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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 March 2017 (23.03.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/048261 A1

(51) International Patent Classification:
G06F 11/08 (2006.01) *G11C 29/00* (2006.01)

(21) International Application Number:
PCT/US20 15/050732

(22) International Filing Date:
17 September 2015 (17.09.201 5)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DE¬ VELOPMENT LP** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **LILLIBRIDGE, Mark;** 1501 Page Mill Road, Palo Alto, California 94304-1 100 (US). **BYRNE, John L.;** 12610 Park Plaza Drive, Cerritos, California 90703 (US).

(74) Agents: **PAGAR, Preetam B.** et al; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, 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, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU,

[Continued on nextpage]

(54) Title: MEMORY STORE ERROR CHECK

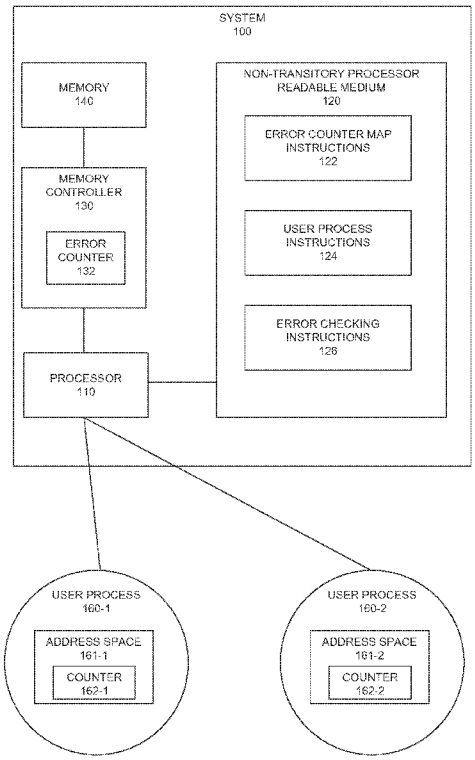

FIG. 1

(57) Abstract: Techniques for memory store error checks are provided. In one aspect, a process running on a processor may execute an instruction to store a first value in memory. The processor may store a plurality of values, including the first value, from a plurality of processes to the memory. A check on a synchronous error notification path may be performed to determine whether an error in storing at least one of the plurality of values occurred.

WO 2017/048261 A1

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

Declarations under Rule 4.17:

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

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

Published:

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

MEMORY STORE ERROR CHECK

BACKGROUND

[0001] Computing systems today may include block-based persistent storage devices. For example, systems may include disk-based or flash-based storage devices. Generally, writing data to such devices involves the use of the computer's Input / Output (i/O) system. Under some circumstances, there may be errors when writing data to a persistent block device using the I/O system.

[0002] Systems may assume that errors in the I/O system may occur and software running on those systems may be designed to receive indications of an error in writing data through the I/O system. The software may then take immediate corrective action.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] FIG. 1 is an example of a system that may utilize the memory store error check techniques described herein.

[0004] FIG. 2 is an example of a high level flow diagram for a process to check for write errors.

[0005] FIG. 3 is another example of a high level flow diagram for a process to check for write errors.

[0006] FIG. 4 is an example of a high live level flow diagram for providing the ability to check for write errors.

[0007] FIG. 5 is another example of a high level flow diagram for providing the ability to check of write errors.

DETAILED DESCRIPTION

[0008] Computing system architectures are being developed in which persistent block based storage is being partially or completely replaced with byte-addressable non-volatile random access memory (NVRAM). Some examples of such memory may include memristor memories, phase change RAM (PCRAM), spin torque transfer (STT) RAM, and others. Other systems may still make use of volatile memory, such as static RAM (SRAM) or dynamic RAM (DRAM), as well as other types of volatile memory. The memory store error check techniques described herein are applicable to both volatile and non-volatile memories. The remainder of the disclosure will refer to memory generally, but it should be understood that memory refers to both volatile and non-volatile memory, and is not limited to any particular type of memory.

[0009] Just as with I/O errors, errors may also occur when writing to memory. However, the current handling of such errors can be problematic. For example, in one write error handling technique, when an error in writing a value to a memory location occurs, a special value, often referred to as poison, is written to the memory location instead. Upon a subsequent read of that memory location, the poison value is detected and corrective action may be taken (if possible).

[0010] In another example of a memory write error handling technique, a process may cause a processor to execute an instruction to store a value in memory. The processor may then issue the store command to a memory controller to actually execute the storage of the value in memory. The process may then continue execution, even though the memory controller may not yet have written the value to the memory. if an error in writing occurs, the process may be notified via some type of interrupt (e.g., a signal generated by the operating system (OS)). However, the timing of the signal may be asynchronous, such that the process does not know when to anticipate the receipt of such a signal. Because the process does not know if or when such a signal may be received (e.g., the error free case), the process cannot be designed to efficiently take corrective action in those cases where the signal is received.

[0011] The error handling techniques described above exhibit problems in that there may be a delay from when an error occurs to when the process is made aware of the error. Because of the delay, the process may not be able to take corrective action. For example, in the case where poison is used, the process is not made aware of the error until the value written is actually read. In the case of a notification via an interrupt, the process may have already moved on to different tasks. In both cases, the process may not be able to take appropriate corrective action because of the delay in notification.

[0012] The techniques described herein provide for memory store error checks that notify a process of an error in a timely fashion, such that the process may take corrective action. A process may first read a current value of an error count that is being maintained by a memory controller. The process may then write a value to the memory. The process may then again check the value of the error counter. A difference in value of the error count indicates an error occurred in writing to the memory.

[0013] The memory controller may be used by all processes running on the computing system. Although the error counter allows the process to determine that an error occurred, this does not mean that the error that occurred was associated with the process. When the process determines an error has occurred, the process may then wait to see if the OS signals the process that the error may have been caused by the process's attempt to write to memory. If so, the process may take the appropriate corrective action. If the error was not associated with the process, the process can continue operation.

[0014] Appropriate error correction may be process dependent. For example, corrective action may include attempting to write the data again. As another example, a process may mirror data between two sections of memory. An error in writing to one of the sections may cause the process to declare that particular mirror invalid, and cause the mirror to be rebuilt elsewhere in memory. The techniques described herein are not limited to any particular type of error correction. It should be understood that the techniques described herein provide the ability for a process to be notified of an error at a time when the

processor can take appropriate corrective action and are not limited to any-particular type of corrective action.

[0015] FIG. 1 is an example of a system that may utilize the memory store error check techniques described herein. System 100 may be any type computing system and is not limited to a particular form. Example computing systems may include servers, laptops, desktops, mobile devices, or any suitable system. In general, system 100 may include a processor 110, a memory controller 130, a memory 140, and a non-transitory processor readable medium 120.

[0016] The processor 110 may be any device capable of executing instructions, such as a central processing unit (CPU), graphics processing unit (GPU), application specific integrated circuit (ASIC), or any other suitable device. The techniques describe herein are not limited to any particular type of processor. The processor may be coupled to a non-transitory processor readable medium 120. The medium may contain thereon a set of instructions, which when executed by the processor may cause the processor to implement the techniques described herein. Although only a single processor is shown, it should be understood that system 100 may contain multiple processors 110.

[0017] For example, the medium 120 may include error counter map instructions 122. The error counter map instructions may allow a processor to map an error counter into a processes' address space, such that the process may access the value of the error counter. Operation of the error counter is described in further detail below. The medium may also include user process instructions 124. The user process instructions may be instructions executed by the processor to implement a user process. A user process may be a user program or a thread of a user program. In general, a user process is a program running on a system 100 to execute some piece of desired functionality.

[0018] Medium 120 may also include error checking instructions 126. Error checking instructions are described in more detail below and with respect to the flow diagrams depicted in FIGS. 5 and 6. Error checking instructions may allow a process to write a value to a memory and determine if an error occurred in writing a value to memory (not necessarily the value being written by the

process). If an error occurred, the error checking instructions may further be used to determine the process / processes impacted by the error, and those processes may be notified of the error.

[0019] System 100 may also include a memory controller 130. Although depicted as a separate element from processor 110, in some implementations, the memory controller may be integrated within the processor. The memory controller may be responsible for reading values from and writing values to memory 140. The processor 110 may command the memory controller to read data from or write data to a particular location (i.e. address) in the memory. The memory controller may be responsible for interfacing with the physical memory to execute the processors commands for reading / writing to the memory.

[0020] The memory controller may also include an error counter 132. The error counter may be used to count the number of errors that have occurred writing to the memory 140. In other words, upon detection of an error in writing to memory, the error counter may be altered (e.g., incremented or decremented). As will be described later, the error counter may be examined to determine whether an error writing values to memory has occurred. In some implementations, the error counter may be implemented as a hardware register.

[0021] The memory 140 may be of any suitable type. Some types of memory, such as memristor or DRAM have been mentioned above. The memory may be packaged in any suitable form. For example, the memory may be included in at least one dual in-line memory module (DIMM). The memory may be in other suitable form factors as well and the techniques described herein are not dependent on any particular memory form factor.

[0022] In operation, the processor 110 may execute user process instructions 124 to create and execute user processes. Two user processes 160-1,2 are shown in FIG. 1, however it should be understood that the techniques described herein are not limited to any number of user processes. Furthermore, the user processes need not all be of the same type. For example, one user process may be a database, while another is a web server. Any other types of processes are also possible. The techniques described herein are independent of the particular types of processes that are running.

[0023] Each user process may have an address space 161-1 ,2. The address space of the process may include data such as the process's executable code, data, text, stack space, and heap. The processor 110, using the error counter map instructions 122 may map the error counter 132 value into the processes address space 162-1 ,2. For example, the error counter map instructions may be a portion of the operating system that maps the hardware error counter register into the address space to make reading the error count more efficient. In other words, the value of the error counter 132 may be mapped into the process's address space, such that the process executable code may read the value of the counter just as if it were being read from any other value in the process's address space. Explained yet another way, the hardware register maintaining the error counter 132 may be mapped into the user process address space such that the user process can read the error counter while still in user mode (as opposed to kernel mode). As such, reading the error count by the process does not require a system call to enter kernel mode and can be implemented with minimal overhead.

[0024] The process 160-1 may desire to write a value to a location (e.g., address) in memory. The process may first read the error count 162-1 to determine the current value of the error counter. The process may then execute a store instruction in order to cause the processor to store the value in the memory. The processor may send the value to be stored (including the address) to the memory controller to be stored in the memory. Although a store is described in terms of a single instruction, it should be understood that multiple processor instructions may actually be executed to effectuate the store. For example, a first instruction may be used to store a value to a processor cache, then a second instruction may be executed to flush the value from the cache to the memory. For purposes of this disclosure storing a value is meant to encompass the value being stored in memory, regardless of the specific number of processor instructions used.

[0025] The memory controller may receive the store that originated from process 160-1 . However, the memory controller may also be receiving stores from any number of other processes, such as process 160-2. The memory

controller generally does not concern itself with the specific process that has requested values to be stored in memory, as the process generating the store request is generally unimportant to the memory controller. If an error occurs when storing a value, regardless of the source of the request to store the value, the memory controller may alter the error count. For example, in one implementation, the error counter may be incremented.

[0026] After the process 160-1 has executed the store instruction, the process may then read the error count again. The process may then compare the current value of the error count with the value read prior to executing the store instruction. If the values are different, then the memory controller experienced an error in storing a value to the memory. It should be noted that the values being different merely indicates that an error in storing a value occurred in the memory controller. It does not specify the process for which the error occurred. In other words, even though process 160-1 becomes aware of an error by reading the current value of the error count, the process does not know if its own store command is the one that experienced the error (e.g., it could have been a write from process 160-2 that experienced the error).

[0027] The detection of an error via the error counter may be referred to as checking a synchronous error path. Because the process executes the store command and then waits for the command to complete before checking the error count, the process is made aware of a possible error at approximately the same time as the store command was executed, hence the check is substantially synchronous with the store command's execution.

[0028] Once the process has determined that an error has potentially occurred based on a difference in the error count, the process may then move to an asynchronous error notification path. In the asynchronous error notification path, the process may wait for an indication that the memory error was due to the memory store executed by the process, as opposed to an error that occurred from a store performed by a different process.

[0029] Once an error occurs, the error checking instructions may execute on the processor to determine the process or processes that may be impacted by the error. For example, the error checking instructions may be implemented

as part of the OS. The OS may obtain from the memory controller the physical address of the memory that was the cause of the error. The OS may then use this physical address to determine a process that had the physical address causing the error mapped into its address space. The determined process may then receive a signal indicating that the change in error count was related to the write from the process.

[0030] In some cases, a plurality of errors may occur very close in time to one another. For example, if a plurality of write are all directed to a single DIMM, and that DIMM should fail, all of the writes would fail as well. As such, in some cases, the OS may obtain an indication of a failing or failed component, and the OS may signal all process that may have attempted to write to that device.

[0031] In some implementations, the memory controller may batch error notifications to the OS. For example, if an error occurs, the memory controller may include details related to the error (e.g., physical address, failing DIMM, address range) in a data structure and then trigger an interrupt to the processor. While the processor is handling the interrupt, additional errors may be occurring. The memory controller may batch these additional errors until processing of the initial interrupt has completed. The memory controller may then issue a new interrupt with the details of the errors that occurred since the previous interrupt.

[0032] The OS may keep track of the errors that have already been signaled to processes. For example, assume the error count is incremented by one upon each occurrence of an error. Also assume all errors have already been signaled to the processes, such that the error count and error signaled count are equal. When the first error occurs, the error count may be incremented by one. The error signaled count would remain at its current value until the error is actually signaled to the process. Once the error is signaled to the process, the error signaled count may be incremented, thus indicating that once again, all errors have already been signaled to their respective process. As explained below, the error signaled count may be used by a process to determine if a given error is associated with the process.

[0033] In some implementations, once the process becomes aware of a possible error through the synchronous path (e.g., when the value of the error count has changed after a store instruction), the process may enter an asynchronous error notification path. In the asynchronous path, the process may wait to receive a signal from the operating system indicating that the error is associated with the given process. However, it is possible that such a signal may never be received (e.g., the error was caused by a different process). The process may use the error signaled count to determine if the error has already been signaled. For example if the error count was initially x and after the process executes a store the error count was x+1 , the process is aware that an error occurred (although not necessarily generated by that process). The process may then wait in the asynchronous path until either the error signal is received or the error signaled count is x+1 . If the error signal is received, the error was caused by this process. Otherwise, the error signaled count being incremented to x+1 indicates that the proper process has been signaled, thus the instant process need no longer wait for the possibility that it might receive an error signal.

[0034] In some examples, the OS may periodically check the error count in the background looking for changes. When it detects a change, it may enter the asynchronous path and signal one or more processes that an error has occurred. This may allow processes that do not bother to check the error count to still receive notifications that an error may have occurred with one of their writes; however, unlike with the synchronous path, they may receive the notification well after the point that the error can be handled gracefully. They thus may be said to have received the error notification asynchronously, whereas the processes checking the error count receive the error notification synchronously.

[0035] FIG. 2 is an example of a high level flow diagram for a process to check for write errors. In block 210, a process running on a processor may execute an instruction to store a first value in a memory. The processor may store a plurality of values, including the first value, from a plurality of processes to the memory. In other words, many processes may be running on a

processor, each of those processes storing values to a memory. A specific process may be storing a specific value, referred to as the first value.

[0036] In block 220, a check on a synchronous error notification path may be performed to determine whether an error in storing at least one of the plurality of values occurred. In other words, the synchronous check may determine if an error occurred in writing any of the values to the memory. However, it should be understood that the error may not necessarily have occurred from the storing the first value, as it could have also occurred from storing any of the others of the plurality of values.

[0037] FIG. 3 is another example of a high level flow diagram for a process to check for write errors. In block 310, a first error count may be read prior to executing the instruction to store the first value in the memory. In other words, the process may first read the error count to determine what the error count was initially. In block 320, just as in block 210, the process may store a first value to the memory. This first value may be one of many values the processor is storing to the memory.

[0038] In block 330, a second error count may be read after executing the instruction to store the first value in the memory. A difference between the first error count and the second error count indicates an error in storing the at least one of the plurality of values. In other words, if the error counts are different, this means an error occurred in storing a value to memory sometime after the process read the first error count. The error may not necessarily be associated with the storing the first value. Put another way, a difference in the error counts indicates that at least one error has occurred between the reading of the first error count and the reading of the second error count. In some implementations, the error count may be maintained by a device, such as a memory controller. Both the first and second error counts may be read from the same device counter.

[0039] In block 340, an asynchronous error notification path may be entered when it is determined from the check on the synchronous error notification path that an error has occurred. In other words, if the error counts are different, indicating a possible error in storing the first value, the process

may enter an asynchronous error notification path to determine if the error was associated with the process.

[0040] In block 350, an indication may be received when the error is associated with the process. Put another way, the synchronous error notification path informs the process that an error in storing the first value may possibly have occurred, while the asynchronous error notification path definitively notifies the process that the error was associated with storing the first value.

[0041] FIG. 4 is an example of a high live level flow diagram for providing the ability to check for write errors. In block 410, access to an error counter may be provided to a process. As described above, the memory controller may provide an error counter, and access to that error counter may be provided to a process. in block 420, a request for error checking may be received from the process. The request may be generated, at least in part, based on the error counter. In other words the process may request to determine if an error indicated by a change in the error counter could actually have been caused by that process.

[0042] FIG. 5 is another example of a high level flow diagram for providing the ability to check of write errors. In block 510, the error counter may be mapped into the process's address space. As mentioned above, the error counter may exist on a device, such as a memory controller. Reading the device by the processor normally may involve a system call, with corresponding overhead. By mapping the error counter into the process's address space, the process may read the error counter just as if it were reading any other variable within the address space. in other words, the value of the error counter is made available in the process's user space.

[0043] In block 520, just as in block 420, a request for error checking may be received form the process. in block 530, details related to an error may be obtained. As mentioned above, the memory controller may include details related to the error, such as a failing physical address, range of addresses, DIMM, or any other such information related to the specific error that may have occurred. This information may be received from the memory controller. In one

implementation, the memory controller may include the information in a data structure, and then signal the processor with an interrupt to read the data structure.

[0044] In block 540, at least one process the error may be associated with may be determined. As mentioned above, the memory controller may indicate an error has occurred, but may have no idea which process is associated with the store instruction that caused the error. In block 540, the associating may be determined. For example, given a failing physical address, or range of physical addresses, it may be determined which processes currently have those physical addresses mapped into their virtual address space. In the case of a failing DIMM, it may be determined which processes have portions of their address space mapped to addresses stored on the failing DIMM.

[0045] In block 560, at least one of the at least one processes may be notified. In other words, once it is determined which process(es) are associated with the error, one or more of the processes may be notified. As explained above, the process that caused the error may be waiting in the asynchronous notification path for a signal indicating that it was the cause of the error. Once it is determined which process caused the error, such a signal can be sent to the process.

We Claim:

1. A method comprising:

executing, by a process running on a processor, an instruction to store a first value in a memory, wherein the processor may store a plurality of values, including the first value, from a plurality of processes to the memory; and

performing a check on a synchronous error notification path to determine whether an error in storing at least one of the plurality of values occurred.

2. The method of claim 1 wherein performing the check on the synchronous error notification path further comprises:

reading a first error count prior to executing the instruction to store the first value in the memory; and

reading a second error count after executing the instruction to store the first value in the memory, wherein a difference between the first error count and the second error count indicates an error in storing at least one of the plurality of values.

3. The method of claim 2 wherein the first and second error counts are read from the same device counter.

4. The method of claim 2 wherein the difference between the first error count and the second error count indicates that at least one error has occurred between the reading of the first error count and the reading of the second error count.

5. The method of claim 2 further comprising:

entering an asynchronous error notification path when it is determined from the synchronous error notification path that an error has occurred; and

receiving an indication when the error is associated with the process.

6. A non-iransitory processor readable medium containing a set of instructions thereon which when executed by a processor cause the processor to:

provide access to an error counter to a process;

receive a request for error checking from the process, the request generated by the process at least in part based on the error counter.

7. The medium of claim 1 wherein providing access comprises instructions to:

map the error counter into the process's address space.

8. The medium of claim 1 wherein the error counter is changed when an error occurs in writing a value to a memory.

9. The medium of claim 1 further comprising instructions to:

obtain details related to an error; and

determine at least one process the error may be associated with.

10. The medium of claim 4 wherein the details related to the error include at least one physical address in the memory experiencing the error and determining at least one process the error may be associated with further comprises instructions to:

determine at least one process whose virtual address space maps to the at least one physical address; and

notify at least one of the at least one processes.

11. The medium of claim 4 wherein the details related to the error include:

an indication of a failed memory device.

12. The medium of claim 4 wherein the details related to the error include:

a range of addresses in which the error occurred.

13. A system comprising:

a memory;

a memory controller to provide an error counter, the error counter changing when an error occurs in writing a value to the memory; and

a processor to execute instructions to map the error counter into a user process's address space.

14. The system of claim 1 wherein the processor to further execute instructions to determine a process associated with an error in writing a value to the memory, wherein the error caused a change in the error counter.

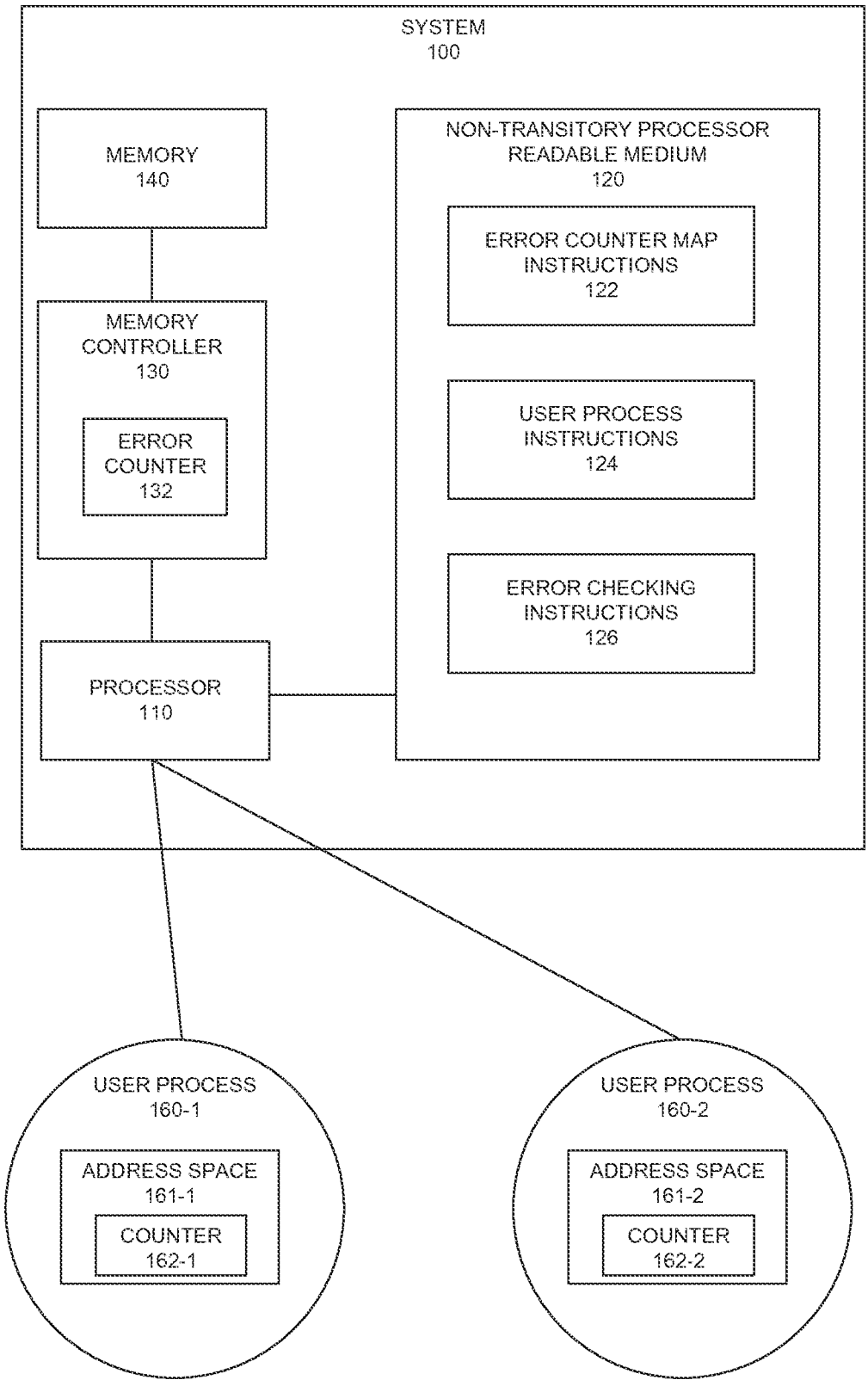
SYSTEM
100
MEMORY
140
MEMORY
CONTROLLER
130
ERROR
COUNTER
132
PROCESSOR
110
NON-TRANSITORY PROCESSOR
READABLE MEDIUM
120
ERROR COUNTER MAP
INSTRUCTIONS
122
USER PROCESS
INSTRUCTIONS
124
ERROR CHECKING
INSTRUCTIONS
126
USER PROCESS
160-1
ADDRESS SPACE
161-1
COUNTER
162-1
USER PROCESS
160-2
ADDRESS SPACE
161-2
COUNTER
162-2

FIG. 1

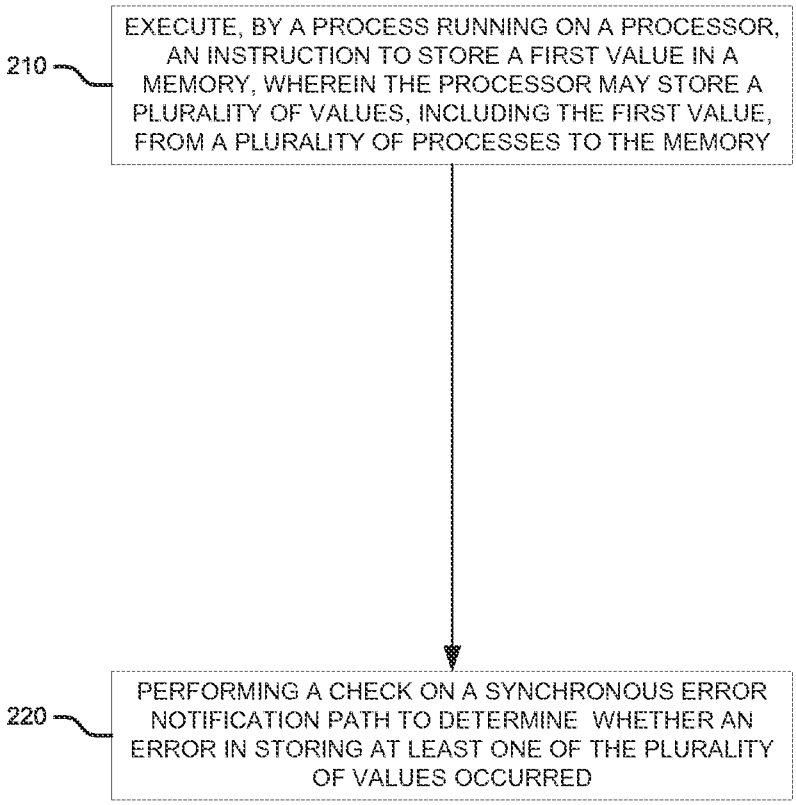
210
EXECUTE, BY A PROCESS RUNNING ON A PROCESSOR, AN INSTRUCTION TO STORE A FIRST VALUE IN A MEMORY, WHEREIN THE PROCESSOR MAY STORE A PLURALITY OF VALUES, INCLUDING THE FIRST VALUE, FROM A PLURALITY OF PROCESSES TO THE MEMORY
220
PERFORMING A CHECK ON A SYNCHRONOUS ERROR NOTIFICATION PATH TO DETERMINE WHETHER AN ERROR IN STORING AT LEAST ONE OF THE PLURALITY OF VALUES OCCURRED

FIG. 2

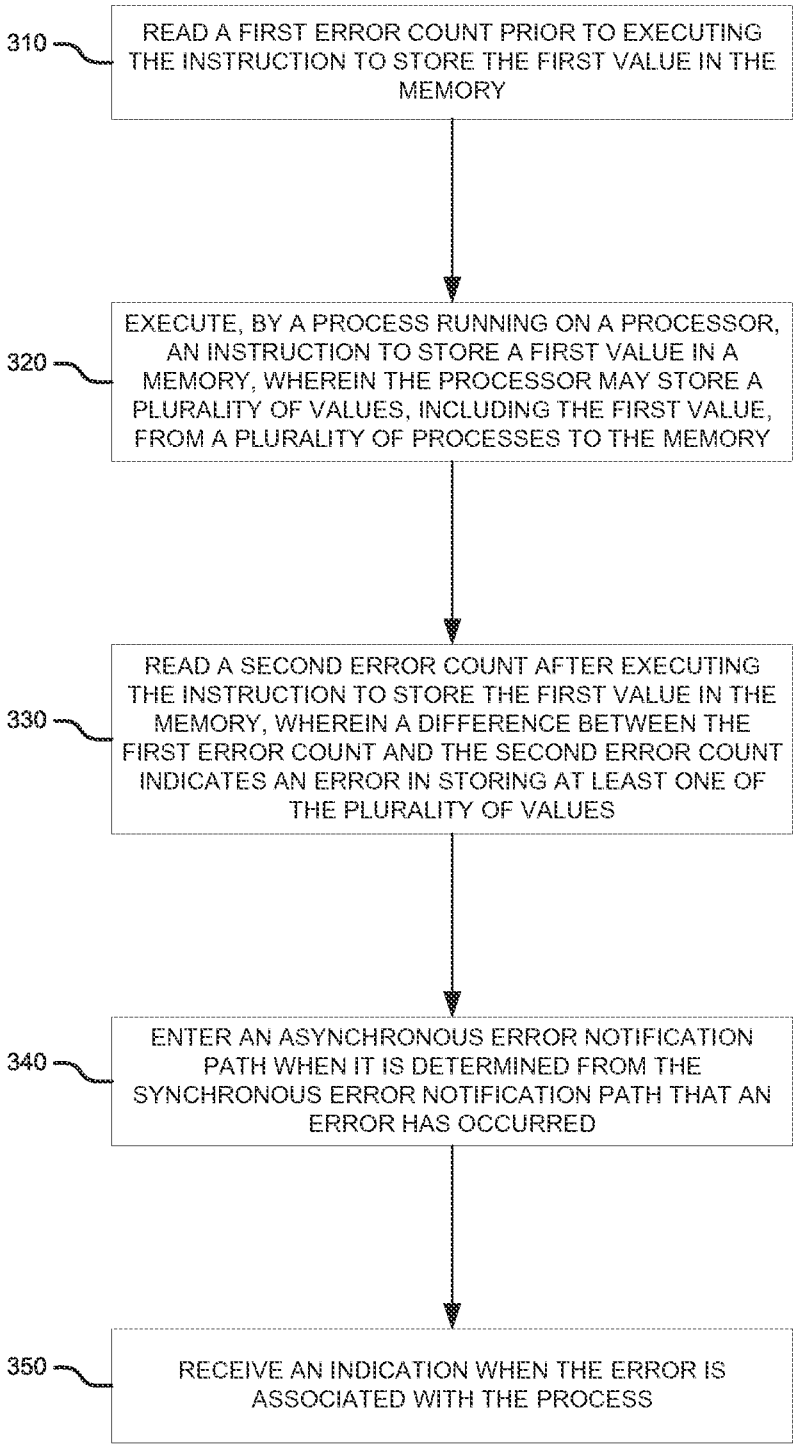
310
READ A FIRST ERROR COUNT PRIOR TO EXECUTING THE INSTRUCTION TO STORE THE FIRST VALUE IN THE MEMORY
320
EXECUTE, BY A PROCESS RUNNING ON A PROCESSOR, AN INSTRUCTION TO STORE A FIRST VALUE IN A MEMORY, WHEREIN THE PROCESSOR MAY STORE A PLURALITY OF VALUES, INCLUDING THE FIRST VALUE, FROM A PLURALITY OF PROCESSES TO THE MEMORY
330
READ A SECOND ERROR COUNT AFTER EXECUTING THE INSTRUCTION TO STORE THE FIRST VALUE IN THE MEMORY, WHEREIN A DIFFERENCE BETWEEN THE FIRST ERROR COUNT AND THE SECOND ERROR COUNT INDICATES AN ERROR IN STORING AT LEAST ONE OF THE PLURALITY OF VALUES
340
ENTER AN ASYNCHRONOUS ERROR NOTIFICATION PATH WHEN IT IS DETERMINED FROM THE SYNCHRONOUS ERROR NOTIFICATION PATH THAT AN ERROR HAS OCCURRED
350
RECEIVE AN INDICATION WHEN THE ERROR IS ASSOCIATED WITH THE PROCESS

FIG. 3

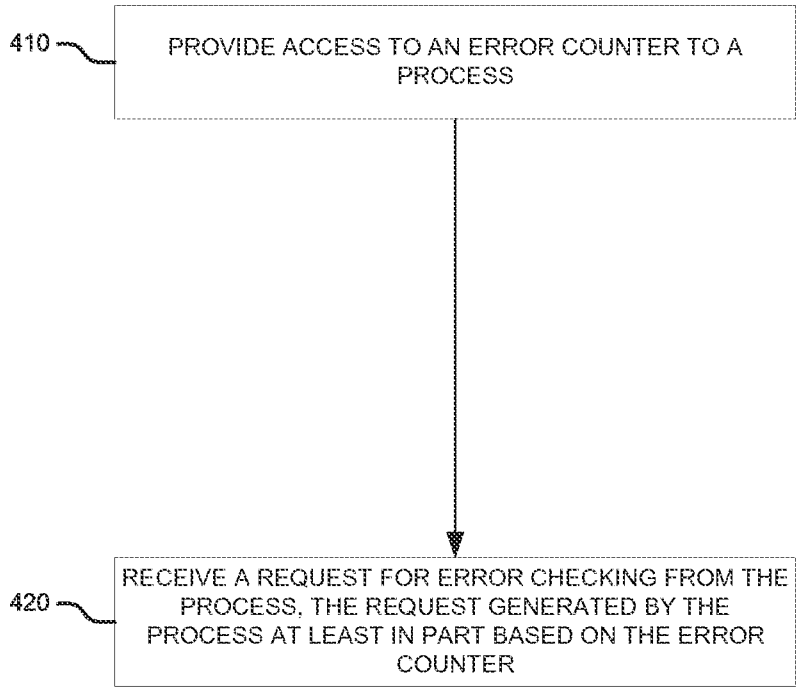
410
PROVIDE ACCESS TO AN ERROR COUNTER TO A PROCESS
420
RECEIVE A REQUEST FOR ERROR CHECKING FROM THE PROCESS, THE REQUEST GENERATED BY THE PROCESS AT LEAST IN PART BASED ON THE ERROR COUNTER

FIG. 4

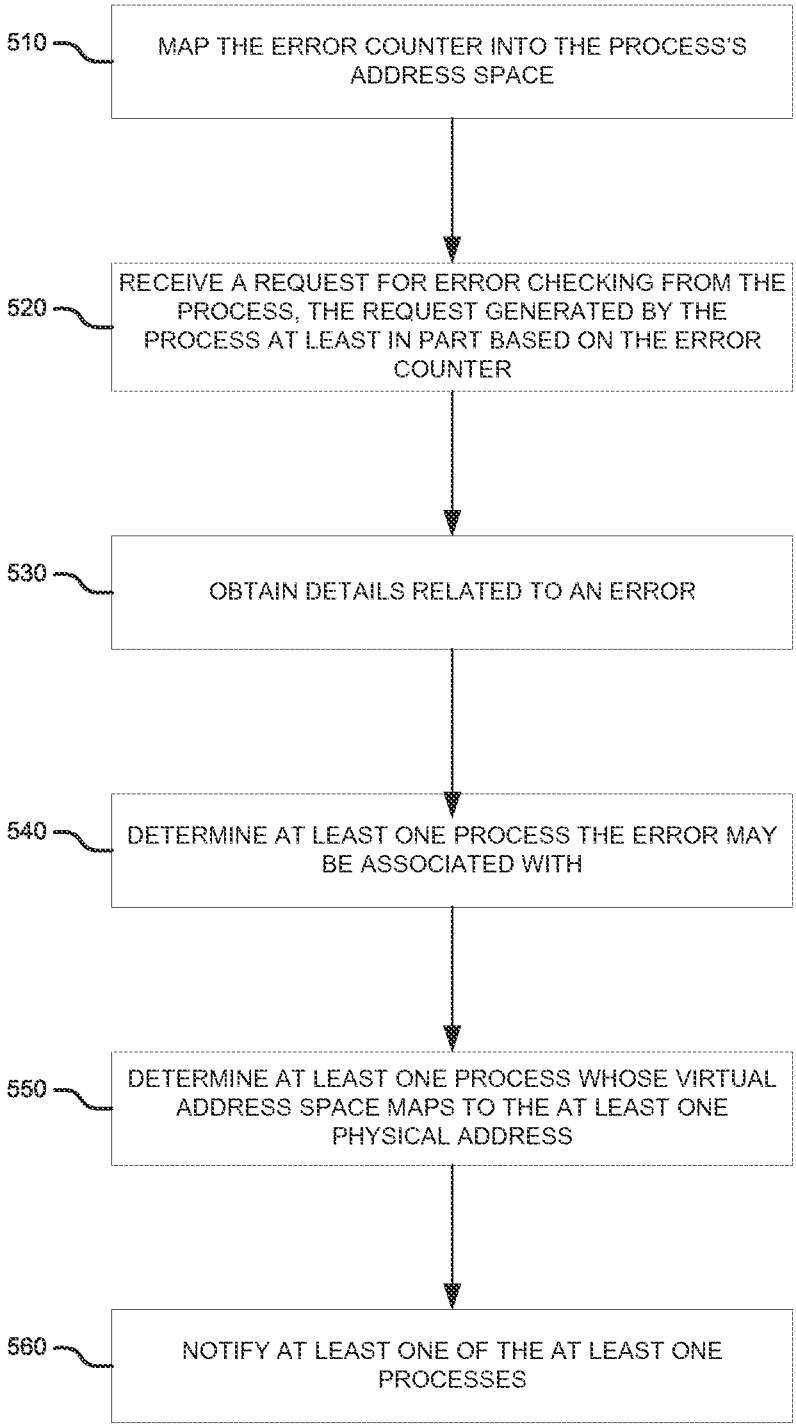
510
MAP THE ERROR COUNTER INTO THE PROCESS'S ADDRESS SPACE
520
RECEIVE A REQUEST FOR ERROR CHECKING FROM THE PROCESS, THE REQUEST GENERATED BY THE PROCESS AT LEAST IN PART BASED ON THE ERROR COUNTER
530
OBTAIN DETAILS RELATED TO AN ERROR
540
DETERMINE AT LEAST ONE PROCESS THE ERROR MAY BE ASSOCIATED WITH
550
DETERMINE AT LEAST ONE PROCESS WHOSE VIRTUAL ADDRESS SPACE MAPS TO THE AT LEAST ONE PHYSICAL ADDRESS
560
NOTIFY AT LEAST ONE OF THE AT LEAST ONE PROCESSES

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/050732

A. CLASSIFICATION OF SUBJECT MATTER

G06F ll/08(2006.01)i, G11C 29/00(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 11/08; H04N 7/167; G06F 11/10; G 11B 20/10; G06F 11/00; G 11C 29/00; G 11B 20/18

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: memory, writing error, error counter, compare, detection, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category'k	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2007-0098163 Al (JOSEPH MACRI et al.) 03 May 2007 See paragraphs [0016H0019] and [0028] ; and figures 1 and 4.	1
A		2-14
A	US 6,098,179 A (PAUL KARL HARTER, JR.) 01 August 2000 See column 7, lines 31-36 ; column 10, line 57 - column 12, line 18; and figures 2 and 5.	1-14
A	US 2013-0124931 Al (TSAN LIN CHEN) 16 May 2013 See paragraphs [0044]- [0045] ; and figures 1 and 3.	1-14
A	US 2014-0317479 Al (CISCO TECHNOLOGY, INC.) 23 October 2014 See paragraphs [0007]- [0019] and figure 6C.	1-14
A	EP 2063428 A2 (KABUSHIKI KAISHA TOSHIBA) 27 May 2009 See paragraphs [0012]- [0020] and figure 1.	1-14

I I 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	Date of mailing of the international search report
16 June 2016 (16.06.2016)	17 June 2016 (17.06.2016)

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/050732

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0098163 Al	03/05/2007	EP 1955163 A2	13/08/2008
		US 7996731 B2	09/08/2011
		US 8661300 Bl	25/02/2014
		WO 2007-052147 A2	10/05/2007
		WO 2007-052147 A3	30/08/2007
US 6098179 A	01/08/2000	None	
US 2013-0124931 Al	16/05/2013	None	
US 2014-0317479 Al	23/10/2014	CN 105122213 A	02/12/2015
		EP 2972871 Al	20/01/2016
		WO 2014-151758 Al	25/09/2014
EP 2063428 A2	27/05/2009	CA 2638421 Al	26/05/2009
		CA 2638421 C	27/01/2015
		EP 2063428 A3	28/04/2010
		JP 2009-129284 A	11/06/2009
		JP 4417994 B2	17/02/2010
		US 2009-0138783 Al	28/05/2009
		us 8631285 B2	14/01/2014