



- (51) International Patent Classification:
G06F 9/40 (2006.01) *G06F 9/38* (2006.01)
- (21) International Application Number:
PCT/US2013/061889
- (22) International Filing Date:
26 September 2013 (26.09.2013)
- (25) Filing Language: English
- (26) Publication Language: English
- (71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; 11445 Compaq Center Drive West, Houston, Texas 77070 (US).
- (72) Inventor: **CHAKRABARTI, Dhruva**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).
- (74) Agents: **SEGARRA, Roosevelt V.** et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (US).

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

(84) Designated States (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

[Continued on next page]

(54) Title: UNDOING CHANGES MADE BY THREADS

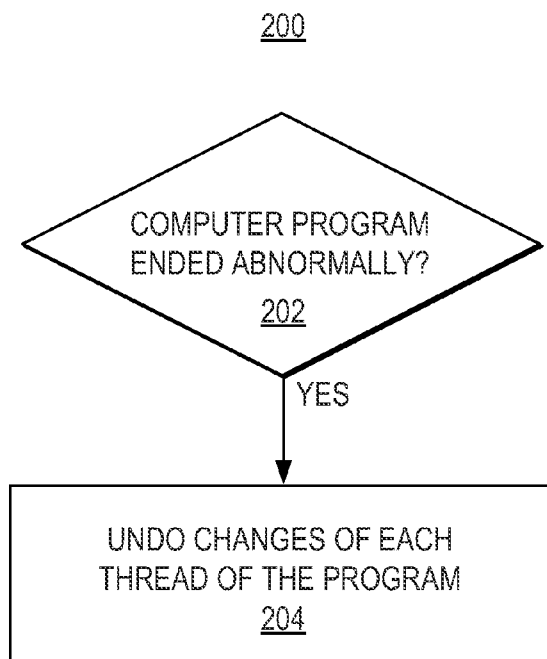


Fig. 2

(57) Abstract: Disclosed herein are a system, non-transitory computer readable medium, and method for recovering from an abnormal failure of a program. Changes made by a plurality of threads of the program are undone in a reverse order in which the changes were made.



TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,
KM, ML, MR, NE, SN, TD, TG).

— *as to applicant's entitlement to apply for and be granted
a patent (Rule 4.17(ii))*

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

Published:

— *with international search report (Art. 21(3))*

UNDOING CHANGES MADE BY THREADS

BACKGROUND

[0001] Software developers heretofore may use multithreading to increase a program's performance. Multithreading is a widespread programming technique that allows multiple sub-programs ("threads") to spawn from the main program. These threads share the main program's resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] FIG. 1 is a block diagram of an example system in accordance with aspects of the present disclosure.

[0003] FIG. 2 is a flow diagram of an example method in accordance with aspects of the present disclosure.

[0004] FIG. 3 is a working example in accordance with aspects of the present disclosure.

[0005] FIG. 4 is a further working example in accordance with aspects of the present disclosure.

DETAILED DESCRIPTION

[0006] As noted above, the threads spawning from a main program may share the main program's resources, but are able to execute independently. Each thread may also execute independently from other threads. Furthermore, a multithreaded program may share and alter the same memory locations. The memory locations may be encoded in the source code as variables. When a given thread alters a memory location shared with other threads, the given thread may lock the memory location. That is, the given thread may obtain exclusive access to the memory location to ensure that other threads do not intervene while it's modifying the memory location. The actions of each thread may be logged so that the log files may be used to undo the activities of each thread in the event of a failure. However, the sequence in which the operations are undone may be complex given that multiple threads may be changing the same memory location. Undoing the

transactions of each thread separately without considering changes made by other threads in between may lead to changes being rolled back out of sequence. In this instance, the program and its shared memory locations may be left in an inconsistent state.

[0007] In view of the foregoing, disclosed herein are a system, non-transitory computer readable medium, and method for recovering from an abnormal failure of a program. In one example, changes made by a plurality of threads of the program may be undone in a reverse order in which the changes were made. In another example, changes to a given memory location made by a first thread of the computer program may be undone while the first thread had exclusive access to the given memory location. In another aspect, it may be determined whether the first thread released exclusive access to the given memory location and it may be determined whether a second thread of the computer program obtained exclusive access to the given memory location after release by the first thread. In yet a further example, changes to a given memory location made by the second thread of the program may be undone while the second thread had exclusive access to the given memory location, if the second thread obtained exclusive access to the given memory location after release by the first thread. In another aspect undo of changes to the given memory location by the first thread may be resumed, if the first thread retained exclusive access to the given memory location after release by the second thread. Thus, the system, non-transitory computer readable medium, and method disclosed herein may rollback changes made by threads of a program while ensuring that the changes are undone in a correct order. The aspects, features and advantages of the present disclosure will be appreciated when considered with reference to the following description of examples and accompanying figures. The following description does not limit the application; rather, the scope of the disclosure is defined by the appended claims and equivalents.

[0008] FIG. 1 presents a schematic diagram of an illustrative computer apparatus 100 depicting various components in accordance with aspects of the present disclosure. The computer apparatus 100 may include all the components normally used in connection with a computer. For example, it may have a keyboard and mouse and/or various other types of input devices such as pen-inputs, joysticks,

buttons, touch screens, etc., as well as a display, which could include, for instance, a CRT, LCD, plasma screen monitor, TV, projector, etc. Computer apparatus 100 may also comprise a network interface (not shown) to communicate with other devices over a network using conventional protocols (e.g., Ethernet, Wi-Fi, Bluetooth, etc.). The computer apparatus 100 may also contain a processor 110, which may be any number of well known processors, such as processors from Intel® Corporation. In another example, processor 110 may be an application specific integrated circuit ("ASIC"). Non-transitory computer readable medium ("CRM") 112 may store instructions that may be retrieved and executed by processor 110. As will be discussed in more detail below, the instructions may include recovery module 114. Non-transitory CRM 112 may be used by or in connection with any instruction execution system that can fetch or obtain the logic from non-transitory CRM 112 and execute the instructions contained therein.

[0009] Non-transitory computer readable media may comprise any one of many physical media such as, for example, electronic, magnetic, optical, electromagnetic, or semiconductor media. More specific examples of suitable non-transitory computer-readable media include, but are not limited to, a portable magnetic computer diskette such as floppy diskettes or hard drives, a read-only memory ("ROM"), an erasable programmable read-only memory, a portable compact disc or other storage devices that may be coupled to computer apparatus 100 directly or indirectly. Alternatively, non-transitory CRM 112 may be a random access memory ("RAM") device or may be divided into multiple memory segments organized as dual in-line memory modules ("DIMMs"). The non-transitory CRM 112 may also include any combination of one or more of the foregoing and/or other devices as well. While only one processor and one non-transitory CRM are shown in FIG. 1, computer apparatus 100 may actually comprise additional processors and memories that may or may not be stored within the same physical housing or location.

[0010] The instructions residing in non-transitory CRM 112 may comprise any set of instructions to be executed directly (such as machine code) or indirectly (such as scripts) by processor 110. In this regard, the terms "instructions," "scripts," and "applications" may be used interchangeably herein. The computer executable instructions may be stored in any computer language or format, such as in object

code or modules of source code. Furthermore, it is understood that the instructions may be implemented in the form of hardware, software, or a combination of hardware and software and that the examples herein are merely illustrative.

[0011] In one example, computer program 116 may instruct processor 110 to generate log entries that specify changes made to memory locations by a plurality of threads spawning from computer program 116. The log entries may further indicate when each thread obtained and released exclusive access to each memory location. In another example, recovery module 114 may determine whether the computer program has ended abnormally and may undo changes to the memory locations in a reverse order in which each thread changed a given memory location while each thread had exclusive access to the given memory location.

[0012] Working examples of the system, method, and non-transitory computer-readable medium are shown in FIGS. 2-4. In particular, FIG. 2 illustrates a flow diagram of an example method 200 for recovering from a program failure. FIGS. 3-4 each show a working example in accordance with the techniques disclosed herein. The actions shown in FIGS. 3-4 will be discussed below with regard to the flow diagram of FIG. 2.

[0013] Referring to FIG. 2, it may be determined whether a computer ended abnormally, as shown in block 202. Referring now to FIG. 3, a computer program 302 is shown executing two threads, thread 304 and thread 306. In this example, the threads write log entries in log 320. FIG. 3 depicts the example steps executed by each thread. In step 307, thread 304 obtains exclusive access ("lock") to two memory locations represented by variables X and Y. In step 309, thread 304 changes the value of X to 1 and then unlocks variables X and Y in step 310. Then, thread 306 obtains a lock on variables X and Y in step 312 and assigns the value of X to Y in step 313. Thread 306 then unlocks variables X and Y in step 314. At step 315, thread 304 again obtains an exclusive lock on variable X and Y and changes the value of X to 2 in step 316. After thread 304 unlocks variables X and Y in step 317, computer program 302 may crash. When computer program 302 crashes, recovery module 322 may read the log entries in log 320 and begin rolling back changes to variables X and Y and attempt to return the variables to a consistent state.

[0014] Referring back to FIG. 2, changes made by the threads of the program may be undone in a reverse order in which the plurality of threads changed the memory locations, as shown in block 204. Referring now to FIG. 4, recovery module 322 is shown undoing the changes made by computer program 302 in FIG. 3. Recovery module 322 may read the example log entries shown in FIG. 4 in a reverse order and may undo the changes based on an analysis of the log entries. The log entries shown in FIG. 4 may capture intra-thread dependences in reverse execution order. For example, an edge from change log entry 418 to lock log entry 422 is added since thread 304 executed the change operation indicated by log entry 418 immediately after acquiring the lock indicated by log entry 422. Inter-thread SYNC edges between log entry 406 to 410 and 414 to 416 capture inter-thread dependences that arise when one thread synchronizes with another. In one example, a second thread may synchronize with a first thread when the first thread releases a lock that the second thread subsequently acquires. Log entry 402 specifies that a first thread released exclusive access to variables X and Y. In one example, when recovery module 322 encounters an unlock log record, it may determine if a second thread obtained exclusive access to the same variables or memory locations that were unlocked. In this example, there is no indication that a second thread obtained a lock on variables X and Y after log entry 402 was recorded. That is, there is no log entry indicating that another thread obtained a lock on variables X and Y. Therefore, after reading log entry 402, recovery module 322 may move on to log entry 404. In another example, whenever a thread changes a variable or memory location, the log entry associated with the change (*i.e.*, the change log entry) may indicate the following: the memory location or the variable that was changed and the old value of the variable before the change.

[0015] In the example of FIG. 4, log entry 404 corresponds to step 316 in FIG. 3. Log entry 404 indicates that variable X had a value of 1 before it was changed to 2. Thus, recovery module 322 may undo the change made in step 316 of FIG. 3 by changing variable X back to 1. Log entry 406 indicates that variables X and Y were previously locked. In one aspect, recovery module 322 may ignore any log entry that indicates a lock. Log entry 416 indicates another unlock of variables X and Y. As noted above, recovery module 322 may check whether a second thread obtained a

lock on the variables, when it encounters an unlock log entry. Here, a second thread does obtain a lock on variables X and Y after log entry 416 was recorded, as indicated by log entry 414. At this point, if the program crashes because of a hardware or software failure, the recovery module 322 may begin to undo some of the changes made by the threads. Log entry 412 corresponds to step 313 of FIG. 3. Thus, in this example, recovery module 322 may rollback the execution of step 313 in FIG. 3 using the corresponding log entry 412. Log entry 412 shows that the value of Y before step 313 was 0; accordingly, recovery module 322 may assign 0 back to variable Y. Log entry 410 indicates that the variables were unlocked again and recovery module may determine whether any other thread obtained a lock on the variables. Here, thread 304 did retain a lock on the variables as indicated by log entry 422. Recovery module 322 may then read log entry 418, which corresponds to step 309 in FIG. 3. Log entry 418 may cause recovery module 322 to roll the value of X back to 0.

[0016] As noted above, the instructions for carrying out the foregoing techniques may comprise any set of instructions to be executed directly or indirectly by at least one processor. In one aspect, given a log entry *e*, a function *prev(e)* may return the log entry that was generated before log entry *e*. For example, applying *prev(e)* to log entry 402 in FIG. 4 may return log entry 404. In a further aspect, given an unlock log entry *e* generated by a first thread, a function *hb_prev(e)*, may return a lock log entry generated by a second thread right after the unlock log entry *e* was generated. For example, applying *hb_prev(e)* to log entry 416 in FIG. 4 may return log entry 414. In yet a further aspect, a function *last_log(t)* may return the next log entry of activity that has yet to be rolled back for a given thread *t*. The following example pseudocode is one illustrative way to utilize the aforementioned example functions:

```
main() {
    for every thread tid
        last_log(tid) = last log created by tid
    for every thread tid
        Recover(tid)
}
```

```
Recover(tid) {
```

```

log_entry = last_log(tid)
while (log_entry) {
    if log type is lock, mark it visited
    else if type is change, apply the undo operation
    else if type is unlock {
        acq_entry = hb_prev(log_entry)
        if (acq_entry is present and acq_entry not already visited) {
            last_log(tid) = prev(log_entry)
            new_tid = thread id of acq_entry
            Recover(new_tid)
        }
    }
    log_entry = prev(log_entry)
}
}

```

[0017] The example pseudocode above is one way to implement the working examples shown in FIGS. 3-4. The pseudocode above starts at an arbitrary thread; obtains its last log entry using the *last_log()* function; and, begins rolling back the activity expressed in the log entries in reverse order. If a lock log entry is encountered, the lock log entry may be marked as visited but no action may be taken. If a change log entry is encountered, the appropriate undo action may be taken (e.g., writing the previous value indicated in the log entry back to the memory location). If an unlock log entry is encountered, it may be determined whether a second thread acquired a lock on the same variables or memory locations using the *hb_prev()* function; If so, a switch may be made to the logs of this second thread and the last log of the second thread that has yet to be rolled back may be obtained using the *last_log()* function; the rollback may begin with the logs created by that second thread. The pseudocode may loop through the log entries until all activities are undone. The *last_log()* entry of a given thread may be tracked and maintained as the pseudocode alternates between threads.

[0018] Advantageously, the foregoing computer apparatus, non-transitory computer readable medium, and method ensure that multithreaded programs are returned to a consistent state after a failure. In this regard, changes to a given

variable or memory location may be undone in a reverse order in which each thread made the change. A recovery module may alternate between prerecorded log records generated by the threads, when it determines that exclusive access to a memory location has changed to another thread. In turn, users may be rest assured that their systems will be returned to a consistent state in the event of a failure.

[0019] Although the disclosure herein has been described with reference to particular examples, it is to be understood that these examples are merely illustrative of the principles of the disclosure. It is therefore to be understood that numerous modifications may be made to the examples and that other arrangements may be devised without departing from the spirit and scope of the disclosure as defined by the appended claims. Furthermore, while particular processes are shown in a specific order in the appended drawings, such processes are not limited to any particular order unless such order is expressly set forth herein. Rather, processes may be performed in a different order or concurrently and steps may be added or omitted.

CLAIMS

1. A system comprising:

a computer program which upon execution generates log entries that specify changes made to memory locations by a plurality of threads spawning from the computer program, the log entries further to indicate when each thread obtained and released exclusive access to each memory location;

a recovery module which upon execution instructs at least one processor to:

determine whether the computer program has ended abnormally; and

undo changes to the memory locations in a reverse order in which the threads changed the memory locations while each thread had exclusive access to a given memory location.

2. The system of claim 1, wherein the recovery module upon execution further instructs at least one processor to undo changes to the given memory location made by a first thread of the computer program while the first thread had exclusive access to the given memory location.

3. The system of claim 2, wherein the recovery module upon execution further instructs at least one processor to:

determine whether the first thread released exclusive access to the given memory location; and

determine whether a second thread of the computer program obtained exclusive access to the given memory location after release by the first thread.

4. The system of claim 3, wherein the recovery module upon execution further instructs at least one processor to undo changes to the given memory location made by the second thread of the program while the second thread had exclusive access to the given memory location, if the second thread obtained exclusive access to the given memory location after release by the first thread.

5. The system of claim 4, wherein the recovery module upon execution further instructs at least one processor to resume undo of changes to the given memory location by the first thread, if the first thread retained exclusive access to the given memory location after release by the second thread.
6. A non-transitory computer readable medium having instructions therein which, if executed, cause at least one processor to:
- determine whether a computer program has ended abnormally;
 - analyze prerecorded log records that specify changes made to memory locations by a plurality of threads spawning from the computer program and which specify when each thread had exclusive access to each memory location;
 - and
 - undo changes to the memory locations in accordance with an analysis of the log records such that the changes are undone in a reverse order in which the plurality of threads changed the memory locations.
7. The non-transitory computer readable medium of claim 6, wherein the instructions therein upon execution further instructs at least one processor to undo changes to a given memory location made by a first thread of a program while the first thread had exclusive access to the given memory location.
8. The non-transitory computer readable medium of claim 7, wherein the instructions therein upon execution further instructs at least one processor to:
- determine whether the first thread released exclusive access to the given memory location; and
 - determine whether a second thread of the program obtained exclusive access to the given memory location after release by the first thread.
9. The non-transitory computer readable medium of claim 8, wherein the instructions therein upon execution further instructs at least one processor to

undo changes to the given memory location made by the second thread of the program while the second thread had exclusive access to the memory location, if the second thread obtained exclusive access to the given memory location after release by the first thread.

10. The non-transitory computer readable medium of claim 9, wherein the instructions therein upon execution further instructs at least one processor to resume undo of changes to the given memory location by the first thread, if the first thread retained exclusive access to the given memory location after release by the second thread.

11. A method comprising
determining, using at least one processor, whether a computer program has ended abnormally;

analyzing, using at least one processor, log files generated by a plurality of threads that spawned from the computer program, the log files specifying changes made to variables by each thread and when each thread had exclusive access to each variable; and

undoing, using at least one processor, changes to the variables such that the changes are undone in a reverse order in which the plurality of threads changed the variables while each thread had exclusive access to a variable.

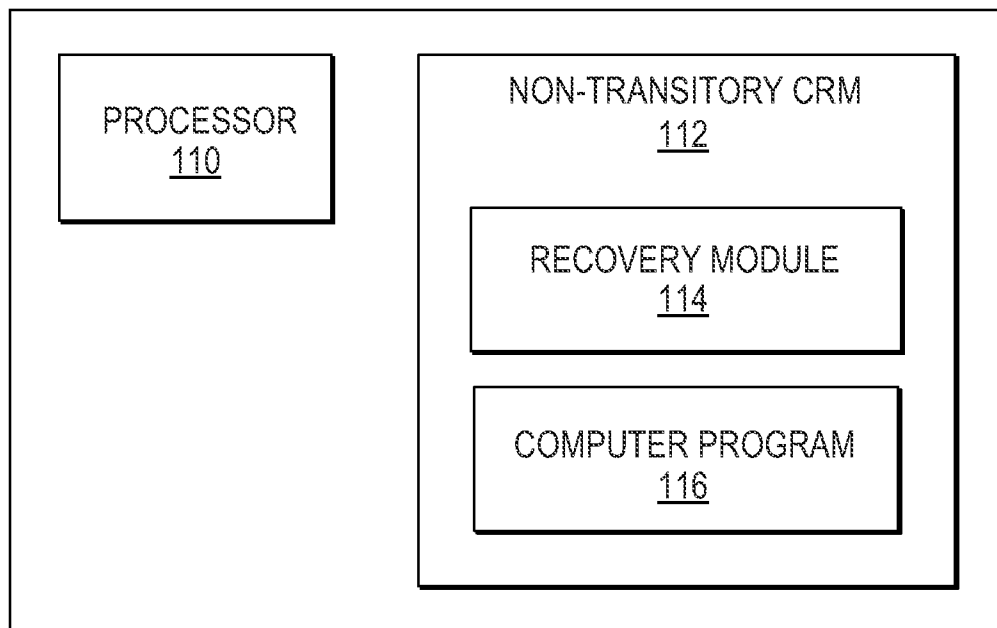
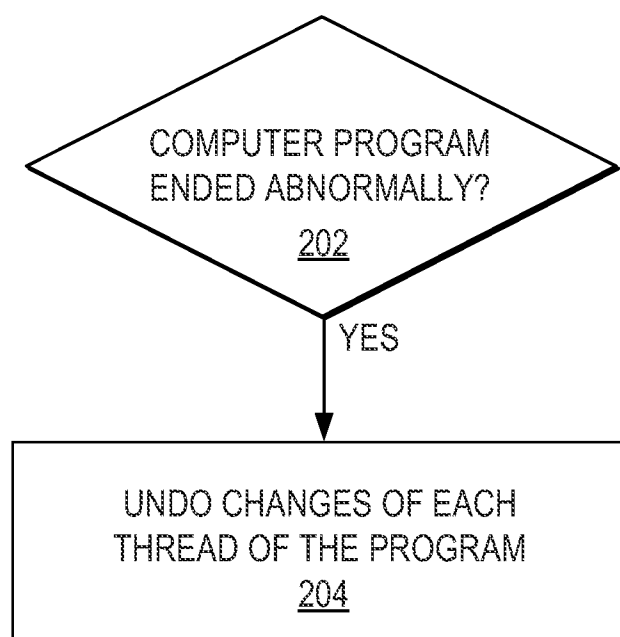
12. The method of claim 11, further comprising undoing, using at least one processor, changes to the variable made by a first thread of a program while the first thread had exclusive access to the variable.

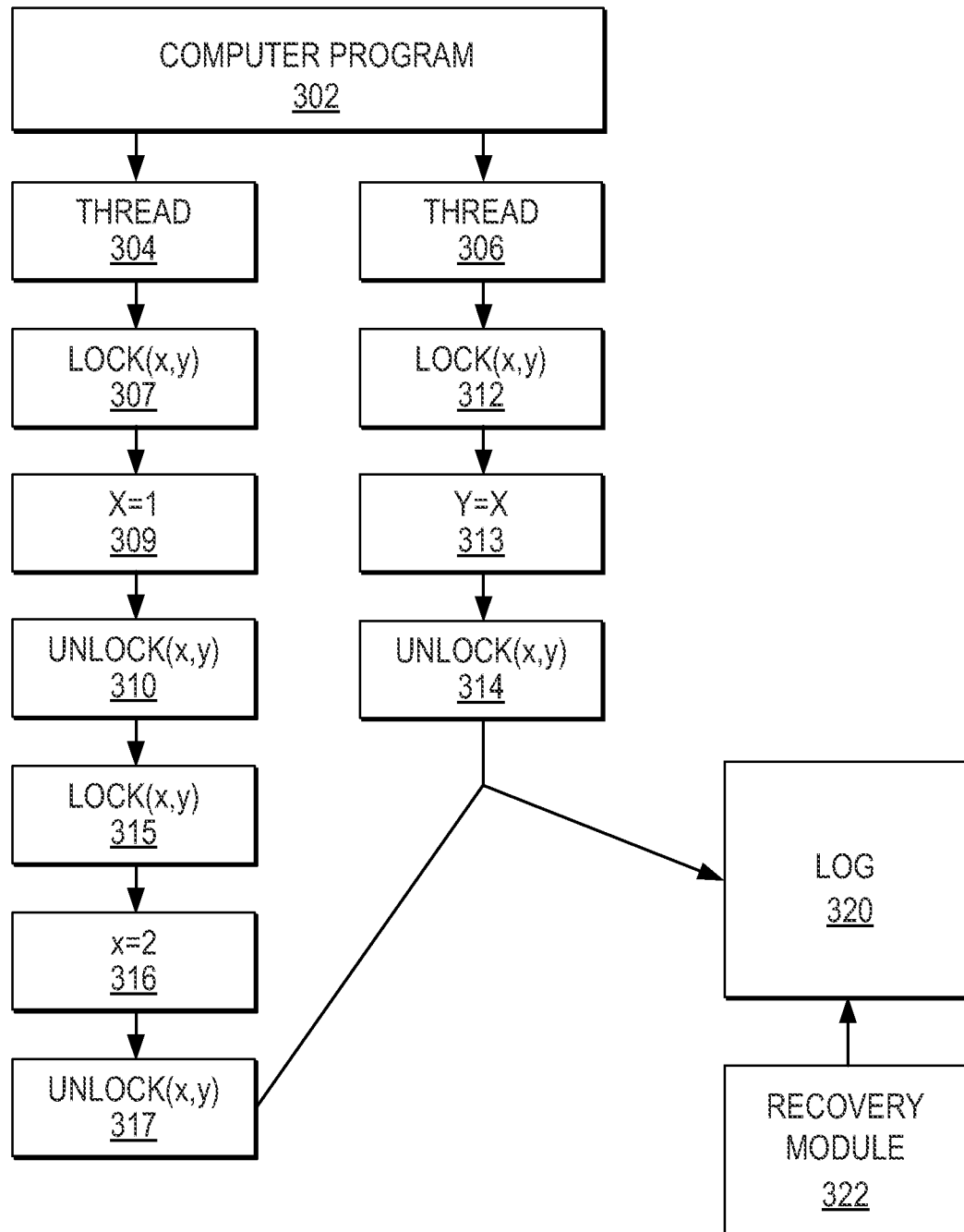
13. The method of claim 12, further comprising:
determining, using at least one processor, whether the first thread released exclusive access to the variable; and
determining, using at least one processor, whether a second thread of the program obtained exclusive access to the variable after release by the first thread.

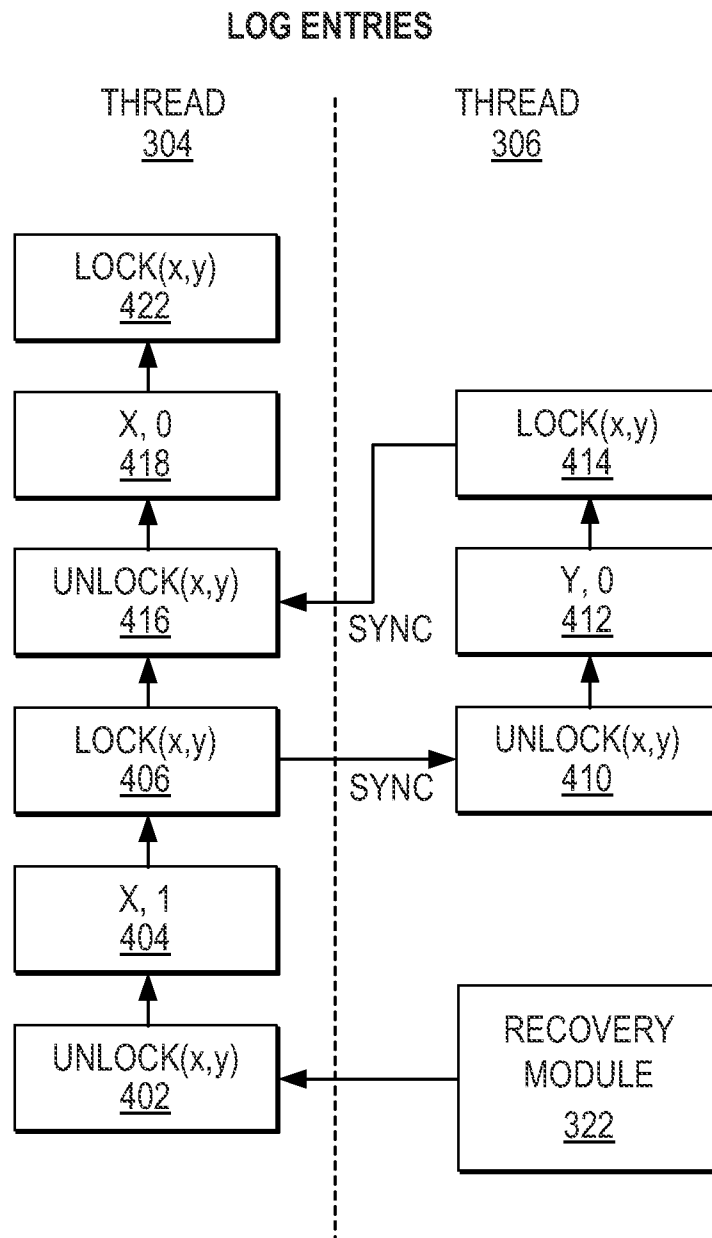
14. The method of claim 13, further comprising undoing, using at least one processor, changes to the variable made by the second thread of the program while the second thread had exclusive access to the variable, if the second thread obtained exclusive access to the variable after release by the first thread.

15. The method of claim 14, further comprising resuming, using at least one processor, to undo changes to the variable by the first thread, if the first thread retained exclusive access to the variable after release by the second thread.

1/3

100**Fig. 1**200**Fig. 2**

**Fig. 3**

**Fig. 4**

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2013/061889**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/40(2006.01)i, G06F 9/38(2006.01)i**

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)

G06F 9/40; G06F 9/44; G06F 12/00; G06F 11/00; G06F 11/36; G06F 11/16; G06F 9/38

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: abnormal failure, recovery, sub-program

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2004-0040019 A1 (CHANG, JIN HA) 26 February 2004 See the abstract, figures 1-5, pages 1-2 and claim 1.	1-15
A	US 6314532 B1 (DAUDELIN; DOUGLAS STREETER et al.) 06 November 2001 See the abstract, figures 1-4, columns 2-3 and claim 1.	1-15
A	US 05134712 A (YAMAMOTO; YUICHI) 28 July 1992 See the abstract, figures 1-6, columns 1-2 and claim 1.	1-15
A	US 2002-0129042 A1 (ROBERT BRADSHAW et al.) 12 September 2002 See the abstract, figures 1-10, pages 1-2 and claim 1.	1-15



Further documents are listed in the continuation of Box C.



See patent family 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 application or patent 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

16 April 2014 (16.04.2014)

Date of mailing of the international search report

16 April 2014 (16.04.2014)

Name and mailing address of the ISA/KR

International Application Division
Korean Intellectual Property Office
189 Cheongsu-ro, Seo-gu, Daejeon Metropolitan City, 302-701,
Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

KIM, Sang Taek

Telephone No. +82-42-481-8211



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2013/061889

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2004-0040019 A1	26/02/2004	CN 1244860 C0 CN 1487411 A CN 1487411 C0 KR 10-0744873 B1 KR 10-2004-0017081 A US 7171658 B2	08/03/2006 07/04/2004 08/03/2006 01/08/2007 26/02/2004 30/01/2007
US 6314532 B1	06/11/2001	None	
US 05134712 A	28/07/1992	EP 0319034 A3 EP 0319034 B1 JP 01-147727 A	06/03/1991 29/05/1996 09/06/1989
US 2002-0129042 A1	12/09/2002	AU 5094201 A CA 2402933 A1 CA 2402933 C CA 2403044 A1 EP 1266289 A1 EP 1266292 A1 EP 1266292 A4 JP 2003-528391 A JP 2003-528391 T JP 2003-528392 T US 2001-0037475 A1 US 2001-027457 A1 US 6480944 B2 US 6609184 B2 WO 01-71501 A1 WO 01-71508 A1	03/10/2001 27/09/2001 10/01/2012 27/09/2001 18/12/2002 18/12/2002 11/06/2003 24/09/2003 24/09/2003 24/09/2003 01/11/2001 04/10/2001 12/11/2002 19/08/2003 27/09/2001 27/09/2001