PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/02 — A1

(11) International Publication Number: **WO 99/32978**

(43) International Publication Date: 1 July 1999 (01.07.99)

(21) International Application Number: PCT/US98/26769

(22) International Filing Date: 17 December 1998 (17.12.98)

(30) Priority Data:
08/994,098 19 December 1997 (19.12.97) US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: SAUNTRY, David, M.; 2008 223rd Place N.E., Redmond, WA 98053 (US). MARKLEY, Michael, E.; 3014 273rd Avenue N.E., Redmond, WA 98053 (US). GILBERT, Mark; 2704 Tybee Pass, Mount Pleasant, SC 29464 (US).

(74) Agent: VIKSNINS, Ann, S.; Schwegman, Lundberg, Woessner & Kluth, P.O. Box 2938, Minneapolis, MN 55402 (US).

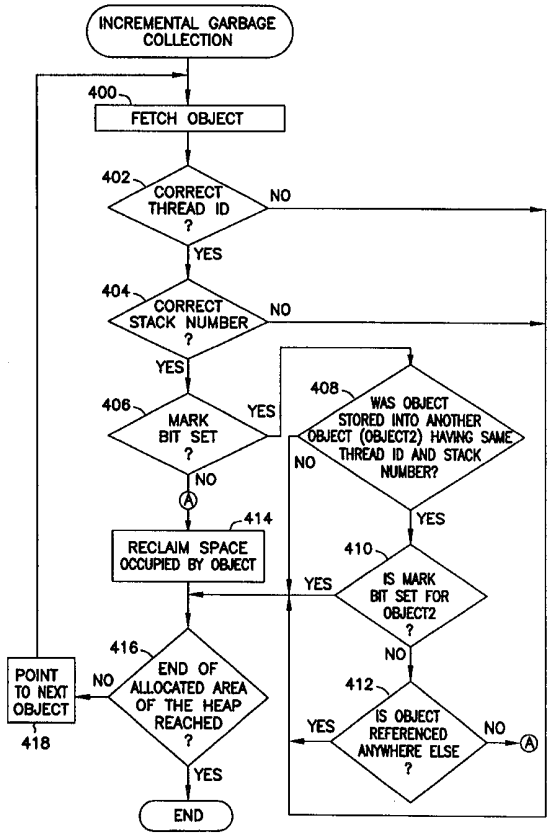
(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

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: INCREMENTAL GARBAGE COLLECTION

(57) Abstract

An incremental garbage collector is disclosed. Upon termination of a function or program, the incremental garbage collector scans the object heap for objects allocated by the function or program that are not referenced outside the function or program that allocated the objects. Memory occupied by such objects is immediately reclaimed without having to wait for the garbage collector.



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.

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

INCREMENTAL GARBAGE COLLECTION

Field of the Invention

The present invention relates generally to computer systems and more specifically to managing the memory portions of such systems.

Background of the Invention

Many computer systems manage information by the use of objects. An object is data that share a particular attribute and occupy a region of random access memory (RAM). Objects are not permitted to overlap in memory. Live objects are those needed in the computational process currently being performed by a computer system. If all objects in a system are live at all times, then there is no concern about memory management. The space assigned to each object at system startup need never be reclaimed. In most systems, however, live objects have varying lifetimes that cannot be predicted in advance. In such systems, some method of recognizing expired or dead objects and evicting them from memory is necessary if memory resources are to be conserved.

Garbage refers to data stored in computer system memory that is no longer being used in the performance of a program, method, function, or subroutine that allocated such data. For purposes of convenience, a program, method, function, or subroutine that allocates data will be referred to simply as a program or function. Garbage collection is the process of locating data in dynamically-allocated memory that is no longer being used and reclaiming the memory to satisfy future memory allocation requests. Garbage collection offers the potential of significant programmer productivity gains because with garbage collection, programmers need not worry about removing data from memory when no longer needed when the program is ended. Hence, garbage collection encourages programmers and system designers to dedicate their efforts to higher-level pursuits, such as the design of fundamental algorithms, user interfaces, and general program functionality. Also, by eliminating many low-level programming concerns, garbage collection reduces the likelihood of programming errors. These benefits of garbage collection combine

together to offer improved software functionality and reliability for lower development costs.

Garbage collection can occur in a number of situations. For example, when the amount of memory remaining in available memory falls below some pre-defined level, garbage collection is performed to regain whatever memory is recoverable. Also, a program or function can force garbage collection by calling the garbage collector. Finally, the garbage collector may run as a background task that searches for objects to be reclaimed. But however they may be invoked, traditional garbage collectors work by periodically halting execution of system programs in order to traverse all of memory in search of memory regions that are no longer in use. Traditional garbage collectors have a number of major shortcomings. One such shortcoming is that, in terms of rates of allocation and deallocation of objects, storage throughput is generally much lower than, for example, stack allocation. Also, the times required to allocate memory are only very loosely bounded — the bounds on allocation times are not tight enough to allow reliable programming of highly-interactive or real-time systems such as mouse tracking, interactive multimedia device control, and virtual reality systems. Finally, in some garbage collectors, the performance penalties associated with memory reads and writes are so high that overall system performance may be unacceptably slow.

These concerns are further exacerbated in systems with inherent limitations and particularities. For example, Microsoft Windows CE is a compact, efficient and scalable operating system that may be used in a wide variety of embedded products, from hand-held PCS to specialized industrial controllers and consumer electronic devices. Many devices that utilize Microsoft Windows CE are intended to have a relatively low amount of random-access memory (RAM), such as one megabyte, to ensure that the devices remain low in cost, compact in size, and efficient in the usage of power. Moreover, devices designed to utilize Microsoft Windows CE typically have less powerful processors than what is typically found on computers designed to run more powerful operating systems like Microsoft Windows NT. For systems with such inherent limitations and particularities, it is

essential to maximize the amount of memory available. There is a need to effectively and efficiently maximize the amount of memory available in such systems.

Summary of the Invention

The present invention is directed to a method for removing as many temporary objects as possible during the execution of a program or function so that a main garbage collector is not triggered.

Certain commands in the program allocate objects, whereas other commands do not. Typically, such objects are allocated from a heap. In one aspect of the present invention, if a program command does allocate an object, information is stored on such object that will facilitate its identification at a later time after the program terminates. Such information comprises, for example, thread identification, stack number, and a mark bit.

The present invention allows for the reclamation of such space without waiting for the main garbage collector. During the execution of a program, if an allocated object is never stored into another object such object can be discarded and the space that it occupied can be reclaimed. In other words, if the main garbage collector is activated, the space occupied by such object would be reclaimed. Hence, one of the advantages of the present invention is the freeing up of memory at a time sooner than when the garbage collector performs its task, because as noted above the present invention allows for the reclaiming of space as soon as the program that allocated such space is terminated. Furthermore, instead of scanning the whole heap, as the garbage collector would, the incremental garbage collector of the present invention allows for the scanning of only the allocated portion of the heap, instead of the entire heap.

Brief Description of the Drawings

Figure 1 is a block diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced.

Figure 2(a) is a block diagram of the system level overview of a technique for removing temporary objects.

Figure 2(b) is a block diagram showing incremental garbage collection module as an embodiment of the present invention.

Figure 3 is a flowchart of the process of allocating objects off the heap.

Figure 4 shows the additional information stored in an object being allocated in the object heap.

Figure 5 shows the steps taken by the incremental garbage collector module as it scans through the heap and reads the additional information stored in the objects.

Detailed Description of the Embodiments

In the following detailed description, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that structural changes may be made without departing from the spirit and scope of the present invention. Therefore, the following detailed description is not to be taken in a limiting sense, and the scope of the present invention is defined by the appended claims.

There are three sections in the detailed description. The first section describes the hardware and operating environment with which embodiments of the invention may be practiced. The second section presents a system level description of one embodiment of the invention. Finally, the third section provides methods for an embodiment of the invention.

Hardware and Operating Environment

Figure 1 is a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced. The description of Figure 1 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer.

Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCS, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

The exemplary hardware and operating environment of Figure 1 for implementing the invention includes a general purpose computing device in the form of a computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that operatively couples various system components include the system memory to the processing unit 21. There may be only one or there may be more than one processing unit 21, such that the processor of computer 20 comprises a single central processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment. The computer 20 may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited.

The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory may also be referred to as simply the memory, and includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within the computer 20, such as during start-up, is stored in ROM 24. The computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a

removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media.

The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for the computer 20. It should be appreciated by those skilled in the art that any type of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24, or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor, computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as remote computer 49. These logical connections are achieved by a communication device coupled to or a part of the computer 20; the invention is not limited to a particular type of communications device. The remote computer 49 may be another computer, a server, a router, a network PC, a client, a peer device or other common network

node, and typically includes many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in Figure 1. The logical connections depicted in Figure 1 include a local-area network (LAN) 51 and a wide-area network (WAN) 52. Such networking environments are commonplace in office networks, enterprise-wide computer networks, intranets and the Internet, which are all types of networks.

When used in a LAN-networking environment, the computer 20 is connected to the local network 51 through a network interface or adapter 53, which is one type of communications device. When used in a WAN-networking environment, the computer 20 typically includes a modem 54, a type of communications device, or any other type of communications device for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It is appreciated that the network connections shown are exemplary and other means of and communications devices for establishing a communications link between the computers may be used.

The hardware and operating environment in conjunction with which embodiments of the invention may be practiced has been described. The computer in conjunction with which embodiments of the invention may be practiced may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited. Such a computer typically includes one or more processing units as its processor, and a computer-readable medium such as a memory. The computer may also include a communications device such as a network adapter or a modem, so that it is able to communicatively couple other computers.

System Level Overview

Figure 2(a) shows a system level overview of a technique for removing temporary objects during the operation of a program or function 204 within the environment of a Java Virtual Machine (JVM) 208. JVM is a native

program running within an operating system to interpret and execute program or function 204. Program or function 204 in this implementation is Java code.

During execution of program or function 204, certain commands can cause the allocation of objects off a heap 212, for example by the referencing by a variable to an object. Heap manager 216 keeps track of an address in heap 212 from where objects can be allocated. As more and more objects get allocated off heap 212, it is possible for a heap manager 216 to request the operating system for additional memory space. When an object is allocated off heap 212, said object also has a reference count associated with it. Whenever a reference goes out of scope, the reference count of the object that the variable referenced is decremented. Any object with a reference count of 0 is a candidate for garbage collection.

Garbage collector 220, when activated, scans heap 212 for objects with reference count of 0 and makes available the memory occupied by the object for future use. The operation of garbage collector 220 is known in the art, but essentially Java performs garbage collection under the following circumstances: (1) whenever it is needed — when the amount of memory remaining in heap 212 falls below some pre-defined level, garbage collection is performed to regain whatever memory is recoverable; (2) whenever garbage collection is requested — garbage collection can be forced in Java by calling *System.gc*, which is the Java garbage collector; or (3) whenever Java executes a background task that searches for objects to be reclaimed.

Figure 2(b) shows incremental garbage collector 270 as an embodiment of the present invention. The incremental garbage collector 270 is an integral part of the JVM. As in the prior art, heap manager 266 keeps track of an address in heap 262 from where objects can be allocated. Once function 254 is activated or executed, the incremental garbage collector 270 saves the address from the heap manager 266 indicating the next memory from which objects can be allocated. Once function 254 exits, the incremental garbage collector 270 saves the address of the last object allocated off heap 262. During execution of function 254 within the environment of JVM 258, when objects are allocated off heap 262,

additional information is stored in the object that will facilitate the identification at a later time of such object by incremental garbage collector module 270.

Moreover, as previously noted, the incremental garbage collector 270 has information concerning the area of the heap allocated by function 254 — essentially the beginning address and ending address of the objects allocated off the heap 262. As soon as program or function 254 terminates or exits, incremental garbage collector module 270, starting from the beginning address to the ending address of the objects allocated off the heap, scans through the heap and reads the additional information stored in the objects. If the incremental garbage collector 270 identifies an object as garbage, it immediately reclaims space occupied by such object and makes it available for use. Hence, the incremental garbage collector 270 reclaims the space occupied by the object without waiting for garbage collector 274 to operate. Additionally, the incremental garbage collector 270 need not scan through the whole heap but only from the beginning address to the ending address of the allocated area of the heap, instead of the entire heap.

Although the garbage collector 220 and 270 were described above in terms of reference counting technique, other techniques for garbage collection are known in the art. These other techniques include, for example, deferred reference counting, mark-sweep collection, mark-compact collection, and copying garbage collection.

Methods of an Embodiment of the Invention

The previous section described on a system level the operation of an embodiment of the invention. This section describes methods performed by a computer of such an embodiment.

In Step 300, in the flowchart of Figure 3, a program or function command or code is executed. Step 302 determines whether the code being executed requires the allocation of objects off the heap. If Step 302 determines that the code does not allocate objects off the heap, control is transferred to Step 306. Otherwise, if Step 302 determines that the code allocates objects off the heap, control is transferred to Step 304.

In Step 304, information about the object is stored in the object being allocated. The information stored in the object is discussed more fully below. Control then proceeds with Step 306, which determines whether the end of the program or function has been reached. If the end of the program or function has not been reached, the next command is fetched in step 308 and control is then transferred to step 300. However, if step 306 determines that the end of the program or function has been reached, incremental garbage collection is performed in step 310.

Figure 4 shows the information added to the object, as discussed above. This information comprises (1) thread identification 315 for the function or program allocating the object, (2) a function or stack number 320, and (3) a mark bit 325, which, if set, indicates that the object is stored outside of the function in any way. The thread identification 315 is retrieved from either the operating system or the JVM, and is known in the art. The function or stack number 320 is a way of indicating which function allocated the object. When a function ("function1") calls another function ("function2"), both will have the same thread identification. It is important that when function2 exits and the incremental garbage collector is activated that objects allocated by function1 are still available to function1.

The mark bit 325 is set, as previously noted, if the object is stored outside of the function in any way. An object that is not stored outside of the function that allocated the object is referred to as local to the function. For example, the mark bit 325 is set when the object is stored into a global variable, returned, or thrown as an exception.

Ordinarily, the mark bit 325 is also set for an object ("object1") if object1 is stored into another object ("object2"). However, if object2 (into which object1 is stored) was allocated in the same function that allocated object1 *and* object2 is not referenced outside of the same function, then the mark bit 325 is not set for either object1 or object2. However, if the mark bit 325 of object2 is later set, then the incremental garbage collector examines all of objects stored in object2. If an object stored in object2 does not have its mark bit set, the incremental garbage collector sets the mark bit at such time. Additionally, the mark bit 325 for the

object being used is also set whenever any of the following commands is executed: AASTORE, ARETURN, ATHROW, PUTFIELD, PUTFIELD_FAST, PUTSTATIC, or PUTSTATIC_FAST. This list is not exhaustive — as a general rule, the mark bit 325 is set for objects used by commands (such as the ones listed) that may or do store the objects outside the function.

Figure 5 shows the steps taken by the incremental garbage collector module as it scans through the heap and reads the additional information stored in the objects. The incremental garbage collector fetches an object off the heap in Step 400, and then in Step 402 determines whether the thread identification stored in the object corresponds to the thread identification of the function or program that called the incremental garbage collector. If the thread identification of the object does not correspond, control is transferred to Step 410. If the thread identification corresponds, control is transferred to Step 404, which determines whether the stack number corresponds to the number assigned by the calling function. If the stack number does not correspond, control is transferred to Step 410. If the stack number corresponds, control is transferred to Step 406, which determines whether the mark bit stored in the object is set. If the mark bit is set, control is transferred to Step 410. If the mark bit is not set, the space occupied by the object is reclaimed in Step 408, and control is transferred to Step 410. Step 410 determines whether the end of the object heap has been reached. If the end of the object heap has not been reached, the incremental garbage collector points to the next object as shown in Step 412 and control is transferred to Step 400. Otherwise, if the end of the object heap has been reached, the incremental garbage collector terminates.

Conclusion

It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reviewing the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

What is claimed is:

1. A method for identifying an object, the method comprising the steps of indicating on the object:

a thread identification;

a stack number; and

whether the object is local.

2. The method of claim 1, wherein the object was allocated by code in a function.

3. A method for reclaiming a memory area occupied by an object in a heap, the object having indicated thereon a thread identification, a stack number, and whether the object is local, the method comprising the steps of:

identifying a reclaimable object; and

indicating on the memory area occupied by the object that the memory area occupied by the object is available for future allocation.

4. The method of claim 3, wherein the object was allocated by code in a function.

5. The method of claim 4, wherein reclaiming the memory occupied by the object is activated by termination of the function that allocated the object.

6. The method of claim 3, wherein the step of identifying a reclaimable object comprises the steps of:

examining the thread identification,

examining the stack number, and

determining whether the object is local.

7. A computerized device comprising:

a virtual machine;

a computer readable medium having stored thereon:

a computer program executed within the environment of the virtual machine; and

an incremental garbage collector executed within the environment of the virtual machine; and

memory having an object heap;

wherein the computer program calls at least one function;

wherein the at least one function allocates at least one object from the memory having an object heap;

wherein upon termination of the at least one function that allocated the at least one object, the incremental garbage collector determines whether the at least one object is local to the at least one function; and

wherein, upon determining that the at least one object is local to the at least one function, the incremental garbage collector reclaims such at least one object.

8. The computerized device of claim 7, wherein the device is compatible with the Windows CE Operating System.

9. The computerized device of claim 7, wherein the incremental garbage collector scans through only an area of the memory having an object heap from which the at least one function allocated the at least one object.

10. A computer readable medium having a computer program stored thereon to cause a suitably equipped computer to perform a method for identifying an object, the method comprising the steps of indicating on the object:

a thread identification;

a stack number; and

whether the object is local.

11. The computer readable medium of claim 10, wherein the object was allocated by code in a function.

12. A computer readable medium having a computer program stored thereon to cause a suitably equipped computer to perform a method for reclaiming a memory area occupied by an object in a heap, the object having indicated thereon a thread identification, a stack number, and whether the object is local, the method comprising the steps of:

identifying a reclaimable object; and

indicating on the memory area occupied by the object that the memory area occupied by the object is available for future allocation.

13. The computer readable medium of claim 12, wherein the object was allocated by code in a function and wherein reclaiming the memory occupied by the object is activated by termination of the function that allocated the object.

14. The computer readable medium of claim 12, wherein the step of identifying a reclaimable object comprises the steps of:

examining the thread identification,

examining the stack number, and

determining whether the object is local.

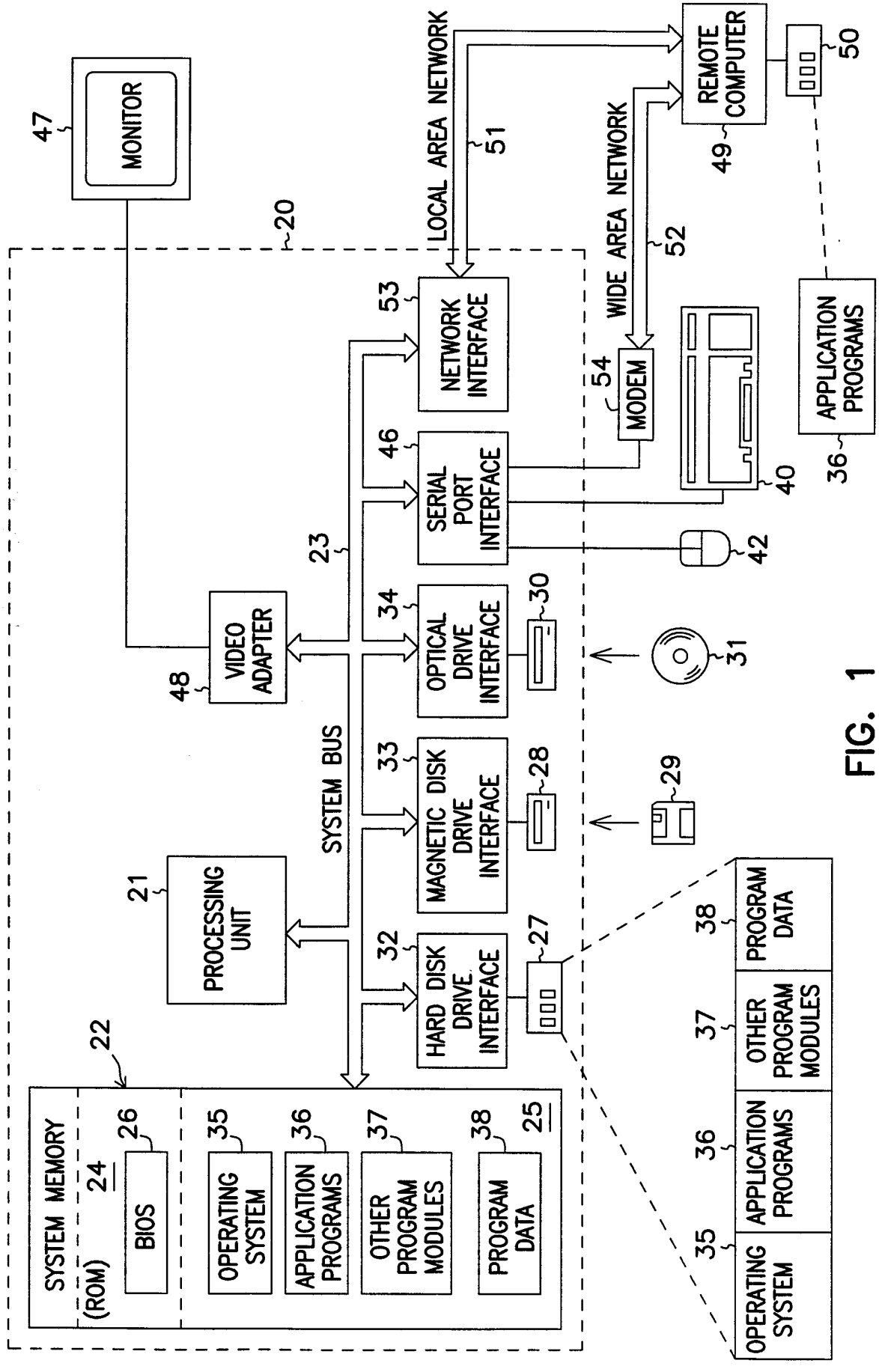

MONITOR
47
20
LOCAL AREA NETWORK
51
REMOTE COMPUTER
49
50
WIDE AREA NETWORK
52
53
NETWORK INTERFACE
54
MODEM
46
SERIAL PORT INTERFACE
40
APPLICATION PROGRAMS
36
42
23
34
OPTICAL DRIVE INTERFACE
30
31
48
VIDEO ADAPTER
SYSTEM BUS
33
MAGNETIC DISK DRIVE INTERFACE
28
29
21
PROCESSING UNIT
32
HARD DISK DRIVE INTERFACE
27
22
SYSTEM MEMORY
(ROM)
24
26
BIOS
35
OPERATING SYSTEM
36
APPLICATION PROGRAMS
37
OTHER PROGRAM MODULES
38
PROGRAM DATA
25
38
PROGRAM DATA
37
OTHER PROGRAM MODULES
36
APPLICATION PROGRAMS
35
OPERATING SYSTEM

FIG. 1

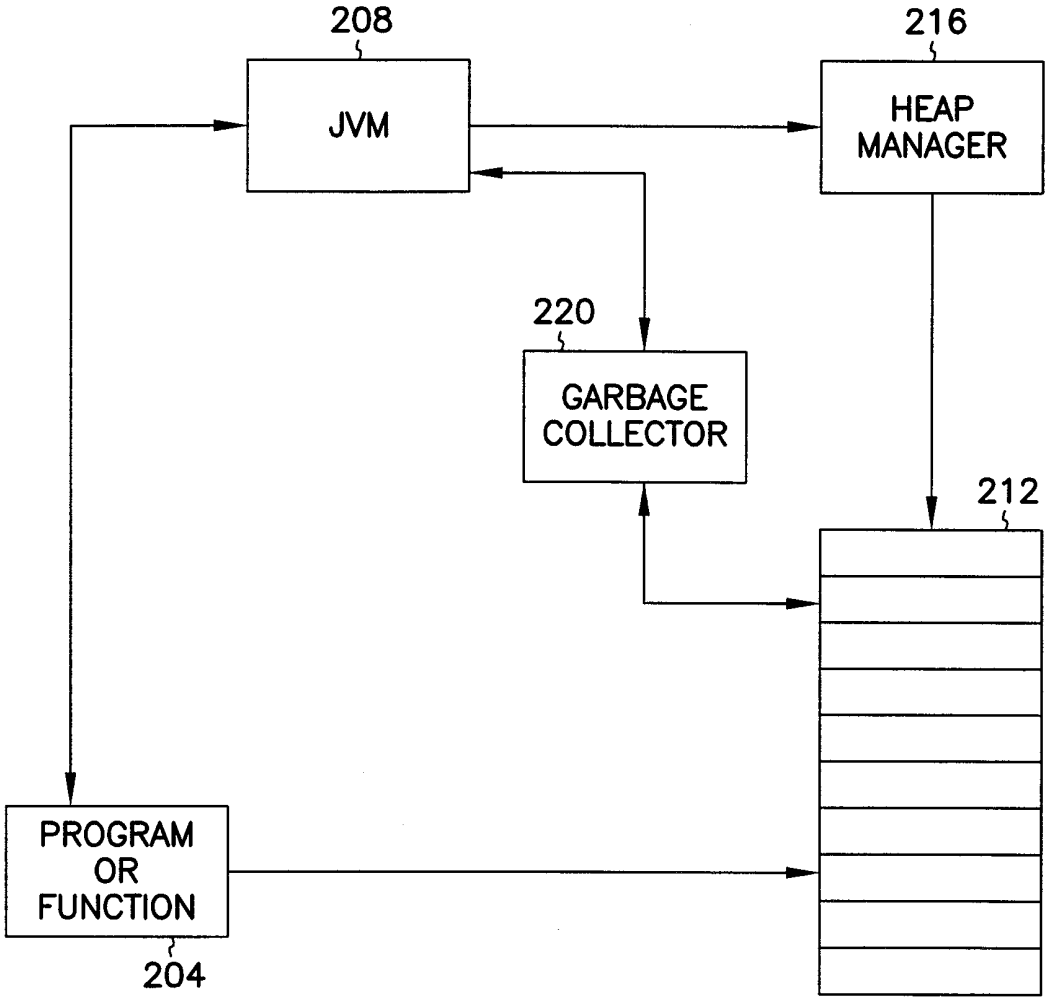

208
JVM
216
HEAP MANAGER
220
GARBAGE COLLECTOR
212
PROGRAM OR FUNCTION
204

FIG. 2(a)

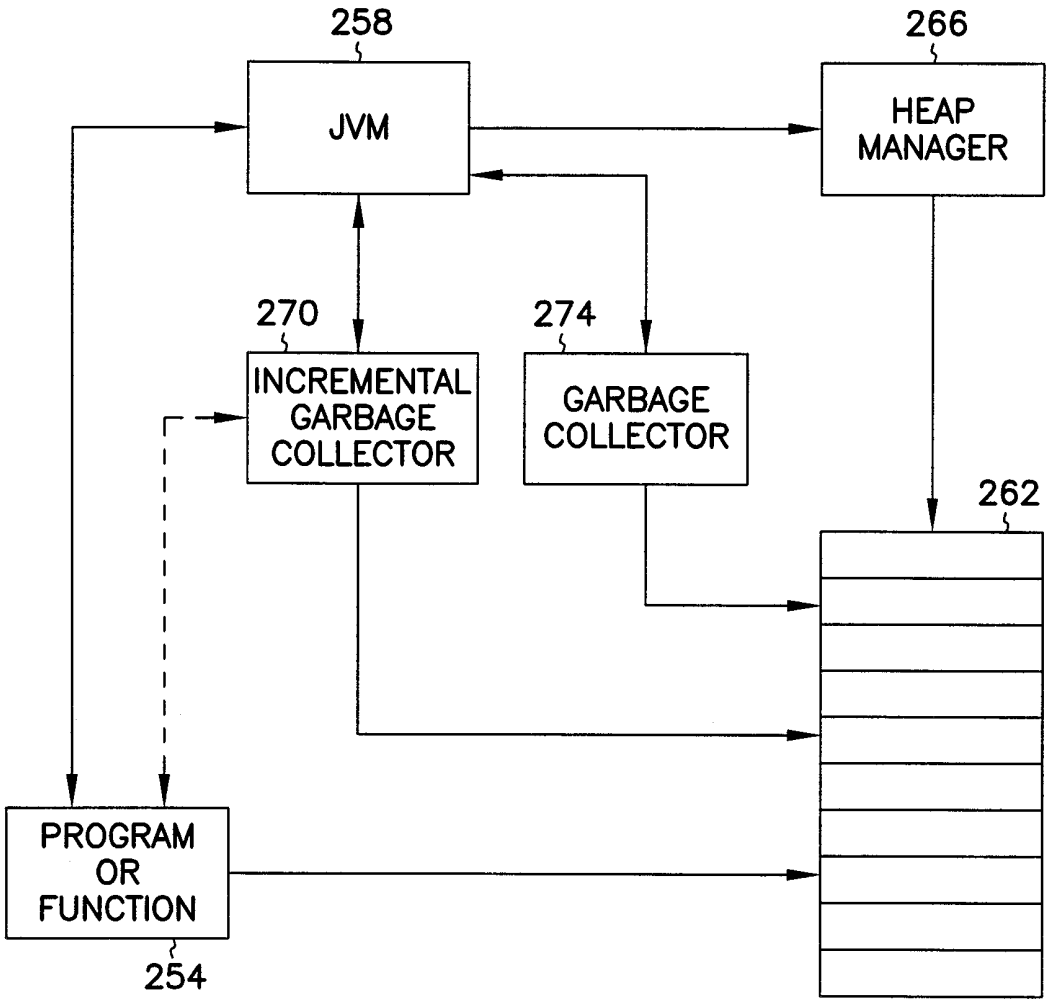

258
JVM
266
HEAP MANAGER
270
INCREMENTAL GARBAGE COLLECTOR
274
GARBAGE COLLECTOR
262
PROGRAM OR FUNCTION
254

FIG. 2(b)

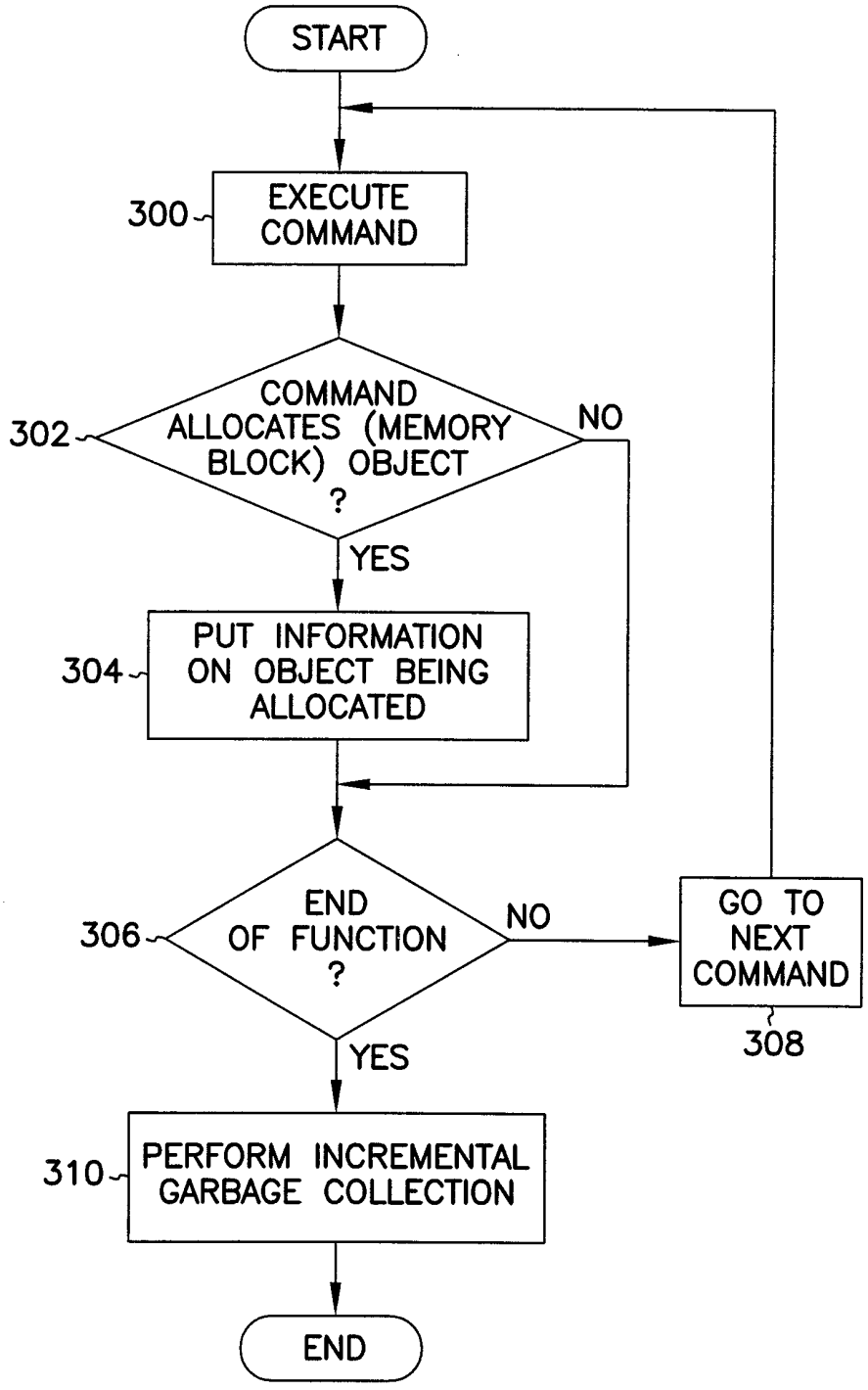

START
300
EXECUTE COMMAND
302
COMMAND ALLOCATES (MEMORY BLOCK) OBJECT ?
NO
YES
304
PUT INFORMATION ON OBJECT BEING ALLOCATED
306
END OF FUNCTION ?
NO
YES
GO TO NEXT COMMAND
308
310
PERFORM INCREMENTAL GARBAGE COLLECTION
END

FIG. 3

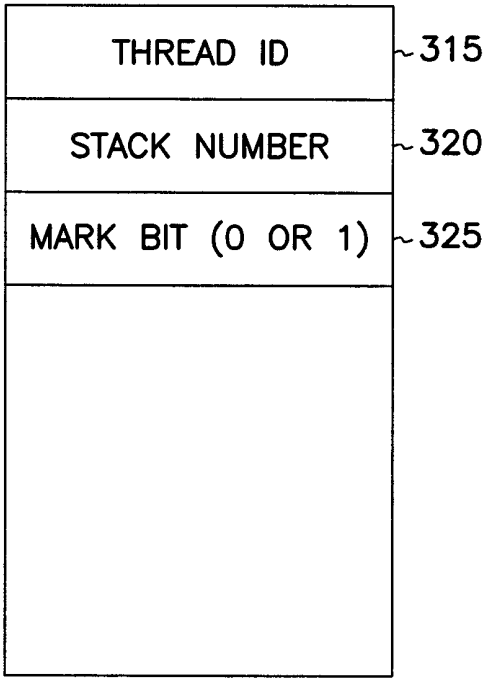


FIG. 4

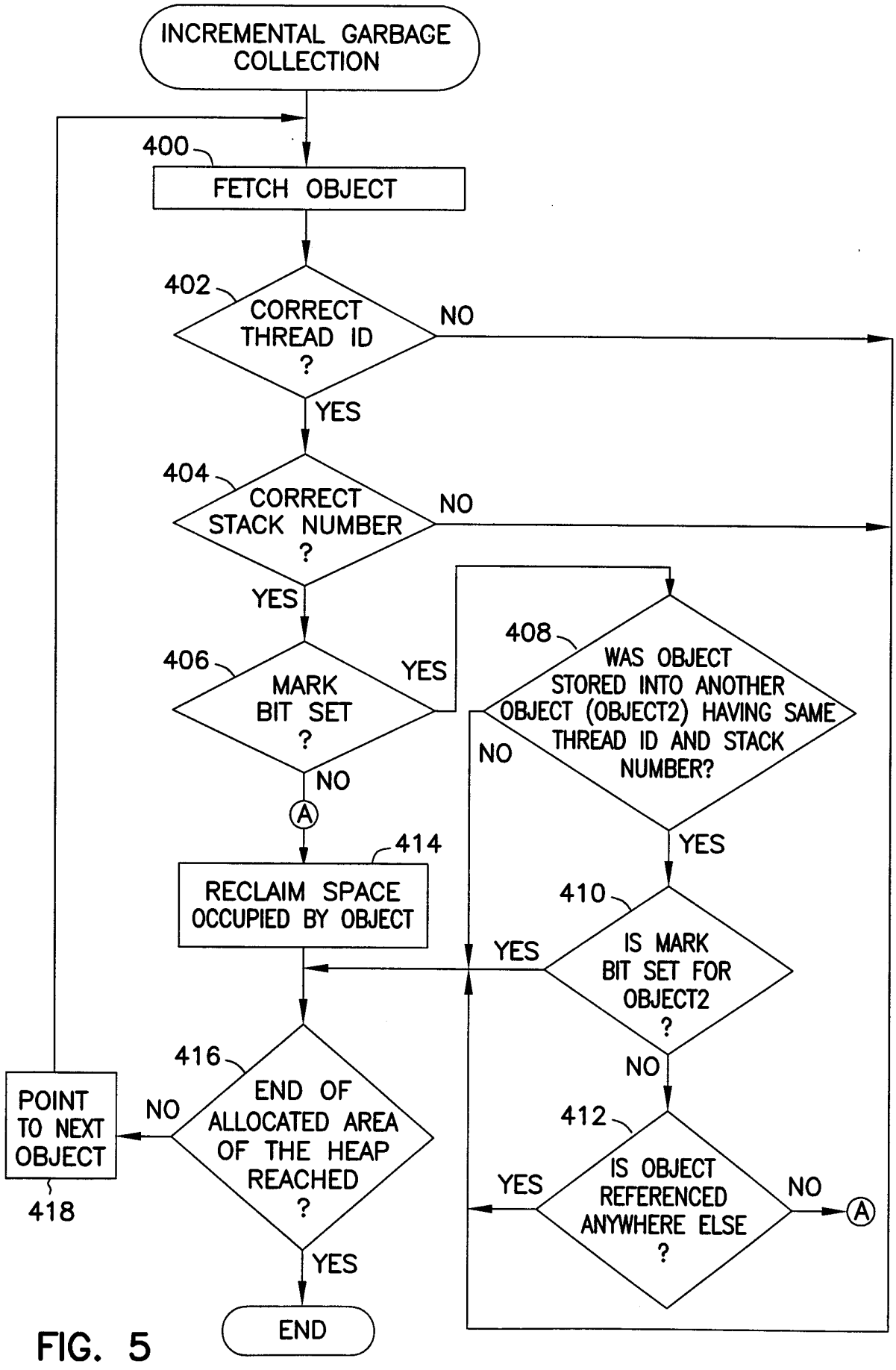

INCREMENTAL GARBAGE COLLECTION
400
FETCH OBJECT
402
CORRECT THREAD ID ?
NO
YES
404
CORRECT STACK NUMBER ?
NO
YES
406
MARK BIT SET ?
YES
NO
A
408
WAS OBJECT STORED INTO ANOTHER OBJECT (OBJECT2) HAVING SAME THREAD ID AND STACK NUMBER?
NO
YES
410
IS MARK BIT SET FOR OBJECT2 ?
YES
NO
412
IS OBJECT REFERENCED ANYWHERE ELSE ?
YES
NO
A
414
RECLAIM SPACE OCCUPIED BY OBJECT
416
END OF ALLOCATED AREA OF THE HEAP REACHED ?
NO
YES
418
POINT TO NEXT OBJECT
END

FIG. 5

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/26769

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F12/02

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)
IPC 6 G06F

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 535 390 A (HILDEBRANDT THOMAS H) 9 July 1996 see column 11, line 1 - column 13, line 62; figure 9 ---	1,3,7
A	SHAPIRO M: "A FAULT-TOLERANT, SCALABLE, LOW-OVERHEAD DISTRIBUTED GARBAGE DETECTION PROTOCOL" 30 September 1991 , PROCEEDINGS OF THE SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, PISA, SEPT. 30 - OCT. 2, 1991, NR. SYMP. 10, PAGE(S) 208 - 217 , INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS XP000266719 see page 211, right-hand column, paragraph 4.4 - page 212, right-hand column, last line --- -/--	1,3,7

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"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)

"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

"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

"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

"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.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 April 1999	19/04/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Ledrut, P

1

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

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/26769

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	KATZBERG J D ET AL: "GARBAGE COLLECTION SOFTWARE INTEGRATED WITH THE SYSTEM SWAPPER IN AVIRTUAL MEMORY SYSTEM" 17 May 1993 , COMMUNICATIONS, COMPUTERS AND POWER IN THE MODERN ENVIRONMENT, SASKATOON, MAY 17 - 18, 1993, PAGE(S) 184 - 191 , INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS XP000419817 see page 187, right-hand column, line 50 - page 188, right-hand column, line 7 -----	1,3,7

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter ional Application No
PCT/US 98/26769

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5535390 A	09-07-1996	NONE	