



- (51) International Patent Classification:
G06F 21/53 (2013.01) *G06F 21/60* (2013.01)
- (21) International Application Number:
PCT/US2016/044319
- (22) International Filing Date:
27 July 2016 (27.07.2016)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
14/849,222 9 September 2015 (09.09.2015) US
- (71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).
- (72) Inventors: XING, Bin; 2756 NE Aurora Drive, Hillsboro, Oregon 97124 (US). SHANAHAN, Mark W.; 2841 Anfield Road, Raleigh, North Carolina 27606 (US). ZHANG, Bo; 4904 Highland Park Ct., Raleigh, North Carolina 27613 (US).
- (74) Agents: AU YEUNG, Al et al.; 1211 SW 5th Avenue, Suite 1500-1900, Portland, Oregon 97204 (US).

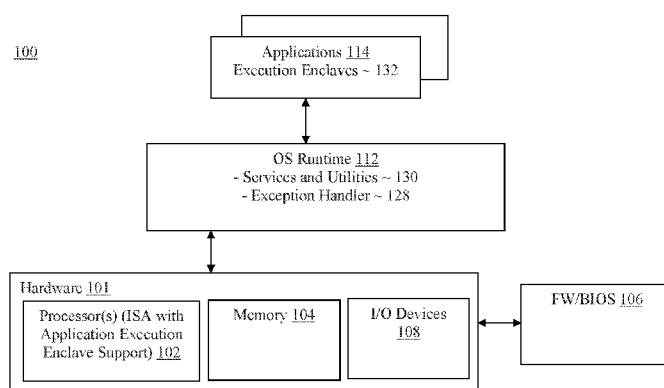
(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, 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, 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).

Published:

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

(54) Title: APPLICATION EXECUTION ENCLAVE MEMORY PAGE CACHE MANAGEMENT METHOD AND APPARATUS

**Figure 1**

(57) Abstract: Apparatuses, methods and storage medium associated with application execution enclave cache management, are disclosed herein. In embodiments, an apparatus may include one or more processors with supports for application execution enclaves; cache memory coupled with the one or more processors to be organized into a plurality of cache pages; and an exception handler to be operated by the one or more processors to handle cache page fault exceptions, wherein to handle cache page fault exceptions includes to handle a cache page fault triggered to request additional allocation of one or more cache pages to an execution enclave of an application. Other embodiments may be described and/or claimed.

Application Execution Enclave Memory Page Cache Management Method and Apparatus

Related Application

5 This application claims priority to U.S. Patent Application 14/849,222, entitled
“APPLICATION EXECUTION ENCLAVE MEMORY PAGE
CACHE MANAGEMENT METHOD AND APPARATUS,” filed September 9, 2015.

Technical Field

10 The present disclosure relates to the field of computing. More particularly, the
present disclosure relates to the application execution enclave memory page cache
management method and apparatus.

Background

15 The background description provided herein is for the purpose of generally
presenting the context of the disclosure. Unless otherwise indicated herein, the materials
described in this section are not prior art to the claims in this application and are not
admitted to be prior art by inclusion in this section.

 Security, protection from malicious software, has emerged as a major concern
of the computing art. It is expected that processors, e.g., some processors from
Intel® Corporation of Santa Clara, CA, will begin to provide support for
20 isolated/protected execution environments to individual applications (also referred to
as application execution enclaves or simply, enclaves). Accesses to the enclave
memory areas will be limited to codes resident in the enclaves only. Codes outside
the enclave will have no access to enclave memory areas. For Intel® processors, the
technology is currently known as SGX (Software Guard Extensions). For further
25 information, see Intel® Software Guard Extensions Programming Reference, dated
October, 2014.

 It is expected that the isolation supports will include support for enclave
dynamic memory management, allowing cache pages to be dynamically added to or
removed from an enclave on an as needed basis. One proposed approach
30 contemplated for dynamically augmenting an enclave with additional cache pages
includes having the enclave send a request to the privileged OS kernel or a privileged
driver for a certain number of cache pages to be mapped to a specified virtual
address. Since, for security reasons, calls to system services from enclaves are not

allowed, an enclave would have to make the request through the non-isolated portion of an application.

In response, the privileged software (i.e., the OS kernel or the privileged driver) will use the support features provided by the processor (e.g., the EAUG instruction for SGX) to allocate and map the required number of cache pages to the requesting enclave. Similar to the request, the privileged software would communicate completion of the allocation and mapping to the enclave through the non-isolated portion of the application. On notification of completion of the allocation and mapping, the enclave is to execute an acknowledgement (e.g., the EACCEPT instruction for SGX) for each newly added cache page.

Brief Description of the Drawings

Embodiments will be readily understood by the following detailed description in conjunction with the accompanying drawings. To facilitate this description, like reference numerals designate like structural elements. Embodiments are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings.

Figure 1 illustrates a computing device having the application execution enclave cache management technology of the present disclosure, according to various embodiments.

Figure 2 illustrates a virtual address space view of an application, according to various embodiments.

Figure 3 illustrates an example process for dynamically adding one or more cache pages to an application execution enclave, according to the various embodiments.

Figures 4 and 5 illustrate an example process of the exception handler for dynamically adding one or more cache pages to an application execution enclave, according to various embodiments.

Figure 6 illustrates an example computer system suitable for use to practice aspects of the present disclosure, according to various embodiments.

Figure 7 illustrates a storage medium having instructions for practicing methods described with references to Figures 3-5, according to various embodiments.

Detailed Description

Apparatuses, methods and storage medium associated with application execution enclave memory page cache management, are disclosed herein. In embodiments, an apparatus may include one or more processors with supports for application execution enclaves; memory coupled with the one or more processors to be organized into a plurality

of memory pages; and an exception handler to be operated by the one or more processors to handle memory page fault exceptions, wherein to handle memory page fault exceptions includes to handle a memory page fault triggered to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application.

5 In the following detailed description, reference is made to the accompanying drawings which form a part hereof wherein like numerals designate like parts throughout, and in which is shown by way of illustration embodiments that may be practiced. It is to be understood that other embodiments may be utilized and structural or logical changes may be made without departing from the scope of the present disclosure. Therefore, the
10 following detailed description is not to be taken in a limiting sense, and the scope of embodiments is defined by the appended claims and their equivalents.

Aspects of the disclosure are disclosed in the accompanying description. Alternate embodiments of the present disclosure and their equivalents may be devised without parting from the spirit or scope of the present disclosure. It should be noted that like
15 elements disclosed below are indicated by like reference numbers in the drawings.

Various operations may be described as multiple discrete actions or operations in turn, in a manner that is most helpful in understanding the claimed subject matter. However, the order of description should not be construed as to imply that these operations are necessarily order dependent. In particular, these operations may not be
20 performed in the order of presentation. Operations described may be performed in a different order than the described embodiment. Various additional operations may be performed and/or described operations may be omitted in additional embodiments.

For the purposes of the present disclosure, the phrase “A and/or B” means (A), (B), or (A and B). For the purposes of the present disclosure, the phrase “A, B, and/or C”
25 means (A), (B), (C), (A and B), (A and C), (B and C), or (A, B and C).

The description may use the phrases “in an embodiment,” or “in embodiments,” which may each refer to one or more of the same or different embodiments. Furthermore, the terms “comprising,” “including,” “having,” and the like, as used with respect to
embodiments of the present disclosure, are synonymous.

30 As used herein, the term “module” may refer to, be part of, or include an Application Specific Integrated Circuit (ASIC), an electronic circuit, a processor (shared, dedicated, or group) and/or memory (shared, dedicated, or group) that execute one or more software or firmware programs, a combinational logic circuit, and/or other suitable components that provide the described functionality.

Referring now to Figure 1, wherein a computing device having the power management package provision technology of the present disclosure, according to various embodiments, is shown. As illustrated, in embodiments, computing device 100 may include hardware 101, firmware (FW)/basic input/output services (BIOS) 106, OS runtime 112 and applications 114, operatively coupled with each other as shown. Hardware 101 may include one or more processors 102 with instruction set architectures (ISA) that support application execution enclaves (i.e., isolated/protected execution areas), and memory 104 having a number of memory pages. Applications 114 may have corresponding execution enclaves 132 having respective memory page caches. OS runtime 112 may include a number of services and utilities 130. In particular, services and utilities 130 may include an exception handler 128 configured to handle one or more exceptions, including exceptions intentionally or unknowingly triggered to request one or more additional memory pages be dynamically added to a memory page cache of an application execution enclave 132. An example of an unknowingly triggered request may be an exception triggered by an unexpected overflow of a heap or stack of an application execution enclave, or violation of memory protection. These and other aspects related to application execution enclave memory page cache management will be further described below with references to Figures 2-5.

Except for their ISA that supports application execution enclaves, processor(s) 102 may otherwise be any one of a number of processors known in the art, having one or more processor cores. Memory 104 may be any known volatile or non-volatile memory in the art, suitable for storing data. Memory 104 may include a hierarchy of cache memory and system memory. Both the cache and system memory may be respectively organized into cache pages and memory pages. In embodiments, hardware 101 may further include I/O devices 108, or other elements (not shown). Examples of I/O devices 108 may include communication or networking interfaces, such as Ethernet, WiFi, 3G/4G, Bluetooth®, Near Field Communication, Universal Serial Bus (USB) and so forth, storage devices, such as solid state, magnetic and/or optical drives, input devices, such as keyboard, mouse, touch sensitive screen, and so forth, and output devices, such as, display devices, printers, and so forth.

FW/BIOS 106 may be any one of a number FW/BIOS known in the art. Except for exception handler 128 configured to handle exceptions triggered to request additional memory pages be dynamically allocated and mapped for a memory page cache of an application enclave, OS runtime 112 may likewise be any one of a number of OS known

in the art, e.g., the Windows OS from Microsoft® Corporation. Applications 114, exception of usage of execution enclaves, may likewise be any one of a number of applications known in the art.

Referring now to Figure 2, wherein a virtual address space view of an application, according to various embodiments, is illustrated. As shown, within virtual address space 202 may be OS runtime 112, isolated/protected execution enclave 132 having its own memory page cache, the non-isolated application code segments 204a-204b. Execution enclave 132 may include entry table 212, enclave heap 214, enclave stack 216, and resident enclave code 218. Entry table 212 may include information about various resources within execution enclave 132, dictating which memory pages to accept. In embodiments, entry table 212 may further include information about locations of dynamic regions 222 of execution enclave 132. Dynamic regions 222 of an execution enclave 132 may specify address ranges of the execution enclave 132 where new memory pages may be mapped. In embodiments, information about a dynamic region 222 may include the starting address of the dynamic region and/or page type 224. Page type 224 may include parameters to be provided to an execution enclave support instruction to augment the memory page cache of an execution enclave, e.g., the EAUG instruction of SGX. Examples of these parameters may include, but are not limited to, read and/or write permissions. Region size 226 may specify a number of memory pages the dynamic region can hold. Address mask and flags 228 may include information on buffer and/or code/data section/segment alignment within the dynamic region (address mask), and information on the growth direction for adding memory pages. In embodiments, a value 0 may denote the new memory pages are to be mapped growing upward (with increasing address values), and a value of 1 may denote the new memory pages are to be mapped growing downward (with decreasing address values).

Enclave heap 214 and enclave stack 216 may be similar to conventional software heap and stack, except they are within the enclave's memory page cache, thus isolated and protected, and may be accessed by enclave code 218 resident within enclave 132 only. As described earlier, on entry, execution may flow within enclave code 218, and on completion, exit back to the non-isolated application code segments 204a-204b. Enclave code 218 is not permitted to make any direct calls to OS services and utilities 130 for services. For security/protection, such calls must go through the hosting application 114. Before further describing the present disclosure, it should be noted that while for ease of understanding, only one execution enclave 132 is shown within virtual address space 202,

the present disclosure is not so limited. Virtual address space 202 of application 114 may include one or more execution enclaves 132 (each having its own memory page cache).

Referring now to Figure 3, wherein an example process for dynamically adding one or more memory pages to a memory page cache of an application execution enclave, according to the various embodiments, is illustrated. As shown, process 300 for
5 dynamically adding one or more memory pages to a memory page cache of an application execution enclave may include operations performed at blocks 302-312. The operations at blocks 302-312 may be performed, e.g., by execution conclave 132 of Figure 1.

Process 300 may start at block 302. At block 302, execution of an application may
10 be transferred into one of its execution conclave. On entry, a block 304, resident conclave code may be executed. At some point, while execution continues (“No” branch of block 306), a need may arise for one or more additional memory pages added to the memory page cache of the execution enclave (“Yes” branch of block 310). The need may arise for any one of a number of reasons, e.g., a realization of needs for additional memory space,
15 or an unexpected overflow of the enclave’s heap or stack. If the need for one or more additional memory pages does not arise (“No” branch of block 310), process 300 may return to block 304 and continue therefrom as earlier described.

If the need for one or more additional memory pages (“Yes” branch of block 310) arises, at block 312, the application enclave may intentionally or unknowingly trigger a
20 memory page fault. In embodiments, the application enclave may intentionally trigger a memory page fault by issuing a memory page allocation acknowledgment for a vacant address to which the application enclave wants one or more memory pages be added. In SGX embodiments, the memory page fault may be intentionally triggered by issuing the EACCEPT instruction for a vacant address to which the application enclave wants one or
25 more cache pages starting to be added.

On return, i.e., handling of the memory page fault (for requesting one or more additional memory pages), at block 314, the execution conclave may acknowledge allocation of the new memory pages. In SGX embodiments, the execution conclave may execute (retry in the case of intentional trigger of the memory page fault) the EACCEPT
30 instruction.

On acknowledgment of the newly added memory page(s), process 300 may return to block 304, and continue therefrom as earlier described. Eventually, execution within the execution conclave completes (“Yes” branch of block 306), then at 308, execution may exit the enclave, and return to one of the non-isolated segments of the application. Process

300 may end.

Referring now to Figures 4 and 5, wherein an example process for an exception handler to dynamically add one or more memory pages to a memory page cache of an application execution enclave, according to various embodiments, is illustrated. As shown, process 400 for dynamically adding one or more memory pages to an application execution enclave may include operations performed at blocks 402-460. The operations at block 402-460 may be performed, e.g., by exception handler 128 of Figure 1.

Process 400 may start at block 402. At block 402, a memory page fault exception may be received. Next, at block 404, a determination may be made on whether the memory page fault exception was intentionally or unknowingly triggered to request one or more additional memory pages to a memory page cache of an application enclave. The determination may be made, e.g., based at least in part on a page fault code (PFEC) of the memory page fault (#PF). If the memory page fault exception was not triggered to request one or more additional memory pages to a memory page cache of an application enclave, at block 406, the memory page fault may be handled as in the past. For example, if the memory page fault is caused because the referenced memory page was paged out, the missing memory page may be fetched and be paged back into the memory. On handling the conventional memory page fault, e.g., paging back a missing memory page, at block 408, the faulting application, i.e., the application that triggered the memory page fault, may be resumed. Thereafter, process 400 may end. In some embodiments, at block 406, depending on OS runtime 112, an exception may be injected into the faulting application, before proceeding to block 408 to resume the faulting application.

Back at block 404, if a result of the determination indicates that the memory page fault was triggered to request one or more additional memory pages to a memory page cache of an application execution enclave, at block 410, a dynamic region of the application execution enclave may be located. In embodiments, the PFEC may include information on locations where information about the dynamic regions of an execution enclave may be found. Next at block 412, a determination may be made to confirm a located dynamic region is not null. If a result of the determination is negative, i.e., the dynamic region is null, at block 414, an exception (e.g., SIGBUS if OS 112 is Linux) may be injected into the faulting application. Thereafter, process 400 may continue at block 408 as earlier described. In embodiments, e.g., SGX embodiments, operations at 410 and 412 may be integrally performed as part of the operations at block 404.

If a result of the determination is positive, i.e., the dynamic region is not null, at

block 416, one or more new memory pages may be allocated and mapped to the execution
conclave to augment the memory page cache previously allocated to the execution
enclave. In embodiments, the earlier described memory page address, memory page type,
memory page mask and flag information of the dynamic region may be gathered and used
5 in the allocation and mapping. In SGX embodiments, the EAUG instruction may be
executed with the memory page type as a parameter at the memory page address.

Next, at block 420, a check may be performed to determine whether the code/data
sections/segments are aligned. If the code/data sections/segments are aligned, at block
422, another check may be performed to determine whether the page fault is caused by a
10 write operation. If the result of the determination is negative, i.e., the page fault is not
caused by a write operation, process 400 may continue at block 408, as earlier described.
If the result of the determination is positive, i.e., the page fault is caused by a write
operation, process 400 may continue at block 414, and then at block 408, as earlier
described. The rationale of block 420 is that in SGX embodiments, EACCEPT causes
15 Read access to a page, and that no Read access should be performed on a newly allocated
page (which is considered as uninitialized data). In other words, a Read access to a newly
allocated page is either by EACCEPT or a software error.

Back at block 420, if the result of the alignment indicates the code/data
sections/segments are not aligned, i.e., there is space for further allocation, process 400
20 may continue at block 452. At block 452, the direction of allocation, i.e., whether the
memory pages are to be allocated (grow) in an upward direction (with increasing
addresses) or a downward direction (with decreasing addresses) may be determined, e.g.,
by examining the flag of the dynamic region. Based on a result of the direction
determination, process 400 may proceed to block 454 or 456 accordingly. At block 454, a
25 new memory page in the upward direction may be allocated and mapped (e.g., by setting
page address += page size). At block 456, a new memory page in the downward direction
may be allocated and mapped (e.g., by setting page address -= page size). In
embodiments, in addition to allocation in an upward or downward direction, allocation of
a predefined region may also be supported. Examples of a pre-defined region may
30 include, but are not limited to, a region as small as a page or as large as a module or
module section/segment.

Next, at block 458, a check may be performed to determine whether the page
address calculation is still within the dynamic region. If a result of the determination
indicates the page address calculation is no longer within the dynamic region, process 400

may continue at block 422 as earlier described. If a result of the determination indicates the page address calculation is still within the dynamic region, at block 460, another check may be performed to determine whether the page address is populated. If a result of the determination indicates the page address is not populated, process 400 may return to block 416 for further allocation, as earlier described. If a result of the determination indicates that the page address is populated, process 400 may return to block 422, and continue therefrom, as earlier described.

Referring back to block 420, in embodiments, the alignment check may be performed by performing an AND operation on the memory page address and the memory page mask. Thus, when the memory page mask is set to zero, process 400 will proceed to block 422, resulting in only one new memory page allocated per page fault. This might be useful for discrete allocation, such as populating bound table pages for protected memory. Similarly, by setting memory page mask to minus 0XF000, process 400 will proceed to block 422, whenever memory page address is 64K aligned, likewise resulting in only one new memory page allocated per page fault. This might be useful, for instance, when allocating memory pages to host 64K aligned code segments loaded at runtime. Additionally, the alignment check may be effectively bypassed by setting the memory page mask to all ones. This might be useful for continuous allocation, such as heaps and stacks, ensuring the gap between the faulting address and the closest populated page within the same dynamic region will always be filled.

Referring back to block 454/456, as described earlier, in SGX embodiments, the application execution enclave may acknowledge the allocation of each new memory page by executing the EACCEPT instruction. In embodiments, the enclave may execute the EACCEPT instruction for the new allocated memory page that is farthest from the committed area first. For example, when extending by m pages a heap currently ending at $addr$ (i.e., requesting pages at $addr$, $addr+0x1000$, $addr+0x2000$, ..., $addr+(m-1)*0x1000$), the enclave may EACCEPT $addr+(m-1)*0x1000$ first, which will cause all m pages to be populated by in the first #PF, and thus avoid further #PFs by subsequent EACCEPTs.

Thus, a novel approach for dynamically allocating additional cache pages to an application execution enclave has been described. The approach has at least the following advantages:

- More efficient – The overhead of each additional memory page request is a memory page fault, which typically costs less than a system call (via the host application of the

application execution enclave).

- More secure – By avoiding the system call via the host application of the execution enclave, the number of paths into or out of the application execution enclave are reduced, thereby reducing the complexity of the enclave interface.

5 - Simpler code – code to otherwise deal with facilitating the system call via the host application of the application execution enclave is no longer necessary.

- More flexible – there are no inherent restrictions to the disclosed approach. The approach works in both normal and exception paths.

Figure 6 illustrates an example computer system that may be suitable for use to practice selected aspects of the present disclosure. As shown, computer 600 may include one or more processors or processor cores 602 with application execution enclave support, read-only memory (ROM) 603, and system memory 604. For the purpose of this application, including the claims, the term “processor” refers to a physical processor, and the terms “processors” and “processor cores” may be considered synonymous, unless the context clearly requires otherwise. Additionally, computer system 600 may include mass storage devices 606. Example of mass storage devices 606 may include, but are not limited to, tape drives, hard drives, compact disc read-only memory (CD-ROM) and so forth. Further, computer system 600 may include input/output devices 608 (such as display, keyboard, cursor control and so forth) and communication interfaces 610 (such as network interface cards, modems and so forth). The elements may be coupled to each other via system bus 612, which may represent one or more buses. In the case of multiple buses, they may be bridged by one or more bus bridges (not shown).

Each of these elements may perform its conventional functions known in the art. In particular, ROM 603 may include basic input/output system services (BIOS) 605, including the PM data package provision service, as earlier described. System memory 604 and mass storage devices 606 may be employed to store a working copy and a permanent copy of the programming instructions implementing the operations associated with applications 114 and/or OS runtime 112, including exception handler 128, as earlier described, collectively referred to as computational logic 622. The various elements may be implemented by assembler instructions supported by processor(s) 602 or high-level languages, such as, for example, C, that can be compiled into such instructions.

The number, capability and/or capacity of these elements 610-612 may vary, depending on whether computer system 600 is used as a mobile device, such as a wearable device, a smartphone, a computer tablet, a laptop and so forth, or a stationary device, such

as a desktop computer, a server, a game console, a set-top box, an infotainment console, and so forth. Otherwise, the constitutions of elements 610-612 are known, and accordingly will not be further described.

As will be appreciated by one skilled in the art, the present disclosure may be embodied as methods or computer program products. Accordingly, the present disclosure, in addition to being embodied in hardware as earlier described, may take the form of an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to as a “circuit,” “module” or “system.” Furthermore, the present disclosure may take the form of a computer program product embodied in any tangible or non-transitory medium of expression having computer-usable program code embodied in the medium. Figure 7 illustrates an example computer-readable non-transitory storage medium that may be suitable for use to store instructions that cause an apparatus, in response to execution of the instructions by the apparatus, to practice selected aspects of the present disclosure. As shown, non-transitory computer-readable storage medium 702 may include a number of programming instructions 704. Programming instructions 704 may be configured to enable a device, e.g., computer 600, in response to execution of the programming instructions, to implement (aspects of) OS runtime 112, e.g., exception handler 128, and/or applications 114. In alternate embodiments, programming instructions 704 may be disposed on multiple computer-readable non-transitory storage media 702 instead. In still other embodiments, programming instructions 704 may be disposed on computer-readable transitory storage media 702, such as, signals.

Any combination of one or more computer usable or computer readable medium(s) may be utilized. The computer-usable or computer-readable medium may be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a non-exhaustive list) of the computer-readable medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a transmission media such as those supporting the Internet or an intranet, or a magnetic storage device. Note that the computer-usable or computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured,

via, for instance, optical scanning of the paper or other medium, then compiled, interpreted, or otherwise processed in a suitable manner, if necessary, and then stored in a computer memory. In the context of this document, a computer-usable or computer-readable medium may be any medium that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device. The computer-usable medium may include a propagated data signal with the computer-usable program code embodied therewith, either in baseband or as part of a carrier wave. The computer usable program code may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc.

Computer program code for carrying out operations of the present disclosure may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

The present disclosure is described with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer-readable medium that can direct a computer or other programmable data processing apparatus to

function in a particular manner, such that the instructions stored in the computer-readable medium produce an article of manufacture including instruction means which implement the function/act specified in the flowchart and/or block diagram block or blocks.

5 The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

10 The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for
15 implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each
20 block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

The terminology used herein is for the purpose of describing particular
25 embodiments only and is not intended to be limiting of the disclosure. As used herein, the singular forms “a,” “an” and “the” are intended to include plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises” and/or “comprising,” when used in this specification, specific the presence of stated features, integers, steps, operations, elements, and/or components, but do not
30 preclude the presence or addition of one or more other features, integers, steps, operation, elements, components, and/or groups thereof.

Embodiments may be implemented as a computer process, a computing system or as an article of manufacture such as a computer program product of computer readable media. The computer program product may be a computer storage medium readable by a

computer system and encoding a computer program instructions for executing a computer process.

The corresponding structures, material, acts, and equivalents of all means or steps plus function elements in the claims below are intended to include any structure, material
5 or act for performing the function in combination with other claimed elements are specifically claimed. The description of the present disclosure has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the disclosure in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill without departing from the scope and spirit of the disclosure. The
10 embodiment was chosen and described in order to best explain the principles of the disclosure and the practical application, and to enable others of ordinary skill in the art to understand the disclosure for embodiments with various modifications as are suited to the particular use contemplated.

Referring back to Figure 6, for one embodiment, at least one of processors 602
15 may be packaged together with memory having aspects of OS runtime 112, e.g., exception handler 128. For one embodiment, at least one of processors 602 may be packaged together with memory having aspects of OS runtime 112, e.g., exception handler 128, to form a System in Package (SiP). For one embodiment, at least one of processors 602 may be integrated on the same die with memory having aspects of OS runtime 112, e.g.,
20 exception handler 128. For one embodiment, at least one of processors 602 may be packaged together with memory having aspects of OS runtime 112, e.g., exception handler 128, to form a System on Chip (SoC). For at least one embodiment, the SoC may be utilized in, e.g., but not limited to, a smartphone or computing tablet.

Thus various example embodiments of the present disclosure have been described
25 including, but are not limited to:

Example 1 may be an apparatus for computing, comprising: one or more processors with supports for application execution enclaves; memory coupled with the one or more processors to be organized into a plurality of memory pages; and an exception handler to be operated by the one or more processors to handle memory page fault
30 exceptions, wherein to handle memory page fault exceptions may include to handle a memory page fault triggered to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application of the apparatus.

Example 2 may be example 1, wherein the supports for application execution enclaves may include supports for an application execution enclave to acknowledge

acceptance of an allocation of one or more memory pages to the memory page cache of the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application may be triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or
5 triggered by an overflow of a heap or stack of the application execution enclave.

Example 3 may be example 2; wherein the vacant address may be a starting address of where the application execution enclave wants one or more additional memory pages to be mapped.

10 Example 4 may be example 2, wherein the exception handler, on handling the memory page fault triggered to request additional allocation of one or more memory pages to the memory page cache of the execution enclave of the application, may resume the application execution enclave to cause the acknowledgement to be retried.

Example 5 may be any one of examples 2 - 4, wherein the supports for application
15 execution enclaves may include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional memory pages; wherein the page exception handler may handle the memory page allocation request using the augmentation support.

Example 6 may be example 5, wherein the exception handler may further determine
20 whether a faulting address associated with a memory page fault points to a previously allocated memory page that has been paged out of the memory, and on determination that the faulting address points to a paged out memory page, may cause the paged out memory page to be paged back into the memory.

Example 7 may be example 5, wherein the exception handler may determine
25 whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is not within a dynamic address range of the application execution enclave, the exception handler may inject an exception to the faulting application, and resume the faulting application.

Example 8 may be example 5, wherein the exception handler may determine
30 whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is within a dynamic address range of the application execution enclave, the exception handler may retrieve at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range.

Example 9 maybe example 8, wherein on retrieval of the at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range, the exception handler may cause the augmentation support to allocate one or more memory pages to the memory page cache of the application
5 execution enclave using the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator, starting at the vacant address.

Example 10 maybe example 9, wherein the exception handler may further check for code/data section/segment alignment.

Example 11 maybe example 10, wherein when a result of the checking indicates
10 code/data section/segment is not aligned, the exception handler may further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected on of an upward or downward address direction.

Example 12 maybe example 10, wherein the retrieved at least one of a memory
15 page type, a memory address mask or a memory page growth direction indicator may include the memory address mask, and the exception handler may further compare the starting address with the memory page mask to check for code/data section/segment alignment.

Example 13 maybe example 12, wherein based at least in part on a result of the
20 comparison of the starting address with the memory page mask, the exception handler may further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, based at least in part on the memory page growth direction indicator.

Example 14 maybe example 10, wherein the exception handler may further
25 determine whether the memory page fault is caused by a write attempt; on determination that the memory page fault is caused by a write attempt, the exception handler may inject an exception into faulting application, and thereafter resume the faulting application; and on determination that the memory page fault is not caused by a write attempt, the
30 exception handler may resume the faulting application.

Example 15 may be a method for computing, comprising: receiving, by an exception handler of a computing device, a memory page fault exception triggered to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application operating on one or more processors of the computing

device, wherein the one or more processors support application execution enclaves; and handling, by the exception handler, the memory page fault exception, causing the requested one or more memory pages to be allocated to the memory page cache of the execution enclave.

5 Example 16 may be example 15, wherein the supports for application execution enclaves may include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave
10 of an application is triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave.

 Example 17 may be example 16; wherein the vacant address may be a starting address of where the application execution enclave wants one or more additional memory
15 pages to be mapped.

 Example 18 may be example 16, further comprising, on servicing the memory page fault triggered to request additional allocation of one or more memory pages to the memory page cache of the execution enclave of the application, resuming the application execution enclave to cause the acknowledgement to be retried.

20 Example 19 may be any one of examples 16 - 18, wherein the supports for application execution enclaves may include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional memory pages; wherein handling the memory page allocation request may comprise using the augmentation support.

25 Example 20 may be example 19, further comprising determining whether a faulting address associated with a memory page fault points to a previously allocated memory page that has been paged out of the memory, and on determining that the faulting address points to a paged out memory page, causing the paged out memory page to be paged back into the memory.

30 Example 21 may be example 19, further comprising determining whether the starting address is within a dynamic address range of the application execution enclave, and on determining that the vacant address is not within a dynamic address range of the application execution enclave, injecting an exception to the faulting application, and resuming the faulting application.

Example 22 maybe example 19, further comprising determining whether the starting address is within a dynamic address range of the application execution enclave, and on determining that the vacant address is within a dynamic address range of the application execution enclave, retrieving at least one of a memory page type, a memory
5 address mask or a memory page growth direction indicator of the dynamic address range.

Example 23 maybe example 22, wherein on retrieving the at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range, causing the augmentation support to allocate one or more memory pages to the memory page cache of the application execution enclave using the retrieved at
10 least one of a memory page type, a memory address mask or a memory page growth direction indicator, starting at the vacant address.

Example 24 maybe example 21, further comprising checking for code/data section/segment alignment.

Example 25 maybe example 24, further comprising when a result of the checking
15 indicates code/data section/segment is not aligned, successively adding or subtracting the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected on of an upward or downward address direction.

Example 26 maybe example 24, wherein the retrieved at least one of a memory
20 page type, a memory address mask or a memory page growth direction indicator may include the memory address mask, and handling may comprise comparing the starting address with the memory page mask to check for code/data section/segment alignment.

Example 27 maybe example 26, further comprising based at least in part on a result of the comparison of the starting address with the memory page mask, successively adding
25 or subtracting the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, based at least in part on the memory page growth direction indicator.

Example 28 maybe example 24, further comprising determining whether the memory page fault is caused by a write attempt; on determining that the memory page
30 fault is caused by a write attempt, injecting an exception into faulting application, and thereafter resuming the faulting application; and on determining that the memory page fault is not caused by a write attempt, resuming the faulting application.

Example 29 may be one or more computer-readable media comprising instructions that cause a computing device, in response to execution of the instructions by one or more

processors of the computing device, to provide an exception handler, to: receive a memory page fault exception triggered to request additional allocation of one or more memory pages to an execution enclave of an application operating on the one or more processors, wherein the one or more processors support application execution enclaves; and handle the memory page fault exception, wherein to handle may include to cause the requested one or more memory pages to be allocated to the memory page cache of the execution enclave.

Example 30 may be example 29, wherein the supports for application execution enclaves may include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application is triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave.

Example 31 may be example 30; wherein the vacant address may be a starting address of where the application execution enclave wants one or more additional memory pages to be mapped.

Example 32 may be example 30, wherein the exception handler, on servicing the memory page fault triggered to request additional allocation of one or more memory pages to the memory page cache of the execution enclave of the application, may resume the application execution enclave to cause the acknowledgement to be retried.

Example 33 may be any one of examples 30 - 32, wherein the supports for application execution enclaves may include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional memory pages; wherein the page exception handler handles the memory page allocation request using the augmentation support.

Example 34 may be example 33, wherein the exception handler may further determine whether a faulting address associated with a memory page fault points to a previously allocated memory page that has been paged out of the memory, and on determination that the faulting address points to a paged out memory page, cause the paged out memory page to be paged back into the memory.

Example 35 may be example 33, wherein the exception handler may determine whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is not within a dynamic address

range of the application execution enclave, the exception handler may inject an exception to the faulting application, and resume the faulting application.

Example 36 may be example 33, wherein the exception handler may determine whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is within a dynamic address range of the application execution enclave, the exception handler may retrieve at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range.

Example 37 may be example 36, wherein on retrieval of the at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range, the exception handler may cause the augmentation support to allocate one or more memory pages to the memory page cache of the application execution enclave using the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator, starting at the vacant address.

Example 38 may be example 35, wherein the exception handler may further check for code/data section/segment alignment.

Example 39 may be example 38, when a result of the checking indicates code/data section/segment is not aligned, the exception handler may further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected one of an upward or downward address direction.

Example 40 may be example 38, wherein the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator may include the memory address mask, and the exception handler may further compare the starting address with the memory page mask to check for code/data section/segment alignment.

Example 41 may be example 40, wherein based at least in part on a result of the comparison of the starting address with the memory page mask, the exception handler may further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, based at least in part on the memory page growth direction indicator.

Example 42 may be example 38, wherein the exception handler may further determine whether the memory page fault is caused by a write attempt; on determination

that the memory page fault is caused by a write attempt, the exception handler may inject an exception into faulting application, and thereafter resume the faulting application; and on determination that the memory page fault is not caused by a write attempt, the exception handler may resume the faulting application.

5 Example 43 may be an apparatus for computing, comprising: exception receiving means for receiving a memory page fault exception triggered to request additional allocation of one or more cache pages to a memory page cache of an execution enclave of an application operating on one or more processors of the computing device, wherein the one or more processors support application execution enclaves; and exception handling
10 means for handling the memory page fault exception, causing the requested one or more memory pages to be allocated to the execution enclave.

 Example 44 may be example 43, wherein the supports for application execution enclaves may include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of
15 the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application is triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave.

20 Example 45 may be example 44; wherein the vacant address may be a starting address of where the application execution enclave wants one or more additional memory pages to be mapped.

 Example 46 may be example 44, further comprising means for resuming the application execution enclave to cause the acknowledgement to be retried, on servicing the
25 memory page fault triggered to request additional allocation of one or more memory pages to the memory page cache of the execution enclave of the application.

 Example 47 may be any one of examples 44 - 46, wherein the supports for application execution enclaves may include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional
30 memory pages; wherein exception handling means may comprise means for using the augmentation support.

 Example 48 may be example 47, further comprising means for determining whether a faulting address associated with a memory page fault points to a previously allocated memory page that has been paged out of the memory, and means for causing the paged out

memory page to be paged back into the memory, on determining that the faulting address points to a paged out memory page.

Example 49 may be example 47, further comprising means for determining whether the starting address is within a dynamic address range of the application execution enclave, and means for injecting an exception to the faulting application, and resuming the faulting application, on determining that the vacant address is not within a dynamic address range of the application execution enclave.

Example 50 may be example 47, further comprising means for determining whether the starting address is within a dynamic address range of the application execution enclave, and means for retrieving at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range.

Example 51 may be example 50, further comprising means for causing, on determining that the vacant address is within a dynamic address range of the application execution enclave, the augmentation support to allocate one or more memory pages to the memory page cache of the application execution enclave using the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator, starting at the vacant address, on retrieving the at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range.

Example 52 may be example 51, further comprising means for checking for code/data section/segment alignment.

Example 53 may be example 52, further comprising means for successively adding or subtracting the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected one of an upward or downward address direction, when a result of the checking indicates code/data section/segment is not aligned.

Example 54 may be example 52, wherein the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator may include the memory address mask, and exception handling means may comprise means for comparing the starting address with the memory page mask to check for code/data section/segment alignment.

Example 55 may be example 54, further comprising means for successively adding or subtracting the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution

enclave, in accordance with at least in part on the memory page growth direction indicator, based at least in part on a result of the comparison of the starting address with the memory page mask.

Example 56 may be example 52, further comprising means for determining whether
5 the memory page fault is caused by a write attempt; means for injecting an exception into faulting application, and thereafter resuming the faulting application, on determining that the memory page fault is caused by a write attempt; and means for resuming the faulting application, on determining that the memory page fault is not caused by a write attempt.

It will be apparent to those skilled in the art that various modifications and
10 variations can be made in the disclosed embodiments of the disclosed device and associated methods without departing from the spirit or scope of the disclosure. Thus, it is intended that the present disclosure covers the modifications and variations of the embodiments disclosed above provided that the modifications and variations come within the scope of any claims and their equivalents.

15

Claims

What is claimed is:

1. An apparatus for computing, comprising:
one or more processors with supports for application execution enclaves;
5 memory coupled with the one or more processors to be organized into a plurality of memory pages; and
an exception handler to be operated by the one or more processors to handle memory page fault exceptions, wherein to handle memory page fault exceptions includes to handle a memory page fault triggered to request additional allocation of one or
10 more memory pages to a memory page cache of an execution enclave of an application of the apparatus.
2. The apparatus of claim 1, wherein the supports for application execution enclaves include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of the
15 application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application is triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave.
- 20 3. The apparatus of claim 2; wherein the vacant address is a starting address of where the application execution enclave wants one or more additional memory pages to be mapped.
4. The apparatus of claim 2, wherein the exception handler, on handling the memory page fault triggered to request additional allocation of one or more memory pages
25 to the memory page cache of the execution enclave of the application, is to resume the application execution enclave to cause the acknowledgement to be retried.
5. The apparatus of any one of claims 2 - 4, wherein the supports for application execution enclaves include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional
30 memory pages; wherein the page exception handler handles the memory page allocation request using the augmentation support.
6. The apparatus of claim 5, wherein the exception handler is to further determine whether a faulting address associated with a memory page fault points to a

previously allocated memory page that has been paged out of the memory, and on determination that the faulting address points to a paged out memory page, cause the paged out memory page to be paged back into the memory.

7. The apparatus of claim 5, wherein the exception handler is to determine
5 whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is not within a dynamic address range of the application execution enclave, the exception handler is to inject an exception to the faulting application, and resume the faulting application.

8. The apparatus of claim 5, wherein the exception handler is to determine
10 whether the starting address is within a dynamic address range of the application execution enclave, and on determination that the vacant address is within a dynamic address range of the application execution enclave, the exception handler is to retrieve at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range.

9. The apparatus of claim 8, wherein on retrieval of the at least one of a
15 memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range, the exception handler is to cause the augmentation support to allocate one or more memory pages to the memory page cache of the application execution enclave using the retrieved at least one of a memory page type, a memory
20 address mask or a memory page growth direction indicator, starting at the vacant address.

10. The apparatus of claim 9, wherein the exception handler is to further check for code/data section/segment alignment.

11. The apparatus of claim 10, wherein when a result of the checking
25 indicates code/data section/segment is not aligned, the exception handler is to further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected one of an upward or downward address direction.

12. The apparatus of claim 10, wherein the retrieved at least one of a memory
30 page type, a memory address mask or a memory page growth direction indicator includes the memory address mask, and the exception handler is to further compare the starting address with the memory page mask to check for code/data section/segment alignment.

13. The apparatus of claim 12, wherein based at least in part on a result of the comparison of the starting address with the memory page mask, the exception handler

is to further successively add or subtract the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, based at least in part on the memory page growth direction indicator.

5 14. The apparatus of claim 10, wherein the exception handler is to further determine whether the memory page fault is caused by a write attempt; on determination that the memory page fault is caused by a write attempt, the exception handler is to inject an exception into faulting application, and thereafter resume the faulting application; and
10 exception handler is to resume the faulting application.

 15. A method for computing, comprising:
 receiving, by an exception handler of a computing device, a memory page
 fault exception triggered to request additional allocation of one or more memory pages to
 a memory page cache of an execution enclave of an application operating on one or
15 more processors of the computing device, wherein the one or more processors
 support application execution enclaves; and

 handling, by the exception handler, the memory page fault exception, causing
 the requested one or more memory pages to be allocated to the memory page cache of
 the execution enclave.

20 16. The method of claim 15, wherein the supports for application execution enclaves include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an
25 execution enclave of an application is triggered by issuance, by the application execution enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave.

 17. The method of claim 16; wherein the vacant address is a starting address of
30 where the application execution enclave wants one or more additional memory pages to be mapped; and the method further comprising, on servicing the memory page fault triggered to request additional allocation of one or more memory pages to the memory page cache of the execution enclave of the application, resuming the application execution enclave to cause the acknowledgement to be retried.

18. The method of claim 16, wherein the supports for application execution enclaves include supports to augment memory page allocation to a memory page cache of an application execution enclave with one or more additional memory pages; wherein handling the memory page allocation request comprises using the augmentation support.

19. The method of claim 18, further comprising determining whether the starting address is within a dynamic address range of the application execution enclave, and on determining that the vacant address is not within a dynamic address range of the application execution enclave, injecting an exception to the faulting application, and resuming the faulting application; and determining whether the starting address is within a dynamic address range of the application execution enclave, and on determining that the vacant address is within a dynamic address range of the application execution enclave, retrieving at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range;

wherein on retrieving the at least one of a memory page type, a memory address mask or a memory page growth direction indicator of the dynamic address range, causing the augmentation support to allocate one or more memory pages to the memory page cache of the application execution enclave using the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator, starting at the vacant address.

20. The method of claim 19, further comprising checking for code/data section/segment alignment.

21. The method of claim 20, further comprising when a result of the checking indicates code/data section/segment is not aligned, successively adding or subtracting the memory page size from the starting address to successively allocate the one or more memory pages to the memory page cache of the application execution enclave, in a selected one of an upward or downward address direction.

22. The method of claim 20, wherein the retrieved at least one of a memory page type, a memory address mask or a memory page growth direction indicator includes the memory address mask, and handling comprises comparing the starting address with the memory page mask to check for code/data section/segment alignment; wherein the method further comprises based at least in part on a result of the comparison of the starting address with the memory page mask, successively adding or subtracting the memory page size from the starting address to successively allocate the one or

more memory pages to the memory page cache of the application execution enclave, based at least in part on the memory page growth direction indicator.

23. One or more computer-readable media comprising instructions that cause a computing device, in response to execution of the instructions by one or more processors
5 of the computing device, to provide an exception handler, to perform any one of the methods of claims 15 - 22.

24. An apparatus for computing, comprising:
exception receiving means for receiving a memory page fault exception triggered to request additional allocation of one or more cache pages to a memory page cache of an
10 execution enclave of an application operating on one or more processors of the computing device, wherein the one or more processors support application execution enclaves; and
exception handling means for handling the memory page fault exception, causing the requested one or more memory pages to be allocated to the execution enclave.

25. The apparatus of claim 24, wherein the supports for application
15 execution enclaves include supports for an application execution enclave to acknowledge acceptance of an allocation of one or more memory pages to the memory page cache of the application execution enclave; wherein the memory page fault to request additional allocation of one or more memory pages to a memory page cache of an execution enclave of an application is triggered by issuance, by the application execution
20 enclave, of an acknowledgement to an allocation of one or more memory pages at a vacant address, or triggered by an overflow of a heap or stack of the application execution enclave; and

wherein the supports for application execution enclaves further include supports to augment memory page allocation to a memory page cache of an application
25 execution enclave with one or more additional memory pages; wherein exception handling means comprises means for using the augmentation support.

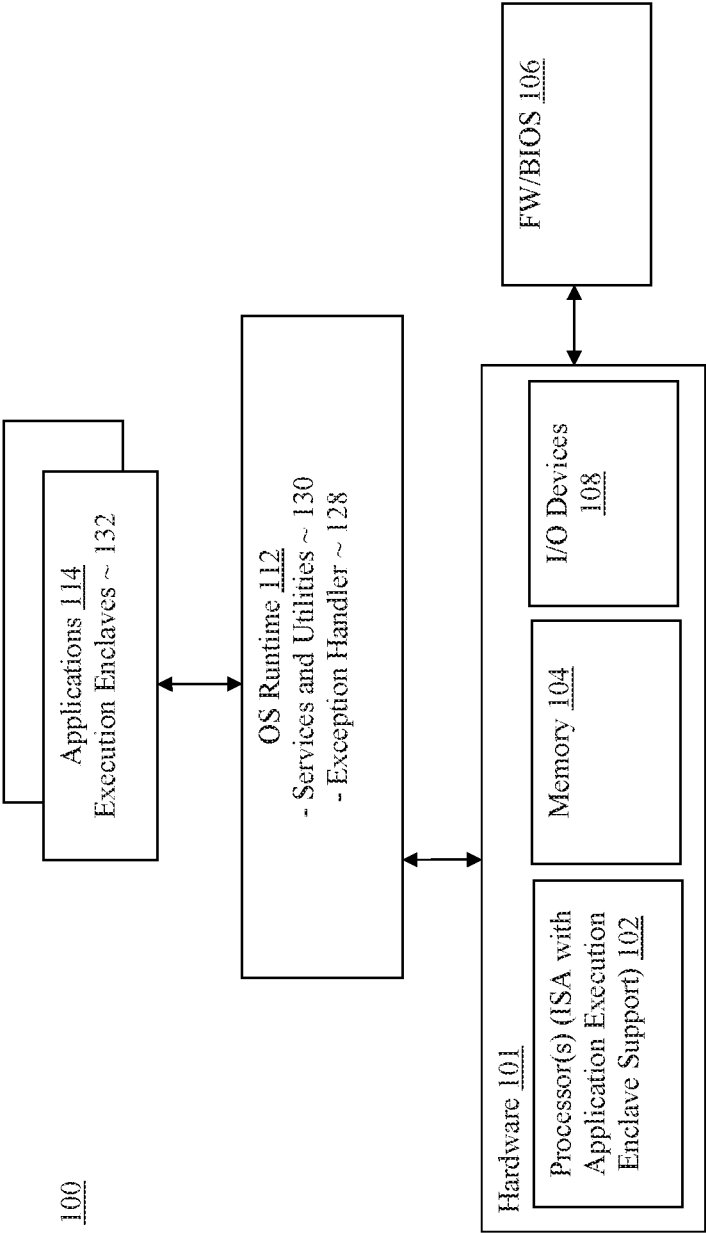


Figure 1

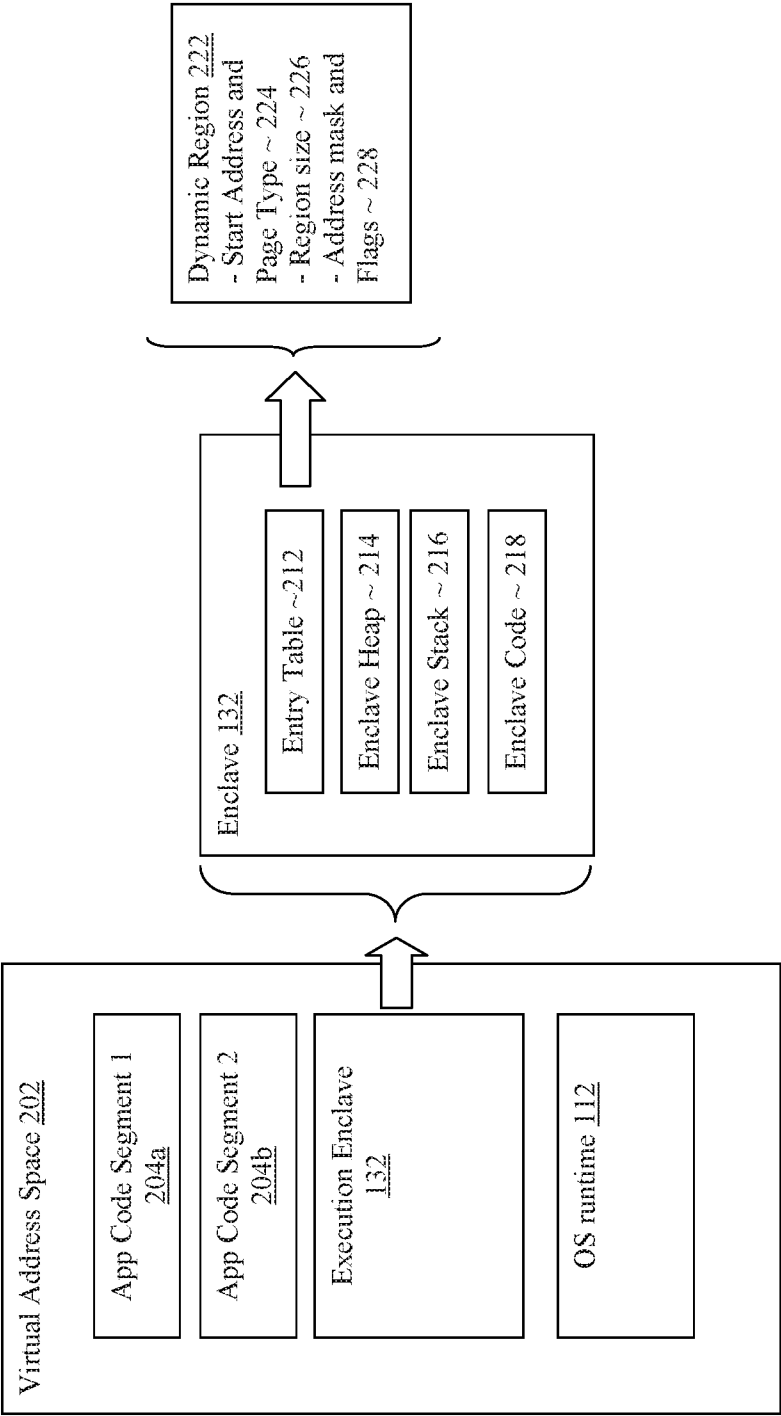


Figure 2

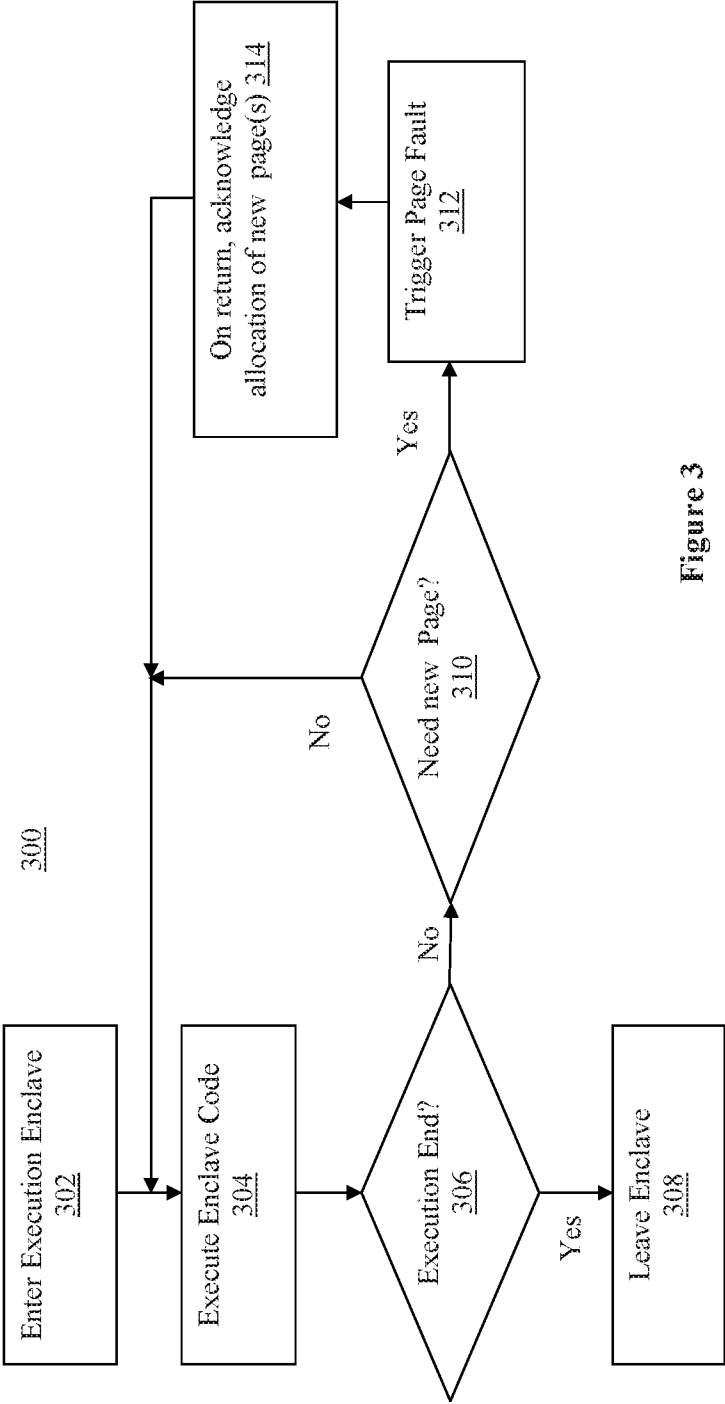
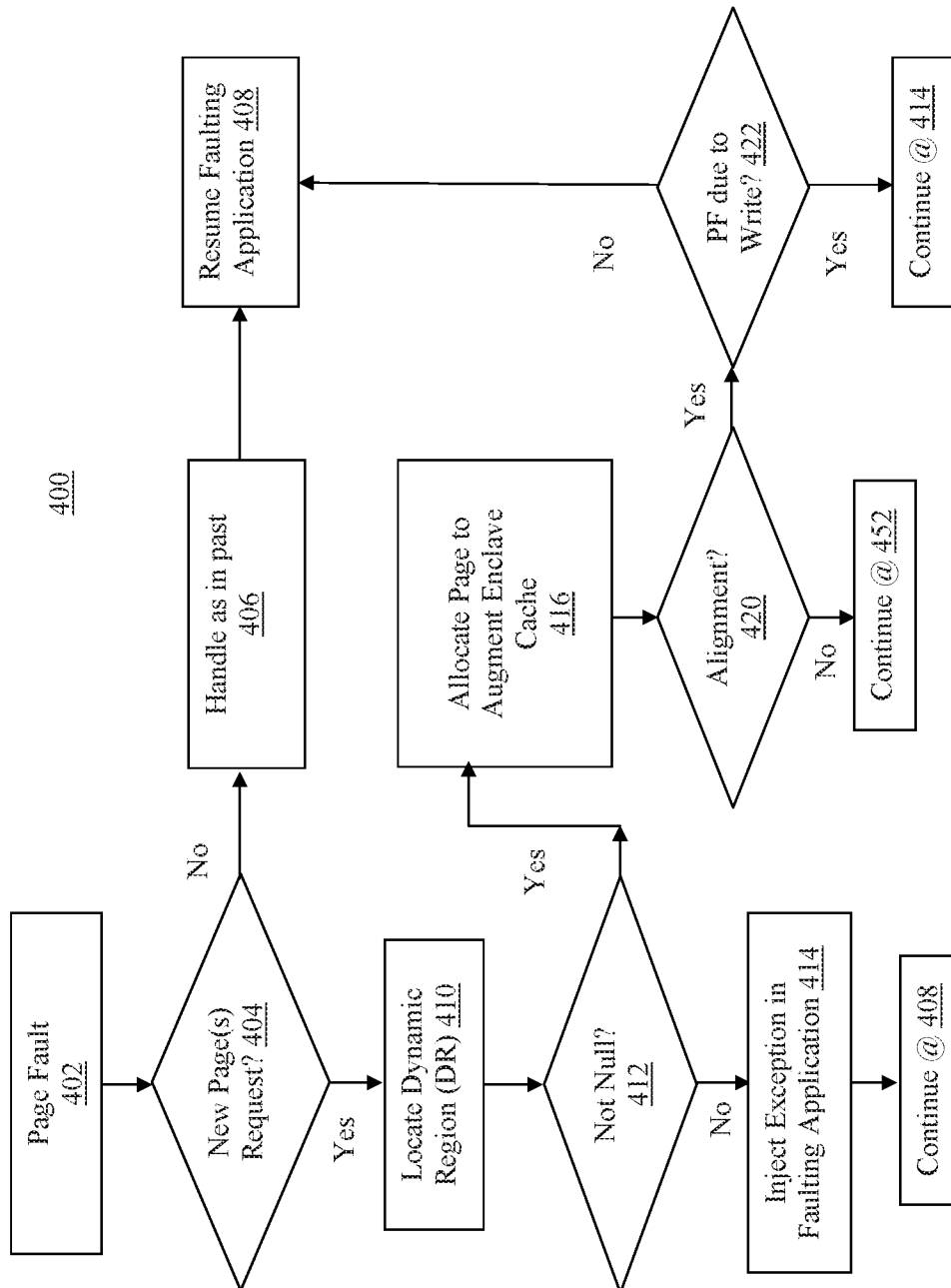


Figure 3

**Figure 4**

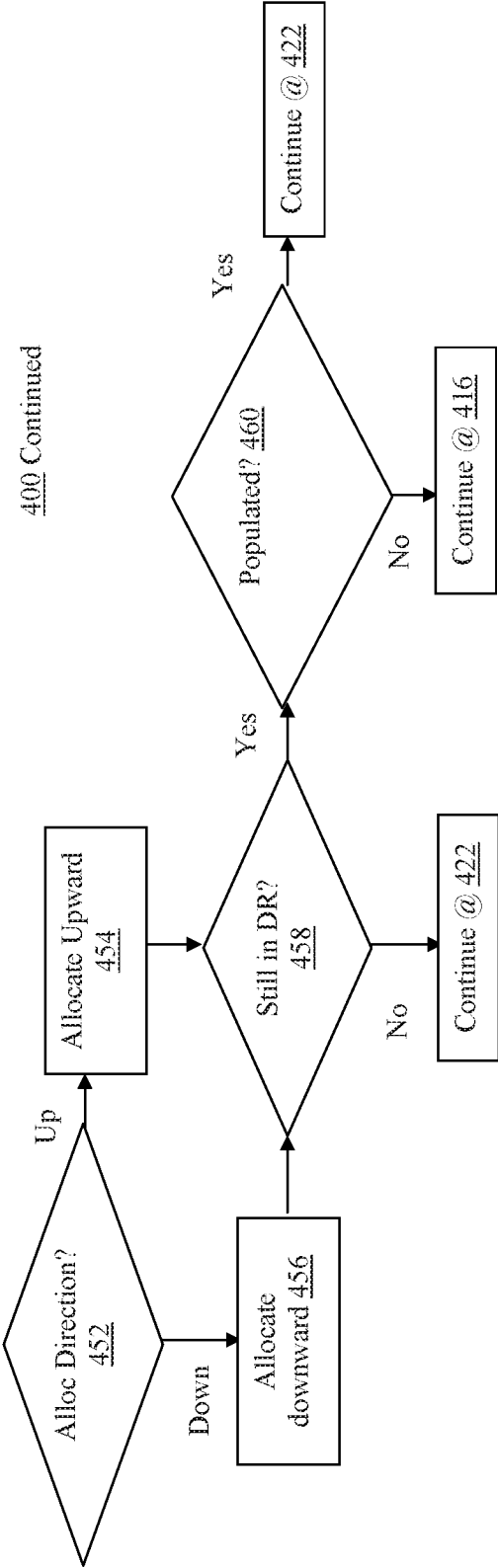


Figure 5

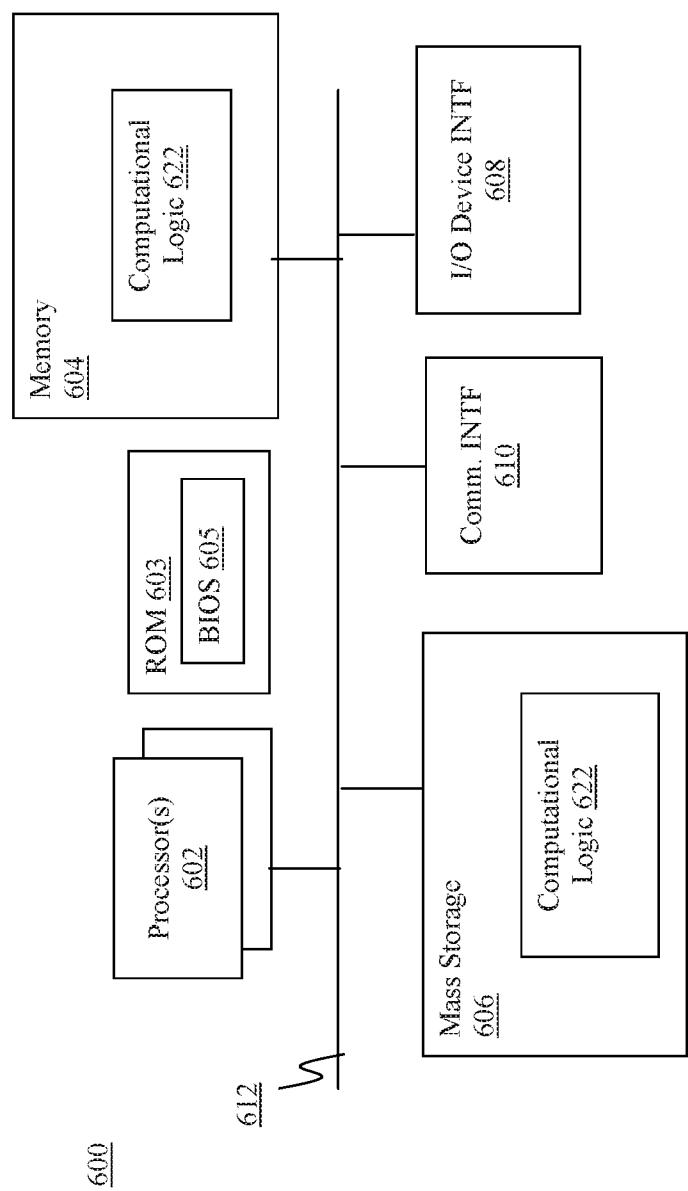


Figure 6

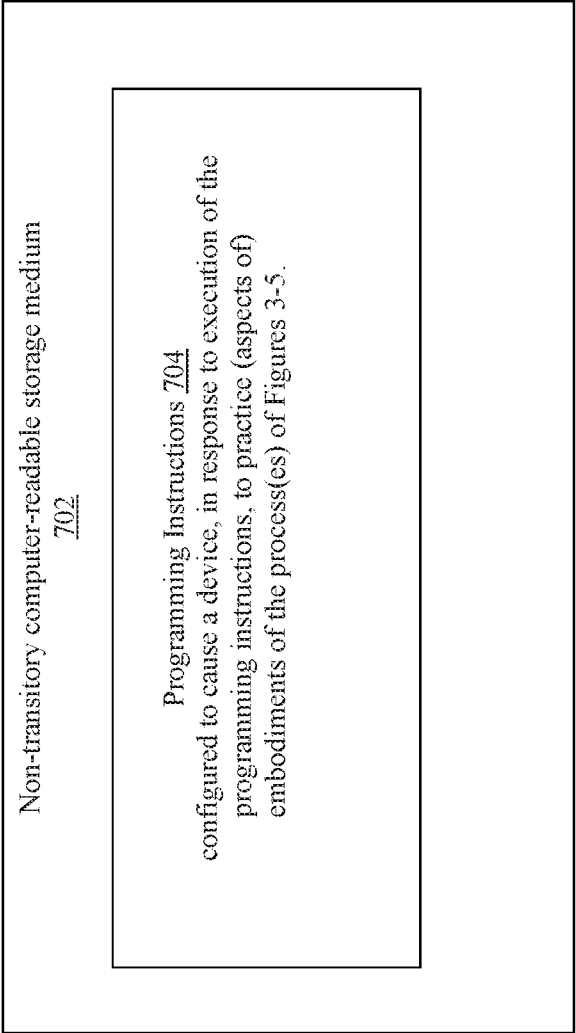


Figure 7

A. CLASSIFICATION OF SUBJECT MATTER**G06F 21/53(2013.01)i, G06F 21/60(2013.01)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 21/53; G06F 12/02; G06F 9/30; G06F 12/14; G06F 9/455; G06F 9/38; G06F 12/08; G06F 21/60

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: application execution enclave, memory page cache management, memory page fault exception, exception handler, additional allocation

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 9128849 B2 (IAN HENDRY et al.) 08 September 2015	1-5, 15-18, 23-25
A	See column 8, lines 62-64, column 11, lines 25-57; and claims 7-8.	6-14, 19-22
Y	US 2014-0189326 A1 (REBEKAH LESLIE et al.) 03 July 2014	1-5, 15-18, 23-25
	See paragraph [0016]; and claim 10.	
A	US 2014-0237199 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 21 August 2014	1-25
	See paragraphs [0053]-[0073]; and claims 1, 4-5, 8.	
A	US 2013-0232315 A1 (CHEN TIAN et al.) 05 September 2013	1-25
	See paragraphs [0017]-[0022]; and claims 1-4, 8.	
A	WO 2014-139125 A1 (INTEL CORPORATION) 18 September 2014	1-25
	See page 6, line 7 - page 8, line 4; and claims 1-3.	



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

04 November 2016 (04.11.2016)

Date of mailing of the international search report

04 November 2016 (04.11.2016)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

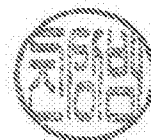
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

CHIN, Sang Bum

Telephone No. +82-42-481-8398



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/044319

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 9128849 B2	08/09/2015	US 2011-0252200 A1	13/10/2011
US 2014-0189326 A1	03/07/2014	CN 104798053 A	22/07/2015
		US 2016-0202976 A1	14/07/2016
		WO 2014-105161 A1	03/07/2014
US 2014-0237199 A1	21/08/2014	EP 2324424 A1	25/05/2011
		US 2010-0030975 A1	04/02/2010
		US 8719541 B2	06/05/2014
		WO 2010-012657 A1	04/02/2010
US 2013-0232315 A1	05/09/2013	KR 10-2013-0100689 A	11/09/2013
WO 2014-139125 A1	18/09/2014	CN 104969183 A	07/10/2015
		EP 2972795 A1	20/01/2016
		JP 2016-510472 A	07/04/2016
		KR 10-2015-0103216 A	09/09/2015
		US 2015-0324240 A1	12/11/2015