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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 October 2014 (02.10.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/159444 A1

(51) International Patent Classification:
G06F 9/48 (2006.01) *G06F 9/455* (2006.01)

(21) International Application Number:
PCT/US2014/023708

(22) International Filing Date:
11 March 2014 (11.03.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/828,183 14 March 2013 (14.03.2013) US

(71) Applicant: **QUALCOMM INCORPORATED** [US/US]; ATTN: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(72) Inventors: **PLONDKE, Erich James**; 5775 Morehouse Drive, San Diego, CA 92121-1714 (US). **CODRESCU, Lucian**; 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

(74) Agent: **TOLER, JEFFREY G.**; 8500 Bluffstone Cove, Suite A201, Austin, Texas 78759 (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, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

WO 2014/159444 A1

(54) Title: SYSTEMS AND METHODS OF USING A HYPERVISOR WITH GUEST OPERATING SYSTEMS AND VIRTUAL PROCESSORS

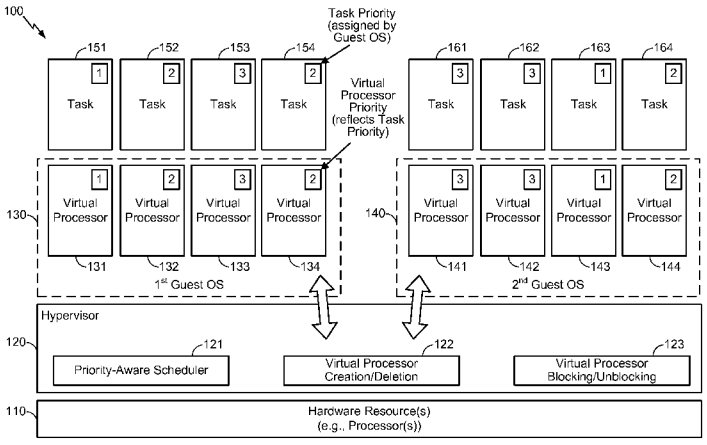

FIG. 1

(57) Abstract: An apparatus includes a processor and a guest operating system. In response to receiving a request to create a task, the guest operating system requests a hypervisor to create a virtual processor to execute the requested task. The virtual processor is schedulable on the processor.

SYSTEMS AND METHODS OF USING A HYPERVISOR WITH GUEST OPERATING SYSTEMS AND VIRTUAL PROCESSORS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority from commonly owned U.S. Non-Provisional Patent Application No. 13/828,183 filed on March 14, 2013, the contents of which are expressly incorporated herein by reference in its entirety.

FIELD

[0002] The present disclosure is generally related to virtualization of operating systems using a hypervisor.

BACKGROUND

[0003] Advances in technology have resulted in smaller and more powerful computing devices. For example, there currently exist a variety of portable personal computing devices, including wireless computing devices, such as portable wireless telephones, personal digital assistants (PDAs), tablet computers, and paging devices that are small, lightweight, and easily carried by users. Many such computing devices include other devices that are incorporated therein. For example, a wireless telephone can also include a digital still camera, a digital video camera, a digital recorder, and an audio file player. Also, such computing devices can process executable instructions, including software applications, such as a web browser application that can be used to access the Internet and multimedia applications that utilize a still or video camera and provide multimedia playback functionality.

[0004] To support such advanced functionality, the computing devices may include an operating system. The operating system may provide abstractions of available hardware resources. For example, an operating system may multiplex system resources (e.g., processor, memory, etc.) amongst various software tasks. The operating system may also schedule tasks for execution and perform memory management. In addition, the operating system may handle blocking and unblocking for events.

[0005] In certain situations, it may be advantageous to execute multiple operating systems, or multiple instances of the same operating system, at a single computing device. A hypervisor (also known as a virtual machine monitor) may provide abstractions of available hardware resources to the operating systems. Thus, the hypervisor may act as an operating system for the multiple operating systems (or multiple instances of the same operating system). When a hypervisor is used, the hypervisor may include a scheduler to determine which operating system and task thereof is granted access to a particular hardware resource at a particular time. Thus, multiple levels of scheduling may be executing within the system: a first level may include the schedulers of each individual operating system and a second level may include the scheduler of the hypervisor.

[0006] Performing multiple levels of scheduling introduces overhead into the system (e.g., due to context switches performed by the hypervisor). Such overhead may be unacceptable when one or more of the operating systems is a real-time operating system that needs to be able to guarantee certain latencies. To include a real-time operating system in a multiple operating system environment, the real-time operating system is usually given "special" priority. For example, the real-time operating system may have a highest available priority. However, this may be inefficient because task priority information within each operating system may not be accessible to the hypervisor. For example, if the real-time operating system is executing an unimportant task but another operating system is executing an important task, the hypervisor may still schedule the relatively unimportant real-time operating system task first.

SUMMARY

[0007] Systems and methods of enabling an operating system (e.g., a real-time operating system) to execute with reduced latency in a hypervisor-controlled computing environment are disclosed. Instead of (or in addition to) conventional guest operating systems that have built-in schedulers, the described techniques utilize "thin" guest operating systems that include limited or no scheduling functions and instead rely on the hypervisor for scheduling. Blocking and unblocking may also be handled by the hypervisor instead of by individual operating systems. Further, the hypervisor may

have access to information regarding the priorities of the tasks of each of the individual operating systems.

[0008] Each time a new task is created at a "thin" operating system, the "thin" operating system may request the hypervisor to create a "virtual processor" for the task. The hypervisor may schedule the virtual processor for execution on underlying physical hardware (e.g., hardware processor(s)), and virtual processors may be maintained as long as the corresponding task exist. For example, each task may be associated with a virtual processor and all scheduling may be handled by the scheduler of the hypervisor. Moreover, the scheduler of the hypervisor may have access to the priorities of individual tasks/virtual processors, which may be used to perform more efficient virtual processor scheduling across all guest operating systems. The described systems and methods may thus simplify individual operating systems and reduce overall latency in the hypervisor-controlled computing environment, thereby enabling the use of one or more real-time operating systems and other operating systems in the hypervisor-controlled computing environment

[0009] In a particular embodiment, an apparatus includes a processor and a hypervisor. The hypervisor is configured to schedule virtual processors for execution by the processor. Each of the virtual processors has a priority that is accessible to the hypervisor and that corresponds to a task priority of a corresponding task. The task priority is assigned by a guest operating system.

[0010] In another particular embodiment, a method includes receiving, at a hypervisor of a computing device, a request from a guest operating system to create a virtual processor to execute a task. The method also includes creating the virtual processor and scheduling the virtual processor for execution on a hardware processor of the computing device. The virtual processor has a priority assigned by the guest operating system, where the priority corresponds to a task priority of the task.

[0011] In another particular embodiment, an apparatus includes means for processing and means for scheduling executable by the means for processing. The means for scheduling is configured to schedule virtual processors for execution by the means for processing. Each of the virtual processors has a priority that is accessible to the means

for scheduling and that corresponds to a task priority of a corresponding task. The task priority is assigned by a first guest operating system.

[0012] In another particular embodiment, a non-transitory computer-readable medium includes instructions that, when executed by a computer, cause the computer to receive, at a hypervisor, a request from a guest operating system to create a virtual processor to execute a task. The instructions, when executed by the computer, also cause the computer to create the virtual processor and to schedule the virtual processor for execution. The virtual processor has a priority corresponding to a task priority of the task. The task priority is assigned by the guest operating system.

[0013] In another particular embodiment, an apparatus includes a processor and a guest operating system. In response to receiving a request to create a task, the guest operating system requests a hypervisor to create a virtual processor to execute the task. The virtual processor is schedulable on the processor.

[0014] In another particular embodiment, a method includes receiving, at a guest operating system executing at a computing device comprising a hypervisor and at least one processor, a request to create a task. The method also includes, in response to the request to create the task, requesting the hypervisor to create a virtual processor to execute the requested task. The virtual processor is schedulable on the at least one processor.

[0015] In another particular embodiment, an apparatus includes a processor and a guest operating system. The guest operating system includes means for requesting a hypervisor to create a virtual processor to execute a task in response to receiving a request to create the task. The virtual processor is schedulable on the processor.

[0016] In another particular embodiment, a non-transitory computer-readable medium includes instructions that, when executed by a computer, cause the computer to receive, at a guest operating system, a request to create a task. The instructions, when executed by the computer, also cause the computer to, in response to the request to create the task, request a hypervisor to create a virtual processor to execute the requested task. The virtual processor is schedulable on a hardware processor.

[0017] In another particular embodiment, an electronic device includes a processor, a hypervisor, a first guest operating system, and a second guest operating system. At least one task of the first guest operating system is executed by a first virtual processor requested by the first guest operating system from the hypervisor. At least one task of the second guest operating system is executed by a second virtual processor requested by the second guest operating system from the hypervisor. The first virtual processor and the second virtual processor are executed by the processor.

[0018] One particular advantage provided by at least one of the disclosed embodiments is an ability to perform prioritized scheduling of tasks from multiple guest operating systems with reduced latency. Another particular advantage provided by at least one of the disclosed embodiments is an ability to use real-time operating system(s) within a hypervisor-controlled computing environment. Other aspects, advantages, and features of the present disclosure will become apparent after review of the entire application, including the following sections: Brief Description of the Drawings, Detailed Description, and the Claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] FIG. 1 is a block diagram of an illustrative embodiment of a system that is operable to use a hypervisor with guest operating systems and virtual processors;

[0020] FIG. 2 is a block diagram of another illustrative embodiment of a system that is operable to use a hypervisor with guest operating systems and virtual processors;

[0021] FIG. 3 is a flowchart of an illustrative embodiment of a method of operation at the hypervisor of FIG. 1;

[0022] FIG. 4 is a flowchart of an illustrative embodiment of a method of operation at one of the guest operating systems of FIG. 1;

[0023] FIG. 5 is a block diagram of a wireless device including components operable to support use of a hypervisor with guest operating systems and virtual processors; and

[0024] FIG. 6 is a block diagram of an illustrative embodiment of a system operable to perform priority inheritance.

DETAILED DESCRIPTION

[0025] FIG. 1 is a block diagram of an illustrative embodiment of a system 100 that is operable to use a hypervisor with guest operating systems and virtual processors. The system 100 includes one or more hardware resources, such as one or more processors 110. The system also includes a hypervisor 120 that is executable by the processors 110. The hypervisor 120 may arbitrate access to the one or more processors 110 by one or more guest operating systems, such as an illustrative first guest operating system 130 and an illustrative second guest operating system 140. As further described herein, one or more of the guest operating systems 130, 140 may be a real-time operating system (RTOS).

[0026] The hypervisor 120 may include various logical and/or functional modules. In a particular embodiment, each such module is implemented using software instructions executable by the processors 110. Alternately, selected functionality of the hypervisor 120 may be implemented using hardware devices, such as a controller, an application-specific integrated circuit (ASIC), a field programmable gate array (FPGA) device, or some other device. In the embodiment of FIG. 1, the hypervisor 120 includes a priority-aware scheduler 121, a virtual processor creation/deletion module 122, and a virtual processor blocking/unblocking module 123. It should be noted, however, that the modules 121-123 are for illustration only. In alternate embodiments, functions described with reference to the hypervisor 120 may be carried out using more or fewer modules.

[0027] The virtual processor creation/deletion module 122 may be configured to receive a request from either of the guest operating systems 130, 140. For example, the request may be received via an operating system-hypervisor communication mechanism, such as an application programming interface (API) or shared memory region. In response to the request, the virtual processor creation/deletion module 122 may create a virtual processor. To illustrate, the first guest operating system 130 is associated with four tasks 151, 152, 153, and 154. Each of the tasks 151, 152, 153, and 154 is associated with a corresponding virtual processor 131, 132, 133, and 134, respectively. Similarly, the second guest operating system 140 is associated with four tasks 161, 162, 163, and 164. Each of the tasks 161, 162, 163, and 164 is associated with a corresponding virtual

processor 141, 142, 143, and 144, respectively. The tasks 151-154 and 161-164 may be associated with user-mode (e.g., application level) or guest-mode (e.g., guest operating system level) routines or subroutines. The tasks 151-154 and 161-164 may correspond to a set of instructions (e.g., program code) that is stored in memory and that is executable by the processors 110. For example, a particular application may be associated with a "main" task and with one or more additional tasks that are generated by the "main task" (e.g., one or more graphical user interface (GUI) tasks, one or more data processing tasks, etc.).

[0028] In the system 100 of FIG. 1, virtual processors may be maintained as long as the corresponding tasks are maintained by the corresponding guest operating systems. When a task concludes, the virtual processor creation/deletion module 122 may delete and/or deallocate the virtual processor for the task. The virtual processor creation/deletion module 122 may detect that a task has concluded based on an exit or return command at the processors 110 or in response to a notification from a guest operating system. In a particular embodiment, creating a virtual processor may include allocating hardware and/or software resources to the virtual processor, and deleting a virtual processor may include deallocating hardware and/or software resources of the virtual processor.

[0029] It should be noted that from the perspective of the hypervisor 120, each of the virtual processors 131-134 and 141-144 is a schedulable entity. Conversely, from the perspective of the guest operating systems 130 and 140, each of the virtual processors 131-134 and 141-144 may be a hardware resource abstraction. Each of the tasks 151-154 and 161-164 may correspond to one or more threads or software routines/sub-routines initiated by the guest operating systems 130 and 140 or applications executing at the guest operating systems 130 and 140.

[0030] The priority-aware scheduler 121 may be configured to schedule the virtual processors 131-134 and 141-144 for execution. The scheduler 121 may be considered "priority aware," because the scheduler 121 has access to priority information of individual virtual processors. For example, in the embodiment of FIG. 1, each virtual processor has a virtual processor priority from 1 to 3. A priority of 1 may be a highest priority and a priority of 3 may be a lowest priority. In alternate embodiments, more or

fewer priority designations may be used. The priority of a virtual processor may be equal to, reflect, or may otherwise be based on a task priority of a task corresponding to the virtual processor. For example, the virtual processor 134 has a task priority of 2, which is equal to the task priority of the corresponding task 154. The task priority of the task 154 may be assigned by the guest operating system 130 during task creation, as further described herein. Thus, the priority priority-aware scheduler 121 has access to task priority information by virtue of having access to virtual processor priority information. Priority information may be accessible to the priority-aware scheduler 121 directly (e.g., by reading certain data structures in operating system memory, shared memory, or in registers) or indirectly (e.g., via an API).

[0031] In a particular embodiment, the priority-aware scheduler 121 may determine whether to schedule a first task or a second task based on a comparison of a first priority of a first virtual processor corresponding to a first task to a second priority of a second virtual processor corresponding to a second task. Further, the priority-aware scheduler 121 may make such comparisons between priorities of virtual processors corresponding to tasks of different operating systems. Thus, instead of automatically prioritizing all tasks of a particular guest operating system (e.g., a real-time operating system (RTOS)) over all tasks of another guest operating system, the priority-aware scheduler 121 may perform task-specific prioritization, which may decrease an overall latency (e.g., measured as an average time between task creation and execution, an average time between beginning and completing task execution, etc.) of the system 100. A RTOS may assign a highest available priority to low-latency (e.g., mission-critical) tasks so that such tasks are prioritized by the priority-aware scheduler 121, but may assign lower priorities to less important RTOS tasks, so that important tasks of other guest operating systems do not have to wait for the less important RTOS tasks to be scheduled.

[0032] In the embodiment of FIG. 1, both guest operating systems support task priorities from "1" to "3." However, these priorities are for illustration only. In alternate embodiments, guest operating systems may support non-overlapping or partially overlapping sets of priorities. For example, the first guest operating system 130 may be a real-time operating system that supports tasks priorities "1" to "3" and the second guest operating system 140 may be a non-real-time operating system that supports task priorities "2" to "4." Thus, real-time operating system(s) (and tasks

thereof) may have access to at least one elevated task priority level (e.g., task priority "1") that is unavailable to non real-time operating system(s) (and tasks thereof).

[0033] The hypervisor 120 may also include the virtual processor blocking/unblocking module 123. The virtual processor blocking/unblocking module 123 may be configured to block and unblock the virtual processors 131-134 and 141-144. It will be appreciated that because there is a one-to-one mapping between tasks and virtual processors in the system 100, the "thin" guest operating systems 130 and 140 may rely on the hypervisor 120 for blocking and unblocking of virtual processors instead of implementing task blocking and unblocking logic within the guest operating systems 130 and 140. This approach simplifies the design and implementations of the guest operating systems 130 and 140. Examples of blocking/unblocking situations may include, but are not limited to, when a task sleeps until a synchronization object (e.g., a mutex or semaphore) becomes available, until data is available or certain processing operations are completed (e.g., by another task), or until some other unblocking condition is satisfied.

[0034] The guest operating systems 130 and 140 may each be a "thin" operating system that includes little or no scheduling logic. Instead, the guest operating systems 130 and 140 rely on the hypervisor 120 for scheduling functions. The guest operating systems 130 and 140 may also rely on the hypervisor 120 for blocking and unblocking of tasks. It should be noted that although two guest operating systems are shown in FIG. 1, this is for example only. In alternate embodiments, more or fewer guest operating systems may be supported by the hypervisor 120. Moreover, the hypervisor 120 may simultaneously support one or more "thin" operating systems and one or more conventional operating systems that include scheduling and blocking/unblocking logic For example, the hypervisor 120 may perform priority-aware scheduling and blocking/unblocking for the "thin" operating systems but not for the full operating systems (e.g., operating systems that include their own scheduler). For example, FIG. 2 illustrates a hypervisor-controlled computing environment 200 in which the "thin" operating system 130 coexists with a conventional operating system 240 that includes a single virtual processor 241 and a "native" operating system scheduler 242. Other components of FIG. 2 may function as described with reference to corresponding components of FIG. 1.

[0035] During operation, the guest operating systems 130 and 140, and/or applications executing at the guest operating systems 130 and 140, may create tasks (e.g., in response to task creation requests issued by applications). In response to a task creation request, the guest operating systems 130 and 140 may request the hypervisor 120 to create a virtual processor to execute the requested task. For example, the virtual processor creation/deletion module 122 may create the virtual processor 134 in response to a request from the first guest operating system 130 associated with the task 154. The virtual processor creation/deletion module 122 may assign the created virtual processor 134 a priority of "2," reflecting the task priority of the task 154. The priority-aware scheduler 121 may schedule the virtual processor 134. For example, scheduling a virtual processor for execution may include placing the virtual processor in an execution queue of the hardware processors 110, allocating execution cycles or time slices of the processors 110 to the virtual processor, etc. The virtual processor blocking/unblocking module 123 may block and unblock the virtual processor 134 in response to requests from the first guest operating system 130 to block and unblock the task 154. Upon completion of the task 154, the virtual processor creation/deletion module 122 may delete the virtual processor 134.

[0036] In a particular embodiment, one or more of the guest operating systems 130 and 140 may be a RTOS. For example, the system 100 may be integrated into a wireless phone and one of the guest operating system 130 or 140 may be an operating system of a modem in the wireless phone that provides certain latency guarantees. To illustrate, the modem operating system may guarantee that scheduling latencies are under 20 microseconds (μs) to meet real-time deadlines. Conventional hypervisors, which may have latencies on the order of milliseconds, may not provide sufficient responsiveness if one of the operating systems is a real-time operating system. However, the techniques described herein may enable the hypervisor 120 of FIG. 1 to support reduced latencies that are acceptable to the modem operating system or other RTOS. The system 100 of FIG. 1 may thus be used in wireless phones and other wireless communication devices (e.g., laptop or tablet computing devices).

[0037] In a particular embodiment, the system 100 may be incorporated into an electronic device that includes one or more multithreaded and/or multi-core processors. For example, the system 100 may be used to implement real-time multithreaded

computing. A multithreaded processor can be implemented via a processor with multiple hardware execution contexts executing concurrently, or as multiple processors with a single hardware execution context (e.g., a symmetric multiprocessing (SMP) cluster).

[0038] The described techniques may be applicable in various electronic devices. To illustrate, mobile electronic devices (e.g., wireless phones) typically include modem software, modem firmware, and audio processing, each of which may utilize different operating systems or operating system images (e.g., due to different latency requirements). The system 100 of FIG. 1 may enable modem firmware and modem software to share the same DSP (e.g., multiple copies of the same operating system may be guest operating systems) while meeting latency requirements. In a particular embodiment, a wireless phone may include a guest RTOS for modem operations and another operating system for applications (e.g., a web server to support a laptop computer that uses the wireless phone for Wi-Fi connectivity), where both operating systems run on the same DSP instead of different processors (e.g., a device that includes a DSP and a separate application processor). It will be appreciated that enabling multiple (or all) operating systems of a device to run on the same processor or set of processors may decrease a production/manufacturing cost associated with the device.

[0039] The system 100 of FIG. 1 may thus provide a hypervisor-controlled computing environment without excessive context switching (e.g., switching between different schedulable entities), because one virtual processor is assigned per task. Each task of an operating system may be performed by a distinct virtual processor that is scheduled by the hypervisor 120 and executed by the processor(s) 110. Moreover, the decreased overhead from implementing a single scheduling layer may provide decreased latency, enabling the use of multiple real time operating systems in the system 100. The system 100 of FIG. 1 may represent a paradigm change in operating system design. Conventionally, an operating system is designed to run "on the metal" (i.e., in view of hardware implementations details), and a hypervisor is designed to support the operating system. In contrast, the system 100 of FIG. 1 reflects an operating system/hypervisor "co-design" paradigm, in which an operating system and a hypervisor are specifically implemented to rely on each other for certain functions,

which may reduce redundancy (e.g., one scheduler instead of multiple schedulers) and decrease latency.

[0040] FIG. 3 is a flowchart to illustrate a particular method 300 of operation at a hypervisor. In an illustrative embodiment, the method 300 may be performed by the hypervisor 120 of FIGS. 1-2.

[0041] The method 300 may include receiving, at a hypervisor, a request from a guest operating system to create a virtual processor to execute a task, where the task has a task priority assigned by the guest operating system, at 302. For example, in FIG. 1, the hypervisor 120 may receive a request from the second guest operating system 140 to create a virtual processor to execute the task 163, where the task 163 has a task priority of "1" assigned by the second guest operating system 140.

[0042] The method 300 may also include creating the virtual processor, where the virtual processor has a priority corresponding to the task priority, at 304. For example, in FIG. 1, the hypervisor 120 may create the virtual processor 143 to execute the task, where the virtual processor 143 has a priority of "1," which corresponds to the task priority of the task 163.

[0043] The method 300 may further include scheduling the virtual processor for execution, at 306, and blocking or unblocking the virtual processor based on a request from the guest operating system, at 308. For example, in FIG. 1, the hypervisor 120 may schedule the virtual processor 143 for execution and block/unblock the virtual processor 143 based on requests from the second guest operating system 140. The hypervisor may also delete or deallocate the virtual processor 143 upon completion of the task 163.

[0044] The method 300 of FIG. 3 may thus enable a hypervisor to create and perform scheduling (e.g., priority-aware scheduling) of virtual processors corresponding to operating system tasks. For example, by performing priority-aware scheduling, the hypervisor may enable a real-time operating system to meet latency requirements without letting the real-time operating system dominate hardware resources and negatively impact the performance of non-real-time operating systems. The method 300 of FIG. 3 may also enable the hypervisor to perform blocking and unblocking. "Thin"

operating systems that do not include scheduling and blocking/unblocking logic may rely on the hypervisor for scheduling and blocking/unblocking functionality, which may reduce redundancy and latency in a computing system.

[0045] The method 300 of FIG. 3 may be implemented by a digital signal processor (DSP), a processing unit such as a central processing unit (CPU), a controller, a field-programmable gate array (FPGA) device, an application-specific integrated circuit (ASIC), another hardware device, a firmware device, or any combination thereof. As an example, the method 300 of FIG. 3 can be performed by a processor that executes instructions, as described with respect to FIG. 5.

[0046] FIG. 4 is a flowchart to illustrate a particular method 400 of operation at an operating system (e.g., a guest operating system in a hypervisor-controlled computing environment). In an illustrative embodiment, the method 400 may be performed by the first guest operating system 130 of FIGS. 1-2 or the second guest operating system 140 of FIG. 1.

[0047] The method 400 may include receiving, at a guest operating system, a request to create a task, at 402. In a particular embodiment, the guest operating system may be a real-time operating system. For example, in FIG. 1, the first guest operating system 130 may be a real-time operating system that receives a request to create a task (e.g., the task 151) from an application executing on the first guest operating system 130.

[0048] The method 400 may also include, in response to the request to create the task, requesting a hypervisor to create a virtual processor to execute the requested task, at 404. For example, in FIG. 1, the first guest operating system 130 may request the hypervisor 120 to create a virtual processor (e.g., the virtual processor 131) to execute the requested task.

[0049] The method 400 may further include assigning a task priority to the task, where a priority of the virtual processor corresponds to the task priority and is accessible to the hypervisor, at 406. For example, in FIG. 1, the first guest operating system may assign a task priority of "1" to the task 151, and the virtual processor 131 may have the priority "1" corresponding to the task priority of the task 151.

[0050] The method 400 of FIG. 4 may thus enable an operating system to use priority-aware hypervisor scheduling of virtual processors corresponding to operating system tasks. "Thin" operating systems that do not include scheduling and blocking/unblocking logic may rely on the hypervisor for scheduling and blocking/unblocking functionality, which may reduce redundancy and latency in a computing system.

[0051] The method 400 of FIG. 4 may be implemented by a digital signal processor (DSP), a processing unit such as a central processing unit (CPU), a controller, a field-programmable gate array (FPGA) device, an application-specific integrated circuit (ASIC), another hardware device, a firmware device, or any combination thereof. As an example, the method 400 of FIG. 4 can be performed by a processor that executes instructions, as described with respect to FIG. 5.

[0052] Referring to FIG. 5, a block diagram of an electronic device 500 is shown. The electronic device 500 includes a processor 510, such as a digital signal processor (DSP), coupled to a memory 532. In a particular embodiment, the electronic device 500, or components thereof, may be included in a set top box, a music player, a video player, an entertainment unit, a navigation device, a communications device, a personal digital assistant (PDA), a fixed location data unit, a computing device, or any combination thereof.

[0053] FIG. 5 also shows a display controller 526 that is coupled to the processor 510 and to a display 528. A coder/decoder (CODEC) 534 can also be coupled to the processor 510. A speaker 536 and a microphone 538 can be coupled to the CODEC 534. FIG. 5 also indicates that a wireless controller 540 can be coupled to the processor 510 and to an antenna 542 (e.g., via a radio frequency (RF) interface).

[0054] The memory 532 may be a tangible non-transitory computer-readable or processor-readable storage medium that includes executable instructions 556. The instructions 556 may be executed by a processor, such as the processor 510 to perform various functions and methods, including the methods 300, 400 of FIGS. 3-4. The memory may also store instructions corresponding to a hypervisor 570 (e.g., the hypervisor 120 of FIGS. 1-2), a first guest operating system 572 (e.g., the first guest operating system 130 of FIGS. 1-2), a second guest operating system 574 (e.g., the

second guest operating system 140 of FIG. 1). Additional operating systems (e.g., the third guest operating system 240 of FIG. 2) may also be included.

[0055] In a particular embodiment, the processor 510, the display controller 526, the memory 532, the CODEC 534, and the wireless controller 540 are included in a system-in-package or system-on-chip device 522. In a particular embodiment, an input device 530 and a power supply 544 are coupled to the system-on-chip device 522. Moreover, in a particular embodiment, as illustrated in FIG. 5, the display 528, the input device 530, the speaker 536, the microphone 538, the antenna 542, and the power supply 544 are external to the system-on-chip device 522. However, each of the display 528, the input device 530, the speaker 536, the microphone 538, the antenna 542, and the power supply 544 can be coupled to a component of the system-on-chip device 522, such as an interface or a controller.

[0056] The described techniques enable the use of multiple guest operating systems, including one or more "thin" real-time operating systems that rely on a hypervisor for scheduling and blocking/unblocking functionality. In a particular embodiment, guest operating systems (and tasks thereof) may communicate with each other and may share synchronization primitives. For example, one guest operating system may be associated with a modem of a wireless communication device (e.g., a cellular telephone) and another guest operating system may be associated with audio processing (e.g., speech encoding/decoding).

[0057] In embodiments in which guest operating systems are able to communicate, security measures may be put in place. The security measures may be designed such that quality of service (QoS) guarantees (e.g., latency requirements for real-time operating systems) are maintained.

[0058] A communication mechanism (e.g., synchronization primitive) may be used for blocking/unblocking tasks/virtual processors (e.g., by the virtual processor blocking/unblocking module 123 of FIG. 1). In a particular embodiment, a single-bit or a multi-bit data value in memory may be used for the synchronization primitive. One or more rules may govern the use of the synchronization primitive. For example, a first rule may indicate that if the synchronization primitive is unlocked (e.g., set to zero), the synchronization primitive can be locked (e.g., set to one) by a thread (e.g., a task of a

guest operating system or a corresponding virtual processor). After the synchronization primitive is locked by a thread (also referred to as the thread "holding" the synchronization primitive), data protected by the synchronization primitive can be written by the thread. A second rule may indicate that if the synchronization primitive is locked, a thread seeking to acquire the synchronization primitive should block until the synchronization primitive is unlocked. In alternate embodiments, the synchronization primitive may be a multi-bit value representing more than two states. For example, an unlocked state may be represented by a first value, a locked-with-no-waiting-threads state may be represented by a second value, and a locked-with-at-least-one-waiting-thread may be represented by a third value. In a particular embodiment, the synchronization primitive may be stored at a user-mode address of memory that is shared between multiple guest operating systems (e.g., the synchronization primitive may be stored in userspace).

[0059] It will be appreciated that in a hypervisor-controlled environment with multiple guest operating systems, tasks in one guest operating system may depend on tasks in another guest operating system. For example, when a first task of a first guest operating system blocks because a second task of a second guest operating system holds a synchronization primitive, the first task of the first guest operating system may not be able to proceed until the second task of the second guest operating system releases the synchronization primitive. The blocked first task may have read-only access to the synchronization primitive but the second task may have read-write access to the synchronization primitive.

[0060] In a particular embodiment, a blocking function call available to tasks executing in guest operating systems may include two arguments: a pointer to the synchronization primitive and an expected value of the synchronization primitive. A hypervisor may check if the value of the synchronization primitive at the address indicated by the pointer is equal to an expected value. If the values are equal, blocking is successful. If the values are not equal, the hypervisor returns without blocking. A wakeup function may include two arguments: a pointer to the synchronization primitive and a maximum number of threads that can be woken up depending on the value of the synchronization primitive.

[0061] In a particular embodiment, a timeout may be implemented for blocking. For example, a task that blocks on a synchronization primitive may automatically wake up if the synchronization primitive is not released within a particular amount of time (e.g., in response to expiration of a timeout period).

[0062] In a particular embodiment, a global set of priorities may be used for operating systems in a hypervisor-controlled computing environment. In some cases, real-time operating system(s) may have access to one or more elevated priorities, of the global set of priorities, that non real-time operating systems do not have access to. When different tasks from different guest operating systems can have different priorities of a global set of priorities, priority inversion may occur. For example, priority inversion may occur when threads of differing priority compete for a synchronization primitive. Priority inversion may result in operating inefficiencies and potential deadlock. For example, a high priority thread (e.g., of a real-time operating system) may wait on a synchronization primitive that is being used by a low priority thread (e.g., of a non real-time operating system). The low priority thread (and by extension, the waiting high priority thread) may be preempted by the medium priority threads. Due to the preemption, the low priority thread may not be able to complete execution to release the synchronization primitive so that the high priority thread may acquire the synchronization primitive and continue execution. Since medium priority threads are effectively preempting a high priority thread, the priorities of the medium priority threads and the high priority thread may be said to be "inverted."

[0063] Priority inheritance is a technique that may be used to avoid priority inversion. In priority inheritance, when a high priority thread (e.g., priority = 1) blocks on a low priority thread (e.g., priority = 3), the low priority thread "inherits" the priority of the blocking high priority thread (e.g., the low priority thread inherits a priority = 1). This inheritance keeps medium priority threads (e.g., priority = 2) from causing priority inversion. In computing environments that include multiple guest operating systems, priority inversion may occur with respect to tasks of a single guest operating system or with respect to tasks of multiple guest operating systems.

[0064] FIG. 6 illustrates an embodiment of a system 600 that is operable to perform priority inversion. The system 600 includes the hypervisor 120 and one or more

multithreaded processors 660 (e.g., the processor(s) 110 of FIG. 1). A multithreaded processor can be implemented via a processor with multiple hardware execution contexts executing concurrently, or as multiple processors with a single hardware execution context (e.g., a symmetric multiprocessing (SMP) cluster). In a particular embodiment, the hypervisor 120 may implement a priority inversion avoidance scheme. According to the scheme, each guest operating system has a maximum allowed priority that is established when the guest operating system is loaded by the hypervisor 120. Each guest operating system can elevate its own tasks/threads (and by extension, virtual processors) to its maximum allowed priority. However, a guest operating system is prohibited from elevating its maximum allowed priority. Instead, only a higher-priority guest operating system can elevate a lower-priority task of another lower-priority guest operating system via the priority inheritance system. This enables the high priority task and guest operating system environment do determine whether task priorities should be adjusted when synchronization primitives are used.

[0065] In a particular embodiment, the hypervisor 120 may implement "lazy" priority inheritance. The hypervisor 120 may maintain control blocks 620 indicating a priority ("Prio"), a temporary priority ("Tprio"), and a status (e.g., "Running," "Blocked," "Ready," etc.) of tasks/virtual processors. When the priority-aware scheduler 121 determines that a first task is blocked from execution by a second lower priority task, the second task may inherit a temporary priority equal to the priority of the first task. The temporary priority may be set without modifying the contents of scheduling data structures.

[0066] For example, scheduling data structures may include a ready list 630 and a run list 640, as shown. The ready list 630 may indicate tasks (or corresponding virtual processors) that are ready for execution (e.g., not blocked on any synchronization primitives). The run list 640 may indicate tasks (or corresponding virtual processors) that are scheduled for execution and/or are being executed. In the embodiment illustrated, the ready list 630 and the run list 640 include one "slot" for each of four priorities available in the system 600. However, this is for illustration purposes only. In alternate embodiments, the ready list 630 and the run list 640 may be organized differently.

[0067] During normal operation when priority inversion does not occur, the "lazy" priority inversion may be low-cost, because only a temporary priority (designated "Tprio" in FIG. 6) is set by the hypervisor 120 and no scheduling data is affected. However, if the second task is preempted by another task, the second task (or corresponding virtual processor) may be rescheduled for execution using the temporary priority. The second task may execute at the higher temporary priority, enabling the system to make progress towards executing the first task.

[0068] In conjunction with the described embodiments, an apparatus is disclosed that includes means for processing. For example, the means for processing may include a hardware processor, such as the one or more processors 110 of FIGS. 1-2 or the processor 510 of FIG. 5, one or more other devices or circuits to process data, or any combination thereof. The apparatus also includes means for scheduling that is executable by the means for processing. The means for scheduling is configured to schedule virtual processors for execution by the means for processing, each of the virtual processors having a priority accessible to the means for scheduling and corresponding to a task priority of a corresponding task, the task priority assigned by a first guest operating system. For example, the means for scheduling may be the priority-aware scheduler 121 of FIGS. 1-2, the hypervisor 570 of FIG. 5, one or more other devices, circuits, modules, or instructions to schedule a virtual processor for execution, or any combination thereof.

[0069] In a particular embodiment, an apparatus includes a processor and a guest operating system. The guest operating system includes means for requesting a hypervisor to create a virtual processor to execute a task in response to receiving a request to create the task. The virtual processor may be scheduled on the processor. For example, the means for requesting may be a component (e.g., an operating system-hypervisor communication mechanism, such as an API or shared memory region) of the guest operating system 130 of FIGS. 1-2, a component of the guest operating system 140 of FIG. 1, one or more other devices, circuits, modules, or instructions to request a hypervisor to create a virtual processor, or any combination thereof.

[0070] Those of skill would further appreciate that the logical blocks, configurations, modules, circuits, and algorithm steps described in connection with the embodiments

disclosed herein may be implemented as electronic hardware, computer software executed by a processor, or combinations of both. Various illustrative components, blocks, configurations, modules, circuits, and steps have been described above generally in terms of their functionality. Whether such functionality is implemented as hardware or processor executable instructions depends upon the particular application and design constraints imposed on the overall system. Skilled artisans may implement the described functionality in varying ways for each particular application, but such implementation decisions should not be interpreted as causing a departure from the scope of the present disclosure. A multithreaded processor can be implemented via a processor with multiple hardware execution contexts executing concurrently, or as multiple processors with a single hardware execution context (e.g., a symmetric multiprocessing (SMP) cluster).

[0071] The steps of a method or algorithm described in connection with the embodiments disclosed herein may be embodied directly in hardware, in a software module executed by a processor, or in a combination of the two. A software module may reside in random access memory (RAM), flash memory, read-only memory (ROM), programmable read-only memory (PROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), registers, hard disk, a removable disk, a compact disc read-only memory (CD-ROM), or any other form of non-transient storage medium known in the art. An exemplary storage medium is coupled to the processor such that the processor can read information from, and write information to, the storage medium. In the alternative, the storage medium may be integral to the processor. The processor and the storage medium may reside in an application-specific integrated circuit (ASIC). The ASIC may reside in a computing device or a user terminal. In the alternative, the processor and the storage medium may reside as discrete components in a computing device or user terminal.

[0072] The previous description of the disclosed embodiments is provided to enable a person skilled in the art to make or use the disclosed embodiments. Various modifications to these embodiments will be readily apparent to those skilled in the art, and the principles defined herein may be applied to other embodiments without departing from the scope of the disclosure. Thus, the present disclosure is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope

possible consistent with the principles and novel features as defined by the following claims.

CLAIMS

1. An apparatus comprising:

a processor; and

a hypervisor,

wherein the hypervisor is configured to schedule virtual processors for execution by the processor, each of the virtual processors having a priority accessible to the hypervisor and corresponding to a task priority of a corresponding task, the task priority assigned by a first guest operating system.

2. The apparatus of claim 1, wherein the hypervisor has access to priority information associated with tasks of the guest operating system and tasks of a second guest operating system.

3. The apparatus of claim 2, wherein the hypervisor is configured to determine whether to schedule a first task of the first guest operating system or a second task of the second guest operating system based at least in part on a comparison of a first priority of a first virtual processor corresponding to the first task to a second priority of a second virtual processor corresponding to the second task.

4. The apparatus of claim 2, wherein at least one of the first guest operating system and the second guest operating system is a real-time operating system.

5. The apparatus of claim 4, wherein the real-time operating system has access to at least one elevated task priority level that is unavailable to non real-time operating systems.

6. The apparatus of claim 1, wherein the hypervisor is configured to block or unblock a virtual processor based on a request from a task of a guest operating system.

7. The apparatus of claim 1, wherein a first task of the first guest operating system and a second task of a second guest operating system are configured to access a common synchronization primitive.

8. The apparatus of claim 7, wherein the common synchronization primitive comprises a data value stored in a memory that is shared by the first guest operating system and the second guest operating system, wherein the data value is stored in a user-mode address of the memory.

9. The apparatus of claim 7, wherein when a task of a particular guest operating system holds the synchronization primitive, the task of the particular guest operating system has read-write access to the synchronization primitive and at least one task of another guest operating system, the task of the other guest operating system blocked on the synchronization primitive, has read-only access to the synchronization primitive.

10. The apparatus of claim 9, wherein the at least one task of the other guest operating system is unblocked upon expiration of a timeout period when the synchronization primitive is not released during the timeout period.

11. The apparatus of claim 1, wherein priorities of tasks of the first guest operating system and priorities of tasks of a second guest operating system are selected from a common set of priorities.

12. The apparatus of claim 11, wherein the first guest operating system has a first maximum priority that the first guest operating system can assign to tasks of the first guest operating system, and wherein the second guest operating system has a second maximum priority that the second guest operating system can assign to tasks of the second guest operating system.

13. The apparatus of claim 12, wherein the first maximum priority is different from the second maximum priority.

14. The apparatus of claim 1, wherein the hypervisor is configured to temporarily elevate a priority of a task of one of the first guest operating system and a second guest operating system when a task of the other of the first guest operating system and the second guest operating system blocks.

15. A method comprising:

receiving, at a hypervisor of a computing device, a request from a guest operating system to create a virtual processor to execute a task;

creating the virtual processor; and

scheduling the virtual processor for execution on a hardware processor of the computing device, wherein the virtual processor has a priority corresponding to a task priority of the task, the task priority assigned by the guest operating system.

16. The method of claim 15, further comprising blocking or unblocking the virtual processor based on a request from the task of the guest operating system.

17. The method of claim 15, further comprising determining to schedule the virtual processor before a second virtual processor associated with a second task.

18. The method of claim 17, wherein determining to schedule the virtual processor before the second virtual processor comprises determining that the priority of the virtual processor is higher than a second priority of the second virtual processor.

19. The method of claim 15, wherein a first task of the guest operating system and a second task of a second guest operating system are configured to access a common synchronization primitive.

20. The method of claim 19, wherein the common synchronization primitive comprises a data value stored in a memory that is shared by the guest operating system and the second guest operating system, wherein the data value is stored in a user-mode address of the memory.

21. The method of claim 19, wherein when a task of a particular guest operating system holds the synchronization primitive, the task of the particular guest operating system has read-write access to the synchronization primitive and at least one task of another guest operating system, the task of the other guest operating system blocked on the synchronization primitive, has read-only access to the synchronization primitive.

22. The method of claim 21, wherein the at least one task of the other guest operating system is unblocked upon expiration of a timeout period when the synchronization primitive is not released during the timeout period.

23. The method of claim 15, wherein priorities of tasks of the guest operating system and priorities of tasks of a second guest operating system are selected from a common set of priorities.

24. The method of claim 15, wherein the guest operating system has a first maximum priority that the guest operating system can assign to tasks of the guest operating system, and wherein a second guest operating system has a second maximum priority that the second guest operating system can assign to tasks of the second guest operating system.

25. The method of claim 24, wherein the first maximum priority is different from the second maximum priority.

26. The method of claim 15, wherein the hypervisor is configured to temporarily elevate a priority of a task of one of the guest operating system and a second guest operating system when a task of the other of the guest operating system and the second guest operating system blocks.

27. An apparatus comprising:

means for processing; and

means for scheduling executable by the means for processing,

wherein the means for scheduling is configured to schedule virtual processors for execution by the means for processing, each of the virtual processors having a priority accessible to the means for scheduling and corresponding to a task priority of a corresponding task, the task priority assigned by a first guest operating system.

28. The apparatus of claim 27, wherein the means for scheduling is integrated into a hypervisor.

29. A non-transitory computer-readable medium comprising instructions that, when executed by a computer, cause the computer to:

receive, at a hypervisor, a request from a guest operating system to create a virtual processor to execute a task;

create the virtual processor; and

schedule the virtual processor for execution, wherein the virtual processor has a priority corresponding to a task priority of the task, the task priority assigned by the guest operating system.

30. The non-transitory computer-readable medium of claim 29, further comprising instructions that, when executed by the computer, cause the computer to block or unblock the virtual processor based on a request from the task of the guest operating system.

31. An apparatus comprising:

a processor; and

a guest operating system,

wherein in response to receiving a request to create a task, the guest operating system requests a hypervisor to create a virtual processor to execute the task, the virtual processor schedulable on the processor.

32. The apparatus of claim 31, wherein the guest operating system relies on the hypervisor to schedule tasks for execution by scheduling virtual processors corresponding to the tasks for execution.

33. The apparatus of claim 31, wherein the guest operating system is a real-time operating system.

34. The apparatus of claim 31, further comprising a second guest operating system executable by the processor.

35. The apparatus of claim 34, wherein each task of the second guest operating system is executed by a corresponding virtual processor that is requested by the second guest operating system.

36. The apparatus of claim 34, wherein a first task of the guest operating system and a second task of the second guest operating system are configured to access a common synchronization primitive.

37. The apparatus of claim 36, wherein the common synchronization primitive comprises a data value stored in a memory that is shared by the guest operating system and the second guest operating system, wherein the data value is stored in a user-mode address of the memory.

38. The apparatus of claim 36, wherein when a task of a particular guest operating system holds the synchronization primitive, the task of the particular guest operating system has read-write access to the synchronization primitive and at least one task of another guest operating system, the task of the other guest operating system blocked on the synchronization primitive, has read-only access to the synchronization primitive.

39. The apparatus of claim 38, wherein the at least one task of the other guest operating system is unblocked upon expiration of a timeout period when the synchronization primitive is not released during the timeout period.

40. The apparatus of claim 34, wherein priorities of tasks of the guest operating system and priorities of tasks of the second guest operating system are selected from a common set of priorities.

41. The apparatus of claim 34, wherein the guest operating system has a first maximum priority that the guest operating system can assign to tasks of the guest operating system, and wherein the second guest operating system has a second maximum priority that the second guest operating system can assign to tasks of the second guest operating system.

42. The apparatus of claim 41, wherein the first maximum priority is different from the second maximum priority.

43. The apparatus of claim 34, wherein the hypervisor is configured to temporarily elevate a priority of a task of one of the guest operating system and the second guest operating system when a task of the other of the guest operating system and the second guest operating system blocks.

44. A method comprising:

receiving, at a guest operating system executing at a computing device comprising a hypervisor and at least one processor, a request to create a task; and

in response to the request to create the task, requesting the hypervisor to create a virtual processor to execute the requested task, the virtual processor schedulable on the at least one processor.

45. The method of claim 44, further comprising assigning a task priority to the task, wherein a priority of the virtual processor corresponds to the task priority and is accessible to the hypervisor.

46. The method of claim 44, wherein the guest operating system is a real-time operating system.

47. The method of claim 46, wherein the real-time operating system has access to at least one elevated task priority level that is unavailable to a non real-time operating system.

48. The method of claim 44, wherein a first task of the guest operating system and a second task of a second guest operating system are configured to access a common synchronization primitive.

49. The method of claim 48, wherein the common synchronization primitive comprises a data value stored in a memory that is shared by the guest operating system and the second guest operating system, wherein the data value is stored in a user-mode address of the memory.

50. The method of claim 48, wherein when a task of a particular guest operating system holds the synchronization primitive, the task of the particular guest operating system has read-write access to the synchronization primitive and at least one task of another guest operating system, the task of the other guest operating system blocked on the synchronization primitive, has read-only access to the synchronization primitive.

51. The method of claim 50, wherein the at least one task of the other guest operating system is unblocked upon expiration of a timeout period when the synchronization primitive is not released during the timeout period.

52. The method of claim 44, wherein priorities of tasks of the guest operating system and priorities of tasks of a second guest operating system are selected from a common set of priorities.

53. The method of claim 44, wherein the guest operating system has a first maximum priority that the guest operating system can assign to tasks of the guest operating system, and wherein a second guest operating system has a second maximum priority that the second guest operating system can assign to tasks of the second guest operating system.

54. The method of claim 53, wherein the first maximum priority is different from the second maximum priority.

55. The method of claim 44, wherein the hypervisor is configured to temporarily elevate a priority of a task of one of the guest operating system and a second guest operating system when a task of the other of the guest operating system and the second guest operating system blocks.

56. An apparatus comprising:

a processor; and

a guest operating system,

wherein the guest operating system comprises means for requesting a hypervisor to create a virtual processor to execute a task in response to receiving a request to create the task, the virtual processor schedulable on the processor.

57. The apparatus of claim 56, wherein the guest operating system is a real-time operating system.

58. The apparatus of claim 56, further comprising a second guest operating system.

59. A non-transitory computer-readable medium comprising instructions that, when executed by a computer, cause the computer to:

receive, at a guest operating system, a request to create a task; and

in response to the request to create the task, request a hypervisor to create a virtual processor to execute the requested task, the virtual processor schedulable on a hardware processor.

60. The non-transitory computer-readable medium of claim 59, wherein the guest operating system is a real-time operating system.

61. The non-transitory computer-readable medium of claim 60, wherein the real-time operating system has access to at least one elevated task priority level that is unavailable to a non real-time operating system.

62. An electronic device comprising:

a processor;

a hypervisor;

a first guest operating system; and

a second guest operating system,

wherein at least one task of the first guest operating system is executed by a first virtual processor requested by the first guest operating system from the hypervisor, the first virtual processor executed by the processor, and

wherein at least one task of the second guest operating system is executed by a second virtual processor requested by the second guest operating system from the hypervisor, the second virtual processor executed by the processor.

63. The electronic device of claim 62, wherein the first guest operating system comprises a real-time operating system, wherein the second guest operating system comprises a non real-time operating system, and wherein a first task of the first guest operating system has access to at least one elevated task priority level that is not available to a second task of the second guest operating system.

64. The electronic device of claim 62, wherein a first task of the first guest operating system and a second task of the second guest operating system are configured to access a common synchronization primitive.

65. The electronic device of claim 64, wherein the common synchronization primitive comprises a data value stored in a memory that is shared by the first guest operating system and the second guest operating system, wherein the data value is stored in a user-mode address of the memory.

66. The electronic device of claim 64, wherein when a task of a particular guest operating system holds the synchronization primitive, the task of the particular guest operating system has read-write access to the synchronization primitive and at least one task of another guest operating system, the task of the other guest operating system blocked on the synchronization primitive, has read-only access to the synchronization primitive.

67. The electronic device of claim 66, wherein the at least one task of the other guest operating system is unblocked upon expiration of a timeout period when the synchronization primitive is not released during the timeout period.

68. The electronic device of claim 62, wherein priorities of tasks of the first guest operating system and priorities of tasks of the second guest operating system are selected from a common set of priorities.

69. The electronic device of claim 62, wherein the first guest operating system has a first maximum priority that the first guest operating system can assign to tasks of the first guest operating system, and wherein the second guest operating system has a second maximum priority that the second guest operating system can assign to tasks of the second guest operating system.

70. The electronic device of claim 69, wherein the first maximum priority is different from the second maximum priority.

71. The electronic device of claim 62, wherein the hypervisor is configured to temporarily elevate a priority of a task of one of the first guest operating system and the second guest operating system when a task of the other of the first guest operating system and the second guest operating system blocks.

72. The electronic device of claim 62, wherein the processor comprises a digital signal processor integrated into a device selected from the group consisting of a set top box, a music player, a video player, an entertainment unit, a navigation device, a communications device, a personal digital assistant (PDA), a fixed location data unit, and a computing device.

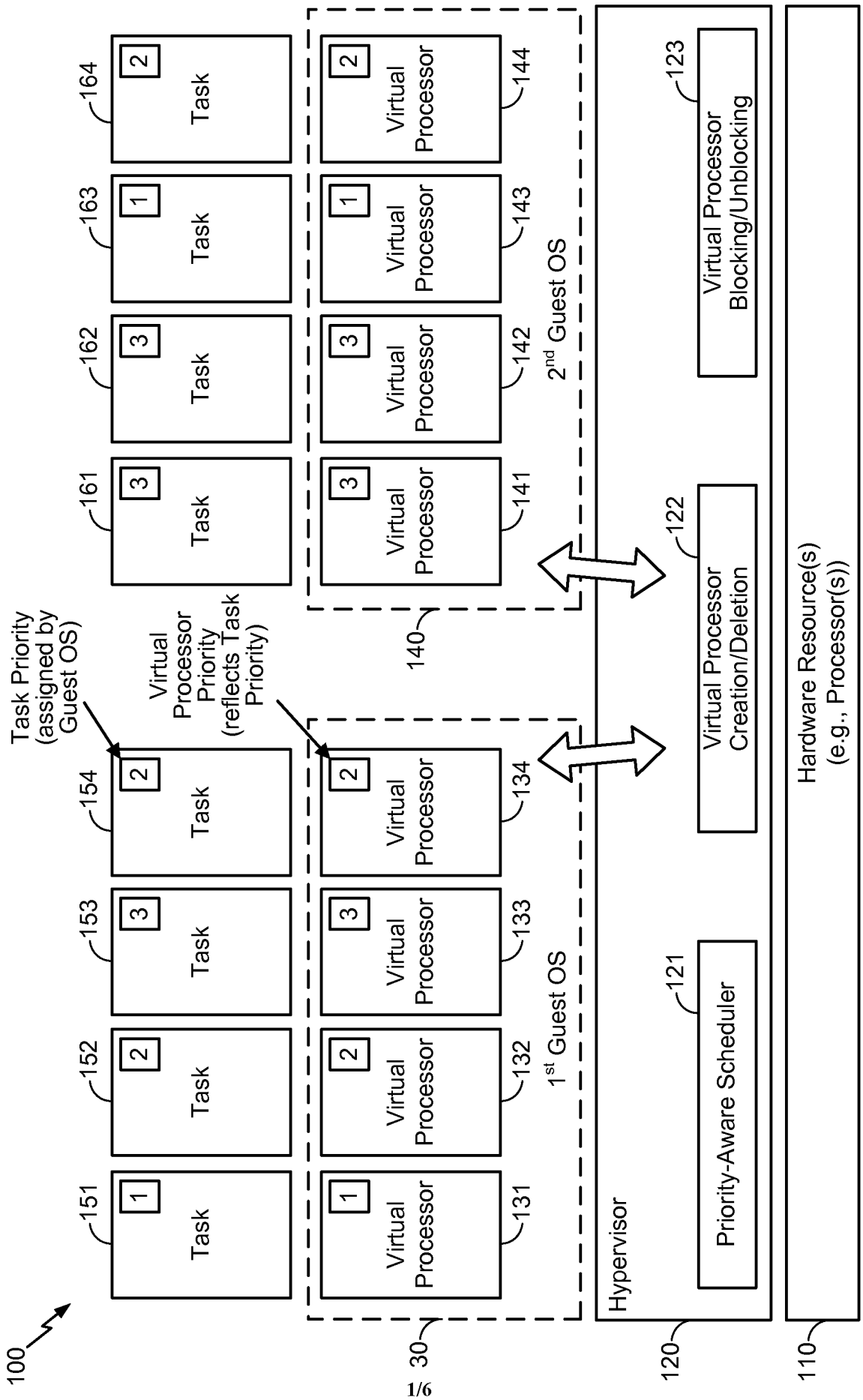
100
30
120
110
1st Guest OS
2nd Guest OS
Hypervisor
Hardware Resource(s)
(e.g., Processor(s))
Task Priority
(assigned by
Guest OS)
Virtual
Processor
Priority
(reflects Task
Priority)
Task
Virtual
Processor
Priority-Aware Scheduler
Virtual Processor
Creation/Deletion
Virtual Processor
Blocking/Unblocking
131
132
133
134
140
141
142
143
144
151
152
153
154
161
162
163
164
121
122
123

FIG. 1

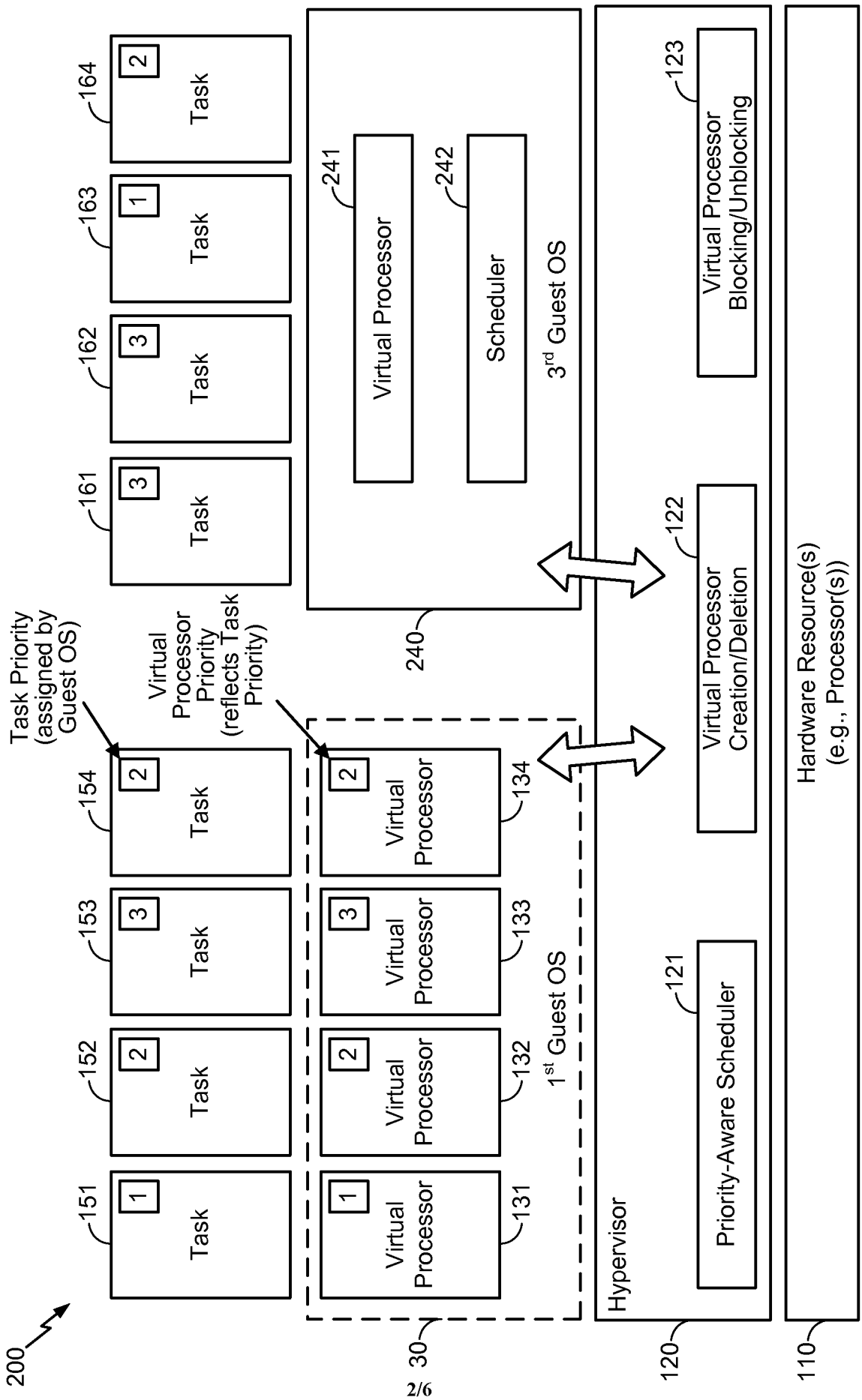
200
30
120
110
Hypervisor
Hardware Resource(s)
(e.g., Processor(s))
Priority-Aware Scheduler
121
Virtual Processor
Creation/Deletion
122
Virtual Processor
Blocking/Unblocking
123
1st Guest OS
3rd Guest OS
240
Virtual Processor
241
Scheduler
242
Virtual Processor
131
Virtual Processor
132
Virtual Processor
133
Virtual Processor
134
Task
151
Task
152
Task
153
Task
154
Task
161
Task
162
Task
163
Task
164
Task Priority
(assigned by
Guest OS)
Virtual
Processor
Priority
(reflects Task
Priority)

FIG. 2

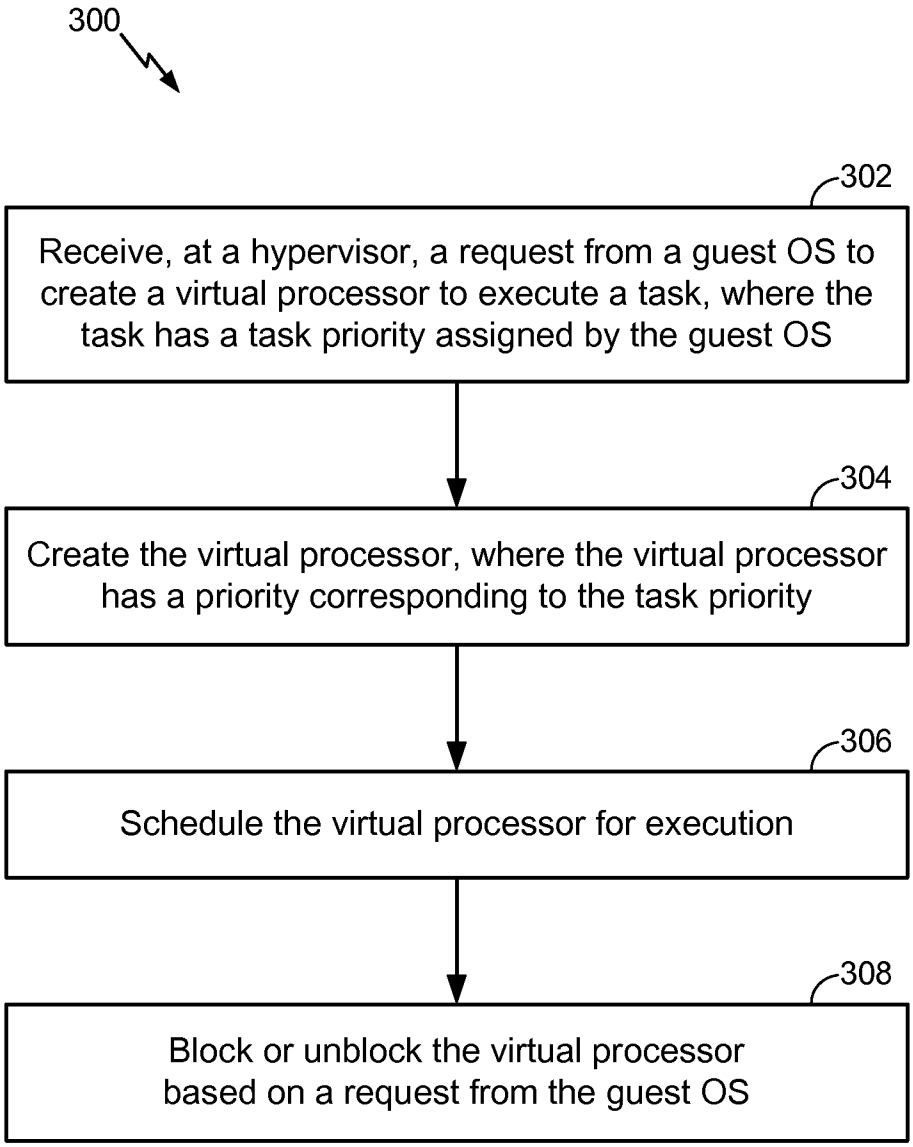
300
302
Receive, at a hypervisor, a request from a guest OS to create a virtual processor to execute a task, where the task has a task priority assigned by the guest OS
304
Create the virtual processor, where the virtual processor has a priority corresponding to the task priority
306
Schedule the virtual processor for execution
308
Block or unblock the virtual processor based on a request from the guest OS

FIG. 3

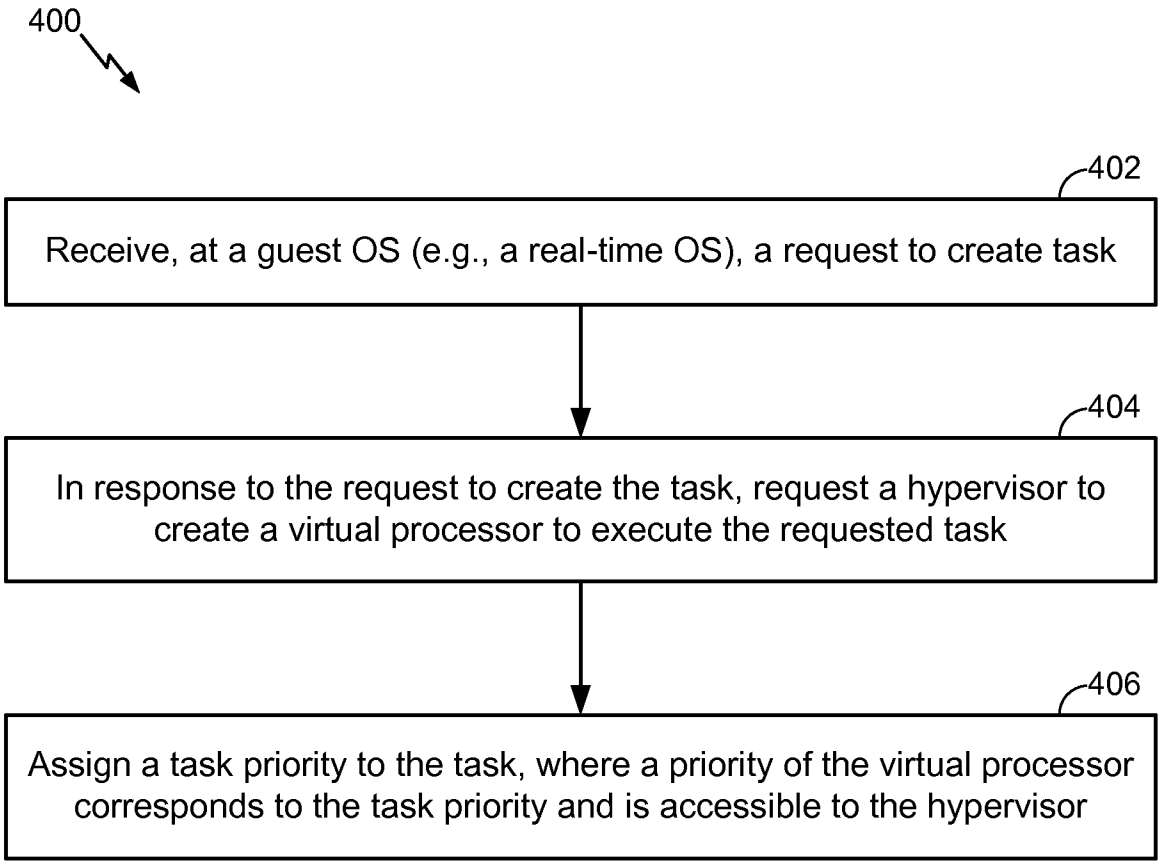
400
402
Receive, at a guest OS (e.g., a real-time OS), a request to create task
404
In response to the request to create the task, request a hypervisor to create a virtual processor to execute the requested task
406
Assign a task priority to the task, where a priority of the virtual processor corresponds to the task priority and is accessible to the hypervisor

FIG. 4

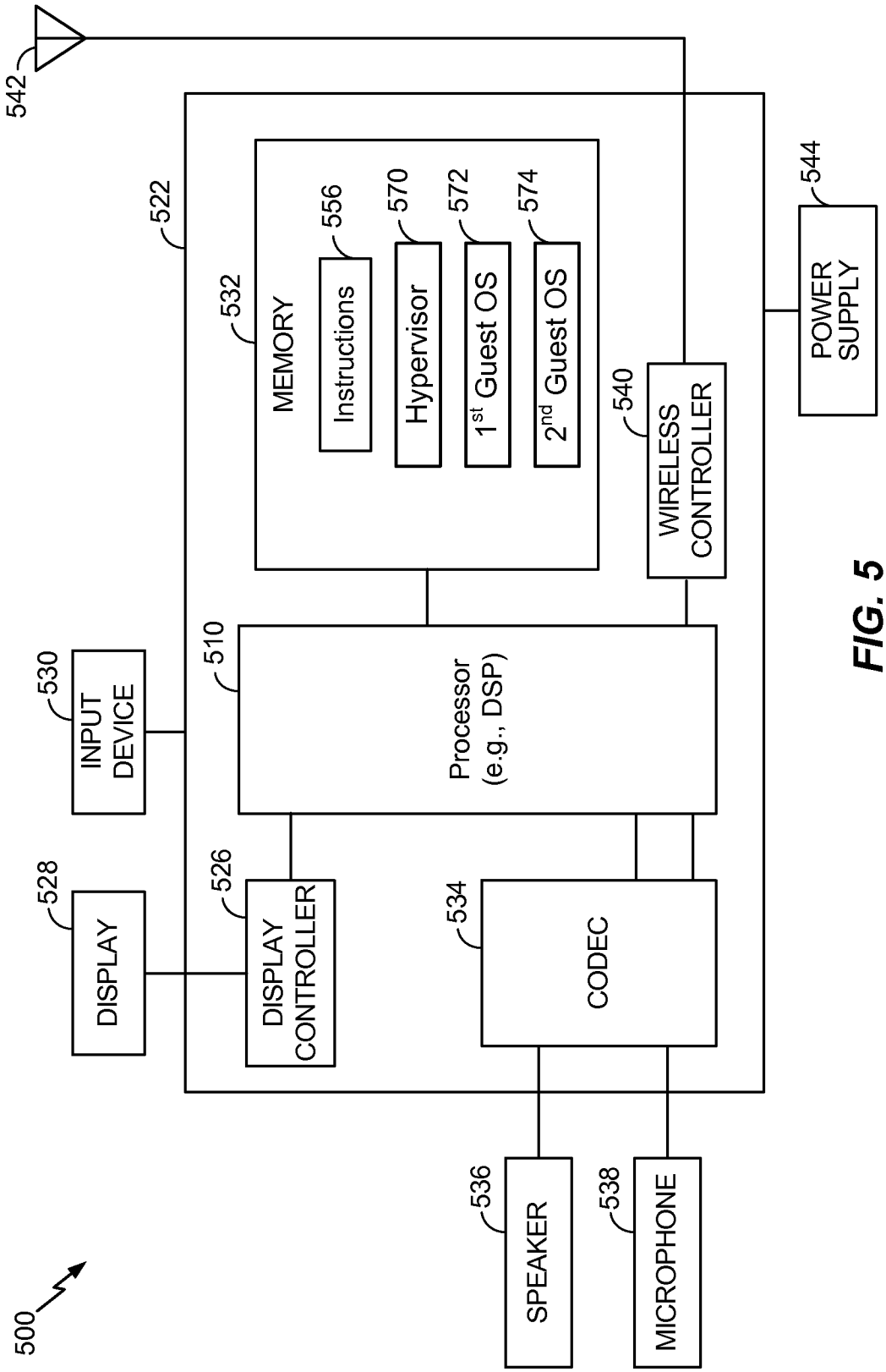
542
522
532
MEMORY
556
Instructions
570
Hypervisor
572
1st Guest OS
574
2nd Guest OS
540
WIRELESS CONTROLLER
544
POWER SUPPLY
530
INPUT DEVICE
510
Processor (e.g., DSP)
528
DISPLAY
526
DISPLAY CONTROLLER
534
CODEC
536
SPEAKER
538
MICROPHONE
500

FIG. 5

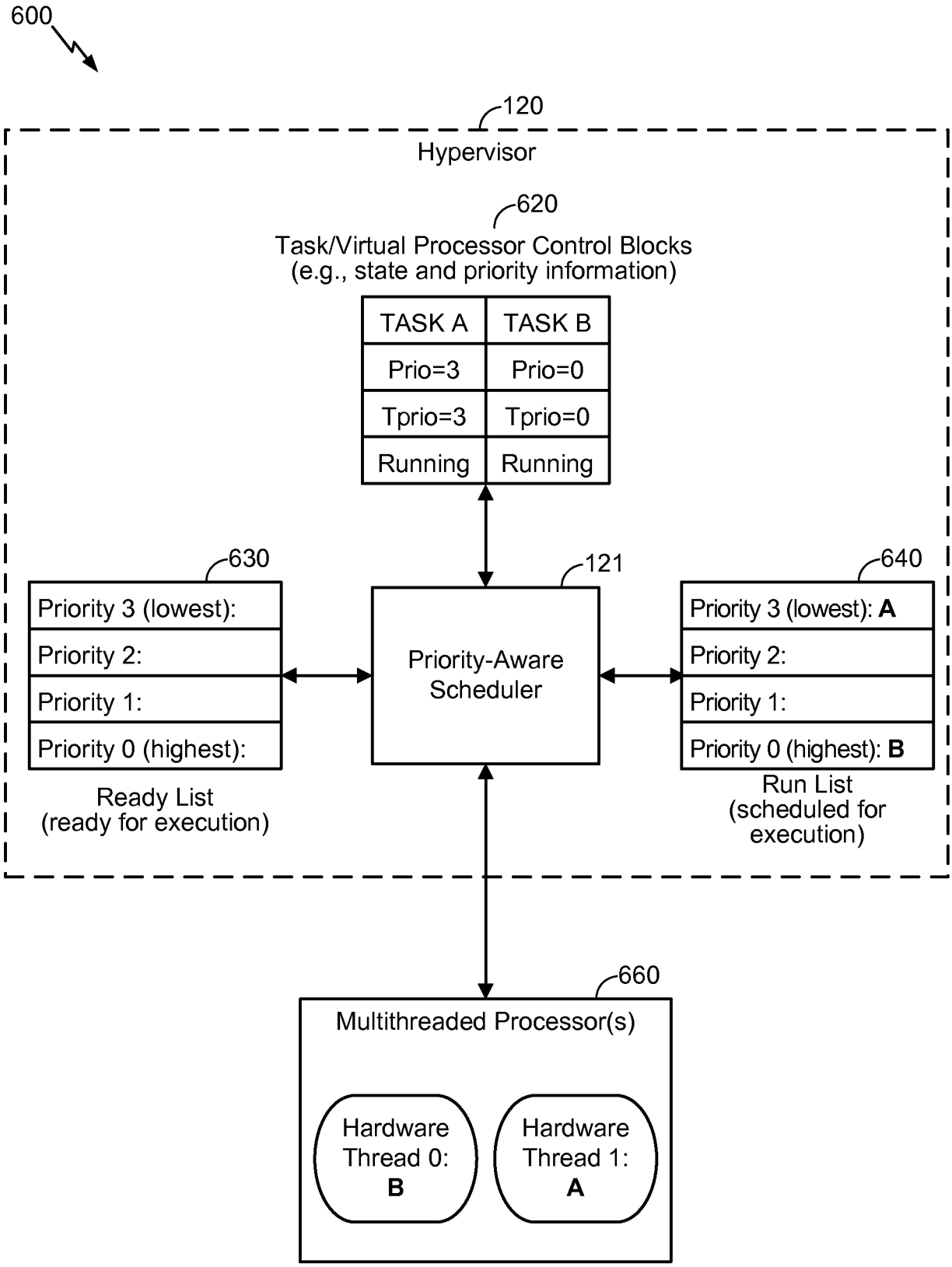
600
120
Hypervisor
620
Task/Virtual Processor Control Blocks
(e.g., state and priority information)
TASK A
TASK B
Prio=3
Prio=0
Tprio=3
Tprio=0
Running
Running
630
121
640
Priority 3 (lowest):
Priority 2:
Priority 1:
Priority 0 (highest):
Priority-Aware
Scheduler
Priority 3 (lowest): A
Priority 2:
Priority 1:
Priority 0 (highest): B
Ready List
(ready for execution)
Run List
(scheduled for
execution)
660
Multithreaded Processor(s)
Hardware
Thread 0:
B
Hardware
Thread 1:
A

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2014/023708

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/48 G06F9/455
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, INSPEC, IBM-TDB, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2012/240112 A1 (NISHIGUCHI NAOKI [JP] ET AL) 20 September 2012 (2012-09-20) paragraph [0003] - paragraph [0007] paragraph [0051] paragraph [0056] - paragraph [0060] -----	1-72

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
24 June 2014	03/07/2014

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Brandt, Jan

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2014/023708

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012240112 A1	20-09-2012	JP 2012194710 A	11-10-2012
		US 2012240112 A1	20-09-2012