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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 August 2015 (06.08.2015)

WIPO | PCT

(10) International Publication Number
WO 2015/114644 A1

(51) International Patent Classification:
G06F 12/00 (2006.01)

(21) International Application Number:
PCT/IN2014/000075

(22) International Filing Date:
30 January 2014 (30.01.2014)

(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) Inventors; and
(71) Applicants *(for US only)*: **GOPALAKRISHNAN, Shyam Sankar** [IN/IN]; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, 560048, Karnataka (IN). **MANGALORE, Pramod Kumar** [IN/IN]; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, 560048, Karnataka (IN). **K E, Prashanth** [IN/IN]; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, 560048, Karnataka (IN). **MADHYASTHA, Sandesh V** [US/IN]; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, 560048, Karnataka (IN).

(74) Agent: **ALANKI, N.V. Pradeep Kumar**; Global IP Services, 198F, 27th Cross, 3rd Block, Jayanagar, Bangalore, 560011, Karnataka (IN).

(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, 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,

[Continued on next page]

(54) Title: PERSISTENT POINTERS FOR PROGRAMS RUNNING ON NVRAM BASED COMPUTERS

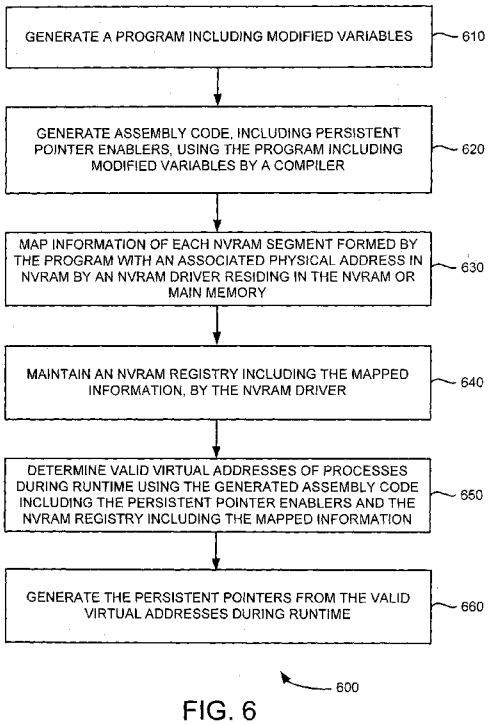

FIG. 6

(57) Abstract: In one example implementation, a method for generating persistent pointers using non-volatile random access memory (NVRAM) compiler directives in a program for NVRAM based computing systems includes generating a program including modified variables. The modified variables include NVRAM compiler directives indicative of persistent pointer type. The method further includes generating assembly code, including persistent pointer enablers, using the program including the modified variables by a compiler. Furthermore, the method includes mapping information of each NVRAM segment formed by the program with an associated physical address in NVRAM by an NVRAM driver residing in the NVRAM or main memory, maintaining an NVRAM registry including the mapped information by the NVRAM driver, determining valid virtual addresses of processes during runtime using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information, and generating the persistent pointers from the valid virtual addresses during runtime.

WO 2015/114644 A1

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))*

— *of inventorship (Rule 4.17(iv))*

Published:

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

PERSISTENT POINTERS FOR PROGRAMS RUNNING ON NVRAM BASED COMPUTERS

BACKGROUND

[1] Non-volatile solid state memory technologies, such as non-volatile random access memory (NVRAM) and the like, have been rapidly maturing in the recent years. Further, the NVRAM can be modeled as a block oriented device. In this case, programming modules are typically implemented using existing memory hierarchy, i.e., RAM ↔ NVRAM based pseudo block devices. However, such implementation may need additional programming instructions to store the data onto the NVRAM based devices via a block mode interface.

[2] Furthermore, NVRAM provides byte oriented access, and can be used as a temporary memory and/or as a persistent memory. This can facilitate a target program/application to access the data structures from the NVRAM. In addition, the data types used in existing computer programming languages may be easily ported to use NVRAM for storage other than for pointers.

[3] Moreover, an emerging trend is to use NVRAMs as replacement to DRAMs in computers. Since NVRAMs have a property of persistence as well as byte addressability, they can be used to store data structures similar to the way it is done for DRAMs. For example, linked-lists may be saved as-is in a persistent manner to avoid rebuilding them upon power-up after a reboot or power-off.

BRIEF DESCRIPTION OF THE DRAWINGS

[4] Figure 1 is a block diagram depicting an example scenario of using/sharing pointers/virtual addresses in a data structure in a non-volatile random access memory (NVRAM) environment across two processes, in the context of the present subject matter.

[5] Figure 2 is a block diagram depicting example modules used to implement example system for generating NVRAM based persistent pointers using NVRAM compiler directives.

[6] Figure 3 depicts a block diagram of an example NVRAM registry format having mapping of physical frame numbers associated with an NVRAM segment.

[7] Figure 4 depicts an example 64 bit field format for NVRAM pointers that is used to implement example systems for generating NVRAM based persistent pointers using NVRAM compiler directives.

[8] Figure 5 depicts an example per process table generated for the mapping of the NVRAM segments into the process address spaces.

[9] Figure 6 is a flow diagram depicting an example method for generating NVRAM based persistent pointers using NVRAM compiler directives.

DETAILED DESCRIPTION

[10] In the following description and figures, some example implementations of systems and/or methods for implementing persistent pointers in a process running on non-volatile random access memory (NVRAM) and/or dynamic random-access memory (DRAM) based computing systems using NVRAM compiler directives are described. Pointers embedded in the data structure during operation may not be valid after power-up and/or reboot. To overcome this, the embedded pointers need to be made valid across reboots/power cycles. Further, the embedded pointers are operating system process (for example, a running program on the system) dependent and may vary across operating system processes for the same memory location. Pointers indicate address of data structures, objects and/or functions. Further, pointers are typically values of addresses in virtual address spaces, which can range from minimum to maximum virtual addresses. Furthermore, pointers may be de-referenced to access data stored in an address location. This flexible but powerful construct may be used in implementing most of the basic data structures, such as trees, link lists and the like.

[11] Various examples described below relate to generating NVRAM based persistent pointers to locations in an NVRAM using NVRAM compiler directives. The persistent pointers include an NVRAM segment identifier and an offset within an NVRAM segment. More specifically, the examples described below relate to a scheme of including NVRAM compiler directives in a source code to signal a compiler that the program variables would point to locations in NVRAM and has to be treated as

persistent pointers. Furthermore, the examples described below relate to creation and storage of NVRAM addresses based on using NVRAM segment numbers and associated offset in the NVRAM segment and generating associated code by the compiler to map NVRAM segments calling into interfaces of an NVRAM driver in response to namespace directives and lookup of NVRAM pointers. The namespace directives are one of the NVRAM compiler directives. In addition, the examples described below generate code to convert NVRAM pointers (as shown in Figure 4) into virtual addresses for each process. This mechanism ensures that NVRAM pointers are valid across reboots/power cycles and are valid across processes.

[12] The terms "computing systems" and "computers" are used interchangeably throughout the document. Further, the terms "program", "computer program" and "user program" are being used interchangeably throughout the document. Furthermore, the term "process" refers to an instance of a computer program that is being executed. In addition, the term "user program including NVRAM compiler directives" refers to source code including NVRAM compiler directives. Moreover, the term "persistent pointer enabled user program" refers to persistent pointer enabled assembly code. Also, the terms "persistent pointers" and "NVRAM pointers" are used interchangeably throughout the document.

[13] Figure 1 is a block diagram 100 depicting an example scenario of using pointers/virtual addresses in a data structure in an NVRAM environment across two processes. Referring to Figure 1, the example scenario depicts a first process 110 for writing a binary data structure onto a portion of shared NVRAM memory130. Since NVRAM segment is being mapped at a virtual address which is different from the first process , the written data structure may not be valid in the second process 120. Furthermore, if tree nodes in the data structure are saved and then reboot the computing system, then the pointers in the tree node may not be valid in both the first process 110 and the second process 120. Various examples described below enable the written data structure to be valid in processes across reboots.

[14] Figure 2 is a block diagram depicting an example system 200 for realizing implementation of persistent pointers in NVRAM based computing systems using NVRAM compiler directives to address the example scenario shown in Figure 1. The

example system 200 generally includes inputting a user program including NVRAM compiler directives 210 into a compiler 235 residing in a server 230 that is communicatively coupled to a computing system 220. In the example shown in Figure 2, the compiler 235 is described as residing in the server 230; however the compiler 235 can also be implemented as part of the computing system 220.

[15] Further, the compiler 235 includes an NVRAM compiler directive code generator 240. The computing system 220 includes an NVRAM 250 and a main memory 260. Example main memory is an NVRAM, DRAM and the like. Further, the main memory 260 includes an NVRAM driver 270. The NVRAM 250 includes an NVRAM registry 280.

[16] In operation, the NVRAM compiler directive code generator 240 receives the user program including the NVRAM compiler directives 210 and generates persistent pointer enabled user program 290 for enabling persistent pointers upon encountering the NVRAM compiler directives in the user program 210 during runtime. The NVRAM compiler directive code generator 240 and the NVRAM driver 270 represent any combination of circuitry and executable instructions to generate code to support NVRAM compiler directives for enabling the persistent pointers.

[17] In this case, the user program 210 including modified variables is generated. The modified variables include the NVRAM compiler directives that are indicative of a persistent pointer type. Example modified variables that can act as directives to treat the variable as an NVRAM pointer variable are as follows:

```
NVRAM NVR1::int *x;
NVRAM int *y;
int z;
z = *x;
```

[18] In the user program 210, modified variables are stored in NVRAM segments to direct the compiler 235 where a variable is to be created or should be looked up in a name space. The NVRAM segments are identified by a NVRAM segment name. When the compiler 235 encounters a construct similar to NVRAM segment NVR1::, the compiler 235 is configured to know that the variable is in the NVRAM segment named NVR1. If a segment name is not encountered, the compiler 235 presumes that the variable is in a pre-defined NVRAM segment "STD". Upon encountering these

directives, the compiler 235 generates source code to map the respective NVRAM segments to the process. If the NVRAM segment already exists, then the NVRAM segment is mapped into the process address space.

[19] The compiler directive code generator 240 then generates associated assembly code, including persistent pointer enablers, using the NVRAM segment name, segment id and other information in the registry 280 upon encountering the NVRAM compiler directives in the generated user program 210 during runtime. In one example, the NVRAM compiler directive code generator 240 generates the assembly code including NVRAM segment numbers and associated offset within the NVRAM segment to form the physical address that is stored in the NVRAM using the user program 230 including the modified variables.

[20] One can envision that similar assembly code can be generated for other processor architectures. The above example assembly code reads current value of the pointer which has the offset from the base of the segment and segment identifier.

[21] Further in operation, the NVRAM driver 270 maps the information of each NVRAM segment formed by the user program with an associated physical address in the NVRAM. For example, the information of each NVRAM segment includes a unique NVRAM segment name, segment identifier, physical address of a NVRAM page that is part of the NVRAM segment and other such attributes assigned to the NVRAM segment by an operating system. The NVRAM driver 270 then maintains an NVRAM registry including the mapped information.

[22] Furthermore, the processes can carve out the NVRAM 250 into NVRAM segments, such as those shown in Figure 1. The information associated with each NVRAM segment includes a unique NVRAM segment name 310 and an associated NVRAM segment number 320 as shown in an example NVRAM registry 280 in Figure 3. Also, it can be seen in Figure 3, the example NVRAM registry 280 maintains associated physical address. Example physical address includes physical frame numbers (PFNs) 330 in the NVRAM 250 and associated attributes 340. For example, the NVRAM registry 280 can be stored either in an allocated NVRAM segment or in an external persistent memory that is mapped to a pre-defined physical address. Example

external persistent memory is an external disk drive. Also, the NVRAM driver can be configured to be a kernel loadable plug-in module.

[23] The NVRAM driver 270 then maintains the NVRAM registry 280 including information of each NVRAM segment formed by the process running the user program 210 with an associated physical address in the NVRAM 250. Further, process interfaces may be realized as function codes to the NVRAM driver 270 so that the NVRAM driver 270 can manage allocation and de-allocation of the NVRAM 250 on the computing system 220. It can be envisioned that the NVRAM driver 270 can be modeled similar to a file system driver that facilitates shared memory in an operating system, such as a Linux.

[24] As shown in an example persistent pointer format 400 of Figure 4, persistent pointers include NVRAM segment number 320 and an associated NVRAM segment offset 410 in place of virtual/physical addresses in the process. The example persistent pointer format 400 shown in Figure 4 is for a 64 bit field processor implementations, which can accommodate a 8 bit wide field for the NVRAM segment number 320 and the remaining 56 bit field may be used for offsets associated with about 256 unique NVRAM segments as shown in Figure 4.

[25] In this case, operating system 295 is modified to create a per-process table 500, as shown in Figure 5, to maintain a mapping of the NVRAM segment numbers 320 to the base virtual address at which the segment is mapped for that process. This table 500 may be referred to as process NVRAM table (PROCNVRAMTBL). As shown in Figure 5, the NVRAM segment number 320 is used to index the PROCNVRAMTBL. This process specific table 500 would have all the NVRAM segments mapped into the processes running the user program 210 (shown in Figure 2). It can be seen that once the base virtual address for an NVRAM segment is extracted, the final virtual address may be obtained by adding the base virtual address to the offset in the NVRAM segment. The NVRAM segment number 320 can be obtained from the NVRAM registry 280, by indexing into the NVRAM registry 280 using the NVRAM segment name 310. Further, the operating system process management and executable image loading are configured to generate a process specific table, PROCNVRAMTBL, which would

include the identification information of the NVRAM segments associated to that process and the process address where the NVRAM segments are mapped to.

[26] Further as shown in Figure 5, the NVRAM compiler directive code generator 240 (shown in Figure 2) generates code that translates the assigned address into a segment identifier and an offset from the base of the NVRAM segment. The table 500 includes a mapping of the NVRAM segment number 320 to a corresponding virtual address used for:

- obtaining the segment number 320 by locating an entry for which the assigned address minus base virtual address is positive and lowest across all other entities;
- computing the offset by subtracting the base virtual address from the assigned address; and
- storing the NVRAM segment number and the offset in the NVRAM pointer.

[27] Further as shown in Figure 4, the NVRAM pointer includes NVRAM segment identification, such as NVRAM segment number and an offset. Furthermore, the compiler 235 (shown in Figure 2) stores the NVRAM segment identification and the associated offset. The above technique works for pointers in a same NVRAM segment as well as pointers across NVRAM segments to make it persistent pointers.

[28] Furthermore in operation, the NVRAM driver 270 determines valid virtual addresses of processes, during runtime, using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information. Also, during the runtime, the NVRAM driver 270 then generates the persistent pointers from the valid virtual addresses.

[29] In the discussion herein, the compiler directive code generator 240 and the NVRAM driver 270 of Figure 2 have been described as a combination of circuitry and executable instructions. Such components can be implemented in a number of fashions. The executable instructions can be processor executable instructions, such as program instructions, stored on memory, such as the NVRAM 250 and/or main memory 260, which is a tangible, non-transitory computer readable storage medium, and the circuitry can be electronic circuitry, and computing system 220, for executing those instructions. The computing system 220, for example, can include one or multiple processors. The NVRAM 250 can store program instructions which enable support for persistent pointers

as a result of the code generated by the NVRAM compiler directive code generator 240, that when executed by the computing system 220 implements persistent pointer support which enables the persistent pointers residing in the NVRAM 250 to be valid across reboots and valid across processes. The NVRAM 250 and/or main memory 260 can be integrated in the same computing system 220 or it can be in a separate computing system that is accessible to the computing system 220.

[30] In one example, the executable instructions can be part of an installation package that when installed can be executed by the computing system 220 to implement the system 200. In that example, the memory resource in the computing system can also be a portable medium such as a CD, a DVD, a flash drive, or memory maintained by a computer device from which the installation package can be downloaded and installed. In another example, the executable instructions can be part of an application or applications already installed. Here, the memory resource in the computing system 220 can include integrated memory such as a drive, NVRAM, DRAM or the like.

[31] Referring now to Figure 2, the NVRAM compiler directive code generator 240 and the NVRAM driver 270 can be stored in a same computing system or distributed across servers, other devices or storage mediums, or a combination thereof. For example, an instance of the NVRAM compiler directive code generator 240 can be executing on each one of the processor resources of the server devices. The engines and/or modules can complete or assist completion of operations performed in describing another engine and/or module. The engines, drivers and/or modules can perform the example methods described in connection with Figure 6.

[32] Figure 6 is a flow diagram 600 depicting an example method for implementing persistent pointers in a user program that can run in an NVRAM based computing systems. Referring to Figure 6, example methods for implementing persistent pointers that is based on NVRAM compiler directives in a program.

[33] At block 610, a program including modified variables is generated. The modified variables include NVRAM compiler directives indicative of persistent pointer type. At block 620, assembly code including persistent pointer enablers is generated using the program including modified variables by a compiler. At block 630, information of each

NVRAM segment formed by the program is mapped with an associated physical address in an NVRAM by an NVRAM driver. The NVRAM driver can reside in the NVRAM, DRAM and the like. At block 640, an NVRAM registry including the mapped information is maintained by the NVRAM driver. At block 650, valid virtual addresses of processes are determined during runtime using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information. The process is an instance of a computer program that is being executed. At block 660, the persistent pointers are generated, during runtime, from the valid virtual addresses. The persistent pointers include an NVRAM segment identifier and an offset of a location within an NVRAM segment. The persistent pointers point to the offset of the location.

[34] The method associated with the flow diagram 600 of Figure 6 is explained in more detail with reference to Figures 1-5 above. Although the flow diagram of Figure 6 illustrates specific orders of execution, the order of execution can differ from that which is illustrated. For example, the order of execution of the blocks can be scrambled relative to the order shown. Also, the blocks shown in succession can be executed concurrently or with partial concurrence. All such variations are within the scope of the present subject matter.

[35] The terms "include," "have," and variations thereof, as used herein, have the same meaning as the term "comprise" or appropriate variation thereof. Furthermore, the term "based on", as used herein, means "based at least in part on." Thus, a feature that is described as based on some stimulus can be based on the stimulus or a combination of stimuli including the stimulus.

[36] The present description has been shown and described with reference to the foregoing examples. It is understood, however, that other forms, details, and examples can be made without departing from the spirit and scope of the invention that is defined in the following claims.

CLAIMS

What is claimed is:

1. A system for generating persistent pointers using non-volatile random access memory (NVRAM) compiler directives in a program for NVRAM based computing systems comprising:

a server; and

a computing system communicatively coupled to the server, wherein the computing system comprises:

an NVRAM; and

a main memory that is communicatively coupled with the NVRAM, wherein the NVRAM includes an NVRAM registry, wherein an NVRAM driver resides in the main memory or the NVRAM, wherein the server includes a compiler, and wherein the compiler includes an NVRAM compiler directive code generator to:

generate a program including modified variables, wherein the modified variables include NVRAM compiler directives indicative of persistent pointer type;

generate assembly code, including persistent pointer enablers, using the program including the modified variables, by the NVRAM compiler directive code generator;

map information of each NVRAM segment formed by the program with an associated physical address in the NVRAM by the NVRAM driver;

maintain the NVRAM registry including the mapped information, by the NVRAM driver;

determine valid virtual addresses of processes during runtime using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information, by the NVRAM driver; and

generate the persistent pointers from the valid virtual addresses during runtime, by the generated assembly code.

2. The system of claim 1, wherein the information of an NVRAM segment comprises a unique NVRAM segment name, segment identifier, physical address of a NVRAM page that is part of the NVRAM segment and attributes assigned to the NVRAM segment by an operating system.

3. The system of claim 1, further configured to:

generating the assembly code, including NVRAM segment numbers and associated offset within the NVRAM segment to form the persistent pointer that is stored in the NVRAM, using the program including the modified variables;

calling process interfaces of the NVRAM driver to map NVRAM segments containing the modified variables; and

generating the virtual addresses of the processes from the physical addresses and vice-versa.

4. The system of claim 1, further configured to:

storing the NVRAM registry in an NVRAM segment or in an external persistent memory that is mapped to a pre-defined physical address.

5. The system of claim 1, wherein the NVRAM driver is a kernel loadable plug-in module.

6. The system of claim 1, wherein the main memory comprises an NVRAM and/or DRAM.

7. A non-transitory computer readable storage medium comprising a set of instructions executable by at least one processor resource to:

generate a program including modified variables, wherein the modified variables include non-volatile random access memory (NVRAM) compiler directives indicative of persistent pointer type;

generate assembly code, including persistent pointer enablers, using the program including the modified variables by a compiler;

map information of each NVRAM segment formed by the program with an associated physical address in an NVRAM by an NVRAM driver residing in the NVRAM or main memory;

maintain an NVRAM registry including the mapped information, by the NVRAM driver;

determine valid virtual addresses of processes during runtime using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information; and

generate persistent pointers from the valid virtual addresses during runtime.

8. The non-transitory computer readable storage medium of claim 7, wherein the information of an NVRAM segment comprises a unique NVRAM segment name, segment identifier, physical address of a NVRAM page that is part of the NVRAM segment and attributes assigned to the NVRAM segment by an operating system.

9. The non-transitory computer readable storage medium of claim 7, wherein generating the assembly code, including persistent pointer enablers, using the program including the modified variables comprises:

generating the assembly code, including NVRAM segment numbers and associated offset within the NVRAM segment to form the persistent pointer that is stored in the NVRAM, using the program including the modified variables;

calling process interfaces of the NVRAM driver to map NVRAM segments containing the modified variables; and

generate the virtual addresses of the processes from the physical addresses and vice-versa.

10. The non-transitory computer readable storage medium of claim 7, comprising:

storing the NVRAM registry in an NVRAM segment or in an external persistent memory that is mapped to a pre-defined physical address.

11. A method for generating persistent pointers using NVRAM compiler directives in a program for non-volatile random access memory (NVRAM) based computing systems, comprising:

generating a program including modified variables, wherein the modified variables include NVRAM compiler directives indicative of persistent pointer type;

generating assembly code, including persistent pointer enablers, using the program including the modified variables by a compiler;

mapping information of each NVRAM segment formed by the program with an associated physical address in an NVRAM by an NVRAM driver residing in the NVRAM or main memory;

maintaining an NVRAM registry including the mapped information, by the NVRAM driver;

determining valid virtual addresses of processes during runtime using the generated assembly code including the persistent pointer enablers and the NVRAM registry including the mapped information; and

generating, during runtime, the persistent pointers from the valid virtual addresses.

12. The method of claim 11, wherein the information of each NVRAM segment comprises a unique NVRAM segment name, segment identifier, physical address of a NVRAM page that is part of the NVRAM segment and attributes assigned to the NVRAM segment by an operating system.

13. The method of claim 11, wherein generating the assembly code, including persistent pointer enablers, using the program including the modified variables comprises:

generating the assembly code, including NVRAM segment numbers and associated offset within the NVRAM segment to form the persistent pointer that is stored in the NVRAM, using the program including the modified variables;

calling process interfaces of the NVRAM driver to map NVRAM segments containing the modified variables; and

generating the virtual addresses of the processes from the physical addresses and vice-versa.

14. The method of claim 11, comprising:

storing the NVRAM registry in an NVRAM segment or in an external persistent memory that is mapped to a pre-defined physical address.

15. The method of claim 11, wherein the NVRAM driver is a kernel loadable plug-in module.

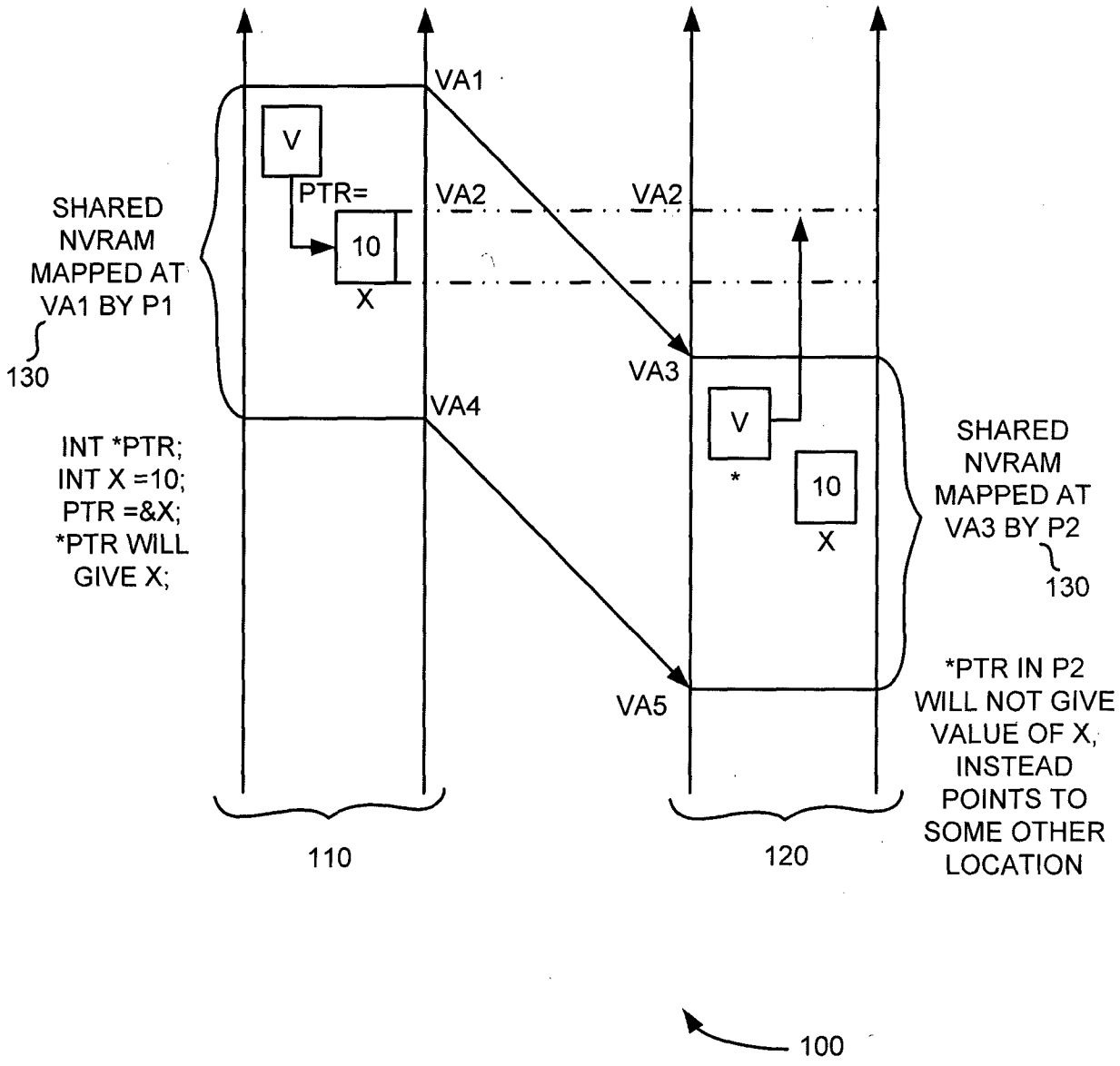
VA1
V
PTR=
VA2
VA2
10
X
SHARED
NVRAM
MAPPED AT
VA1 BY P1
130
VA3
VA4
V
*
10
X
INT *PTR;
INT X =10;
PTR =&X;
*PTR WILL
GIVE X;
SHARED
NVRAM
MAPPED AT
VA3 BY P2
130
VA5
*PTR IN P2
WILL NOT GIVE
VALUE OF X,
INSTEAD
POINTS TO
SOME OTHER
LOCATION
110
120
100

FIG. 1

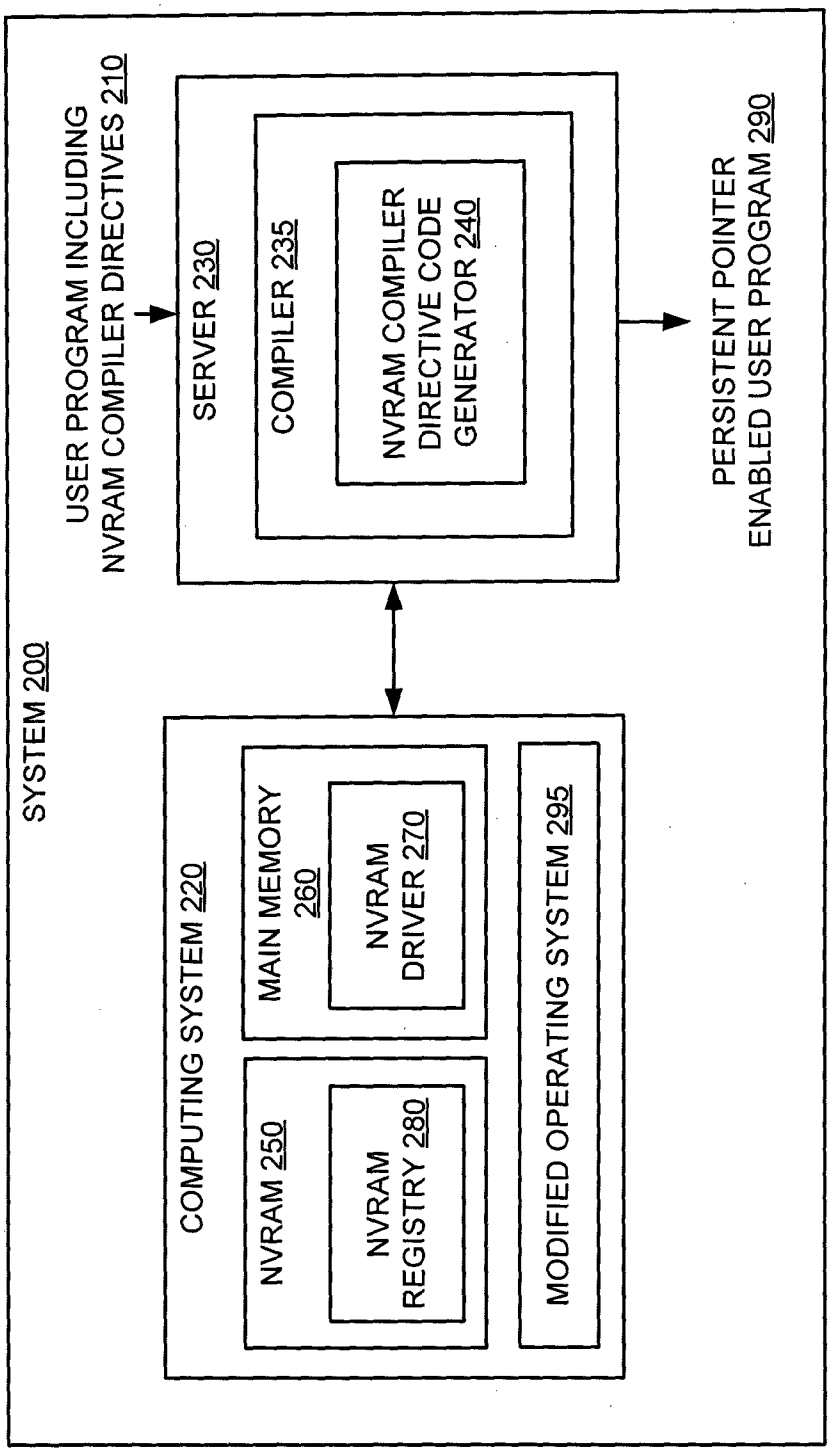
SYSTEM 200
USER PROGRAM INCLUDING NVRAM COMPILER DIRECTIVES 210
SERVER 230
COMPILER 235
NVRAM COMPILER DIRECTIVE CODE GENERATOR 240
PERSISTENT POINTER ENABLED USER PROGRAM 290
COMPUTING SYSTEM 220
NVRAM 250
NVRAM REGISTRY 280
MAIN MEMORY 260
NVRAM DRIVER 270
MODIFIED OPERATING SYSTEM 295

FIG. 2

310 NVRAM SEGMENT NAME	320 NVRAM SEGMENT NUMBER	330 NVRAM PFNs	340 ATTRIBUTES
STD	1	10	80000000
NVR1	10	100	80000000

280

FIG. 3

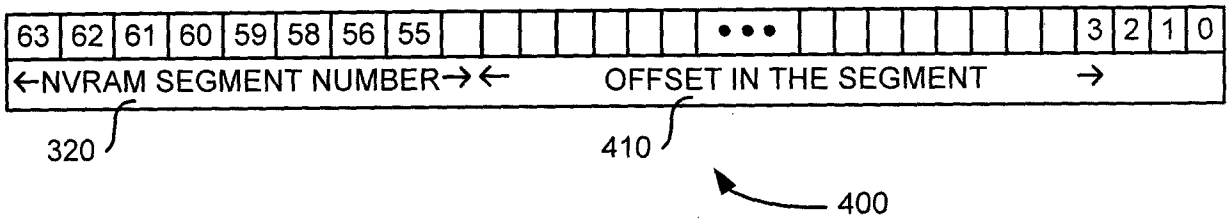

FIG. 4

320 NVRAM SEGMENT NUMBER	PROCESSBASE VA START	PROCESSBASE VA END
1	FFFFFFFF80000000	FFFFFFFF80008000
10	FFFFFFFF70000000	FFFFFFFF70008000

500

FIG. 5

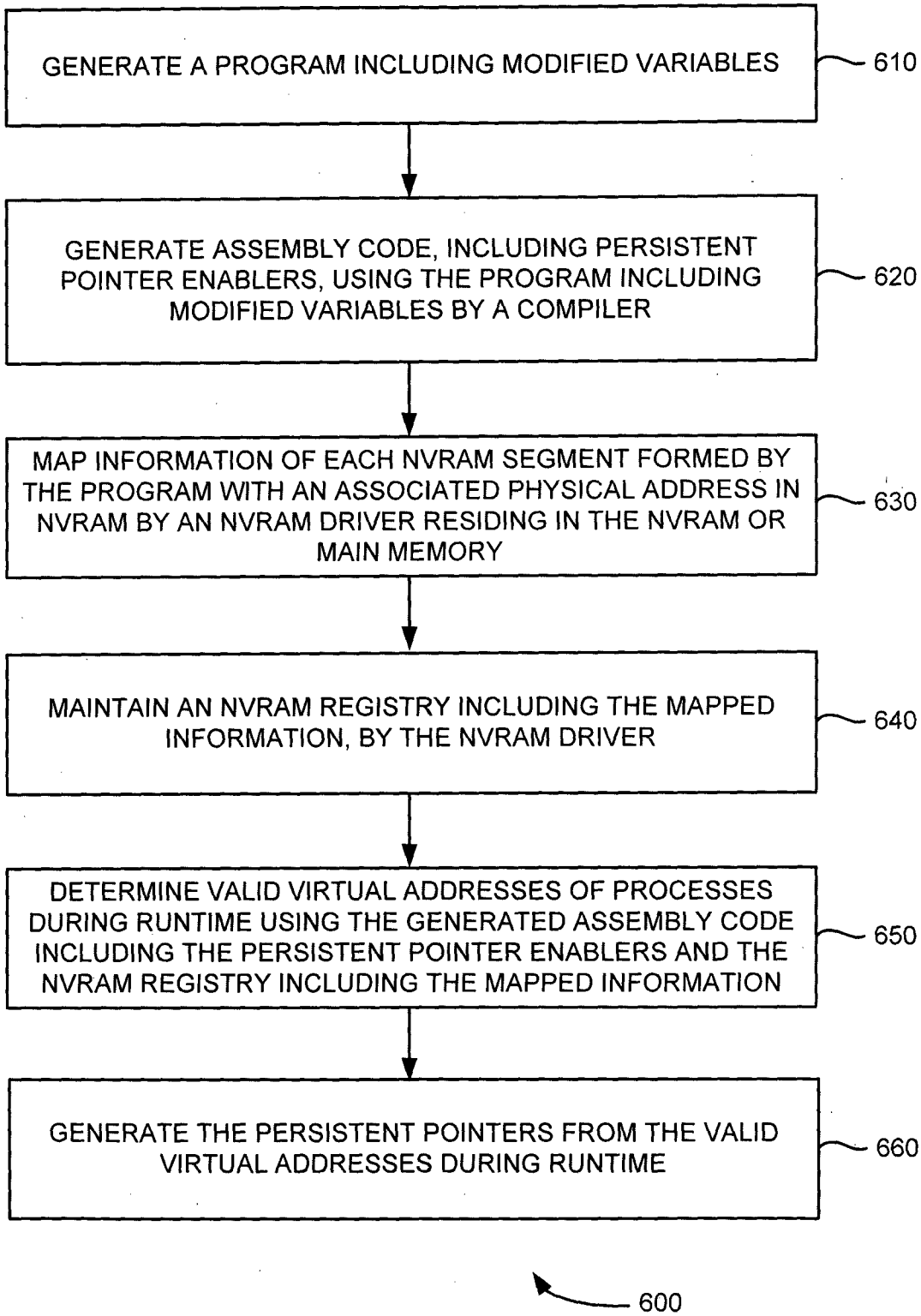
GENERATE A PROGRAM INCLUDING MODIFIED VARIABLES
610
GENERATE ASSEMBLY CODE, INCLUDING PERSISTENT POINTER ENABLERS, USING THE PROGRAM INCLUDING MODIFIED VARIABLES BY A COMPILER
620
MAP INFORMATION OF EACH NVRAM SEGMENT FORMED BY THE PROGRAM WITH AN ASSOCIATED PHYSICAL ADDRESS IN NVRAM BY AN NVRAM DRIVER RESIDING IN THE NVRAM OR MAIN MEMORY
630
MAINTAIN AN NVRAM REGISTRY INCLUDING THE MAPPED INFORMATION, BY THE NVRAM DRIVER
640
DETERMINE VALID VIRTUAL ADDRESSES OF PROCESSES DURING RUNTIME USING THE GENERATED ASSEMBLY CODE INCLUDING THE PERSISTENT POINTER ENABLERS AND THE NVRAM REGISTRY INCLUDING THE MAPPED INFORMATION
650
GENERATE THE PERSISTENT POINTERS FROM THE VALID VIRTUAL ADDRESSES DURING RUNTIME
660
600

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IN2014/000075

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/00(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

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

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

EPODOC,WPI,CNABS,CNTXT,IEEE,GOOGLE:pointer, code, nvram, assembl+, memory, regis+, segment, address, variable, directive, driver, physic+, persis+

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2013111151 A1 (ORACLE INTERNATIONAL CORPORATION) 02 May 2013 (2013-05-02) abstract, description, paragraphs [0035]-[0059], [0078]-[0086], and claims 1-20	1-15
A	CN 1869936 A (SAMSUNG ELECTRONICS CO., LTD.) 29 November 2006 (2006-11-29) the whole document	1-15
A	US 6049667 A (INTERNATIONAL BUSINESS MACHINES CORPORATION) 11 April 2000 (2000-04-11) the whole document	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	Date of mailing of the international search report
29 September 2014	**04 November 2014**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA(ISA/CN)** **6,Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088 China** Facsimile No. **(86-10)62019451**	Authorized officer **LIANG,Jing** Telephone No. **(86-10)62414423**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IN2014/000075

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2013111151 A1	02 May 2013	EP 2771794 A1	03 September 2014
		CN 103959257 A	30 July 2014
		WO 2013062948 A1	02 May 2013
CN 1869936 A	29 November 2006	US 2006271778 A1	30 November 2006
		KR 20060122064 A	30 November 2006
US 6049667 A	11 April 2000	Non e	