



- (51) International Patent Classification:
G06F 9/54 (2006.01)
- (21) International Application Number:
PCT/US2016/051872
- (22) International Filing Date:
15 September 2016 (15.09.2016)
- (25) Filing Language:
English
- (26) Publication Language:
English
- (30) Priority Data:
62/252,692 9 November 2015 (09.11.2015) US
15/265,267 14 September 2016 (14.09.2016) US
- (71) Applicant: NEC LABORATORIES AMERICA, INC.
[US/US]; 4 Independence Way, Suite 200, Princeton, NJ
08540 (US).
- (72) Inventors: ZHANG, Hui; 69 Southfield Road, Princeton
Junction, NJ 08550 (US). JIANG, Guofei; 5 Danby Court,
Princeton, NJ 08540 (US). RHEE, Junghwan; 115 Cas-
cade Court, Princeton, NJ 08540 (US). ARORA, Nipun;
3205 Fox Run Drive, Plainsboro, NJ 08536 (US).
- (74) Agent: BITETTO, James, J.; TUTUNJIAN & BITETTO,
P.C., 425 Broadhollow Road, Suite 302, Melville, NY
11747 (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,

[Continued on next page]

(54) Title: REQUEST PROFILE IN MULTI-THREADED SERVICE SYSTEMS WITH KERNEL EVENTS

(57) Abstract: Methods and systems for profiling requests include generating request units (202) based on collected kernel events that include complete request units and half-open request units. The generated request units are sequenced (204) based on a causality relationship set that describes causality relationships between kernel events.

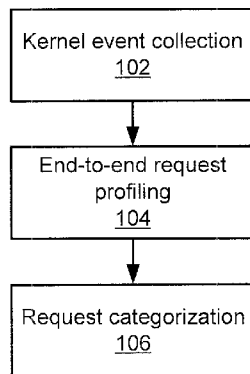


FIG. 1

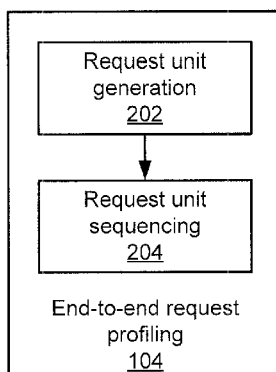


FIG. 2



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, KW, 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.

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:

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH,

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

REQUEST PROFILE IN MULTI-THREADED SERVICE SYSTEMS WITH KERNEL EVENTS

RELATED APPLICATION INFORMATION

[0001] This application claims priority to U.S. Provisional Application No. 62/252,692 filed on November 9, 2015, incorporated herein by reference in its entirety.

BACKGROUND

Technical Field

[0002] The present invention relates to multi-threaded request profiling and, more particularly, to using end-to-end request profiling that builds request traces based on low-level kernel events.

Description of the Related Art

[0003] As software complexity increases, predicting the internal states of all relevant components becomes increasingly difficult when unexpected problems occur. Retracing the history of execution flows is cumbersome and the overwhelming number of different hardware and software combinations, different workload characteristics, and different end user usage patterns further complicate the problem.

[0004] Existing systems for request processing path discovery use a variety of approaches, for example by monitoring thread and communication activities of the components of a system. Existing systems are often dependent on particular system implementations and specific software being present in the target system.

SUMMARY

[0005] A method for profiling requests includes generating request units based on collected kernel events that include complete request units and half-open request units using a processor. The generated request units are sequenced based on a causality relationship set that describes causality relationships between kernel events.

[0006] A system for profiling requests includes a request unit generating module that includes a processor configured to generate request units based on collected kernel events that include complete request units and half-open request units. A request unit sequencing module is configured to sequence the generated request units based on a causality relationship set that describes causality relationships between kernel events.

[0007] These and other features and advantages will become apparent from the following detailed description of illustrative embodiments thereof, which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF DRAWINGS

[0008] The disclosure will provide details in the following description of preferred embodiments with reference to the following figures wherein:

[0009] FIG. 1 is a block/flow diagram of a method of request profiling in accordance with embodiments of the present invention;

[0010] FIG. 2 is a block/flow diagram of a method of request unit generation in accordance with embodiments of the present invention;

[0011] FIG. 3 is a block/flow diagram of a method of request unit slicing in accordance with embodiments of the present invention;

[0012] FIG. 4 is a block/flow diagram of a method of searching virtual complete request units in accordance with embodiments of the present invention;

[0013] FIG. 5 is a block/flow diagram of a method of request unit sequencing in accordance with embodiments of the present invention;

[0014] FIG. 6 is a block/flow diagram of a method of request unit sequencing in accordance with embodiments of the present invention;

[0015] FIG. 8 is a block diagram of a request profiling system in accordance with embodiments of the present invention; and

[0016] FIG. 9 is a block diagram of a processing system in accordance with embodiments of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0017] Embodiments of the present principles provide precise request processing path discovery. While the present embodiments are discussed with respect to distributed applications, it should be understood that the present principles may be applied to any appropriate processing system. A given request unit may belong to service processes distributed over multiple machines, to processes residing on a same device, or to multiple threads within a single process. The present embodiments reconstruct the kernel execution traces of a request in a service system by instrumenting core kernel functions, including system-level operating system calls.

[0018] Two types of request units are defined based on a configurable marking event set. The collected kernel event traces are sliced, transforming the event traces into multiple per-thread request units based on the event set. A graph search builds virtual request units across threads in a thread pool based on their per-thread request units. All request units invoked by the same request are grouped together based on event causality. In this manner, the present embodiments target low-overhead, end-to-end

tracing captures the flow (including the path and timing) of individual requests within and across the components of a distributed service system.

[0019] Referring now in detail to the figures in which like numerals represent the same or similar elements and initially to FIG. 1, a method for request analysis is shown. Block 102 performs kernel event collection, collecting the kernel event traces from a target system. The kernel event traces can be obtained from individual service machines by instrumenting core kernel functions, including system-level operating system calls as well by other routes.

[0020] Block 104 performs end-to-end request profiling by analyzing the kernel event traces collected by block 102. Block 104 constructs end-to-end request traces which include kernel events belonging to service processes distributed over multiple machines, processes residing on a single device, or multiple threads within a single process (e.g., in the case of a pool of threads executing a single request cooperatively and collectively).

[0021] The terms “cooperative” and “collective” refer to the group behavior characteristics of threads in a system. Cooperative behavior refers to a request that is processed by more than one thread cooperatively. In cooperative behavior, one thread may, for example, accept a connection while a worker thread handles the sending and receiving of packets. Collective behavior refers to a group of threads/processes taking a similar role and alternating in the handling of requests in a pool. A given request may involve multiple threads in a non-deterministic way.

[0022] Block 106 categorizes the request traces output by block 104. The traces may be categorized by, e.g., grouping requests based on the marking kernel events used by block 104. Other categorizing processes may use, e.g., data mining clustering to learn categories.

[0023] The output, end-to-end request traces that capture the flow (i.e., the path and timing) of individual requests across the components of a distributed system, may be used for many different purposes. One such purpose is for debugging during and after development of the software, where the request traces make it possible to determine where bugs occur and how to address them. Another potential use for request traces is bottleneck analysis, determining where performance bottlenecks occur. The request traces can also be used to determine whether a given device was involved in processing a given request and how much time was spent on each component. Machine learning and data mining can be used to detect anomalies with request traces, recognizing when abnormal activity occurs.

[0024] In addition, automated system management based on the request traces can be implemented, where administrative actions are taken based on request traces. Exemplary administrative actions include system resource allocation, for example by providing additional computing resources to devices that have a high system load due to handling request units; security mitigation actions, for example raising a security level and issuing security alerts when anomalous activity is detected; and environmental control actions, for example increasing cooling or changing coolant flow distribution when particular device are under heavy load.

[0025] Referring now to FIG. 2, additional detail on the end-to-end request profiling of block 104 is shown. In particular, block 202 generates two types of request units from the collected kernel events based on a configurable marking event set. Block 202 also builds virtual request units across threads in a thread pool based on their per-thread request units. Block 204 constructs full-path request traces by composing relevant request units based on the causality relationship between request units and which is defined on the marking event set and temporal relationships.

[0026] Referring now to FIG. 3, additional detail on the request unit generation of block 202 is shown. In particular, block 302 pre-processes the collected kernel events. The kernel events are organized by their per-thread event sequence ordered by the time when the kernel events occur. This can be accomplished based on context switch kernel events which tell the time at which a thread or process runs on a processor. The original kernel events include the information on when and at which processor the kernel events occur. After pre-processing, a 5-tuple event record is generated to present each kernel event. An exemplary 5-tuple format is: <owner ID, time_begin, time_end, event type, event data>.

[0027] The term “owner ID” refers to the identity of the event owner, which may include a machine name (e.g., the machine MAC or IP address), a process ID (e.g., the thread identifier if the event is generated by a thread), and optionally a process name. This identifier describes at which machine and by which process the event is generated.

[0028] The term “time_begin” refers to the time when the kernel event starts.

[0029] The term “time_end” refers to the time when the kernel event ends.

[0030] The term “event type” refers to the kernel event type. Examples of kernel event types include, but are not limited to, network events (e.g., TCP_SEND and TCP_RECV), process events (e.g., PROCESS_CREATE), and system call events. Application program execution events such as entering or exiting an application function can also be collected as kernel events through special logging tools. Special event types such as AP_FUNCTION_ENTER and AP_FUNCTION_EXIT can be defined for those application events.

[0031] The term “event data” refers to any extra information associated with kernel event types. Examples of event data include, but are not limited to, network event data (e.g., network flow data for TCP_SEND and TCP_RECV such as a destination IP

address, a source IP port, a destination IP port, packet size, etc.), inter-process communication event data (e.g., <ID, size> for PIPE_WRITE and PIPE_READ), process event data (e.g., child process identification for PROCESS_CREATE), system call event data (e.g., system call function names such as read and write), and application event data (e.g., function identifiers for AP_FUNCTION_ENTER and AP_FUNCTION_EXIT).

[0032] The output of block 302 is a set of threads described by owner IDs, with each thread having a time-ordered event record sequence where the event records have the same owner ID as the thread.

[0033] Block 304 defines a marking event set that includes request unit marks represented by a 4-tuple record: <begin_event_type, end_event_type, owner_filter, event_data_filter>.

[0034] The term “begin_event_type” refers to the event type that the first event of a request unit must exactly match. Examples include, but are not limited to, network event types (e.g., TCP_RECV), inter-process communication event types (e.g., PIPE_WRITE), process event types (e.g., PROCESS_CREATE), system call event types (e.g., KWAKEUP), and application event types (e.g., AP_FUNCTION_ENTER).

[0035] The term “end_event_type” refers to the event type that the last event of a request unit must exactly match. As with begin_event_type, examples include but are not limited to network event types, inter-process communication event types, process event types, system call event types, and application event types.

[0036] The term “owner_filter” refers to the owner ID that the first and last events of a request unit must partially or exactly match. Examples include, but are not limited to, a specific thread name (e.g., httpd), a specific thread ID number, or a wild card (e.g., ‘*’, which matches any owner ID).

[0037] The term “event_data_filter” refers to event data that the first and last events of a request unit must partially or exactly match. Examples include, but are not limited to, network event data (e.g., [source IP port = 80] for TCP_SEND or TCP_RECV), inter-process communication event data (e.g., a wild card for PIPE_WRITE or PIPE_READ), and application event data (e.g., [Function ID = 1000] for AP_FUNCTION_ENTER and AP_FUNCTION_EXIT). A wild card in this field matches any event data. The event data filters can be different for the first and last events.

[0038] The output of block 304 is a set of request unit marks which will be used to detect the beginning and ending of request units in the event record sequences from block 302.

[0039] Block 306 slices request units. One request unit is defined through a time-ordered event sequence data structure $\langle S \rangle$, where S is an event record sub-sequence of one thread. Slicing keeps track of two types of request units: half-open request units which have either the beginnings or endings detected, but not both, and complete request units which have both the beginnings and the endings detected. Block 306 scans one event after another in a thread’s event sequence, checks the event with event marks on the begin event type and end event type, creates a new half-open request unit if its first beginning or ending event is detected, and closes an open request unit as a complete request unit when the ending event of an open request unit having only a beginning event is detected. Additional detail on request unit slicing will be provided below. Block 306 outputs the two detected types of request units.

[0040] Block 308 builds virtual complete request units across threads in a given machine based on the per-thread half-open request units supplied by block 306. Block 308 first finds the pairs of request units based on their beginning or ending event

records. For each pair, block 308 builds a causality graph based on kernel events and the two events from the request units. Block 308 searches a path in the causality graph (a default path to search may be the shortest path in traversed links) from the beginning event to the ending event of the two request units. If such a path exists, block 308 constructs a virtual complete request unit which corresponds to the discovered path.

[0041] Referring now to FIG. 4, additional detail on the slicing of request units in block 306 is shown. Block 402 selects a head event in a thread. This initialized empty request unit sets $RU_{\text{globalComplete}}$ and $RU_{\text{globalHalfOpen}}$ and selects the first event record e_c from the thread p 's event record sequence E_p . For each request unit mark M_i in the marking event set M from block 304, block 404 compares the event record e_c with the marking event set by checking if the owner_filter of M_i matches p 's owner ID and, if so, checking if the begin_event_type of M_i matches e_c 's event type and if the event_data_filter in M_i matches e_c 's event data.

[0042] If both are matched, then this is not a new request unit and block 414 checks whether any request unit R in RU_{halfOpen} has the beginning event record in its event record sequence S that matches the begin_event_type and event_data_filter in M_i . If there is such a request unit R , then block 414 checks whether R has any event record in its event record sequence S that matches the end_event_type and event_data_filter in M_i . If so, event record e_R is assigned to be the last such event record. If $\text{time\_begin}(e_c) - \text{time\_begin}(e_R) > \Delta$ (a configurable time threshold), block 416 moves R from RU_{halfOpen} into RU_{complete} to create the complete request unit, a new request unit N is created in RU_{halfOpen} in block 418, e_c is inserted as the beginning event record in N 's event record sequence, and M_i is inserted as the beginning mark in N 's request unit mark set. Block 410 determines whether there are more events in the thread. If so, processing returns to block 404 with the next event in the thread p 's event sequence.

[0043] If block 406 instead detects a new request unit, block 408 checks if the `end_event_type` in M_i matches e_c 's event type and if `event_data_filter` in M_i matches e_c 's event data. If both are matched, and if any request unit R in $RU_{halfOpen}$ has the beginning event record in its event record sequence S matching the `begin_event_type` and `event_data_filter` in M_i , block 406 updates e_c as the ending event record in N 's event record sequence if it has no duplicate copy in the sequence and inserts M_i as the ending mark in N 's request unit mark set. Otherwise, block 406 creates a new request unit N in $RU_{halfOpen}$, with e_c being marked as the ending event record in N 's event record sequence S and inserts M_i as the ending mark in N 's request unit mark set. Block 410 determines whether there are more events in the thread p and, if so, processing returns to block 404 with a new event selected.

[0044] Any request units in $RU_{halfOpen}$ that have both beginning and ending event records are migrated into $RU_{globalComplete}$. Migrating a request unit involves filling in the intermediate events between the beginning and ending event records from the corresponding thread's event record sequence. Any remaining request units in $RU_{halfOpen}$ are moved to $RU_{globalHalfOpen}$.

[0045] Once all of the events in a thread have been processed, block 412 determines whether there are more threads to process. If so, a new thread p is selected and processing returns to block 402. If not, the two types of request units are output, with request units in $RU_{globalComplete}$ being output as the detected complete request units and with request units in $RU_{globalHalfOpen}$ being output as the detected half open request units, where either the beginning or ending event record is detected, but not both.

[0046] Referring now to FIG. 5, additional detail on the search of virtual complete request units in block 308 is shown. Block 502 begins by selecting a half-open request unit R_i from the set of request units RU_{begin} that have only beginning records. For each

such half-open request unit, block 504 determines whether there is a matching request unit R_j from the set of request units that have only ending event records. Block 504 determines that R_i matches R_j if the two events belong to two different threads on the same device, if the `begin_event_type` for a request unit mark M_i matches R_i beginning event type, if the `event_data_filter` in M_i matches R_i 's beginning event data, if the `end_event_type` in M_i matches R_j 's ending event type, and if the `event_data_filter` in M_i matches R_j 's ending event data. If there is no match, block 509 removes R_i from RU_{begin} .

[0047] If block 504 finds a match, block 506 constructs a causality graph connecting R_i and R_j . The event e_{begin} is assigned as R_i 's beginning event record and the event e_{end} is assigned as R_j 's ending event record. The causality graph $G = (V, E)$ is constructed based on the collected kernel event traces E . For each thread in the same machine as R_i , the thread's subgraph is formed from the event nodes where each event e_c satisfies $time\_begin(e_c) \geq time\_begin(e_{begin})$ and $time\_end(e_c) \leq time\_end(e_{end})$ (i.e., each event that occurs between the begin event and the end event), where e_c is one of e_{begin} , e_{end} , an inter-process communication event, a process event, or a context switch event. Directed links connect each event node in the subgraph to the next event node in time order.

[0048] Between subgraphs of two threads, block 506 forms a directed link from inter-process communication events to their nearest counterpart events in time order that have matched communication parameters. A directed link is formed from a process event to its nearest counterpart event. A directed link is formed from a context switch event to its nearest counterpart event is formed in time order with matching parameters.

[0049] Based on the constructed causality graph G , block 508 attempts to find a path from R_i to R_j . Any appropriate method may be used to find the shortest path in traversed links, such as Dijkstra's algorithm. If such a path is discovered, block 510 creates a

complete request unit and adds that request unit to $RU_{globalComplete}$, with the beginning event as e_{begin} , with the ending event as e_{end} , and with intermediate events being the events of the discovered path. Block 509 then removes R_i from RU_{begin} , regardless of whether a path is discovered. If there are more request units in RU_{begin} , a new R_i is selected and processing returns to block 502. If not, block 512 outputs $RU_{globalComplete}$ as the set of detected complete request units.

[0050] Referring now to FIG. 6, additional detail on the request unit sequencing of block 204 is shown. Block 602 defines a causality relationship set that includes event association rules, each presented as a 5-tuple record: $\langle \text{causing\_event\_type}, \text{caused\_event\_type}, \text{time\_rule}, \text{owner\_rule}, \text{event\_data\_rule} \rangle$. The causality relationships are defined through event association rules that can be provided through domain knowledge.

[0051] The term “causing_event_type” refers to the type of events that can cause the occurrence of other events. Examples include, but are not limited to, network event types (e.g., TCP_SENDCV), inter-process communication event types (e.g., PIPE_WRITE), process event types (e.g., PROCESS_CREATE), system call event types (e.g., KWAKEUP), and application event types (e.g., AP_FUNCTION_ENTER).

[0052] The term “caused_event_type” refers to the type of events that are caused by other events. Examples include, but are not limited to, network event types (e.g., TCP_RECV), inter-process communication event types (e.g., PIPE_READ), system call event types (e.g., KSLEEP), and application event types (e.g., AP_FUNCTION_EXIT).

[0053] The term “time_rule” refers to a temporal relationship that can relate a causing_event_type event and a caused_event_type event. Examples include, but are not limited to, an overlap temporal relationship (e.g., $\text{time\_begin}(\text{causing\_event}) <$

time_begin(caused_event) and time_end(causing_event) > time_begin(caused_event)), a nested temporal relationship (e.g., time_begin(causing_event) < time_begin(caused_event) and time_end(causing_event) > time_end(caused_event)), and disjoint temporal relationship (e.g., time_end(causing_event) < time_begin(caused_event)).

[0054] The term “owner_rule” refers to an association of a causing_event_type and a caused_event_type based on owner IDs. Examples include, but are not limited to, a specific machine name (e.g., causing_event_owner_machineName = srcIPAddress (caused_event_data)), a specific thread name (e.g., causing_event_owner_processName = httpd), a specific thread ID (e.g., caused_event_owner_threadID = causing_event_owner_threadID), or a wild card (*, which matches any owner ID).

[0055] The term “event_data_rule” refers to an association between a causing_event_type event and a caused_event_type event based on their event data. Examples include, but are not limited to, network event data (e.g., dstIPport(caused_event_data) = srcIPport (causing_event_data)), inter-process communication event data (e.g., PIPE_ID (caused_event_data) = PIPE_ID(causing_event_data)), and application event data (e.g., Function_ID(caused_event_data) = Function_ID(causing_event_data)). A wild card in this field matches any event data.

[0056] A rule is called time-causality rule if owner_rule requires both causing and caused request units to be from the same thread or the thread pool (for a virtual complete request unit that traverse a pool of threads), and their causality relationship is decided by the time_rule. Rules matching request units from different processes are called event-causality rules.

[0057] For each request unit mark M, a rule pair CRP:(CR_{begin}, CR_{end}) is defined so that the begin_event_type in M uses CR_{begin} to find the associated events in other request units, and the end_event_type in M uses CR_{end} to find the associated events in

other request units. Either rule can be empty. Block 602 outputs a set of event association rule pairs which will be used to detect the set of relevant require units and to compose full-path requests. The event association rules can be predefined as a default set or can be dynamically configured upon user request.

[0058] Block 604 sequences the request unit. One request R is defined through a two-item data structure $\langle E, C \rangle$, where E is a set of request units and C is a set of relationships describing the causality between the request units in E . Sequencing includes two main procedures: for each request unit, the request units causing it are found and, after that search, a full-path transaction for each request unit with no causing request unit is constructed. The transaction is rooted at that request unit and expanded over all request units caused by that root in one hop or multiple hops. Request unit sequencing will be described in greater detail below. Block 606 outputs the sequenced requests.

[0059] Referring now to FIG. 7, additional detail on the request unit sequencing of block 604 is shown. An empty request set RS_{global} is initialized and block 702 selects a new request unit u_i from the request unit set U output by block 306. For each request unit u_i , two empty request unit sets RU_i^{parent} and RU_i^{child} are initialized.

[0060] Block 704 finds parent request units for u_i that belong to other processes. For the beginning event record e_i^f of u_i 's event record sequence, for each rule pair p_x in the event association rule pair set output by block 602, block 704 checks if the event type of e_i^f is the `caused_event_type` of rule CR_{begin} in p_x . If CR_{begin} is a time causality rule, block 704 then checks if there is any request unit u_j in U that has the same owner as u_i and has a beginning event record e_j^f that can be associated with e_i^f by testing CR_{begin} . It should be noted that u_i and u_j having the same owner indicates that the two request units

are from the same process. If the time of e_i^f and e_j^f can yield “True” for the time_rule in CR_{begin} , block 704 yields a match.

[0061] If the event type of e_i^f is the caused_event_type of rule CR_{begin} in p_x and CR_{begin} is not a time causality rule, block 704 checks if there is any request unit u_j having a first event record e_j^f that can be associated with e_i^f by testing the rule CR_{begin} on both event records. If the event type of e_i^f is the causing_event_type of CR_{begin} , and the time, owner IDs, and event data of e_i^f and e_j^f yield “True” on the rules time_rule, owner_rule, and event_data_rule, block 704 also yields a match.

[0062] If block 704 yields a match, such that CR_{begin} is satisfied, block 704 checks if the last event record e_i^l in u_i can be associated with the last event record e_j^l in u_j by testing the rule CR_{end} on both last event records. If the two last event records can be associated under CR_{end} , u_j is a causing event unit for u_i and is added to RU_i^{parent} while u_i is added to RU_j^{child} . This process is repeated for every request unit, so block 708 determines whether there are more request units. If so, block 702 selects a new request unit. If not, block 710 selects a request unit that lacks a parent.

[0063] For each parentless request unit u_i , block 712 searches for all child request units. A new request R_x is created as $\langle E_x, C_x \rangle$, with u_i being inserted into E_x and with the relation “NULL- $\rightarrow$ parent(u_i)” being inserted into C_x . For each request unit u_j in RU_i^{child} , if both u_i and u_j have only one event in their event sequence, block 714 inserts into u_j all the events that belong to the same owner of u_j , have time_begin values larger than the time_begin value of u_i ’s single event, and have time_begin values smaller than the value of u_j ’s single event. Block 714 inserts u_j into E_x and inserts the relation “ u_i - $\rightarrow$ parent(u_j)” into C_x . Each request unit u_k in RU_j^{child} is inserted into RU_i^{child} and the relation “ u_j - $\rightarrow$ parent(u_k)” is added into C_x . The request unit u_j is removed from RU_i^{child} and the new request is added to RS_{global} . Block 716 determines whether there are more

request units—if so, another parentless request unit is selected processing returns to block 710. If not, processing ends and the requests in RS_{global} are output.

[0064] Embodiments described herein may be entirely hardware, entirely software or including both hardware and software elements. In a preferred embodiment, the present invention is implemented in software, which includes but is not limited to firmware, resident software, microcode, etc.

[0065] Embodiments may include a computer program product accessible from a computer-usable or computer-readable medium providing program code for use by or in connection with a computer or any instruction execution system. A computer-usable or computer readable medium may include any apparatus that stores, communicates, propagates, or transports the program for use by or in connection with the instruction execution system, apparatus, or device. The medium can be magnetic, optical, electronic, electromagnetic, infrared, or semiconductor system (or apparatus or device) or a propagation medium. The medium may include a computer-readable storage medium such as a semiconductor or solid state memory, magnetic tape, a removable computer diskette, a random access memory (RAM), a read-only memory (ROM), a rigid magnetic disk and an optical disk, etc.

[0066] Each computer program may be tangibly stored in a machine-readable storage media or device (e.g., program memory or magnetic disk) readable by a general or special purpose programmable computer, for configuring and controlling operation of a computer when the storage media or device is read by the computer to perform the procedures described herein. The inventive system may also be considered to be embodied in a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner to perform the functions described herein.

[0067] A data processing system suitable for storing and/or executing program code may include at least one processor coupled directly or indirectly to memory elements through a system bus. The memory elements can include local memory employed during actual execution of the program code, bulk storage, and cache memories which provide temporary storage of at least some program code to reduce the number of times code is retrieved from bulk storage during execution. Input/output or I/O devices (including but not limited to keyboards, displays, pointing devices, etc.) may be coupled to the system either directly or through intervening I/O controllers.

[0068] Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modem and Ethernet cards are just a few of the currently available types of network adapters.

[0069] Referring now to FIG. 8, a profiling system 800 is shown. The profiling system 800 includes a hardware processor 802 and memory 804. The profiling system 800 also includes one or more functional modules. In some embodiments, the functional modules may be implemented as software that is stored in memory 804 and executed by hardware processor 802. In alternative embodiments, the functional modules may be implemented as one or more discrete hardware components in the form of, e.g., application specific integrated chips or field programmable gate arrays.

[0070] A kernel request module 805 collects kernel event traces, including operating system calls. A request unit generating module 806 generates two types of request units from collected kernel events, half-open request units and complete request units. A request unit sequencing module 808 then constructs full-path request traces by composing the request units based on a causality relationship set between request units.

The profiling system 800 can thereby obtain full-path traces across multiple threads with low overhead.

[0071] Referring now to FIG. 9, an exemplary processing system 900 is shown which may represent the transmitting device 100 or the receiving device 120. The processing system 900 includes at least one processor (CPU) 904 operatively coupled to other components via a system bus 902. A cache 906, a Read Only Memory (ROM) 908, a Random Access Memory (RAM) 910, an input/output (I/O) adapter 920, a sound adapter 930, a network adapter 940, a user interface adapter 950, and a display adapter 960, are operatively coupled to the system bus 902.

[0072] A first storage device 922 and a second storage device 924 are operatively coupled to system bus 902 by the I/O adapter 920. The storage devices 922 and 924 can be any of a disk storage device (e.g., a magnetic or optical disk storage device), a solid state magnetic device, and so forth. The storage devices 922 and 924 can be the same type of storage device or different types of storage devices.

[0073] A speaker 932 is operatively coupled to system bus 902 by the sound adapter 930. A transceiver 942 is operatively coupled to system bus 902 by network adapter 940. A display device 962 is operatively coupled to system bus 902 by display adapter 960.

[0074] A first user input device 952, a second user input device 954, and a third user input device 956 are operatively coupled to system bus 902 by user interface adapter 950. The user input devices 952, 954, and 956 can be any of a keyboard, a mouse, a keypad, an image capture device, a motion sensing device, a microphone, a device incorporating the functionality of at least two of the preceding devices, and so forth. Of course, other types of input devices can also be used, while maintaining the spirit of the present principles. The user input devices 952, 954, and 956 can be the

same type of user input device or different types of user input devices. The user input devices 952, 954, and 956 are used to input and output information to and from system 900.

[0075] Of course, the processing system 900 may also include other elements (not shown), as readily contemplated by one of skill in the art, as well as omit certain elements. For example, various other input devices and/or output devices can be included in processing system 900, depending upon the particular implementation of the same, as readily understood by one of ordinary skill in the art. For example, various types of wireless and/or wired input and/or output devices can be used. Moreover, additional processors, controllers, memories, and so forth, in various configurations can also be utilized as readily appreciated by one of ordinary skill in the art. These and other variations of the processing system 900 are readily contemplated by one of ordinary skill in the art given the teachings of the present principles provided herein.

[0076] The foregoing is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from the Detailed Description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that those skilled in the art may implement various modifications without departing from the scope and spirit of the invention. Those skilled in the art could implement various other feature combinations without departing from the scope and spirit of the invention. Having thus described aspects of the invention, with the details and particularity required by the patent laws, what is claimed and desired protected by Letters Patent is set forth in the appended claims.

WHAT IS CLAIMED IS:

1. A method for profiling requests, comprising:
generating request units (202) based on collected kernel events that include complete request units and half-open request units using a processor; and
sequencing the generated request units (204) based on a causality relationship set that describes causality relationships between kernel events.
2. The method as recited in claim 1, wherein generating request units comprises creating half-open request units based on beginning events and ending events.
3. The method of claim 2, wherein generating request units further comprises creating complete request units based on ending events that match respective beginning events.
4. The method of claim 2, wherein generating request units further comprises generating virtual complete request units based on a causality graph that characterizes causative relationships between kernel events.
5. The method of claim 4, wherein generating request units further comprises searching a shortest path in the causality graph between a beginning event in a first half-open request unit and an ending event in a second half-open request unit.

6. The method of claim 5, wherein generating request units further comprises determining that the beginning event and the ending event match one or more association rules.

7. The method of claim 1, wherein the causality relationship set comprises event association rules that establish at least a caused event type and a causing event type.

8. The method of claim 7, wherein each event association rule is a 5-tuple that includes a caused event type, a causing event type, a time rule, an owner rule, and an event data rule.

9. The method of claim 1, further comprising taking an administrative action responsive to the sequenced request units.

10. A system for profiling requests, comprising:
a request unit generating module (806) comprising a processor configured to generate request units based on collected kernel events that include complete request units and half-open request units; and
a request unit sequencing module (808) configured to sequence the generated request units based on a causality relationship set that describes causality relationships between kernel events.

11. The system of claim 10, wherein the request unit generating module is further configured to create half-open request units based on beginning events and ending events.

12. The system of claim 11, wherein the request unit generating module is further configured to create complete request units based on ending events that match respective beginning events.

13. The system of claim 11, wherein the request unit generating module is further configured to generate virtual complete request units based on a causality graph that characterizes causative relationships between kernel events.

14. The system of claim 13, wherein the request unit generating module is further configured search to a shortest path in the causality graph between a beginning event in a first half-open request unit and an ending event in a second half-open request unit.

15. The system of claim 14, wherein the request unit generating module is further configured to determine that the beginning event and the ending event match one or more association rules.

16. The system of claim 10, wherein the causality relationship set comprises event association rules that establish at least a caused event type and a causing event type.

17. The system of claim 16, wherein each event association rule is a 5-tuple that includes a caused event type, a causing event type, a time rule, an owner rule, and an event data rule.

1/8

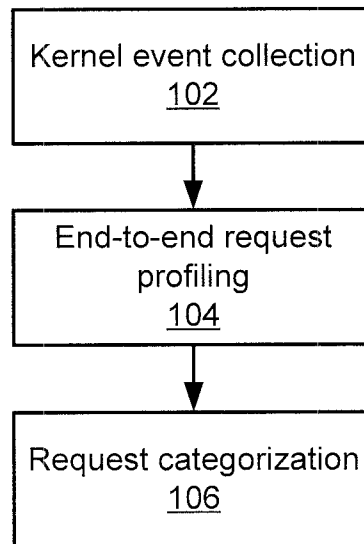


FIG. 1

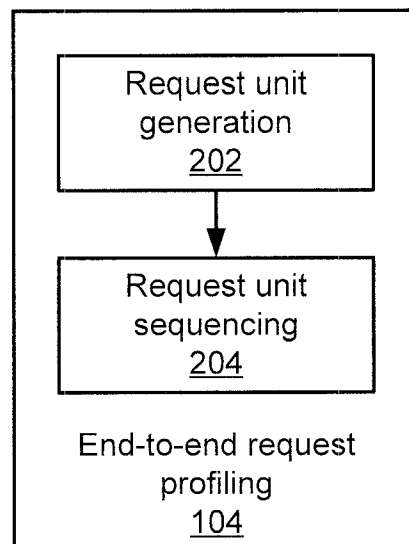


FIG. 2

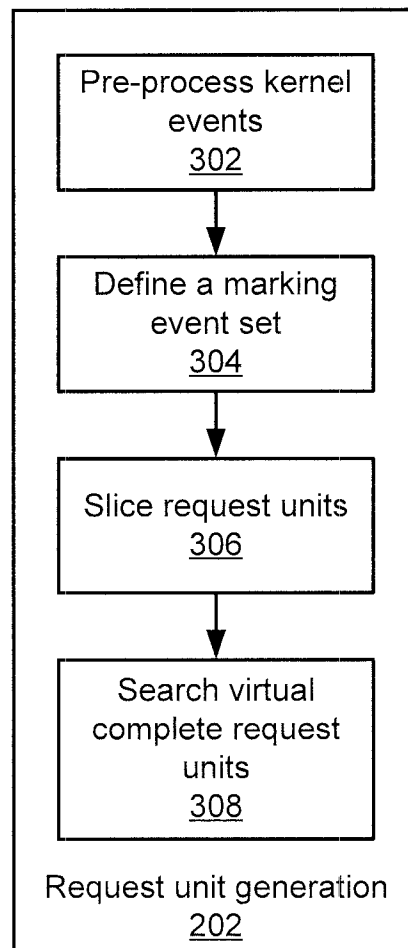


FIG. 3

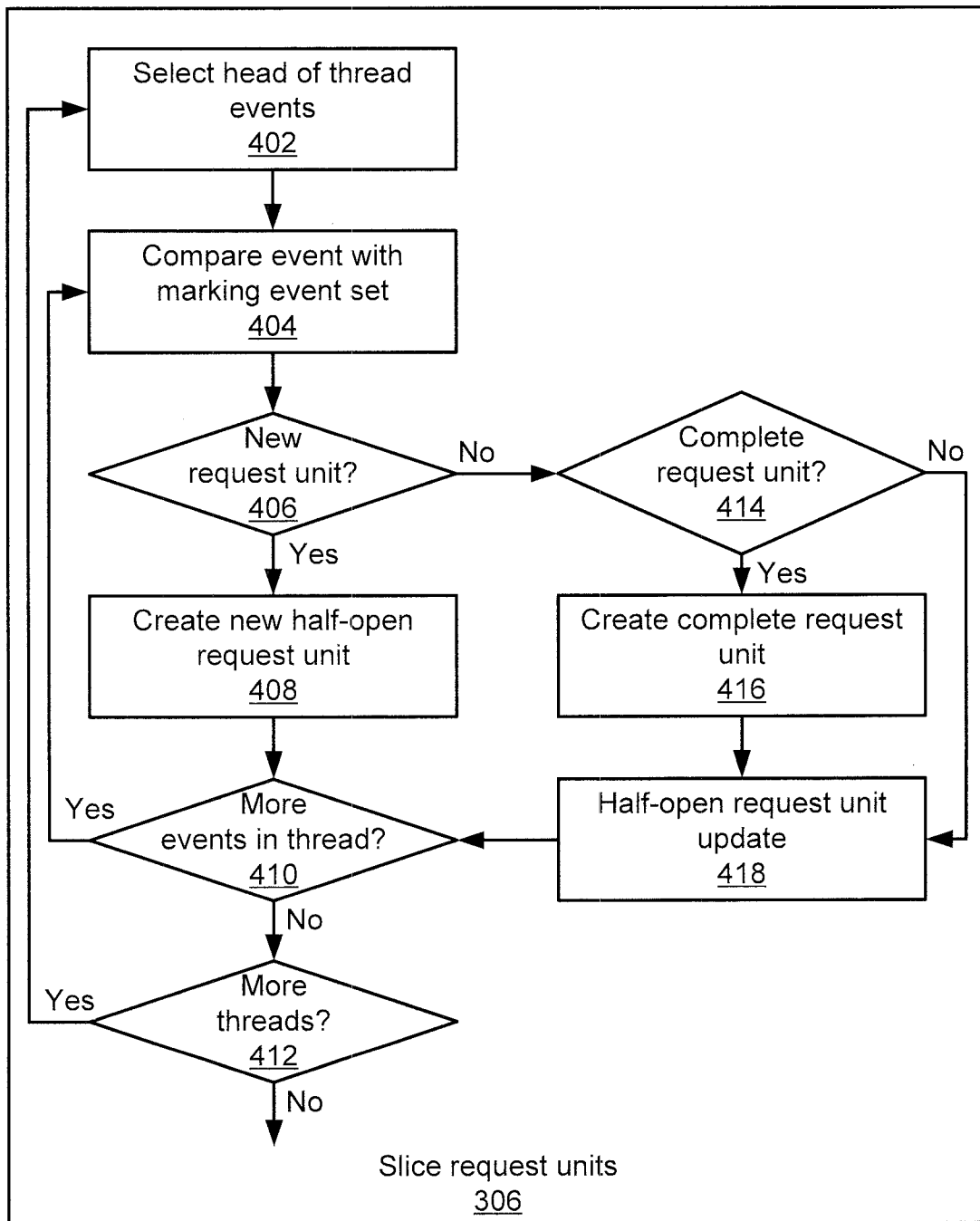


FIG. 4

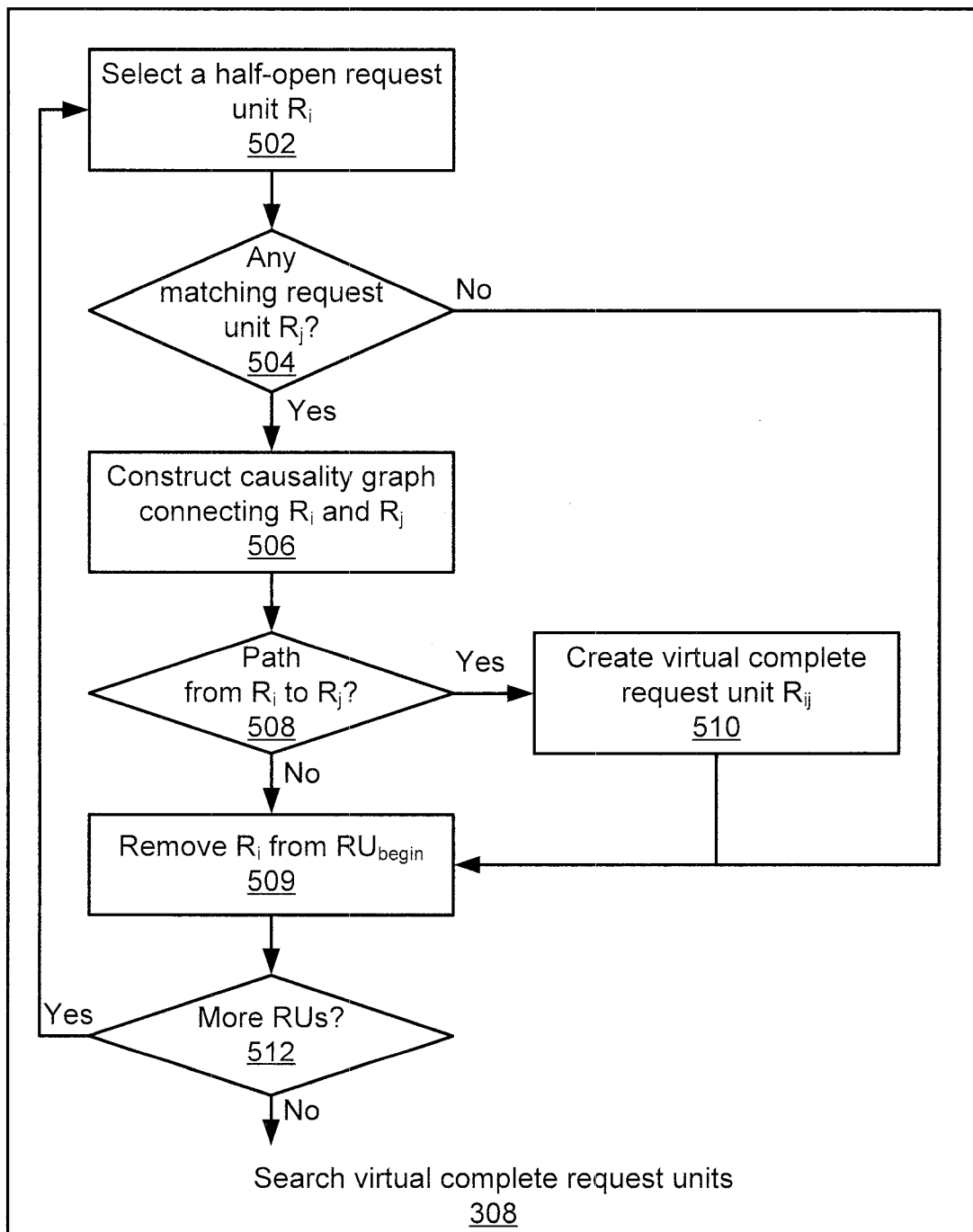


FIG. 5

5/8

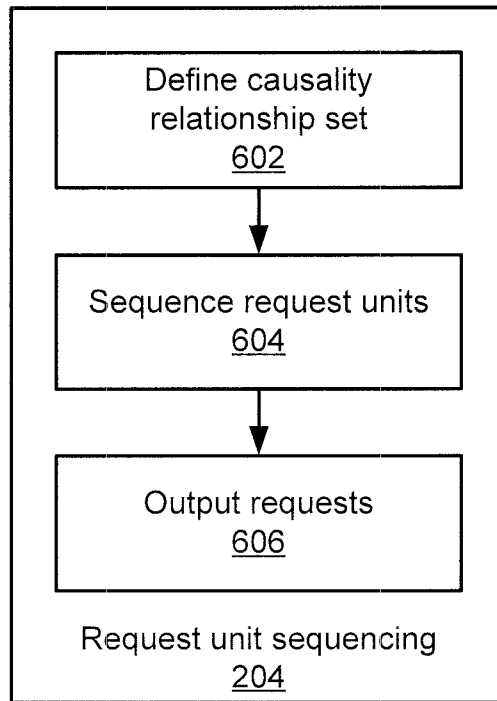


FIG. 6

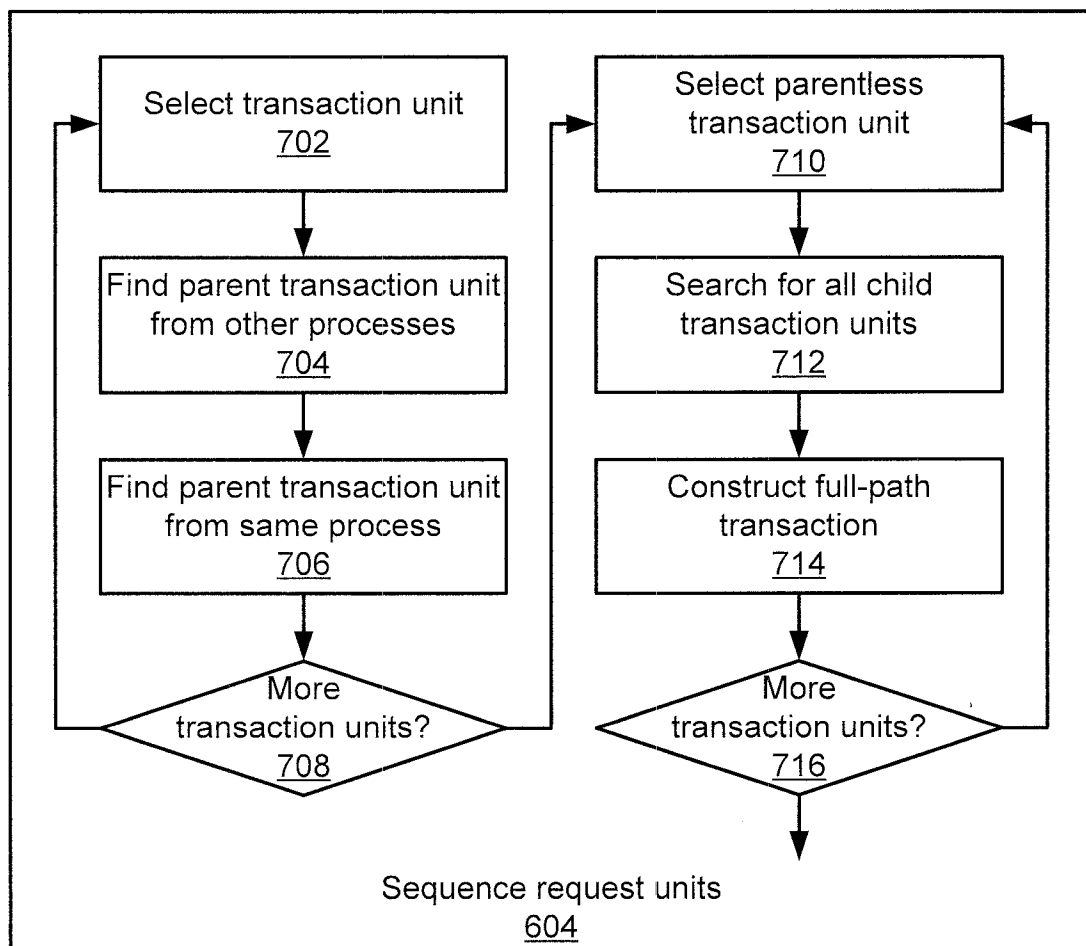


FIG. 7

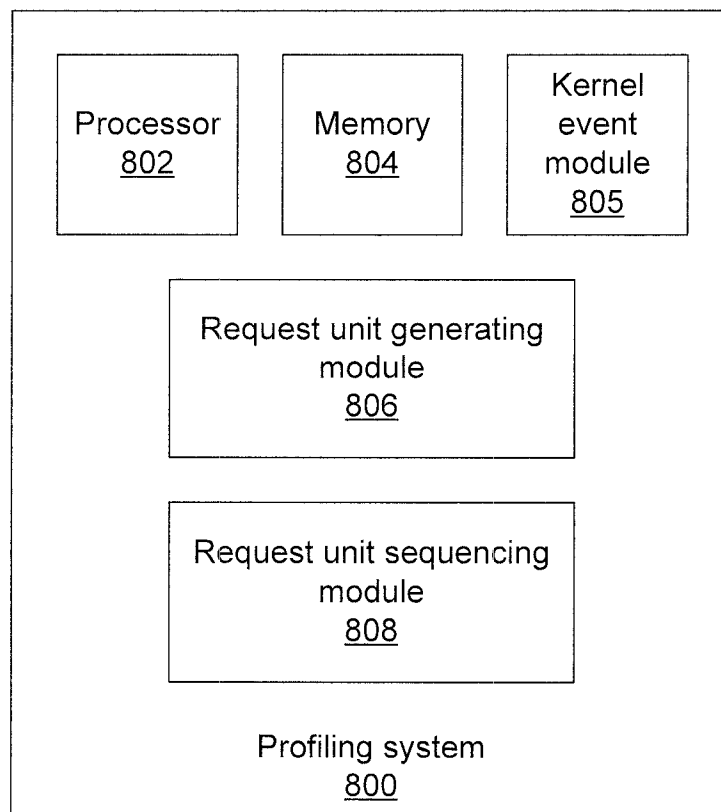


FIG. 8

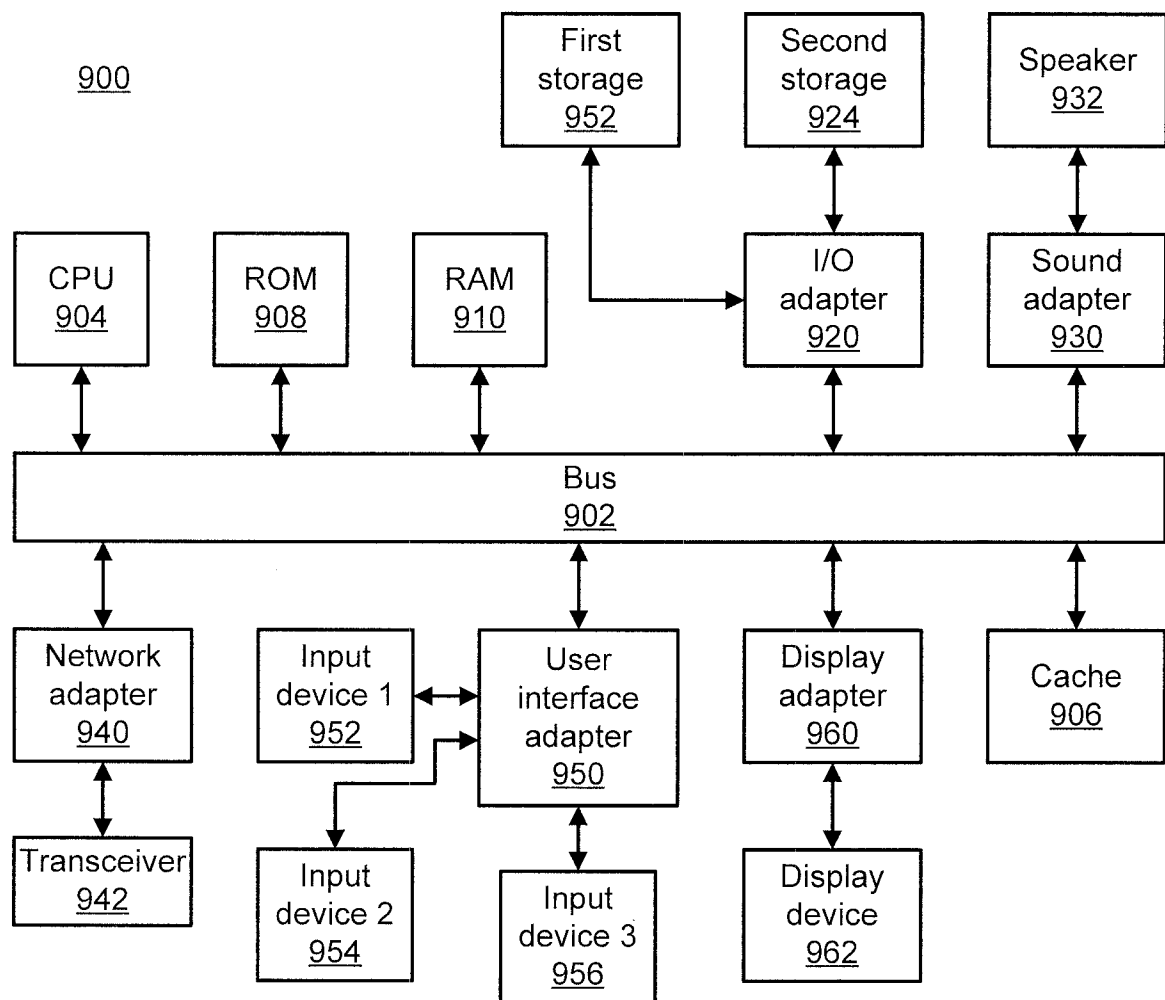


FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/051872**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/54(2006.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/54; G06F 15/173; G06F 9/44; G06F 12/14

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: kernel, event, sequence, profile, trace, begin, end, request

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014-0109112 A1 (NEC LABORATORIES AMERICA, INC.) 17 April 2014 See paragraph [0022]; and claims 3, 9-11.	1-5, 7-14, 16-17
Y		6, 15
Y	US 2011-0238812 A1 (JEFFREY A. VAUGHT) 29 September 2011 See claim 4.	6, 15
A	US 2013-0159977 A1 (CHRISTOPHER CROSETTO et al.) 20 June 2013 See paragraphs [0014]-[0029]; and figures 1-2.	1-17
A	US 2009-0070878 A1 (HAO WANG et al.) 12 March 2009 See paragraphs [0049]-[0084]; and figures 3-6.	1-17
A	US 2005-0021708 A1 (MELUR K. RAGHURAMAN et al.) 27 January 2005 See paragraphs [0041]-[0067]; and figures 2-4e.	1-17



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

13 December 2016 (13.12.2016)

Date of mailing of the international search report

13 December 2016 (13.12.2016)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

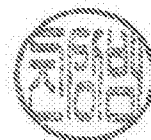
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

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/US2016/051872

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0109112 A1	17/04/2014	US 8875158 B2	28/10/2014
US 2011-0238812 A1	29/09/2011	EP 1652080 A2	03/05/2006
		US 2005-010930 A1	13/01/2005
		US 7984129 B2	19/07/2011
		US 8504674 B2	06/08/2013
		WO 2005-008489 A2	27/01/2005
		WO 2005-008489 A3	14/04/2005
		WO 2005-008489 A8	02/06/2005
US 2013-0159977 A1	20/06/2013	None	
US 2009-0070878 A1	12/03/2009	US 8065728 B2	22/11/2011
		WO 2009-035902 A2	19/03/2009
		WO 2009-035902 A3	14/05/2009
US 2005-0021708 A1	27/01/2005	US 7516209 B2	07/04/2009