REAL-TIME SCHEDULING OF VIRTUAL MACHINES

BACKGROUND

1. Field

This disclosure relates to virtual machines and, more particularly, to real-time computations within virtual machines.

2. Background Information

Virtual computing machines, which are commonly referred to as virtual machines (VMs) have typically been employed in a number of specific situations. A virtual machine may emulate, for example, a processor or microprocessor. VMs, at a high level, are typically software implementations of electronic hardware. Current embodiments of VMs are typically employed in batch-processing, time-sharing applications and security applications. Other uses of VMs include emulating hardware that may still be in development and not yet implemented. Such a use of a VM may be useful in debugging hardware during its development prior to actual implementation. This use of a VM may reduce development time and cost for the hardware that is being emulated in software. In this regard, a processor, which may currently be in development, could be implemented as a VM by employing a system that embodies a prior generation processor.

Prior implementations of VMs may be referred to as non-real-time. "Non-real-time," in this context, means that a VM operates without any requirement that any specific computation or operation is accomplished by a well-defined deadline. Contrariwise, "real-time" means, in this context, that computations or operations upon data available at one substantially predetermined time are to be completed by another substantially predetermined time, which may be referred to as a deadline, as previously indicated. Typically, real-time applications comprise a hierarchy of such deadlines.

Emerging technologies, such as, for example, multimedia applications, typically contain real-time subsystems. These subsystems may include software components that employ such real-time deadlines. These deadlines, typically, would need to be met for such applications to function satisfactorily. Therefore, a need

may exist for virtual machine configurations that may be employed for use with such technologies.

BRIEF DESCRIPTION OF THE DRAWINGS

The subject matter regarded as the invention is particularly pointed out and distinctly claimed in the concluding portion of the specification. The invention, the however, both as to organization and method of operation, together with objects, features, and advantages thereof, may best be understood by reference to the following detailed description when read with accompanying drawings in which:

FIG. 1 is a block diagram illustrating an embodiment for determining interrupt periodicity, Y, in embodiments of virtual machine (VM) configurations in accordance with the invention.

FIG. 2 is a block diagram of an embodiment of a state machine that may be employed by the embodiment of FIG. 1, for example.

FIG. 3 is a block diagram illustrating an embodiment of a feedback loop that may be employed for determining VM resource requirements in accordance with the invention.

FIG. 4 is a block diagram illustrating an embodiment of a virtual machine configuration in accordance with the invention.

FIG. 5 is tables illustrating an embodiment of a real-time schedule for embodiments of virtual machine configurations in accordance with the invention, such as illustrated in FIG. 4, for example.

FIG. 6 is a block diagram illustrating a prior art embodiment of a virtual machine configuration.

DETAILED DESCRIPTION

In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known methods, procedures, components and circuits have not been described in detail so as not to obscure the present invention.

As was indicated above, virtual machines (VMs) have typically been employed in time- sharing and batch processing applications. Additionally, VMs have been employed in security applications as well. The use of virtual machines is, of course, not limited to these situations and the foregoing are provided only by way of example. Current VMs, however, are not adequate to support real-time applications. As the processing capability of computing systems and use of real-time applications, such as multimedia applications, increase, techniques for employing VMs for use with real-time applications are desirable.

FIG. 6 illustrates a prior art embodiment, 600, of a virtual machine configuration that may be employed in batch or time-sharing applications, as previously discussed. In this particular embodiment, bare machine 610 comprises a computing platform, such computing platforms or systems typically comprise electronic hardware. This hardware may be capable, for example, of executing a standard operating system (OS) or a virtual machine monitor (VMM), such as 615. Such a VMM, though typically implemented in software, may export a bare machine interface, such as an emulation, to higher level software. Such higher level software may comprise a standard or real-time OS, although the invention is not limited in scope in this respect and, alternatively, for example, a VMM may be run within, or on top of, another VMM. VMMs and their typical features and functionality are well-known by those skilled in the art and may be implemented, for example, in software, firmware or by a combination of various techniques.

In such prior art embodiments, a VMM, at a minimum, typically schedules the VMs. This scheduling may allow the VMs to share the computing resources of, for example, bare machine 610. Such scheduling is typically done with static or predetermined allocation sequences. For example, VM A 620 may be allocated 25 percent of bare machines 610's computing resources and VM B 630 may be allocated 75 percent of bare machine 610's computing resources. However, the actual allocation may vary and depend, at least in part, on the particular embodiment and the applications being run within the VMs. In such an embodiment, the actual computing resource requirements of the VMs are not accessible to the VMM. Additionally, VM A 620 and VM B 630 are typically configured such that the VMs would not interface with one another. In this respect, the VMs are configured such

that it is not apparent to each VM that the bare machine resources are being shared. As part of scheduling VM A 620 and VM B 630 in this embodiment, VMM 615 is capable of suspending one VM and retaining information necessary for restarting that VM. For example, at the end of a period of time that VM A 620 is utilizing the bare machine's computing resources, VMM 615 may suspend VM A 620's activities, store data related to the current activities of VM A 620, restore data related to a prior suspension of VM B 630 and allocate, or schedule, VM B its 75% share of bare machine 610's computing resources, for this example. However, scheduling percentages in such an embodiment are not guaranteed over any specific time period. In this respect, for this embodiment, VM A or VM B may not receive any share of bare machine 610's computing resources over a specific period of time due to, for example, contention for various resources, such as a disk drive, as one example.

In this particular embodiment, each VM would then allocate its share of bare machine 610's computing resources to any user applications running in that particular VM, such as application 1 640 and application 2 650 for VM A 620, for example. In this context, user applications are software that may be used for example, by a personal computer user. An example of such a user application may be a word processing program, as one example. As was previously indicated, because VMM 615 typically does not have access to information regarding how each VM allocates its share of computing resources, scheduling of the VMs to achieve real-time deadlines, such as displaying a certain number of frames of a video clip in a certain period of time, is problematic.

FIG. 4 illustrates an embodiment, 400, of a virtual machine configuration that may be employed by embodiments of methods of real-time scheduling in accordance with the invention. The virtual machine configuration illustrated by embodiment 400 comprises an interface, which may be implemented in software. This interface may, for example, couple OS 1 425, application 1 450 and application 2 455 to VMM 415. In this particular embodiment, this interface further comprises a resource management application, such as 475 or 480, although the invention is not limited in scope to use of such resource management applications, and embodiments without such a resource management application may exist. Resource management

applications are well known in the art and the invention is not limited in scope to the use of any particular resource management application, nor to the use of a resource management application at all. In such embodiments, scheduling information, which is discussed in more detail hereinafter, may be communicated to a VMM, such as 415, from an OS, an application programming interface (API); such as APIs 430 and 445, a user application, or a resource management application.

In embodiments such as 400, an OS, API, user application or resource management application may communicate one or more parameters that may be used by a scheduler embodied in a VMM, such as 415 to schedule VMs such that real-time deadlines of applications executing within a VM may be met. For such an embodiment, when employed by a method in accordance with the invention, it is assumed that computing activities in the VMs are typically event driven and that device interrupts are typically the root-level events upon which such computing activities are, at least in part, based. Thus real-time scheduling of VMs is typically based on such interrupts. These interrupts are typically periodic and, for example, may be a clock interrupt or video display interrupt. The invention is, of course, not limited in scope to these particular interrupts, nor the to use of any particular interrupts.

In such embodiments of methods in accordance with the invention, it is assumed that computing activities in VMs are typically interrupt driven, as was previously indicated, and that real-time scheduling of VMs is typically based on periodic interrupts. One embodiment of such a method for scheduling VMs may comprise the following. A user application, OS, API, or resource management application may communicate a resource requirement (X_i) and an interrupt period (Y_i) for each interrupt source, although the invention is not limited in scope in this respect. Alternatively, for example, each X_i and Y_i may correspond to a specific VM. In this respect, the subscripts, i, for X and Y may, for example, correspond to a respective interrupt or VM, although the invention is not limited in scope in this respect. For such embodiments X_i may be expressed as a percentage of the computing resources of a bare machine, such as 410 and Y_i may be expressed as the frequency at which the resource requirement should be allocated. For example, a resource management application, such as 475, may communicate to the VMM a

X_i of 20 and a Y_i of 10. This may then indicate to a scheduler embodied in such a VMM that it is desirable that this particular VM receive, for example, 2 microseconds (μS) of computing resources every 10 μS.

In such embodiments, a VMM would schedule VMs based, at least in part, on the various X_i and Y_i values communicated to it by, for example, a resource management application, such as 475. This scheduling may be accomplished by employing a number of different real-time scheduling techniques. Such techniques are well-known in the art and the invention is not limited in scope to any particular technique for scheduling VMs. For this particular embodiment, a VMM may suspend VMs and initiate allocations of computing resources to VMs in a substantially similar manner as was previously described with regard to embodiment 600.

FIG. 5 illustrates an embodiment of a schedule for a particular set of X_i and Y_i values for an embodiment employing three VMs, A, B and C. In FIG. 5, table 510 indicates the respective X_i and Y_i values for VMs A, B and C. Table 520 indicates an embodiment of a schedule that may result from a real-time scheduling technique employed in a VMM scheduler. For ease of explanation, Y_i values will be expressed in terms of μS. For this particular scenario, VM A's resource requirements, which may be referred to as X_A, are 20% every 10 μS, which, likewise, may be referred to as Y_A. Similarly, VM B's resource requirements, X_B, are 50% every 2 μS, or Y_B, and VM C's resource requirements, X_C, are 25% every 4 μS, or Y_C. Simplifying this scenario, VM A's real-time resource requirements are 2 of every 10 μS, VM B's real-time resource requirements are 1 of every 2 μS and VM C's real-time resource requirements are 1 of every 4 μS. The real-time schedule illustrated in table 520 satisfies the resource requirements indicated by these respective X_i and Y_i values and may, therefore, be used to schedule the respective VMs.

An alternative embodiment of a method for real-time scheduling of VMs in accordance with the invention that may employ an embodiment such as 400 may comprise the following. Rather than communicating both X_i and Y_i values from an OS, user application, API, or resource management application, such an embodiment may communicate X_i values in such a manner and employ a technique for determining the interrupt period, Y_i, such as the technique illustrated in FIG. 1. In this particular embodiment, 100, interrupts 115 are communicated from hardware

(H/W), e.g. a bare machine or devices 110, to a H/W interrupt virtualizer, such as 120. H/W interrupt virtualizer 120 may perform various functions. Of course, the invention is not limited in scope to any particular H/W interrupt virtualizer configuration. For this embodiment, however, H/W interrupt virtualizer 120 may filter known aperiodic interrupts, as was previously indicated as desirable. Such known aperiodic interrupts typically include interrupts from, for example, disk drives, keyboards and pointing devices, although the invention is not limited in scope to these particular aperiodic interrupts. H/W interrupt virtualizer 120 may also communicate virtualized interrupts to an appropriate VM. In this particular embodiment, as is typical when employing VMs, the VMs would handle such virtualized interrupts as though they were actual hardware interrupts. It is desirable that VMs handle virtualized interrupts as actual hardware interrupts, at least in part, due to the fact that VMs are typically configured such that it is not apparent to an individual VM that computing resources are being shared. H/W interrupt virtualizer 120 may communicate all interrupt not known to be aperiodic, such as previously discussed, to interrupt period detector (IPD) 130. Certain interrupts, of the interrupts not known to be aperiodic, may appear periodic for a given duration, then may change their period or become non-periodic. That is, these interrupts typically do not have a substantially consistent periodicity. IPD 130, for this embodiment, may be capable of rejecting such aperiodic interrupts, and therefore, the VMM may not consider such interrupts in scheduling VMs 140 and 150, for example.

For this particular embodiment, IPD 130 may then, in turn, iteratively determine Y_i based, at least in part, on interrupt information from H/W interrupt virtualizer 120, VM A 140 and VM B 150. In this regard, IPD 130 may comprise a state machine, such as 200 illustrated in FIG. 2. For this embodiment, such a state machine may be employed for each virtualized interrupt and may be implemented in software, although the invention is not limited in scope in this respect. It may be desirable, in employing such a state machine in this manner, to initialize the state machine with an approximate Y_i. Such initialization may decrease the time employed to converge on an actual Y_i for a given interrupt. This decreased convergence time may, in turn, improve the overall performance of a virtual machine scheme employing such an embodiment. State machines, such as 200, are well-known in the art,

although, of course, the invention is not limited in scope to any particular state machine configuration or to the use of a state machine at all.

As previously indicated, for this embodiment, the state machine illustrated in FIG. 2 may be employed to converge on an actual Y_i for a given interrupt. As was also indicated above, it is desirable to initialize such a state machine with an estimated Y_i value for each interrupt. For a given interrupt in such an embodiment, the state machine would initially be at state 0, indicated in FIG. 2 by element 240. If the expected Y_i value, which may be an initialized value, is less than an actual Y_i value, state machine 200 may then transition to state 1, indicated by element 250. For this particular embodiment, if the next actual Y_i matches the expected Y_i, then state machine 200 would persist in state 1, for this specific situation, while if the expected Y_i is greater than the next actual Y_i, the state machine would then transition back to state 0. However, if the expected Y_i is again less than the subsequent actual Y_i value, state machine 200 may then transition to state 2, indicated by element 260. In similar fashion, if the next actual Y_i value matches the expected Y_i, then state machine 200 would persist in state 2, for this specific situation, while if the expected Y_i is greater than the next actual Y_i, the state machine would then transition back to state 1. However, if the expected Y_i is again less than the subsequent Y_i, state machine 200 may then transition to state 3, indicated by element 270 and the expected Y_i may be adjusted upward and state machine 200 would then transition back to state 0. This upward adjustment could be a fixed value or alternatively a percentage of the previously expected Y_i. The invention is, of course, not limited in scope to any particular technique for adjusting the expected Y_i in such an embodiment. Continuing such an iterative process would allow IPD 130 to converge on an actual Y_i for periodic interrupts.

A substantially similar iterative process may occur for a situation when an expected Y_i is greater than an actual Y_i for a given interrupt with transitions to states -1, -2 and -3, respectively indicated by elements 230, 220 and 210. For this embodiment, such transitions would result in corresponding downward adjustments in the expected Y_i and transitions back to state 0. While the invention is not limited in scope to determining an expected Y_i value in this manner and alternative techniques may exist, this technique is advantageous as it may be tolerant of minor

variations in a periodic interrupt signal. Such variations are commonly referred to as signal jitter and are well-known to those of skill in the art.

Embodiments of IPDs, such as 130, may also employ a number of techniques for rejecting aperiodic interrupts. In such embodiments, such a rejection technique may override a Y_i determined by a state machine associated with an aperiodic interrupt and prevent IPD 130, in this particular embodiment, from communicating such a Y_i to a VMM scheduler, for example. In this specific situation, a schedule generated by a VMM scheduler, such as 180 may not depend on aperiodic interrupts, as is desired. Various techniques for rejection of aperiodic interrupts exist, and the invention is not limited in scope to any particular technique for such rejection. Alternatively, for example, such rejection may be performed by H/W interrupt virtualizer 120.

Embodiments for scheduling VMs in accordance with the invention that employ techniques such those described above with respect to FIG. 1 may schedule VMs using real-time scheduling techniques based, at least in part, on the X_i values communicated from VMs, as previously described, and Y_i values that may be determined by the foregoing technique. The invention is, of course, not limited in scope to the particular techniques described and alternative manners of determining Y_i may exist. Such embodiments may be employed, for example, when implementing a VM that may include a specialized instruction set. Though the invention is not limited in scope in this respect, such a VM may emulate, for example, a digital signal processor (DSP). Because such VMs typically have substantially static computing resource requirements, use of the foregoing embodiment of real-time scheduling of VMs may be advantageous as a static computing resource requirement may be communicated to a VMM by such a VM, which, for this embodiment, may be expressed as X_i.

While the embodiments described above may address the desire to allow real-time scheduling of virtual-machines, an interface for communicating X_i and Y_i may result in a custom implementation of both a VMM and any corresponding VMs implemented by such an embodiment. FIG. 3 illustrates an embodiment, 300, employing a VMM that may allow real-time scheduling of virtual machines without the addition of such an interface. This particular embodiment comprises a VMM that

includes a proportional integral derivative (PID) controller, 330, and an idle detector, 320, both of which may be implemented in software, for example. PID controllers, such as 330, are typically employed in feedback loops, such as 300, and are well-known by those skilled in the art. While the invention is not limited in scope to the use of any particular configuration of a PID controller, or the use of a PID controller at all, in this particular embodiment, PID controller 330 may be employed in determining a resource requirement value, X_i, for VMs A and B, designated 340 and 350 respectively.

In this particular embodiment, an idle detector 320 may also be employed to determine whether the VMs are achieving their real-time deadlines with the computing resources they are allocated. OSs typically employ an idle loop, and such loops are well known in the art. Such an idle loop may be indicated by a halt instruction (HLT), which then may be trapped by the VMM in an idle detector, for example. Such trapping may indicate to a VMM that a VM has encountered an idle loop and that, for example, an OS executing within the VM has no useful work to do. Of course, the invention is not so limited in scope and alternative methods of providing such an indication may exist. Alternatively, as one example, the VM may load a specific application that executes at relatively to low priority to, for example, any real-time applications. Such a specific application may then issue a predefined illegal instruction. Such an instruction may then, in turn, be used to indicate to the VMM that the VM has received a sufficient allocation of computing resources for real-time applications. In this respect, a VM may still have useful non-real-time work to do. Such a technique may also be employed to more efficiently allocate computing resources in certain embodiment or certain situations.

For this embodiment, if a VM does not reach the idle loop, that is does not issue a HLT, during its computing resource allocation, this may indicate to PID controller 330 an increase in computing resource allocation is desired for that particular VM. Conversely, if a VM reaches the idle loop, e.g. issues a HLT, before the completion of its computing resource allocation, this may indicate to PID controller 320 that a decrease in computing resource allocation is desired, for this embodiment. Although the invention is not limited in scope in this respect, it may be advantageous to target reaching an idle loop at some predetermined point during a

given computing resource allocation such as, for example, 95%. Such a target may be advantageous as it may allow real-time deadlines to be achieved while reducing the amount of unutilized computing resources consumed by idle loops. Also, employing methods such as the foregoing may reduce overall power consumption in virtual machine schemes employing such methods. This reduction in power consumption may result, for example, at least in part, from more efficient use of computing resources in such a virtual machine scheme.

In embodiments such as 300, it may be advantageous to initialize the PID controller with an estimated computing resource allocation. This may be advantageous as it may reduce the convergence time for such a feedback loop to establish a X_i that may allow VMs, such as 340 and 350, to achieve their real-time deadlines consistently. Reduction of convergence time for determining a specific X_i may also be advantageous as it may improve overall performance of a VM scheme employing embodiments such as feedback loop 300. Embodiments such as this are also potentially advantageous, as they may allow real-time scheduling of virtual machines while being substantially transparent to an OS executing in a VM. In this context, transparent means that an OS and any user applications running under that OS may not need additional software interfaces to communicate with, for example, a VMM, such as previously discussed with respect to embodiment 400.

An article in accordance with the invention may comprise, for example, a computer readable medium capable of storing instructions that when executed by a computing system may schedule VMs based on a resource requirement and an interrupt period, such as previously discussed. One embodiment of a configuration in which such an article may be employed is illustrated in FIG. 7, although, of course, the invention is not limited in scope to this, or any particular configuration. However, such articles may be employed as a memory, such as 730. In this particular embodiment, such memory, which is well-know to those of skill in the art, may comprise, as some examples, a compact disk read only memory (CDROM), a disk-drive, a digital versatile disk (DVD), static random access memory (SRAM); such as cache memory, or any number of various dynamic RAM (DRAM) configurations. Such DRAM may include synchronous DRAM (SDRAM), extended data-out (EDO)

DRAM, or double data rate (DDR) DRAM. The invention is, of course, not limited in scope to these particular configurations or examples of memory devices.

In the embodiment illustrated in FIG. 7, instructions stored on/in such a memory for scheduling VMs may be executed by the processor system, 720, of computing system 710. Computing system 710 is also coupled to the Internet, 710, via communications medium 740, although the invention is not so limited. Both the Internet and such communication media are well known in the art. Examples of such communication media may include, though are not limited to, modem connections, digital subscriber line (DSL) connections and broadband network access connections. In this respect, instructions, such as those previously described, may be obtained from the Internet via communications medium 740 for execution by processor system 720. Such instructions may be stored, either temporarily or substantially permanently in the memory and then executed, as was previously discussed. The invention is, of course, not limited in scope in this respect and alternatively, for example, such instructions may be downloaded from the Internet and executed without employing memory 730. In such a situation, a computing system included in the Internet from which such instructions are obtained may be analogous in function, for this embodiment, to memory 730.

While certain features of the invention have been illustrated and described herein, many modifications, substitutions, changes and equivalents will now occur to those skilled in the art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes as fall within the true spirit of the invention.

What is claimed is:

1. A method for scheduling a plurality of virtual machines comprising:

determining a respective resource requirement (X_i) for each virtual machine (VM);

determining a respective interrupt period (Y_i) for each VM; and

scheduling said plurality of VMs based, at least in part, on said respective X_i and Y_i values.

2. The method of claim 1 wherein, determining said respective X_i and Y_i comprises communicating said respective X_i and Y_i from an operating system (OS) running within said respective VM.

3. The method of claim 1 wherein, determining said X_i and said Y_i comprises communicating said X_i and said Y_i from an application running within an operating system (OS) running within said respective VM.

4. The method of claim 3 further comprising:

dynamically maintaining values for said X_i and said Y_i, wherein said application is a resource management application.

5. The method of claim 1 wherein, determining said X_i comprises communicating said X_i from an operating system (OS) running within said respective VM.

6. The method of claim 5 wherein, determining said X_i comprises communicating said X_i from an application running within an operating system (OS) running within said respective VM.

7. The method of claim 6, wherein said application is a resource management application, which dynamically maintains said respective X_i.

8. The method of claim 1, wherein determining a respective X_i comprises:

monitoring whether a VM reaches an idle loop;

increasing said respective X_i if said idle loop is not reached;

decreasing said respective X_i if said idle loop is reached before a predetermined percentage of said resource requirement has been utilized.

9. The method of claim 8, wherein determining respective Y_i values comprises:

filtering non-periodic interrupts;

rejecting aperiodic interrupts;

estimating said respective Y_i values for periodic interrupts; and

converging said respective Y_i values to be substantially equivalent to actual periods for said periodic interrupts.

10. An article comprising: a storage medium having stored thereon instructions that, when executed, result in a computing platform having the capability to:

schedule a plurality of virtual machines (VMs) implemented in said computing platform based, at least in part, on a respective resource requirement (X_i) and an a respective interrupt period (Y_i) for each VM of said plurality.

11. The article of claim 10, wherein said instructions, when executed result in the capability to communicate said respective X_i from an application running within a VM of said plurality.

12. The article of claim 11, wherein said instructions, when executed result in the capability to communicate said respective Y_i from an application running within a VM of said plurality.

13. The article of claim 10, wherein said instructions, when executed result in the capability to communicate said respective X_i and said respective Y_i from an operating system running within a VM of said plurality.

14. The article of claim 10, wherein said instructions, when executed result in the capability to communicate said respective X_i and said respective Y_i from a resource management application running within a VM of said plurality.

15. The article of claim 10, wherein said instructions, when executed result in the capability to determine said respective Y_i by comparing an expected interrupt period with an actual interrupt period and adjusting said respective Y_i based, at least in part, on said comparison.

16. The article of claim 10, wherein said instructions, when executed result in the capability to determine said respective X_i by detecting the occurrence of an idle loop within a VM of said plurality and adjusting X_i based, at least in part, on whether said idle loop occurs.

17. A method for determining interrupt period values comprising:
initializing said interrupt period values;
generating virtualized interrupts by virtualizing hardware interrupts;
filtering non-period interrupts;
rejecting aperiodic interrupts; and
adjusting said interrupt period values iteratively until substantially equivalent to actual interrupt periods.

18. The method of claim 17, further comprising:
acquiring resource requirement values; and
scheduling a plurality of virtual machines (VMs) to achieve real-time deadlines based, at least in part, on said interrupt period values and resource requirement values.

19. The method of claim 18, wherein said resource requirement values are acquired from said plurality of VMs.

20. The method of claim 17, further comprising determining resource requirement values, wherein determining said resource requirement values comprises:

initializing said resource requirement values; and

adjusting said resource requirement values iteratively based, at least in part, on a determination of an occurrence of a respective predetermined instruction.

21. The method of claim 20, wherein adjusting said resource requirement values comprises:

increasing said resource requirement values if said respective predetermined instruction does not occur;

decreasing said resource requirement values if said respective predetermined instruction occurs prior to a target time; and

scheduling a plurality of virtual machines (VMs) based, at least in part, on said interrupt period values and said resource requirement values.

22. An article comprising: a storage medium having stored thereon instructions that, when executed, result in a computing system having the capability to:

initialize interrupt period values;

generate virtualized interrupts by virtualizing hardware interrupts;

filter non-period interrupts;

reject aperiodic interrupts; and

adjust said interrupt period values iteratively until substantially equivalent to actual interrupt periods.

23. The article of claim 22, wherein said instructions, when executed, further result in the capability to:

acquire resource requirement values; and

schedule a plurality of virtual machines (VMs) to achieve real-time deadlines based, at least in part, on said interrupt period values and resource requirement values.

24. The article of claim 23, wherein said resource requirement values are acquired from said plurality of VMs.

25. The article of claim 22, wherein said instructions, when executed, result in said computing platform having the further capability to:

determine resource requirement values, wherein determining said resource requirement values comprises:

initializing said resource requirement values; and

adjusting said resource requirement values iteratively based, at least in part, on a determination of an occurrence of a respective predetermined instruction.

26. The article of claim 25, wherein adjusting said resource requirement values comprises:

increasing said resource requirement values if said respective predetermined instruction does not occur;

decreasing said resource requirement values if said respective predetermined instruction occurs prior to a target time; and

scheduling a plurality of virtual machines (VMs) based, at least in part, on said interrupt period values and said resource requirement values.

27. A system comprising:

a computing platform;

said computing platform being adapted to implement, at least, a virtual machine monitor (VMM) and a plurality of virtual machines (VMs);

said VMM being capable of scheduling said VMs to execute real-time applications based, at least in part, on a resource requirement (X_i) for each VM and an interrupt period (Y_i) for each VM.

28. The system of claim 27, further comprising:

an interface capable of communicating respective X_i and Y_i values for said each VM to said VMM.

29. The system of claim 27, wherein said VMM comprises:

a feedback loop capable of determining a respective X_i for said each VM;

a hardware interrupt virtualizer capable communicating device interrupts to said plurality of VMs and filtering non-periodic interrupts;

an interrupt period detector (IPD) capable of determining said periods for periodic interrupts and communicating said periods to a scheduler; and

said scheduler being capable of said scheduling of said plurality of VMs.

30. The system of claim 29, wherein said feedback loop comprises:

a detector capable of determining whether each of said VMs issues a predetermined instruction and indicating said determinations to a proportional integral derivative (PID) controller;

said PID being capable of adjusting said respective X_i for said each VM based, at least in part, on said determination and communicating said adjusted respective X_i to said scheduler.