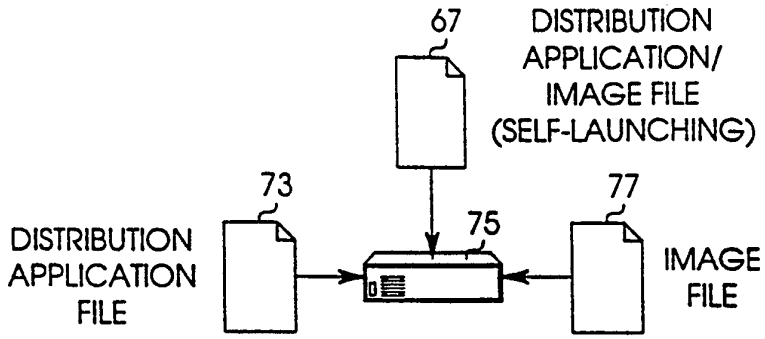

67
DISTRIBUTION APPLICATION/ IMAGE FILE (SELF-LAUNCHING)
73
77
75
DISTRIBUTION APPLICATION FILE
IMAGE FILE

Fig. 4B

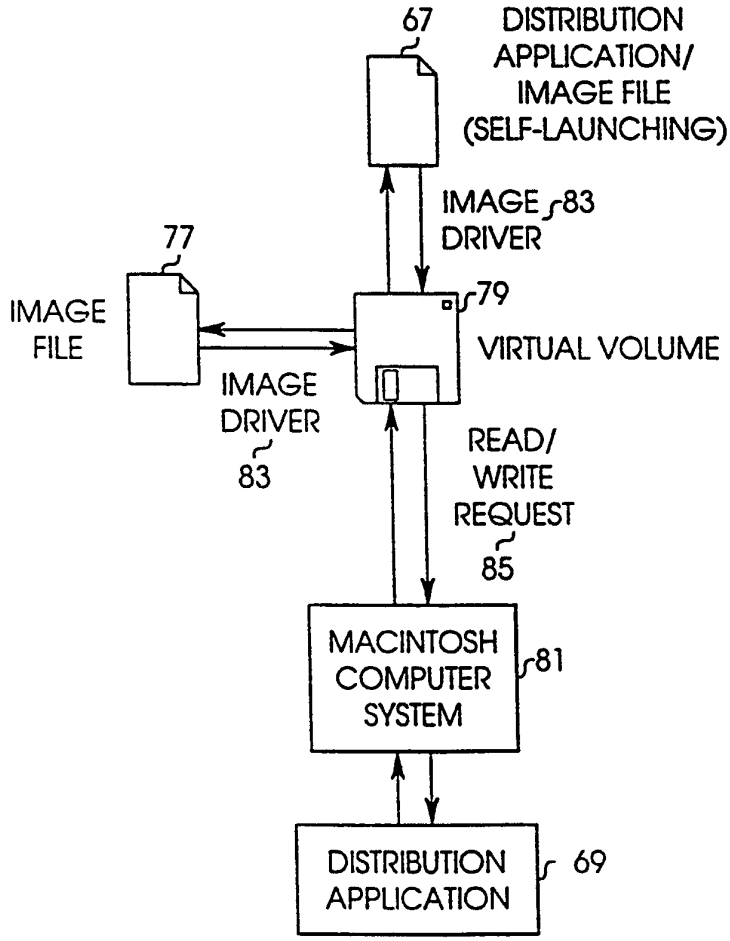

67
DISTRIBUTION APPLICATION/ IMAGE FILE (SELF-LAUNCHING)
IMAGE DRIVER 83
77
79
IMAGE FILE
VIRTUAL VOLUME
IMAGE DRIVER 83
READ/ WRITE REQUEST 85
MACINTOSH COMPUTER SYSTEM 81
DISTRIBUTION APPLICATION 69

Fig. 4C

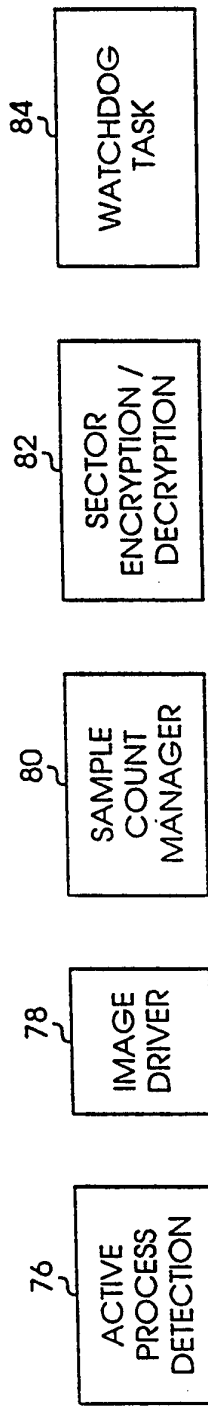

84
WATCHDOG TASK
82
SECTOR ENCRYPTION / DECRYPTION
80
SAMPLE COUNT MANAGER
78
IMAGE DRIVER
76
ACTIVE PROCESS DETECTION

Fig. 5

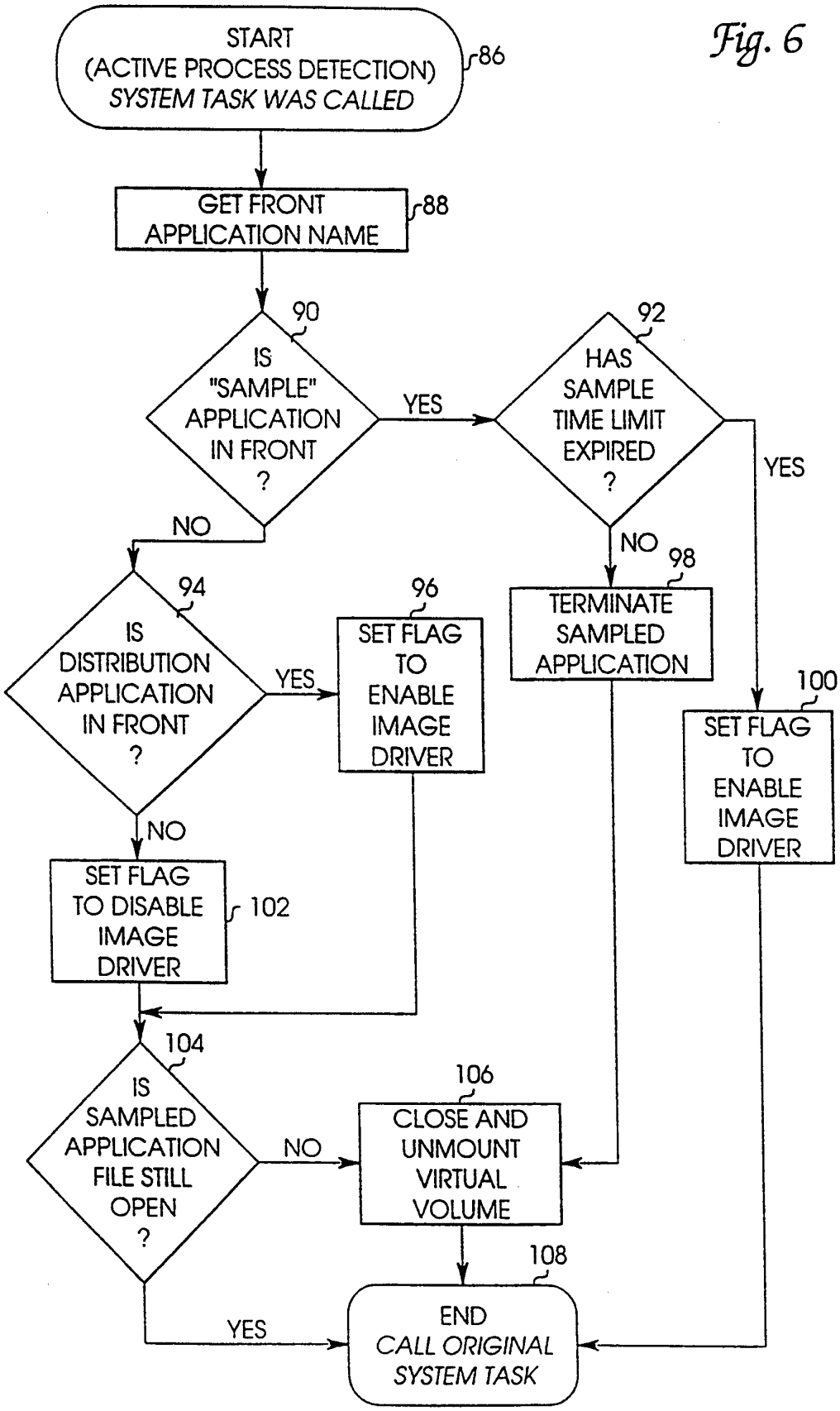

START
(ACTIVE PROCESS DETECTION)
SYSTEM TASK WAS CALLED
86
GET FRONT APPLICATION NAME
88
90
IS "SAMPLE" APPLICATION IN FRONT ?
YES
92
HAS SAMPLE TIME LIMIT EXPIRED ?
YES
NO
NO
94
IS DISTRIBUTION APPLICATION IN FRONT ?
YES
96
SET FLAG TO ENABLE IMAGE DRIVER
98
TERMINATE SAMPLED APPLICATION
100
SET FLAG TO ENABLE IMAGE DRIVER
NO
SET FLAG TO DISABLE IMAGE DRIVER
102
104
IS SAMPLED APPLICATION FILE STILL OPEN ?
NO
106
CLOSE AND UNMOUNT VIRTUAL VOLUME
108
YES
END
CALL ORIGINAL SYSTEM TASK

Fig. 6

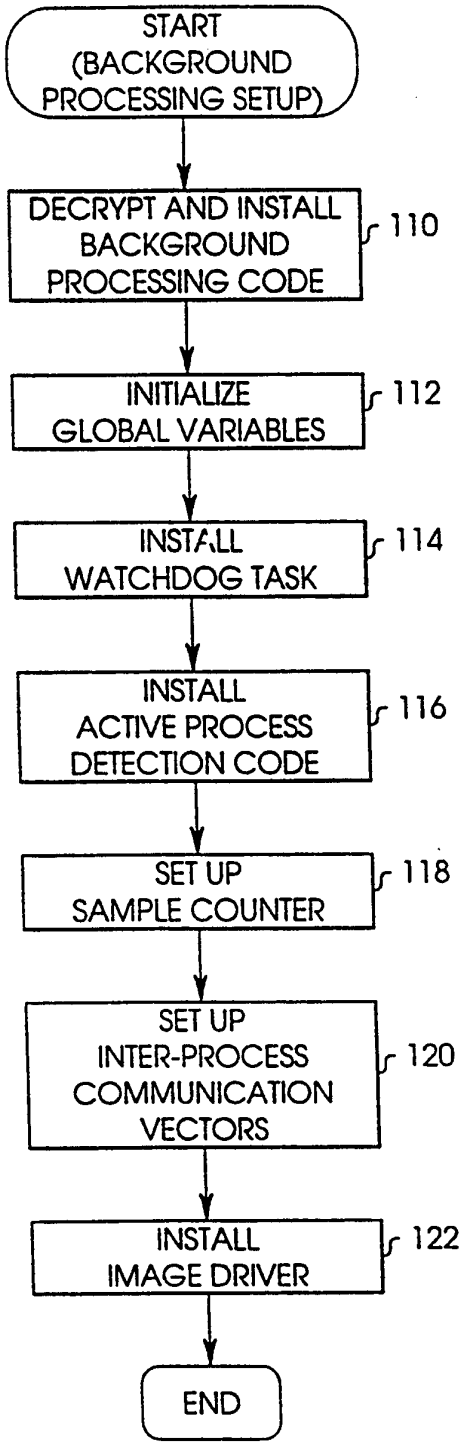

START (BACKGROUND PROCESSING SETUP)
DECRYPT AND INSTALL BACKGROUND PROCESSING CODE
110
INITIALIZE GLOBAL VARIABLES
112
INSTALL WATCHDOG TASK
114
INSTALL ACTIVE PROCESS DETECTION CODE
116
SET UP SAMPLE COUNTER
118
SET UP INTER-PROCESS COMMUNICATION VECTORS
120
INSTALL IMAGE DRIVER
122
END

Fig. 7

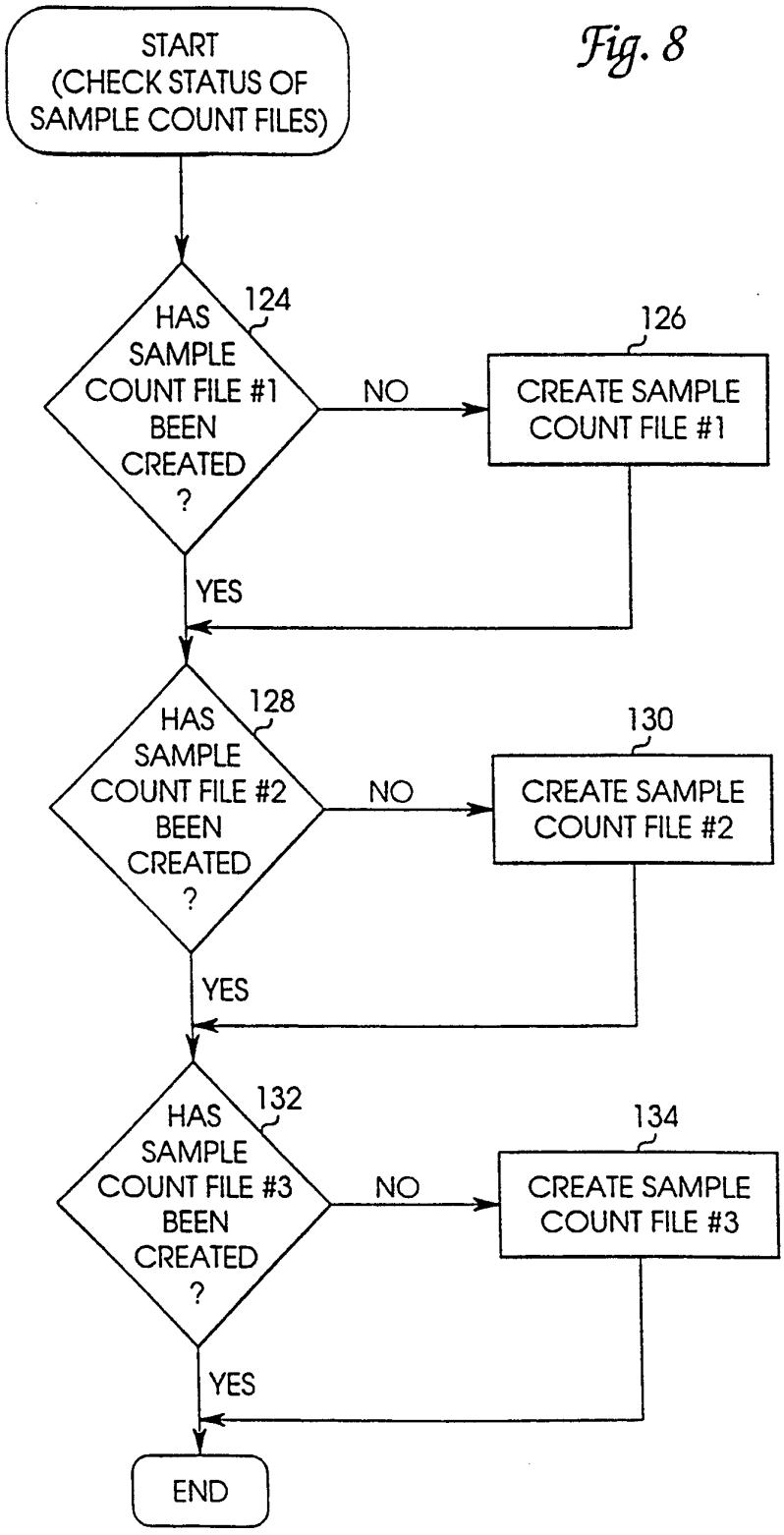

START
(CHECK STATUS OF
SAMPLE COUNT FILES)
124
HAS
SAMPLE
COUNT FILE #1
BEEN
CREATED
?
NO
126
CREATE SAMPLE
COUNT FILE #1
YES
128
HAS
SAMPLE
COUNT FILE #2
BEEN
CREATED
?
NO
130
CREATE SAMPLE
COUNT FILE #2
YES
132
HAS
SAMPLE
COUNT FILE #3
BEEN
CREATED
?
NO
134
CREATE SAMPLE
COUNT FILE #3
YES
END

Fig. 8

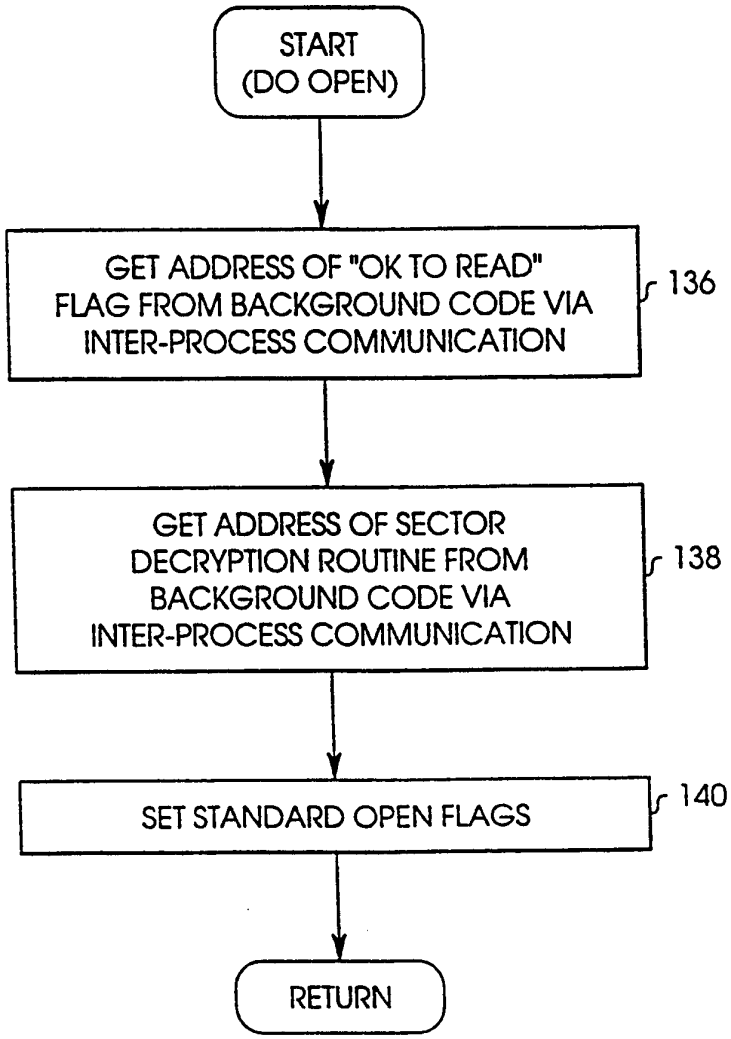
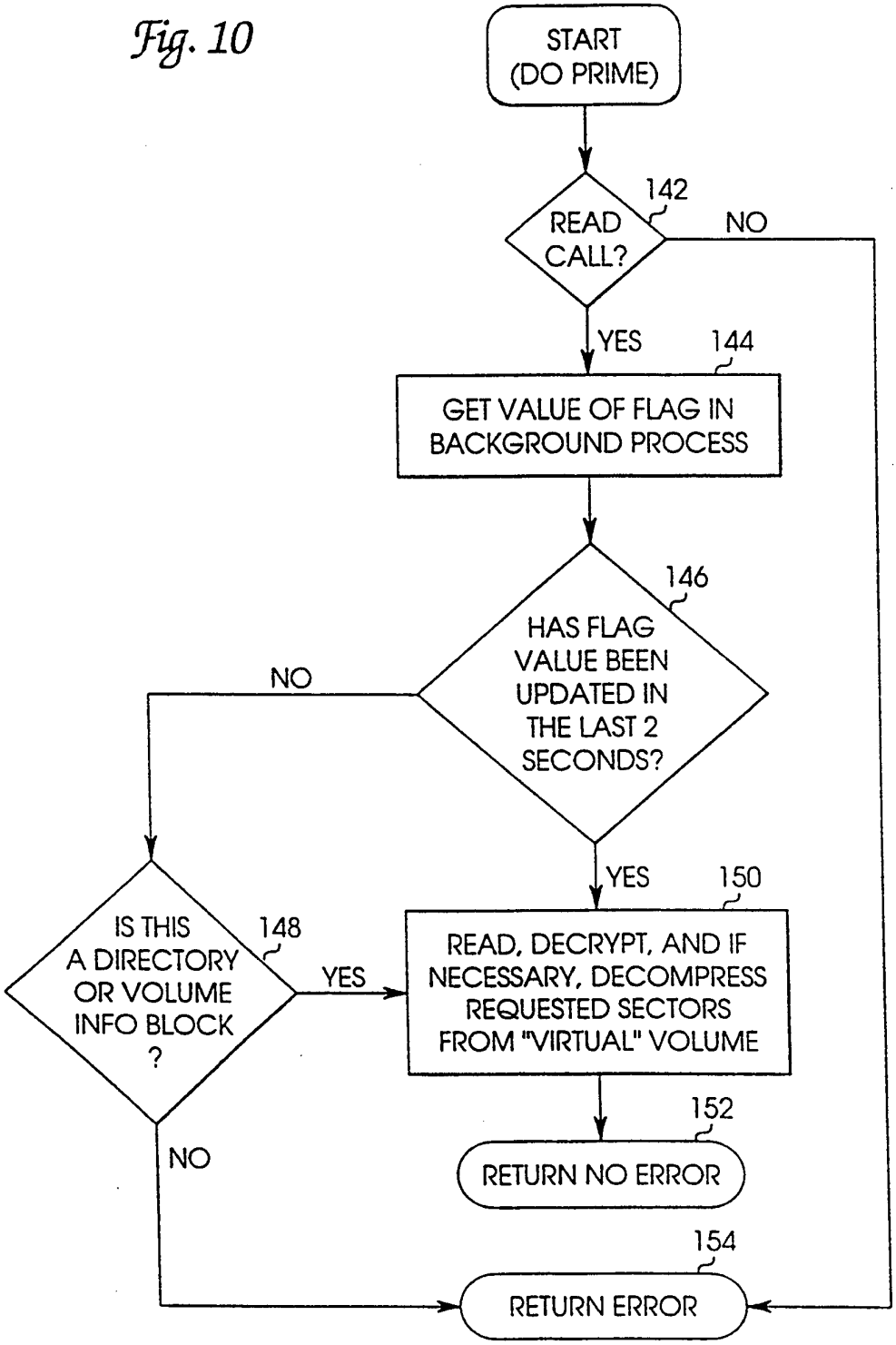

START
(DO OPEN)
GET ADDRESS OF "OK TO READ"
FLAG FROM BACKGROUND CODE VIA
INTER-PROCESS COMMUNICATION
136
GET ADDRESS OF SECTOR
DECRYPTION ROUTINE FROM
BACKGROUND CODE VIA
INTER-PROCESS COMMUNICATION
138
SET STANDARD OPEN FLAGS
140
RETURN

Fig. 9

Fig. 10


START
(DO PRIME)
142
READ
CALL?
NO
YES
144
GET VALUE OF FLAG IN
BACKGROUND PROCESS
146
HAS FLAG
VALUE BEEN
UPDATED IN
THE LAST 2
SECONDS?
NO
YES
150
IS THIS
A DIRECTORY
OR VOLUME
INFO BLOCK
?
148
YES
READ, DECRYPT, AND IF
NECESSARY, DECOMPRESS
REQUESTED SECTORS
FROM "VIRTUAL" VOLUME
152
NO
RETURN NO ERROR
154
RETURN ERROR

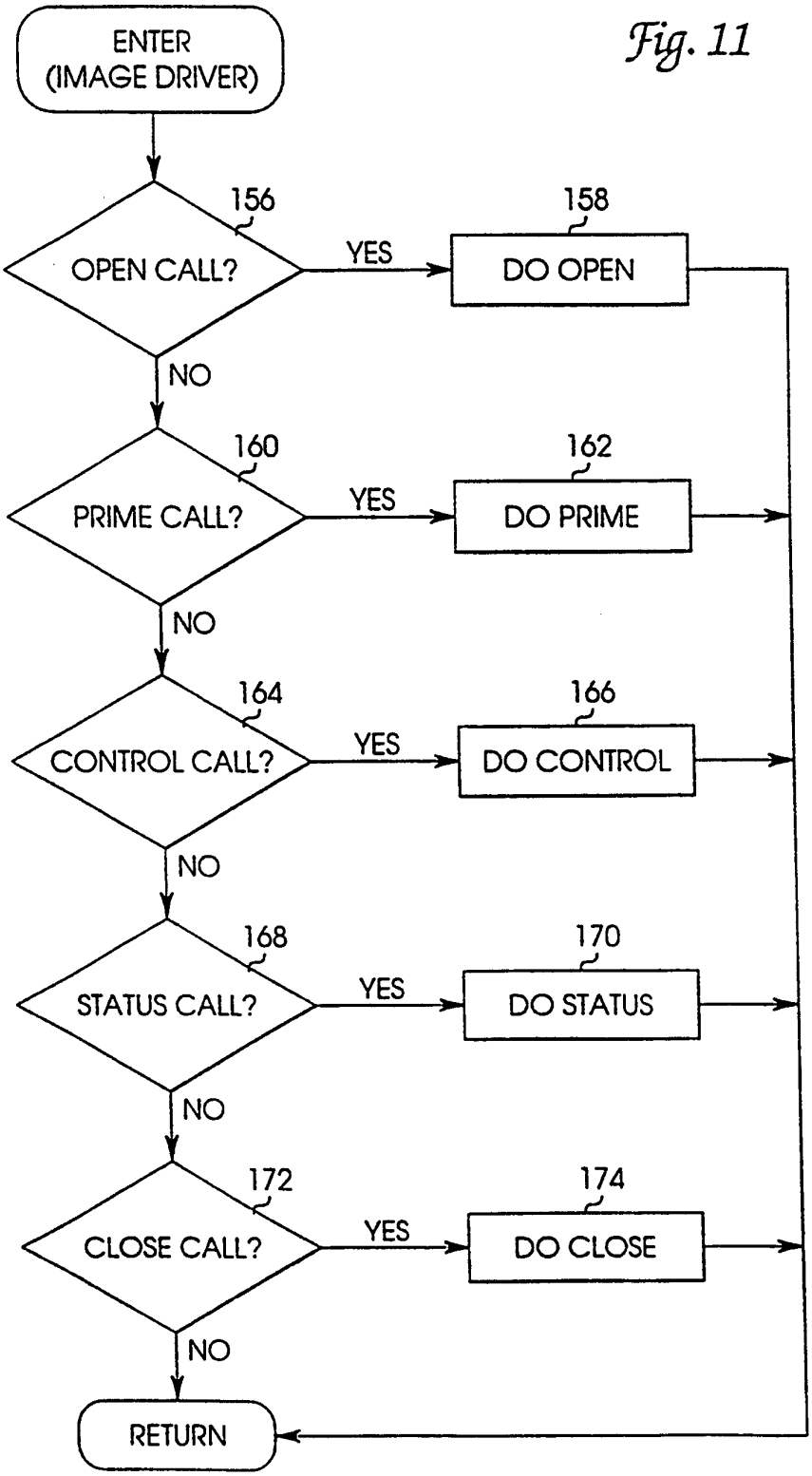

ENTER
(IMAGE DRIVER)
156
OPEN CALL?
YES
158
DO OPEN
NO
160
PRIME CALL?
YES
162
DO PRIME
NO
164
CONTROL CALL?
YES
166
DO CONTROL
NO
168
STATUS CALL?
YES
170
DO STATUS
NO
172
CLOSE CALL?
YES
174
DO CLOSE
NO
RETURN

Fig. 11

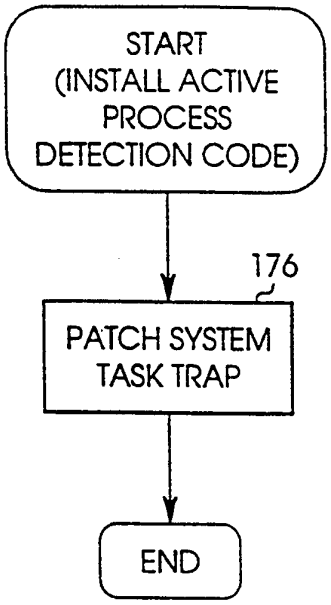


Fig. 12

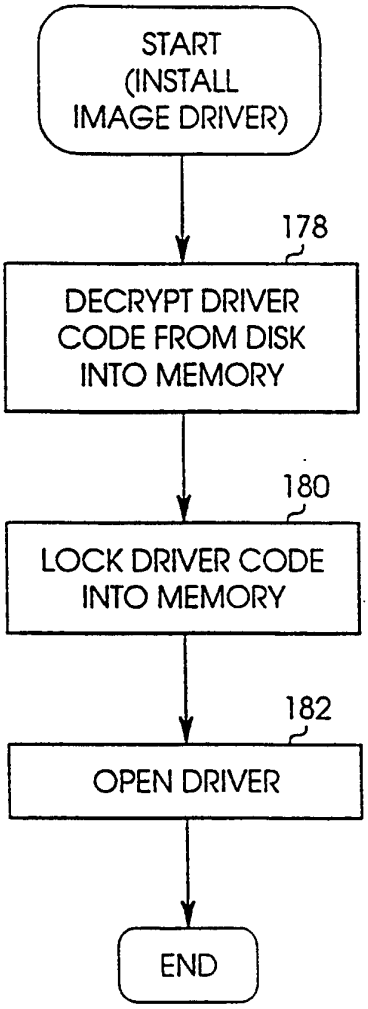


Fig. 13

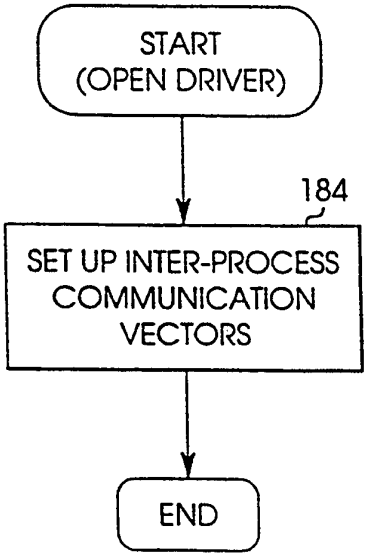


Fig. 14

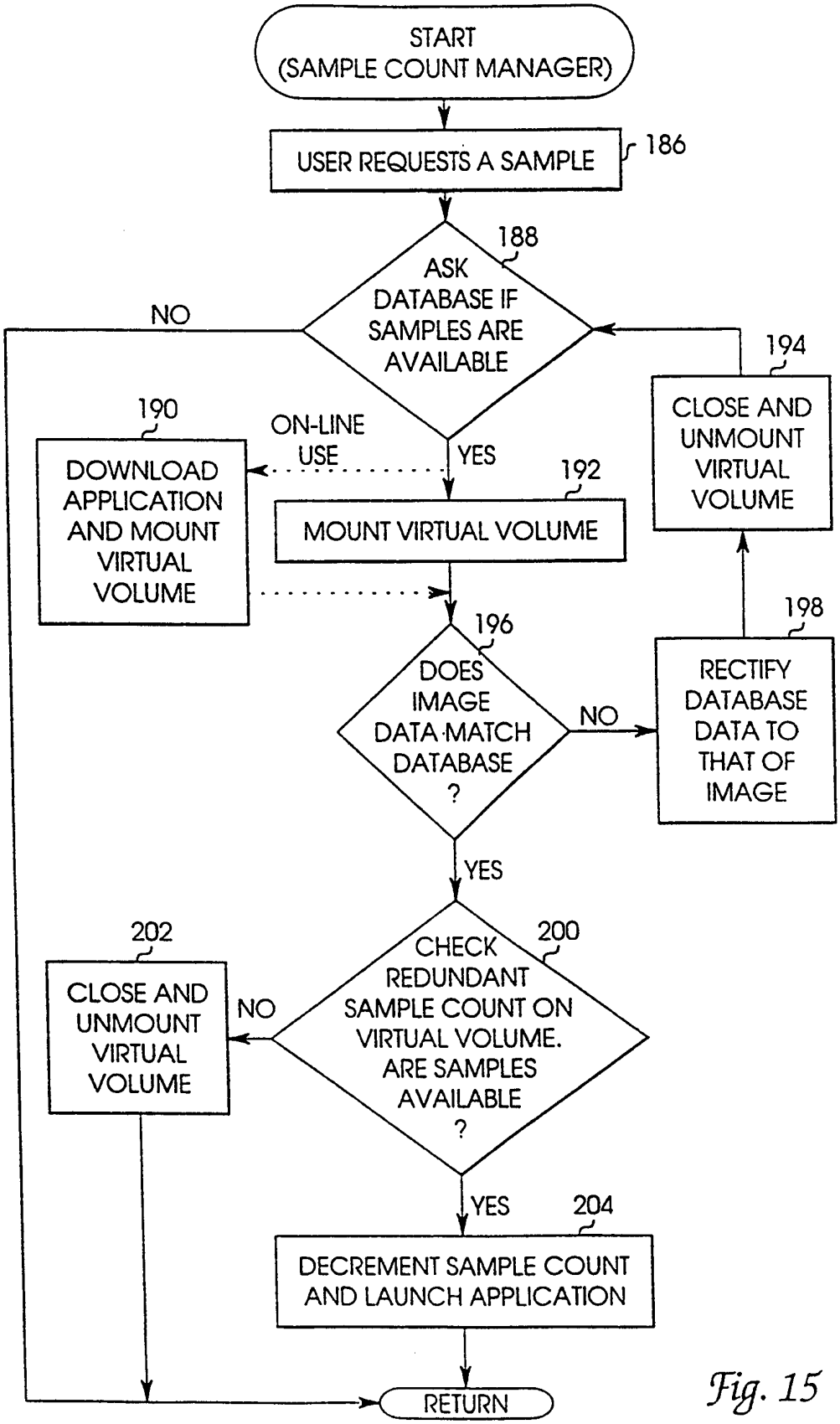

START
(SAMPLE COUNT MANAGER)
USER REQUESTS A SAMPLE
186
ASK
DATABASE IF
SAMPLES ARE
AVAILABLE
188
NO
194
CLOSE AND
UNMOUNT
VIRTUAL
VOLUME
190
ON-LINE
USE
YES
DOWNLOAD
APPLICATION
AND MOUNT
VIRTUAL
VOLUME
192
MOUNT VIRTUAL VOLUME
198
196
DOES
IMAGE
DATA MATCH
DATABASE
?
NO
RECTIFY
DATABASE
DATA TO
THAT OF
IMAGE
YES
202
200
CHECK
REDUNDANT
SAMPLE COUNT ON
VIRTUAL VOLUME.
ARE SAMPLES
AVAILABLE
?
CLOSE AND
UNMOUNT
VIRTUAL
VOLUME
NO
YES
204
DECREMENT SAMPLE COUNT
AND LAUNCH APPLICATION
RETURN

Fig. 15

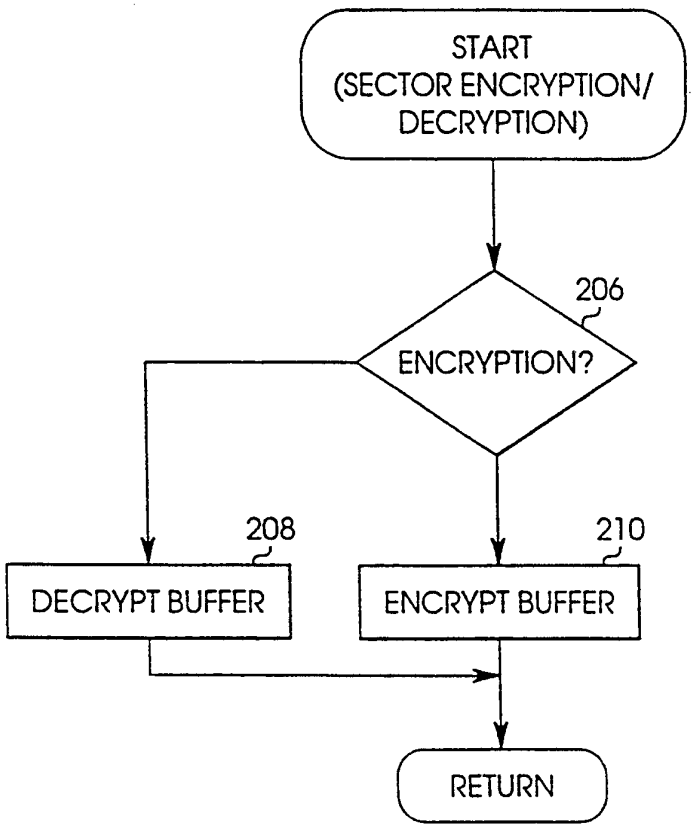

START
(SECTOR ENCRYPTION/
DECRYPTION)
206
ENCRYPTION?
208
210
DECRYPT BUFFER
ENCRYPT BUFFER
RETURN

Fig. 16A

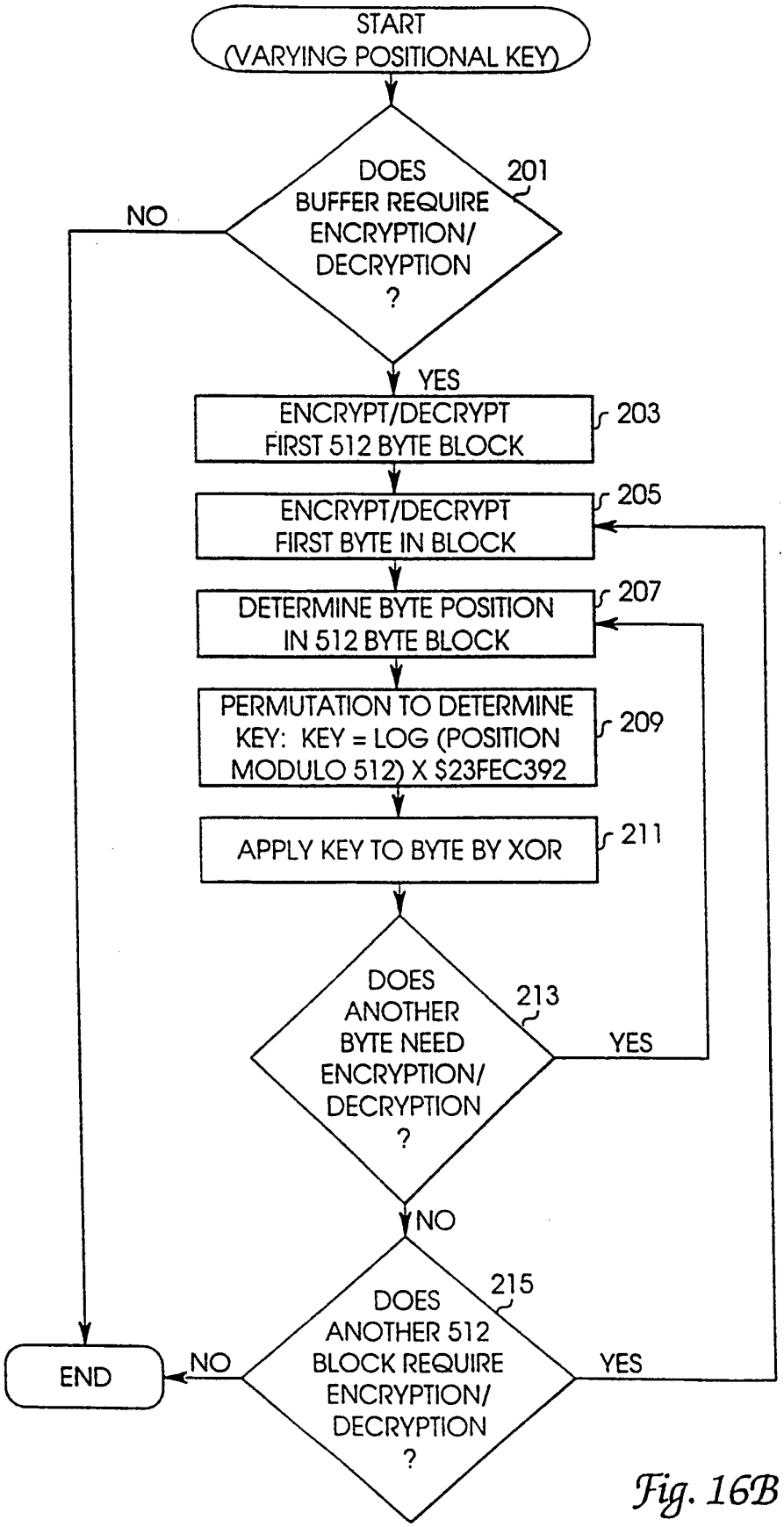
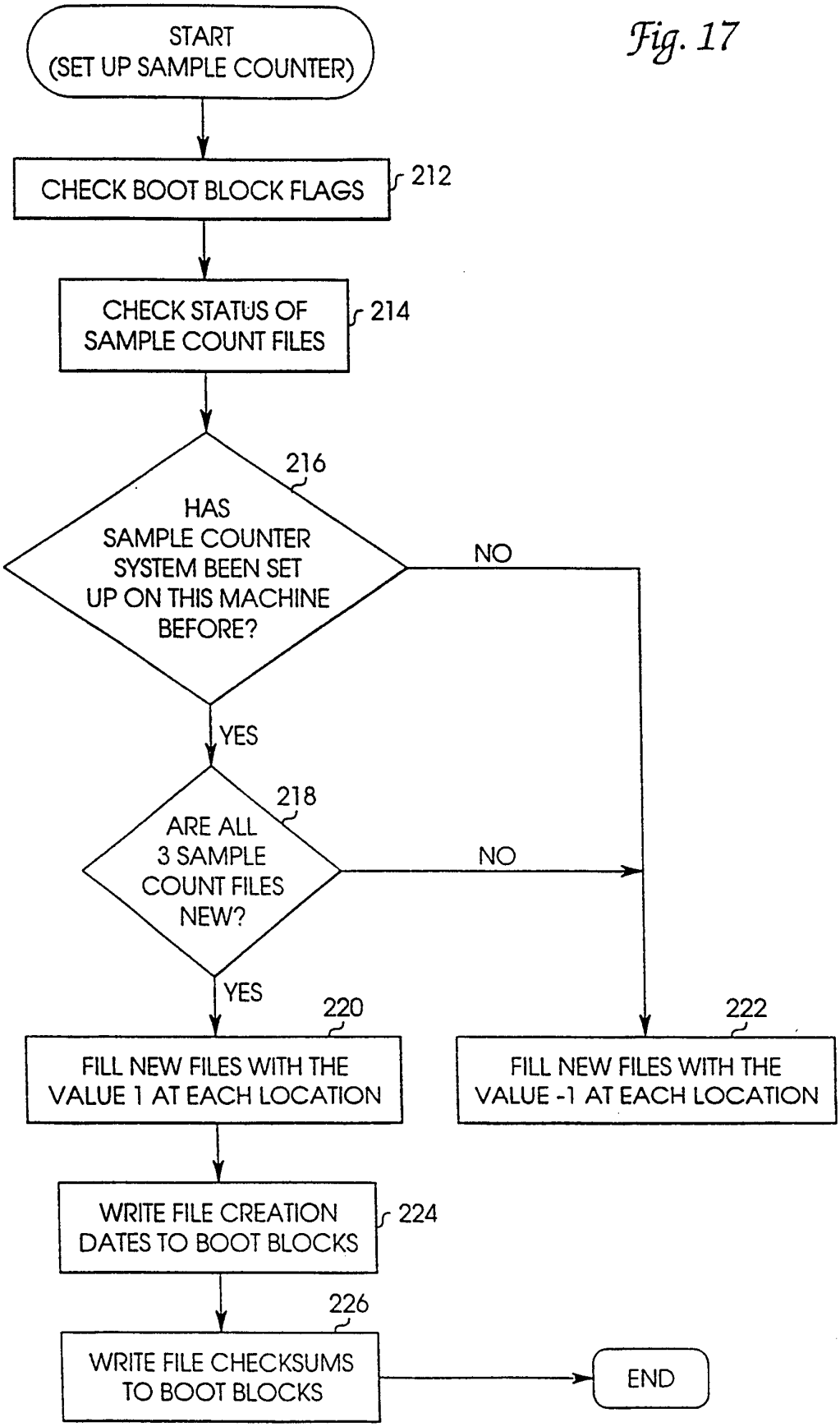

START
(VARYING POSITIONAL KEY)
DOES BUFFER REQUIRE ENCRYPTION/ DECRYPTION ?
201
NO
YES
ENCRYPT/DECRYPT FIRST 512 BYTE BLOCK
203
ENCRYPT/DECRYPT FIRST BYTE IN BLOCK
205
DETERMINE BYTE POSITION IN 512 BYTE BLOCK
207
PERMUTATION TO DETERMINE KEY: KEY = LOG (POSITION MODULO 512) X $23FEC392
209
APPLY KEY TO BYTE BY XOR
211
DOES ANOTHER BYTE NEED ENCRYPTION/ DECRYPTION ?
213
YES
NO
DOES ANOTHER 512 BLOCK REQUIRE ENCRYPTION/ DECRYPTION ?
215
NO
YES
END

Fig. 16B

Fig. 17


START
(SET UP SAMPLE COUNTER)
CHECK BOOT BLOCK FLAGS
212
CHECK STATUS OF
SAMPLE COUNT FILES
214
216
HAS
SAMPLE COUNTER
SYSTEM BEEN SET
UP ON THIS MACHINE
BEFORE?
NO
YES
218
ARE ALL
3 SAMPLE
COUNT FILES
NEW?
NO
YES
220
FILL NEW FILES WITH THE
VALUE 1 AT EACH LOCATION
222
FILL NEW FILES WITH THE
VALUE -1 AT EACH LOCATION
WRITE FILE CREATION
DATES TO BOOT BLOCKS
224
226
WRITE FILE CHECKSUMS
TO BOOT BLOCKS
END

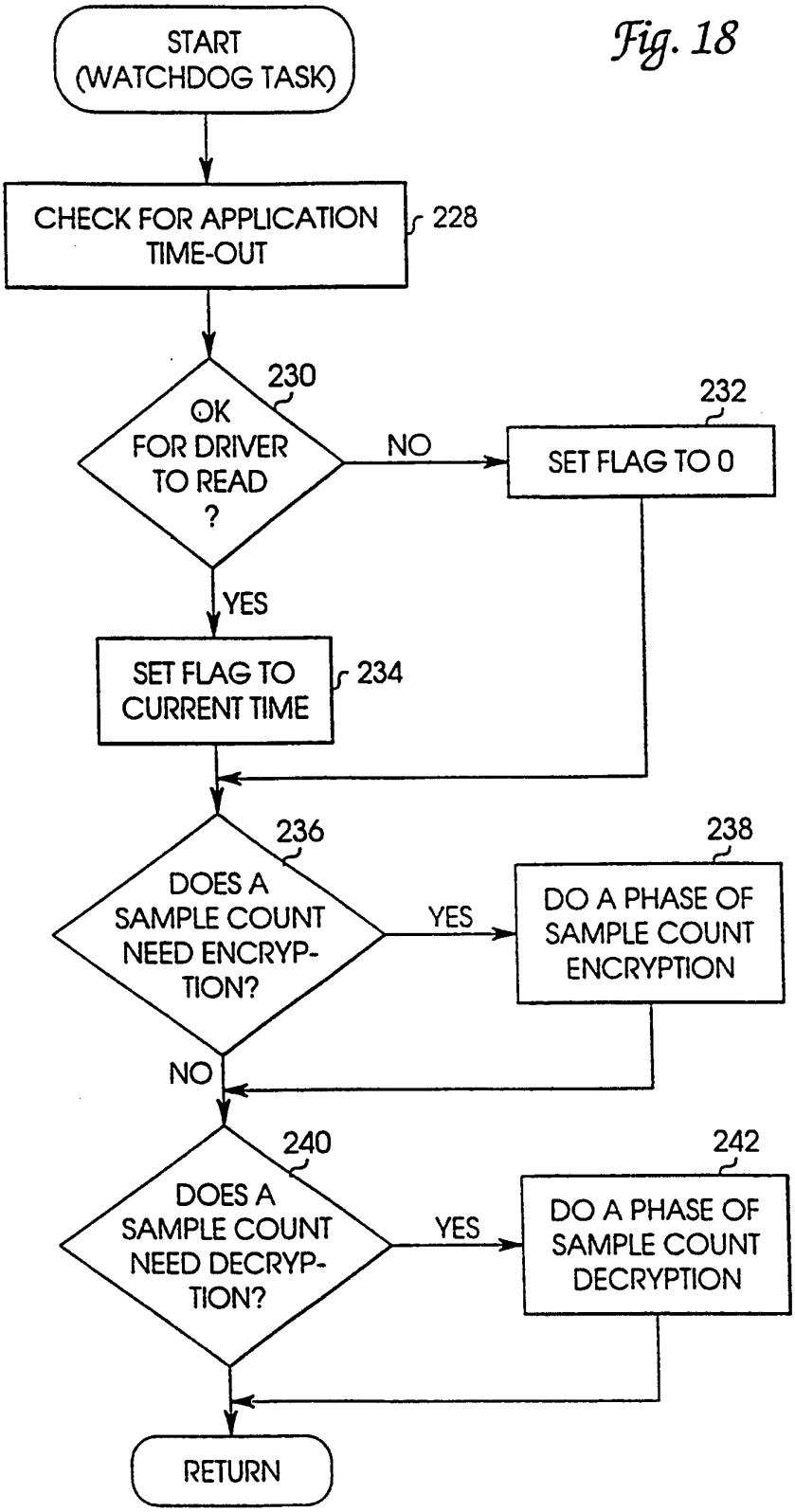

START (WATCHDOG TASK)
CHECK FOR APPLICATION TIME-OUT
228
230
OK FOR DRIVER TO READ ?
NO
232
SET FLAG TO 0
YES
SET FLAG TO CURRENT TIME
234
236
DOES A SAMPLE COUNT NEED ENCRYP-TION?
YES
238
DO A PHASE OF SAMPLE COUNT ENCRYPTION
NO
240
DOES A SAMPLE COUNT NEED DECRYP-TION?
YES
242
DO A PHASE OF SAMPLE COUNT DECRYPTION
RETURN

Fig. 18

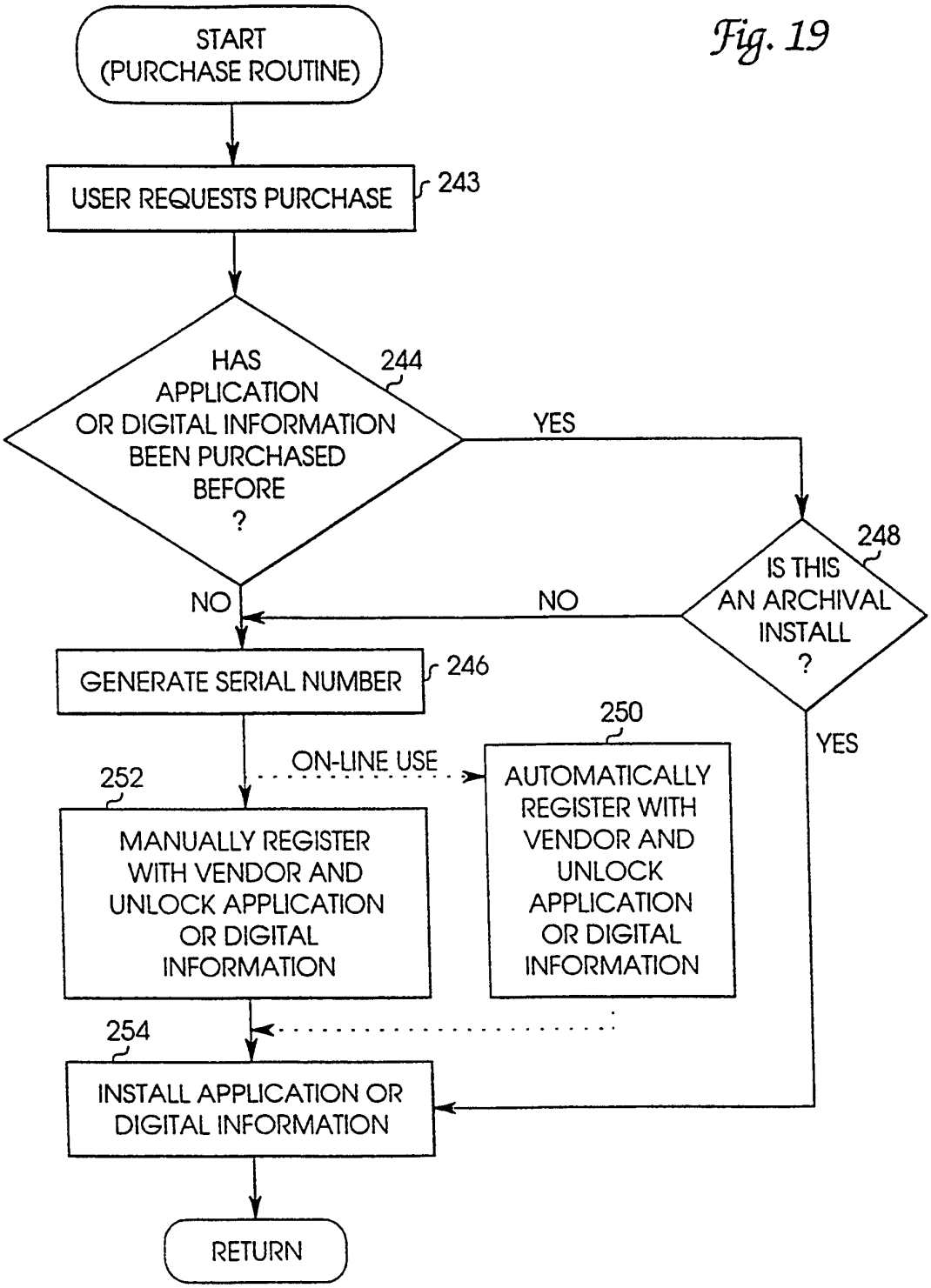
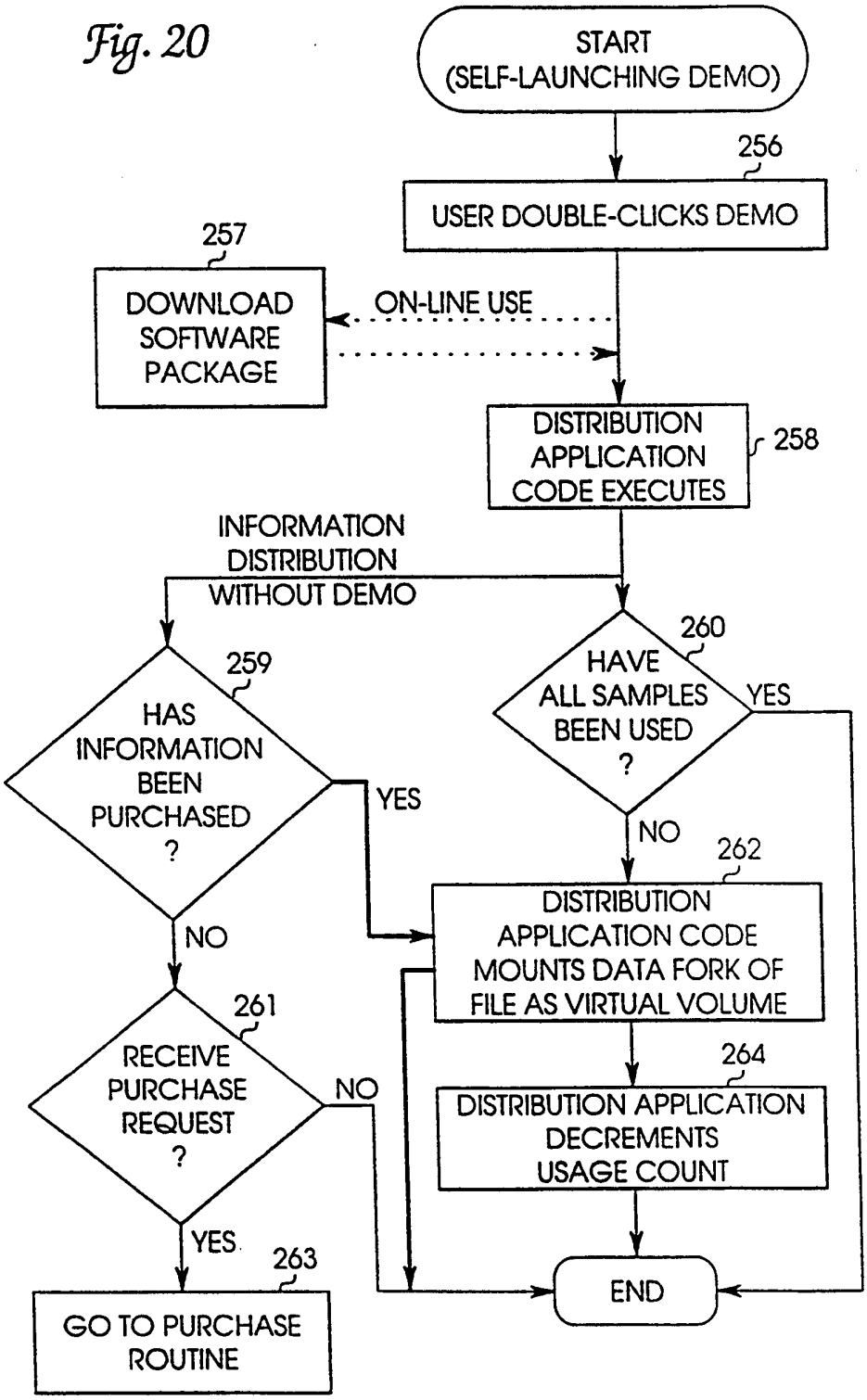

START
(PURCHASE ROUTINE)
USER REQUESTS PURCHASE
243
HAS APPLICATION OR DIGITAL INFORMATION BEEN PURCHASED BEFORE ?
244
YES
NO
IS THIS AN ARCHIVAL INSTALL ?
248
NO
YES
GENERATE SERIAL NUMBER
246
ON-LINE USE
250
252
MANUALLY REGISTER WITH VENDOR AND UNLOCK APPLICATION OR DIGITAL INFORMATION
AUTOMATICALLY REGISTER WITH VENDOR AND UNLOCK APPLICATION OR DIGITAL INFORMATION
254
INSTALL APPLICATION OR DIGITAL INFORMATION
RETURN

Fig. 19

Fig. 20


START
(SELF-LAUNCHING DEMO)
256
USER DOUBLE-CLICKS DEMO
257
DOWNLOAD SOFTWARE PACKAGE
ON-LINE USE
DISTRIBUTION APPLICATION CODE EXECUTES
258
INFORMATION DISTRIBUTION WITHOUT DEMO
259
HAS INFORMATION BEEN PURCHASED ?
YES
NO
260
HAVE ALL SAMPLES BEEN USED ?
YES
NO
262
DISTRIBUTION APPLICATION CODE MOUNTS DATA FORK OF FILE AS VIRTUAL VOLUME
261
RECEIVE PURCHASE REQUEST ?
NO
YES
263
GO TO PURCHASE ROUTINE
264
DISTRIBUTION APPLICATION DECREMENTS USAGE COUNT
END

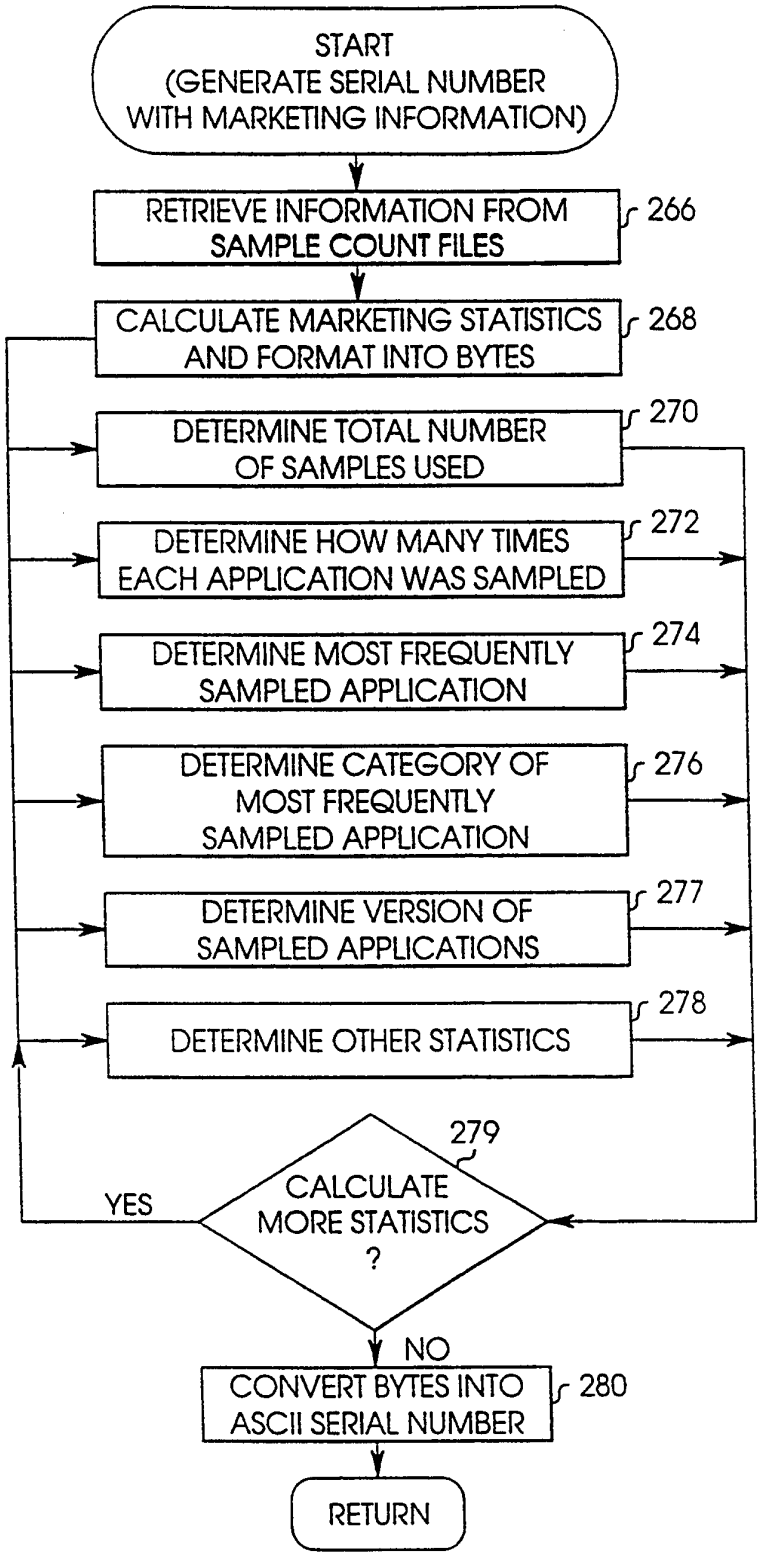
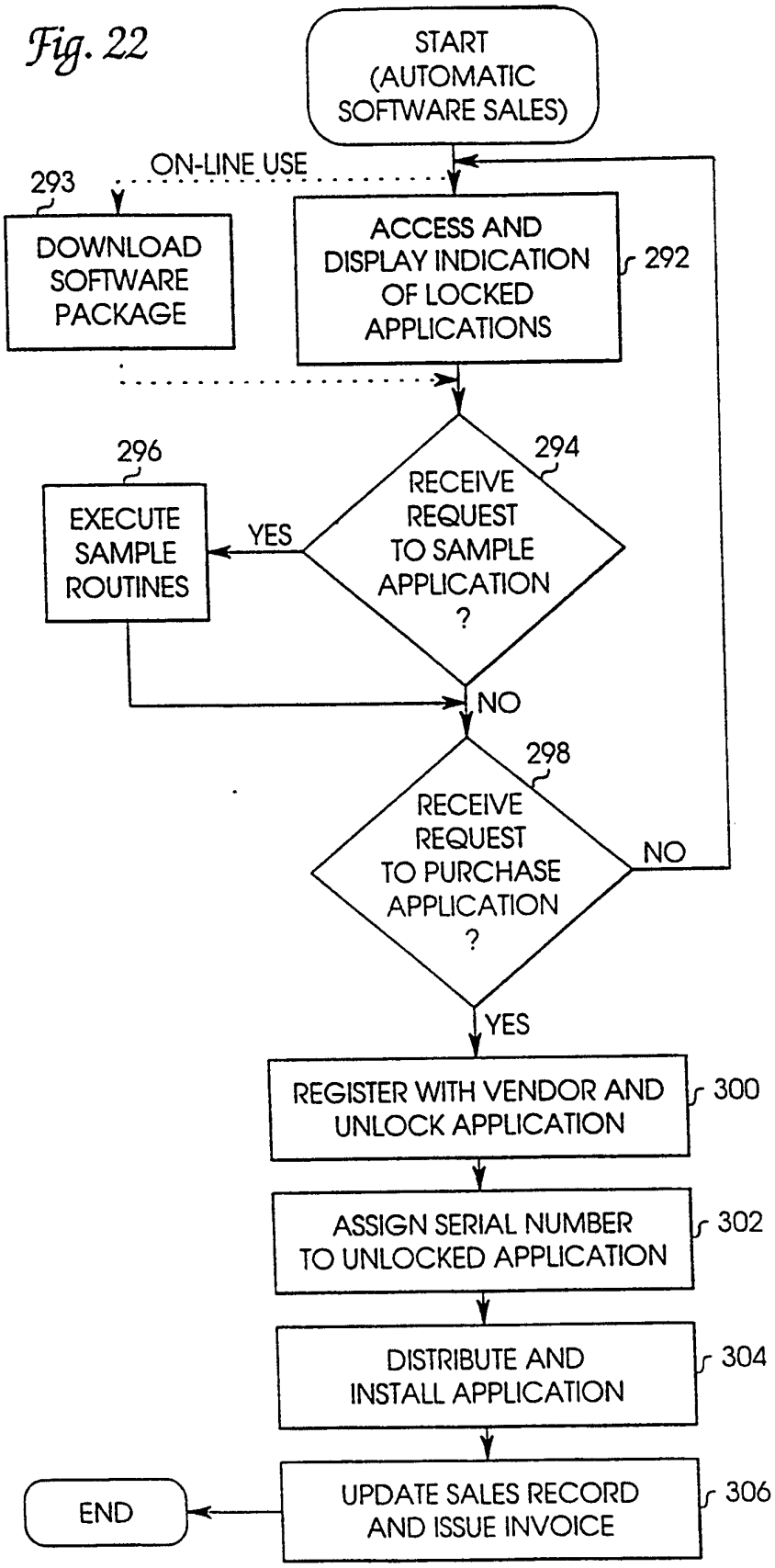

START
(GENERATE SERIAL NUMBER
WITH MARKETING INFORMATION)
RETRIEVE INFORMATION FROM
SAMPLE COUNT FILES
266
CALCULATE MARKETING STATISTICS
AND FORMAT INTO BYTES
268
DETERMINE TOTAL NUMBER
OF SAMPLES USED
270
DETERMINE HOW MANY TIMES
EACH APPLICATION WAS SAMPLED
272
DETERMINE MOST FREQUENTLY
SAMPLED APPLICATION
274
DETERMINE CATEGORY OF
MOST FREQUENTLY
SAMPLED APPLICATION
276
DETERMINE VERSION OF
SAMPLED APPLICATIONS
277
DETERMINE OTHER STATISTICS
278
279
CALCULATE
MORE STATISTICS
?
YES
NO
CONVERT BYTES INTO
ASCII SERIAL NUMBER
280
RETURN

Fig. 21

Fig. 22


START (AUTOMATIC SOFTWARE SALES)
ON-LINE USE
293
DOWNLOAD SOFTWARE PACKAGE
ACCESS AND DISPLAY INDICATION OF LOCKED APPLICATIONS
292
296
EXECUTE SAMPLE ROUTINES
YES
294
RECEIVE REQUEST TO SAMPLE APPLICATION ?
NO
298
RECEIVE REQUEST TO PURCHASE APPLICATION ?
NO
YES
REGISTER WITH VENDOR AND UNLOCK APPLICATION
300
ASSIGN SERIAL NUMBER TO UNLOCKED APPLICATION
302
DISTRIBUTE AND INSTALL APPLICATION
304
UPDATE SALES RECORD AND ISSUE INVOICE
306
END

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/09916

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04L 9/00; H04K 1/00; G06F 9/00, 15/00
US CL : 380/4, 25; 395/186, 218, 226, 227, 700; 364/400; 340/825.31
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : Please See Extra Sheet.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS
Search terms: SELF, LAUNCH, PROGRAM, SOFTWARE, KIOSK, VENDING

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	Please See Continuation of Second Sheet.	

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 20 SEPTEMBER 1996	Date of mailing of the international search report 15 OCT 1996
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer for PINCHUS M. LAUFER Telephone No. (703) 306-4177

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/09916

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X ---- Y	US, A, 5,341,429 (STRINGER ET AL.) 23 August 1994, see entire document.	1-3, 5, 7-12, 14-16, 18, 20-25, 27, 28, 30-36, 38-42, 44-45, 47-50, 52-53, 55-58, 60, 61-67, 69, 70-76, 78, 79, 81-83, 85, 87-89, 91,101-103, 105-107, 109-115, 117-119, and 121-125 ------- 6, 19, 29, 37, 46-47, 51, 54-55, 59, 69, 77, 101-103, 105-108, 110-115, 117-120, 122-125
X --- Y	US, A, 4,658,093 (HELLMAN) 14 April 1987, see entire document.	1, 5, 7, 11-14, 18, 20, 24-28, 30, 35, 36, 38, 43 79, 81-83, 85, 87-89, 91 ---- 6, 19, 29, 37
X --- Y	US, A, 4,446,519 (THOMAS) 01 May 1984, see entire document.	1, 2, 11, 14, 15, 24, 27, 33, 39, 41 ---- 6, 19, 29, 37
X	US, A, 4,465,901 (BEST) 14 August 1984, see entire document.	1, 2, 5, 7-9, 14, 15, 18, 20-22, 27, 28, 30-33, 35, 36, 38-41

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/09916

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US, A, 4,654,799 (OGAKI ET AL) 31 March 1987, see entire document.	1, 2, 4, 10-12, 14, 15, 17, 23-25, 27, 33-35, 40-41, 44, 49-50, 52, 57-58, 61-67, 70-76, 101-106, 112-118, and 124 ------ 4, 6, 17, 19, 29, 37, 46, 47, 51, 54, 55, 59, 68, 77, 101-103, 105-108, 110-115, 117-120, 122-125
X --- Y	US, A, 5,327,563 (SINGH) 05 July 1994, see entire document.	1-3, 5, 9, 12, 14-16, 18, 22, 25, 27, 28, 32, 33, 35, 36, 40, 41 ---- 6, 19, 29, 37
X --- Y	US, A, 5,109,413 (COMERFORD ET AL.) 28 April 1992, see entire document.	1-3, 5, 8-9, 11, 14-16, 18, 21-22, 24, 27-28, 31-33, 35-36, 39-41 ---- 6, 19, 29, 37
X	US, A, 5,388,211 (HORNBUCKLE) 07 February 1995, see entire document.	1-3, 5, 9, 11-12, 14-16, 18, 22, 24-25, 27-28, 32-33, 35-36, 40-41
Y, E	US, A, 5,530,865 (OWENS ET AL.) 25 June 1996, see entire document.	92-100
Y, P	US, A, 5,495,411 (ANANDA) 27 February 1996, see entire document.	92-100
Y	US, A, 4,787,050 (SUZUKI) 22 November 1988, see entire document.	4, 17

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/09916

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US, A, 4,740,890 (WILLIAM) 26 April 1988, see entire document.	61-62, 64-66, 68-71, 73-75, 77-78 ----- 6, 19, 29, 37, 46, 54, 108, 120
X --- Y	US, A, 4,796,220 (WOLFE) 03 January 1989, see entire document.	79, 81-83, 85, 87-89, ----- 102, 114
X	US, A, 3,990,710 (HUGHES) 09 November 1976, see entire document.	61-63, 65-67, 70-72, 74-76
X --- Y	US,A, 5,355,302 (MARTIN ET AL) 11 October 1994, see entire document.	61-78 79, 81-83, 85, 87-89, 91 ---- 51, 59 68, 77
X	US, A, 5,166,886 (MOLNAR ET AL) 24 November 1992, see entire document.	44-60, 61-78
X --- Y	US, A, 4,827,508 (SHEAR) 02 May 1989, see entire document.	79, 81-83, 85, 87-89 ---- 47, 55
X	US, A, 5,014,234 (EDWARDS JR.) 07 May 1991, see entire document.	79, 81-83, 85, 87-89, 91
A	US, A, 5,010,571 (KATZNELSON) 23 April 1991.	
A	US, A, 4,490,810 (HON) 25 December 1984	
A, P	US, A, 5,509,070 (SCHULL) 16 April 1996	
A	US, A, 5,237,157 (KAPLAN) 17 August 1993	

INTERNATIONAL SEARCH REPORT

Inte..ational application No.
PCT/US96/09916

B. FIELDS SEARCHED
Minimum documentation searched
Classification System: U.S.

380/4, 25
364/949.81, 969, 969.4, 286.6, 400, 401R, 726, 856
395/186, 187.01, 188.01, 490, 491, 650, 700
340/825.31, 825.34

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING
This ISA found multiple inventions as follows:

Group I. Claims 1-43: Systems and Methods for demonstrating software and obtaining marketing information.

Group II. Claims 44-60: Self launching system and method for demonstrating software.

Group III. Claims 61-78: Self launching systems and methods for distributing software and digital information.

Group IV. Claims 79-91: System and method for storing a code within an operating system .

Group V. Claims 92-100: System and method for preventing unauthorized duplication of an executing software program.

Group VI. Claims 101-126: Computer based system and method for automatic sales of software.

There is no common special technical feature relating the groups.

Groups 1, 2, 3, 4, and 6 are related as subcombinations usable together.

Group 5 is related to groups 1, 2, and 6 as a subcombination not essential to the combination.Groups 5, 3, and 4 are related to each other as subcombinations usable together.